University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Federated Learning for Dementia Classification in a European Multicentre Dementia Study

Master's Thesis in Robotics and Signal processing
by

Ruben Hesseberg

Petter Minne

Supervisors

Ketil Oppedal

Álvaro Fernández Quílez

July 15, 2020

Abstract

Every year around 10 million people are diagnosed with dementia worldwide. Higher life expectancy and population growth could inflate this number even further in the near future. Currently the diagnostic process of dementia relies heavily on medical experts on an individual basis. As the prevalence of the disease grows, so does the need for reliable diagnosis systems. Medical institutions around the world hold massive amounts of medical patient data. Large portions of this data can not be shared between institutions due to patient privacy concerns.

This thesis explores some solutions to these obstacles. Computer-aided diagnosis systems based on various deep neural networks trained on magnetic resonance imaging is investigated. The use of generative adversarial networks to generate usable samples for deep neural networks without compromising patient privacy is explored. A federated structuring of deep neural networks where patient data is kept locally is tested. Data for all experiments are based on a class-balanced dataset of 690 brain scans from patients diagnosed with Alzheimer's disease, dementia with Lewy bodies and normal control subjects.

An accuracy of 78.65% was achieved for a three class differentiation of 171 test subjects. This is a formidable result, especially compared to related deep learning based approaches. The generative adversarial network approach of generating new data achieved fairly good results, but due to memory limitations this data is of lower resolution and could not be used in the final evaluation. The federated structuring of deep neural networks yielded in part promising results and could be an important way of accessing medical data while protecting privacy in the future.

Acknowledgements

This thesis marks the end of our Master's degree in Robotics and Signal processing at UiS. We would like to give a special thanks to our supervisors Ketil Oppedal and Álvaro Fernández Quílez, for their advice and feedback during this semester. In particular, we are grateful for Álvaro being available to help us in our time of need long into the summer vacation. We also have to thank Simen Larsen for a helping hand with getting started with his code, and Theodor Ivesdal for his technical support during the quarantine, and making us able to work from home.

In addition, we want to thank our cohabitants for the support at home during the pandemic. I (Petter) wish to express my gratitude to my partner Ragnhild Austbø Kjønsøy for spellchecking the thesis multiple times and making some delicious dinners while I was working. And many thanks to Nora for being such a good dogger and providing Ruben with countless happy borks.

Finally, we want to thank our fellow students for two fun and exciting years, and a special thanks to ISI ("Interesseorganisasjonen for Sivilingeniørstudentene I Informasjonsteknologi") for providing us with coffee, good times and many fun games with Super Smash during this period.

Contents

Abbreviations

AD	**A**lzheimer's **D**isease
DLB	**D**ementia with **L**ewy **B**odies
NC	**N**ormal **C**ontrol
MRI	**M**agnetic **R**esonance **I**maging
DL	**D**eep **L**earning
CV	**C**ross **V**alidation
NN	**N**eural **N**etworks
DNN	**D**eep **N**eural **N**etwork
ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
SD	**S**tandard **D**eviation
GPU	**G**raphics **P**rocessing **U**nit
GD	**G**radient **D**ecent
SGD	**S**tochastic **G**radient **D**ecent
ReLU	**Re**ctified **L**inear **U**nit
BN	**B**atch **N**ormalization
FL	**F**ederated **L**earning
E-DLB	**E**uropean **D**ementia with **L**ewy **B**odies consortium
VAE	**V**ariational **A**uto-**E**ncoder
GAN	**G**enerative **A**dversarial **N**etwork

Chapter 1

Introduction

1.1 Motivation

An early accurate diagnosis for a patient can often mean the difference between life and death. In cases where diseases might not be fatal, an early diagnosis is often very important to improve quality of life for the patient. For serious medical conditions such as cancers and brain diseases, we rely on the use of medical imaging to aid in the diagnostic process. Some of the most commonly used types of medical imagery utilized for serious conditions is computed tomography(CT) and magnetic resonance imaging(MRI).[1]

The use of machine learning(ML) algorithms, specifically deep learning(DL) to aid in the diagnostic process based on medical imagery looks promising. There are however several challenges related to the access, quantity and privacy of the data needed to train a robust DL algorithm. In the world of machine learning and statistics, more data generally means a better model. In the field of medicine however, large sets of data can be very difficult to acquire. Medical centers possess large amounts of patient data, but a lot of this data cannot be shared across institutions due to privacy regulations.[2][3]

1.2 Problem Definition

The purpose of this thesis is to explore some of the proposed solutions to the problems surrounding data quantity and availability, mainly exploring a federated learning(FL) approach and the use of generative adversarial networks(GAN). This thesis will conduct experiments using a dataset of dementia patients, some of which are diagnosed with Alzheimer's disease(AD), some which are diagnosed with dementia with Lewy bodies(DLB) and some normal control(NC) samples.

1.3 Thesis division

This thesis is written in collaboration by two students: Ruben Hesseberg and Petter Minne. As a general focus, Ruben has worked mainly on a federated approach, and Petter has experimented extensively with GANs and various other augmentation techniques for improving the machine learning model. For clarification on which individual has worked on/written which part of this thesis a table is provided below.

Thesis division		
Section	Ruben Hesseberg	Petter Minne
1. Introduction	Yes	-
2.1 Dementia	Yes	-
2.2 MRI	Yes	-
2.2.1 MRI Markers	-	Yes
2.3 Preprocessing	-	Yes
2.4 Deep Learning	-	Yes
2.4.7 Optimizers	Yes	-
2.4.8 Activation Functions	Yes	-
2.4.9 Overfitting	Yes	-
2.4.17 Federated Learning	Yes	-
2.4.18 Federated Averaging	Yes	-
2.5 Software	Yes	-
2.5.5 Nipype	-	Yes
2.6 Previous work on detecting AD..	-	Yes
3.1 Reproduce Larsens results..	-	Yes
3.2 Data	-	Yes
3.2.3 Federated Learning Dataset	Yes	Yes
3.3 Models	-	Yes
3.3.1 Federated Learning Models	Yes	-
3.4 Augmentation	-	Yes
3.5 Existing Approaches/Baselines	Yes	-
4. Experimental Evaluation/Results	-	Yes
4.10 Federated Learning Exp. Setup	Yes	-
4.11 Federated Learning Exp. Results	Yes	-
5. Discussion	-	Yes
5.1.1 Federated Learning Dataset	Yes	-
5.6 Federated Learning	Yes	-
6.1.1 GAN and improve the existing..	-	Yes
6.1.2 Federated Learning	Yes	-
6.2.1 GAN	-	Yes
6.2.2 Visualizing the Model..	-	Yes
6.2.3 Federated Learning	Yes	-

Table 1.1: Thesis division table. *(The person credited with a chapter has also written all subchapters unless otherwise specified.)*

1.4 Thesis Outline

Chapter 2 - Background

The contents of chapter two will give an understanding of the different subjects, method and tools used as background theory and in development of the project. The various types of dementia diagnoses will be explained, as well as the machine learning techniques and software utilized in this thesis.

Chapter 3 - Materials and Method

Chapter three will cover the data used in this project, including how it is prepared and processed.

Chapter 4 - Experiments and Results

Chapter four will contain the experiments and results conducted during this project. The experiments will be listed in the order they were performed, this is useful for understanding the process as the grounds for later experiments and models may be based on earlier results.

Chapter 5 - Discussion

Chapter five will include a discussion around the results from chapter four and how it compares to related work and research.

Chapter 6 - Conclusion and Future Directions

Chapter six is the final chapter of this thesis and will contain a conclusion based on the results of conducted experiments. The final section will propose directions for future research.

Chapter 2

Background

2.1 Dementia

Dementia is an overall term for medical conditions which causes abnormal changes in the brain. These changes causes various degrees of decline in the cognitive abilities of the patient. The vast majority of dementia victims are elderly people, and the risk of being diagnosed with the disease increases with age. However, it should not necessarily be regarded as a normal part of the aging process, as many people in their 90s live with no signs of dementia. It is estimated that 5-8% of the population over the age of 60 has some sort of dementia, and up to half the population over 85 might have the disease in some form.[4] The most common form of dementia is Alzheimer's disease (AD) witch accounts for 60 to 70 percent of all dementia cases [5]. Other common types are vascular dementia and dementia with Lewy bodies.[6][7] There is currently no way to cure dementia, but there are ways to improve the lives of those who have it by temporary suppressing symptoms.[8]

As of 2020 it is estimated that around 50 million people live with some form of dementia. There are close to 10 million new cases every year.[5] The number of people with dementia is estimated to reach 82 million by 2030 and 152 million by 2050.[5] Dementia has notable social and medical care costs. The total global cost in term of GDP is estimated to be around 1.1% with even higher proportions in high-income countries.[5]

2.1.1 Alzheimer's disease

Alzheimer's disease(AD) is a chronic brain disease and is the most common form of dementia. Typical symptoms for AD are reduced short term memory, then later reduced long term memory. The diagnosis often comes after a combination of mental tests, blood tests and PET scans. Except for some rare inherited forms of AD there are no other known risk factors that statistical increases the chances for getting AD [9].

Alzheimer's disease is named after the man who discovered it, the german medical doctor Alois Alzheimer. In 1906 he noticed something unusual when he was examining the brain tissue of a woman who had died of a mental illness. She had suffered from memory loss, language problems, and unpredictable behavior. During the post-mortem examination Alzheimer found abnormal clumps and tangled bundles of fibers, now known as plaques and tangles.[10]

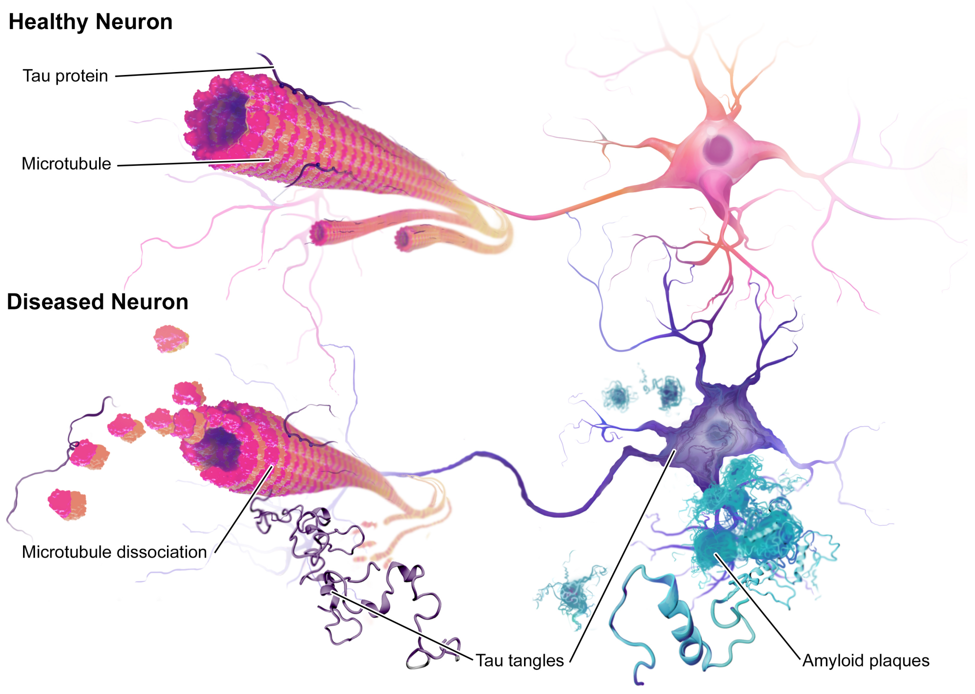


Figure 2.1: Illustration showing plaques and tangles interfering with the brain cells[11]

For patients with Alzheimer's disease, connections between nerve cells in the brain are lost. This occurs due to a buildup of proteins which causes abnormal structures referred to as plaques and tangles. Over time nerve cells die and brain tissue is lost. The brain contains chemicals to aid in the signaling between cells. Patients with AD produce less of some of these chemicals which reduces the communication between cells. Some drug treatments can help boost the production of some of these chemicals to reduce the severity of the symptoms. Alzheimer's is a progressive disease, and over time, more and more functionality of the brain will be lost.[12]

2.1.2 Dementia with Lewy bodies

Dementia with Lewy bodies(DLB) is estimated by most experts to be the third most common cause of dementia after Alzheimer's disease and vascular dementia, accounting for between 5 and 10 percent of all dementia cases.[7] Lewy bodies are found in 10 to 15 percent of post mortem examinations of dementia patients.[13] DLB is associated with a protein called alpha-synuclein being abnormally deposited in the brain. These deposits are called Lewy bodies and affects chemical processes in the brain, which in turn may lead to problems with thinking, movement, behavior and mood in patients.[14] Lewy body deposits are named after Fredereich H. Lewy, a neurologist who discovered them while working in Alois Alzheimer's laboratory during the early 1900s. Lewy bodies are not exclusively found in patients with DLB, but also in patients with AD and Parkinson's disease dementia.[7]

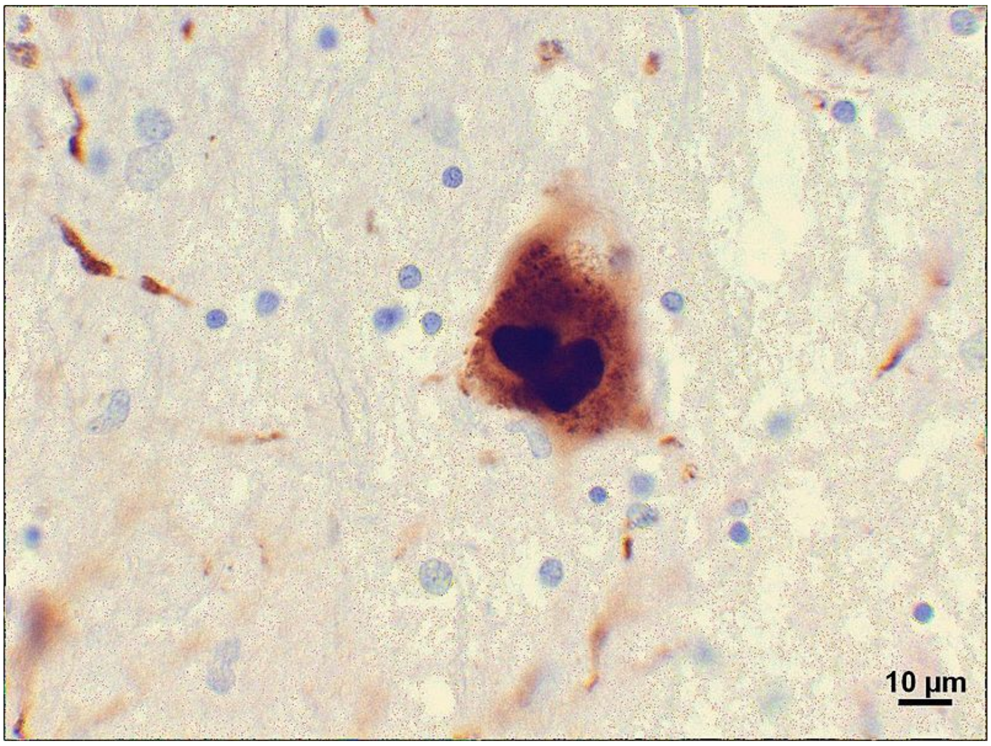


Figure 2.2: Biopsy showing lewy body deposits in the brain[15]

Overlapping symptoms with other brain diseases can make accurate diagnosis difficult, especially during early stages of the disease. DLB is also not mutually exclusive with other brain diseases, so comorbidity can occur in patients, further complicating the diagnosis and treatment process. The disease does not seem to run in families, although this might happen in very rare cases.[16]

2.2 MRI

To differentiate the different types of dementia or NC brains, Magnetic resonance imaging (MRI) is used. MRI is a nonintrusive way to inspect the subjects brain. In short the MRI produces a 3D image of the brain. It utilizes technology which excites and detects change in the direction of the rotational axis of protons in water molecules found in organic tissue. Powerful magnets are employed in MRIs to produce a magnetic field that forces the protons in the body to align with the magnetic field. Radiofrequency current is then pulsed through the subject, the protons are stimulated and spin out of equilibrium, staining against the magnetic field. When the radiofrequency current is turned off, MRI sensors detect the energy released as the protons realign with the magnetic field. The energy released and the time it takes for a proton to realign with the magnetic field changes depending on the chemical nature of the molecules and the surrounding environment. Physicians are able to distinguish between differing types of tissue based on these observed properties.[17]

2.2.1 MRI Markers

There are no known sets of biomarkers in the MRI images which are good enough to make a confident diagnosis of a patient with either AD[18] or DLB[19]. Both AD and DLB are in general characterized by atrophy throughout the brain, this can be seen by comparing the different diagnosis in figure 2.3.

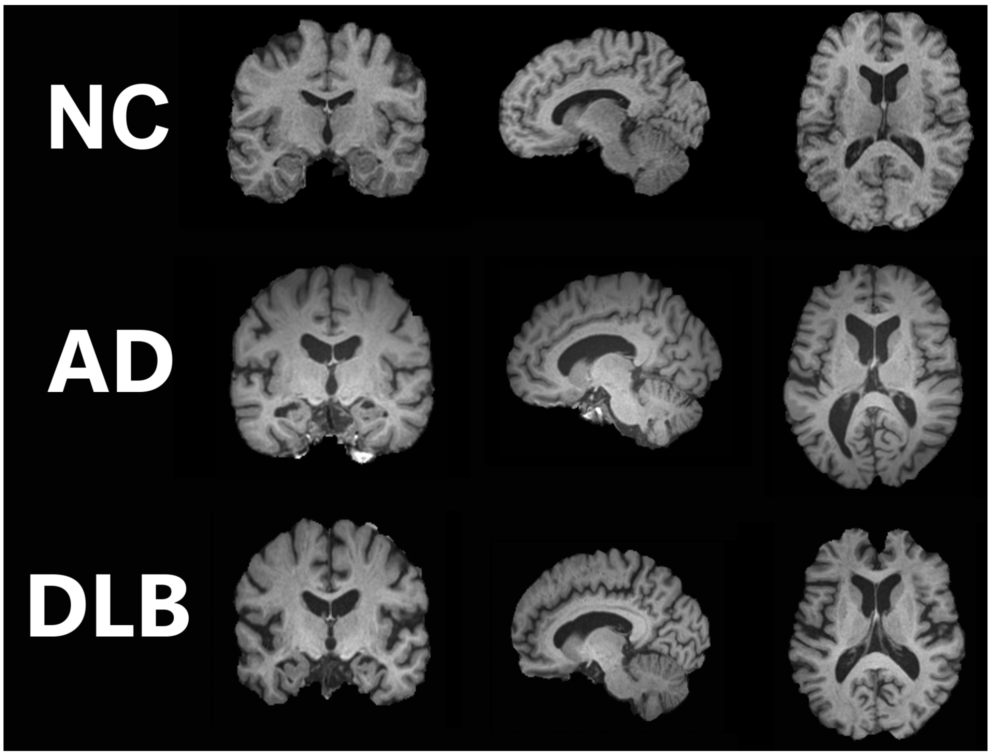


Figure 2.3: MRI scan of NC, AD and DLB brains

It has been shown that in parts of the brain in AD subjects, especially in the medial temporal lobe, there is more atrophy than the DLB subjects [20] [21].

As demonstrated in this paper[22], both AD and DLB showed significant atrophy in the hippocampus relative to NC. But the DLB group had significantly lower rates of atrophy in the CA1, and fimbria compaired to the AD subjects, see figure 2.4.

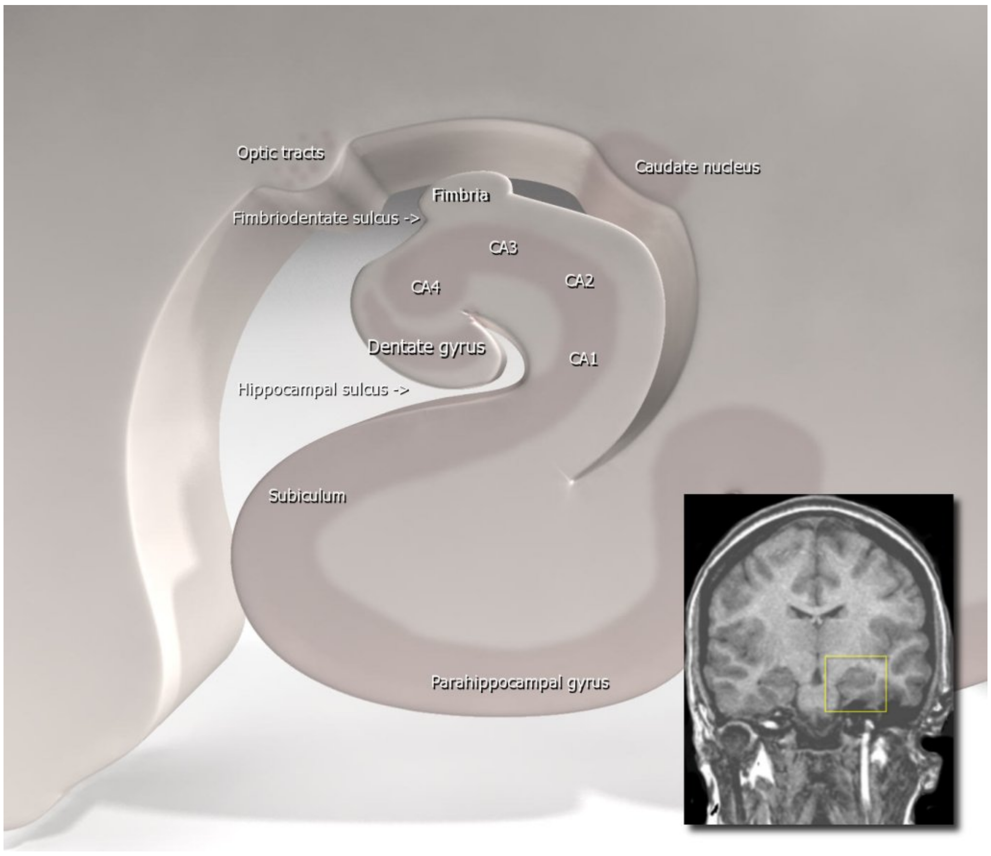


Figure 2.4: Highlighted Hippocampus with detailed anatomy. Case courtesy of Assoc Prof Frank Gaillard, Radiopaedia.org, rID: 10770

2.3 Preprocessing

The main objective of preprocessing is to reduce the irrelevant information from the data and make the relevant information easier to analyze. In machine learning, this is a crucial step to ensure the data is "clean" so that when an algorithm learns to recognize patterns in the data, these patterns are relevant to the problem the algorithm is trying to solve. As an example, when training a machine-learning algorithm to differentiate between images of apples and oranges. If the apples are centered, and the oranges are shifted to the lower left side of the pictures. The algorithm would learn that if an object is centered it must be an apple, and if an object is positioned to the lower left it must be an orange. Where the object is positioned is not a desired pattern for the algorithm to use when differentiating apples and oranges, and that is why preprocessing is essential.

2.3.1 Spatial Normalization

Spatial normalization is a procedure that normalizes how the brains are presented in 3D space. The spatial normalization procedure does this by reshaping all the brains in the dataset to a standard template, and then centering them. This means that one location in one brain corresponds to the same location in all the other brains. The procedure makes all the brains the same size and has the same position in the coordinate system.

2.3.2 Brain Extraction

Brain extraction is the process of removing any part of the MRI scan that is not brain matter. This is a crucial step to minimizing the irrelevant data of the MRI images. The brain extraction is also referred to as skull stripping.

2.3.3 Data Normalization

When training a neural networks(NN) it is common to normalize the data before using it in training. The normalization process makes all the values of the data to have a mean of zero and a unit standard deviation. The process makes the data easier for a model to learn relevant patterns.

2.4 Deep Learning

2.4.1 Artificial Neural Networks

The Artificial Neural Network (ANN) is a popular computer framework inspired by the brain's biological nervous system. The nervous system in the brain consists of a network like structure of many interconnected neurons. The neurons receive signals from their neighboring neurons, which they process before they pass it on. These biological networks are capable of learning numerous different things and perform a variety of complicated tasks. The artificial neuron mimics the biological neuron as it takes numerous inputs(x), which are individually weighted ($\mathrm{w_k}$), sums them, and processes them through an activation function(φ), see figure 2.5. (The matematical function i shown in equation 2.1).

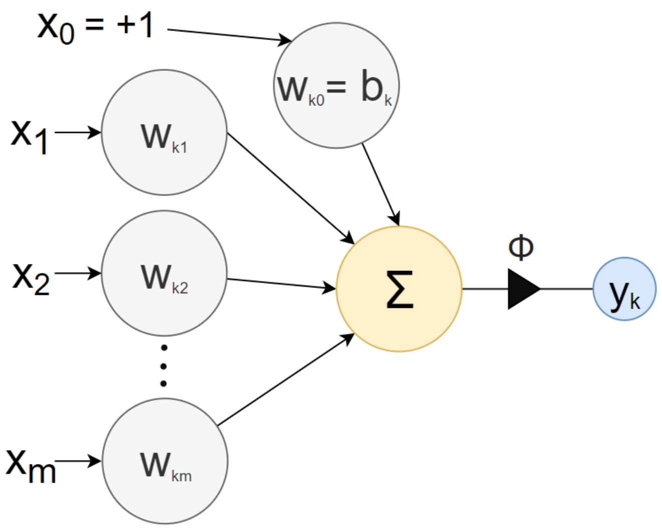


Figure 2.5: Ilustration of an artificial neuron. x = inputs, w_k = weights, φ= activation function, x_0 = +1 which makes is a bias with $w_{k,0} = b_k$

$$y_k = \phi \sum_{j=0}^{M} (w_{kj} x_j) \tag{2.1}$$

An ANN is composed of these neurons which are interconnected in a network structure called the "Hidden layers", illustrated in figure 2.6.

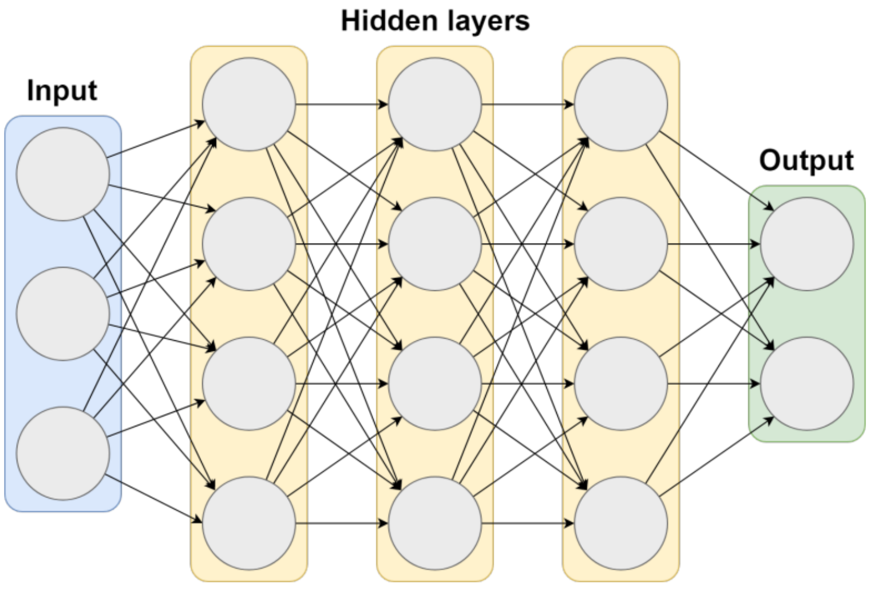


Figure 2.6: Illustration of an ANN structure.(*Picture is from [23] used with Larsens consent).*

When training an ANN, forward propagation is used to test the model, and backward propagation is used to learn from the test results. In the forward propagation, the network maps the input data through the hidden layers out to the output layer, where the nodes conclude what the data should be classified as. The backward propagation then calculates the loss(chapter 2.4.5) of the network, and with the use of an optimizing function(chapter 2.4.7), the training goes backward through the network and calculates new/updated weights in all the neurons to make the next prediction better than the last.

2.4.2 Convolutional Nerural Networks

Convolutional neural networks (CNN) are ANNs that are popular to use when analyzing images. CNNs are good at finding patterns in the data and make sense of them. They does so by using convolutional layers in the hidden layers of an ANN.

convolutional layers

A convolutional layer consists of one or more filters used to convolve over the input to calculate a convolved feature, figure 2.7. The filters use a set size and stride that is defined when creating the CNN. Zero padding is often used to avoid size-reduction when convolving over the input, see figure 2.8. The convolutional layer's output is referred to as a "feature map", which is passed on to the next layer. The convolutions at the start of the hidden layer are usually simpler filters that detect basic shapes. The deeper layers go, the more complex features the filters learn to detect. When applying backpropagation to a CNN, the weights of the filters are updated.

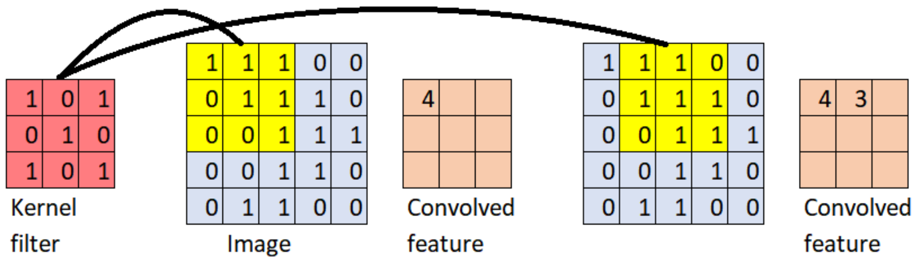


Figure 2.7: Convolution operation on a 5x5 image, with a filter of 3x3 and a stride of one

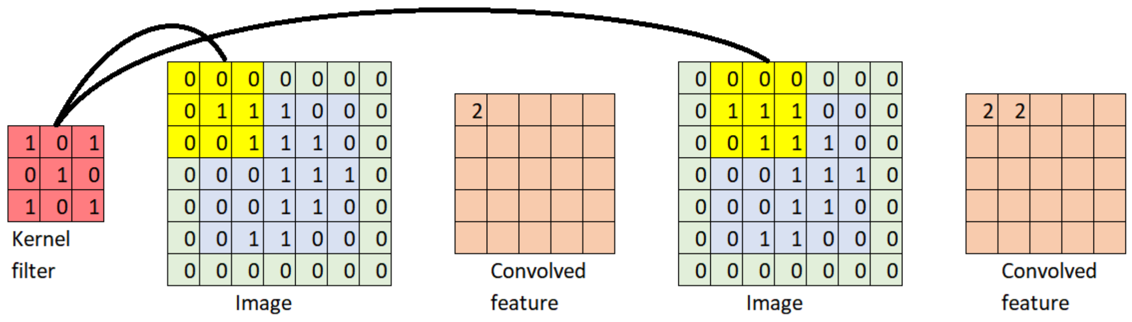


Figure 2.8: Convolution operation on a 5x5 image, with a filter of 3x3 and a stride of one. The green zeroes in the "image" matrix are the padding that is applied

2.4.3 Pooling Layer

After a convolutional layer, it is common to apply a pooling layer. The pooling operation downsamples the input data by selecting a region given height and width and outputting a single desired value from it. The most common pooling functions are average pool (which takes the average of all the values in the region and outputs it) figure 2.9, and maxpool (which takes the highest value in the region and outputs it) figure 2.10. The purpose of the pooling layer is to reduce the dimensions of the feature maps, thus reducing the memory consumption and computational strain on the system.

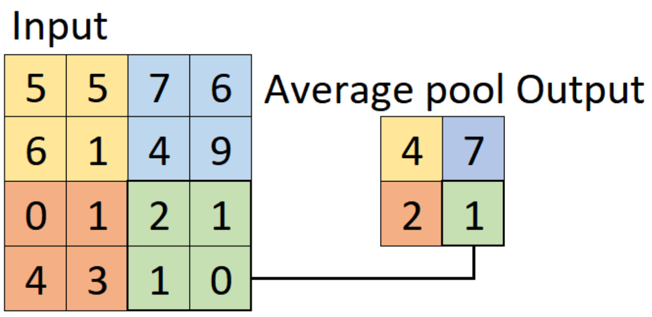


Figure 2.9: Average pooling with a 2x2 region and a stride of 2. The line from the green part of the input to the green part of the output is to illustrate that the output is calculated from this part of the input

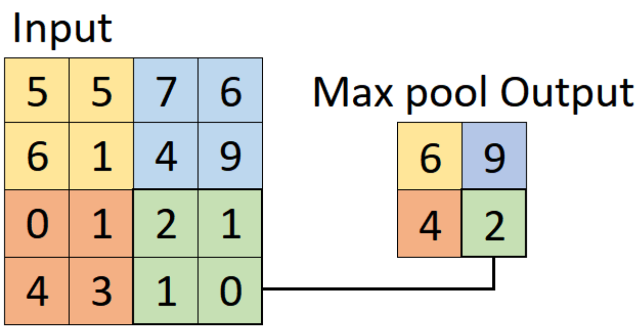


Figure 2.10: Max pooling with a 2x2 region and a stride of 2. The line from the green part of the input to the green part of the output is to illustrate that the output is calculated from this part of the input

2.4.4 Fully Connected Layer

In CNNs the last layer is usually one or more Fully Connected (FC) layers. The feature maps from the convolutional layers are flattened into a vector and used as input in the

FC layer. The FC then maps the vector to the correct outputs. The number of FC layers added to the end of the CNN may vary from different architectures. Basha et. al [24] concluded that deeper CNN architectures need fewer FC layers with fewer nodes than shallower architectures.

2.4.5 Loss Function

When training an ANN a metric is needed to quantify how well the model performs when training. The loss function does this by comparing the output of the model with the desired output and then quantifies how successful/unsuccessful the model was in its prediction. There are many ways to calculate the loss, and the loss function that is the best depends on the application. For classification problems the most common loss function to use is cross-entropy.

Cross entropy

The cross-entropy (equation 2.2) loss function calculates the model's loss based on how confident the model was in its prediction.

$$CE(t,p) = -\sum_{c=1}^{M} t_{o,c} \log(p_{o,c}) \tag{2.2}$$

"t" is the target vector containing the desired output and "p" is the model's output probability for a given class.

2.4.6 Batch Normalization

The batch normalization (BN) uses the same principles as explained in data normalization (chapter 2.3.3), except it is added to the feature map of a convolutional layer. Unlike the normalization of the input, BN data does not necessarily have a mean of zero and a standard deviation of one, because this is not always desirable inside the network. The BN, therefore, has two parameters (one for the mean and one for the standard deviation) that are scaled with the training when doing backpropagation.

2.4.7 Optimizers

Optimizers update the weights of all the nodes in the network to minimize the loss of the model. The loss function guides the optimizer to where it needs to go. There are many different optimizers in use, but in this thesis the stochastic gradient descent (SGD) was mainly used because of its better generalization [25]. If the model struggled to perform while training the Adam function was used instead of the SGD.

Stochastic Gradient Descent (SGD)

Stochastic gradient decent is an established optimizer which is based on the gradient decent algorithm.[26] Gradient decent based algorithms is by far the most common optimization method for neural networks. Gradient decent computes the gradient of the cost function with respect to the parameters of the entire dataset. In contrast, SGD performs a parameter update for each sample $x^{(i)}$ and label $y^{(i)}$.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}). \tag{2.3}$$

SGD performs frequent updates with high variance which causes the function to fluctuate a lot. This rapid fluctuation enables jumping to new potential minima's quickly. However, it also complicates the convergence to the exact minimum due to overshooting. Decreasing learning rate can help counter this issue.

Adam

Adam is a gradient based optimizer specifically designed for DNNs. It uses squared gradients to scale the learning rate and takes advantage of momentum by using moving average of the gradient.[27] Compared to SGD with momentum, inclusion of squared gradients makes the algorithm more robust to large relative differences between derivatives of system parameters. The adam optimizer can achieve significant performance gains compared to the SGD optimizer, however, this will not always be the case and it can indeed perform worse in certain instances.

2.4.8 Activation Functions

The activation functions add non-linearity in the neurons, this is what makes the network able to learn complex non-linear functions.

ReLU

The Rectified Linear Unit(ReLU) is a widely used activation function in deep learning models. The function returns 0 for negative input values, but positive input values are returned unchanged.[28] The ReLU function can be expressed like this: $\boldsymbol{f(x) = max(0,x)}$. The ReLU6 activation function, shown (2.11), is identical to ReLU, except that it limits the maximum output value to 6.

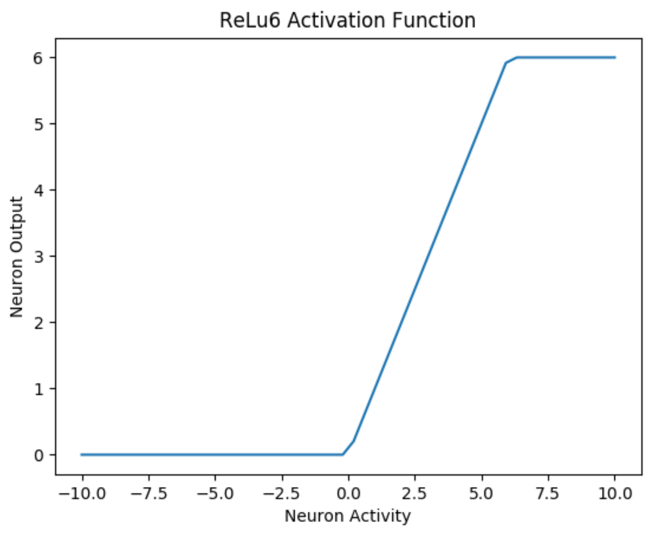


Figure 2.11: ReLU6 activation function

2.4.9 Overfitting

The term overfitting in machine learning is used to describe scenarios where the overall cost becomes small, but the generalization of the model is unreliable.[29] In other words, the model becomes extremely good at guessing correctly on the validation set, but will lose accuracy in a test set, because it is so specialized. An example of overfitting can be seen in (2.12). The validation error should be the global minimum in a balanced model, this is not the case here as seen from the figure.

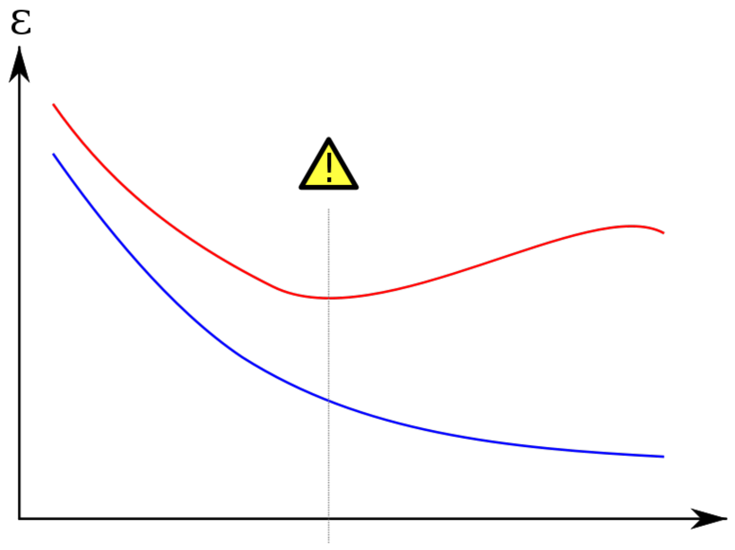


Figure 2.12: Overfitting example.[30] Training error in blue, validation error in red.

2.4.10 Dropout

Dropout is a method that introduces randomness to the DL model by randomly discarding nodes in the network. The amount of nodes the dropout function discards is manually set in as when designing the model. Too much dropout will lead to bad performance because the model fails to learn due to losing to many vital nodes. When tuned correctly, dropout has proven to lower overfitting and improve generalization in many applications**??**.

2.4.11 Augmentation

Data augmentation is the process of altering the available data in ways that do not change its ground truth. This is useful when there is a limited dataset, and new data is hard to gather. Augmenting the data will often lead to improved generalization because the model will learn to detect the data in more scenarios than before. Some simple augmentation techniques are rotating, mirroring, and translating.

2.4.12 K-Fold Cross Validation

Normally when training a model a part of the training data will be used for validation. K-Fold Cross Validation (CV) is a method to ensure that all the training data, even the part that is used in the validation process, is used to train the model. It does so by splitting the training data into K folds, then train the model on all the folds except one which is used for validation. It then trains the model again for each combination of the

K folds. This makes the reported results more robust because the results can report the average of the K models from the CV.

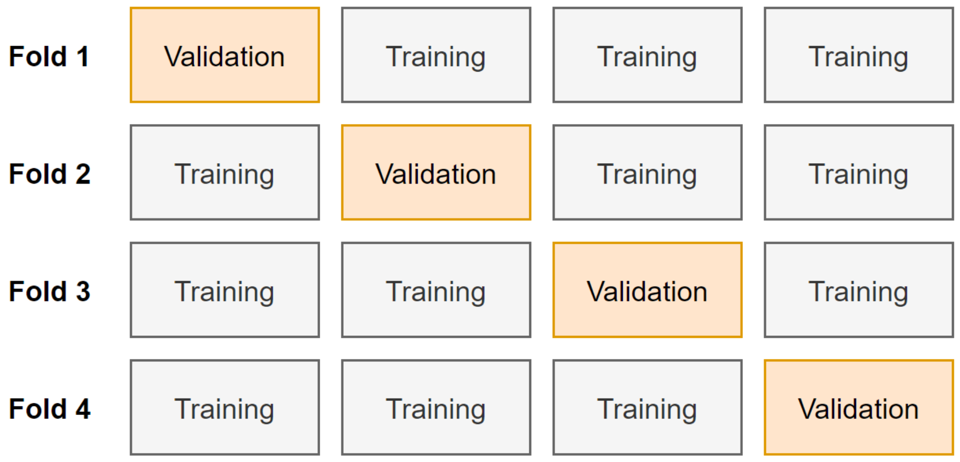


Figure 2.13: 4-Fold Cross Validation example.

2.4.13 hyperparameters

Before training a DL model, some hyperparameters needs to be set, like the learning rate, the number of different layers in the model, the amount of dropout, the parameters in the optimizer, etc. These parameters do not change while training, so it is vital for the DL model that these parameters are well chosen. The difference in training a DL model with bad and good hyperparameters are substantial. There is no correct way of finding the best hyperparameters for a model, but there are different methods that help to search for them methodically.

Manual Search

Manual search is the method of manually inputting the hyperparameters, testing them on the DL model, and trying new values. This is a very time-consuming method as it depends on manually plotting inn new parameters restarting the DL model.

Grid Search

Grid search is a traditional way of searching for the right hyperparameters. It works by making grids/list of all the different parameters to test, and then it iterates through all possible combinations of these while reporting the results.

Random Search

It is a method that randomly chooses the hyperparameters, then trains the model, and logs the result before doing the same again in a loop. This method can outperform grid search, especially if there are a low number of hyperparameters used[31].

Bayesian Optimization

Takes advantage of the information the model learns during the optimization process. The idea is that the Bayesian optimization has some prior beliefs about how the different hyperparameters affect the training outcome. The optimization uses these prior beliefs to make an educated guess when choosing new hyperparameters to test. Based on the latest test results, it updates its prior beliefs and makes a new educated guess, and does this until it converges. In short, the Bayesian optimization remembers all the previous hyperparameters and then chooses to test new hyperparameters close to where it previously has shown to increase the performance.

2.4.14 Models

SimenNet

SimenNet is a model designed by Simen Larsen [23]. The model is designed with six convolution blocks and three linear blocks, see figure 2.14 for details. Each of the convolution blocks uses the 3D convolution layers followed by max pool, ReLU, batch normalization, and dropout. When moving through the layers, more filters were used in the convolutions as the feature map decreased in size from the max pooling. The model ends with three fully connected blocks that interpret the output from the last convolution block and determine the diagnosis.

ResNet

The ResNet model was proposed by Kaiming He et. al in 2015 [32]. The model won first place on the ILSVRC 2015 classification task and won several other first places in the COCO 2015 competitions. Usually, deep neural networks will get better by stacking more layers on top of each other, but at a certain point the model accuracy will drop. The reason why this is happening is not clear, but Kaiming He et. al assume that the deep plain nets may have exponentially low convergence rates, which impacts the reduction of the training error. The ResNet model avoids this problem by using a reference from the

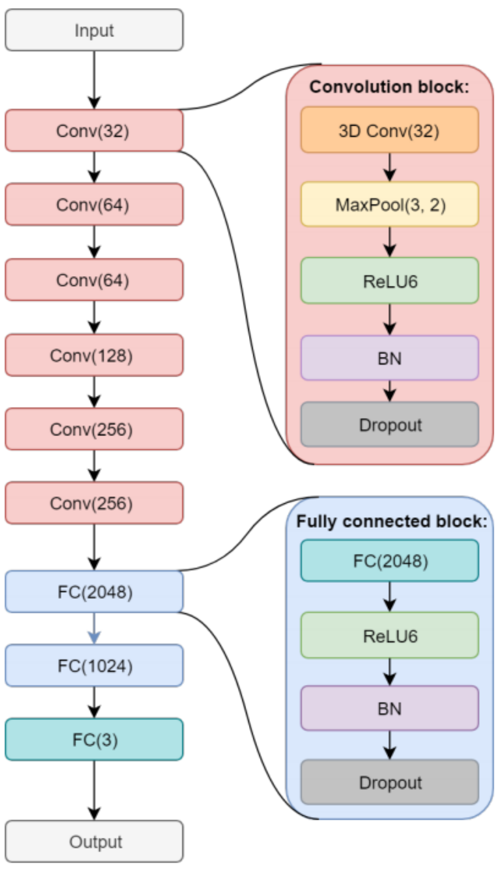


Figure 2.14: SimenNet network structure [23].

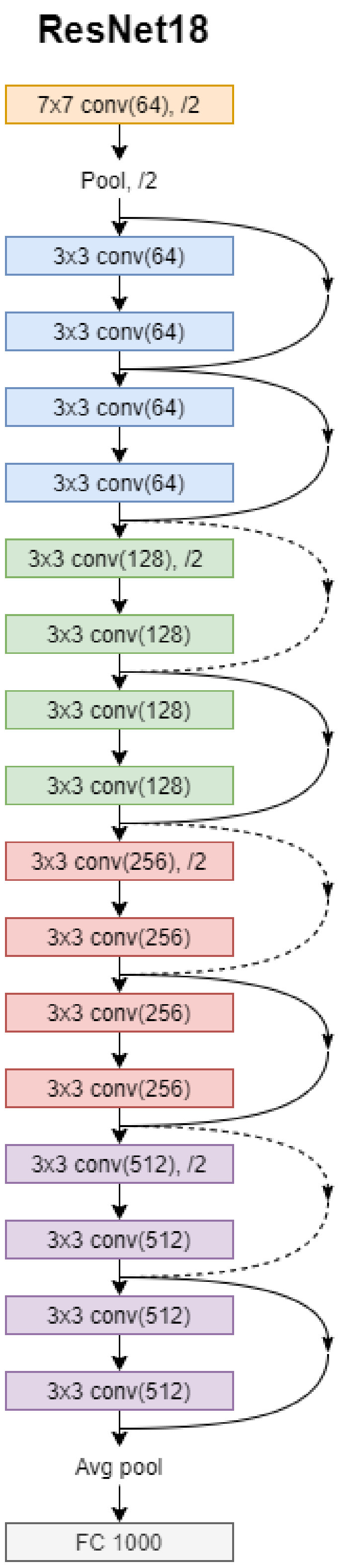


Figure 2.15: ResNet18 network structure. The dashed lines represent a skip connection with dimension matching.

previous layer and adds it to the current layer. This makes the model stack many more layers on top of each other without the descending accuracy problem.

The reference in the ResNet layer is shown in figure 2.17, as the single line that goes from X to the summation. The output of that block is calculated as Y = f(x) + X, where Y is the output, X the input, and f(x) the mapping of the layers. In a plain net the output is Y = f(x) see 2.16. The intuition behind this is that instead of expecting a few stacked layers to fit a desired underlying mapping X directly, these layers explicitly fit a residual mapping by adding X in the output.

In the ResNet18 model, the reference called "skip connection" skips over blocks of two layers at a time. When the dimensions of the blocks change, the skip connection needs to match the new dimensions so by performing a linear projection to the shortcut connection to match the dimensions. Figure 2.15 visualises the ResNet18 model.

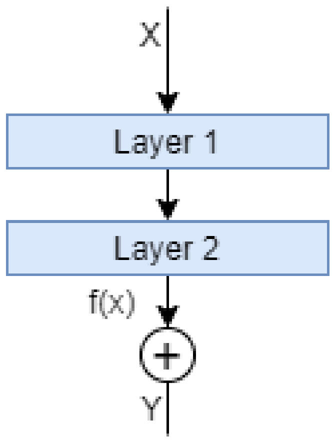


Figure 2.16: PlainNet calculation.

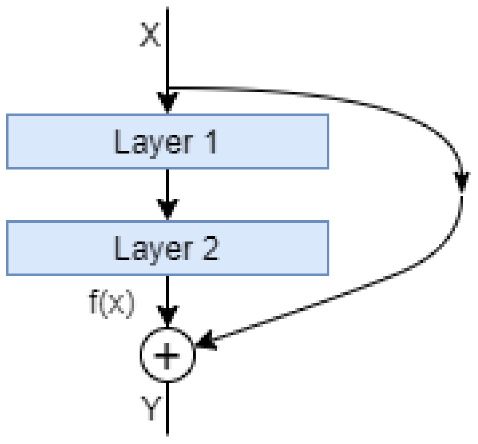


Figure 2.17: ResNet calculation.

2.4.15 Evaluation Metrics

Loss is used when training a model. The final value the loss has after training is not very interesting, but, the graph of the loss values during training is useful to get an overview of how the model performed while training and to spot overfitting.

Accuracy is used for measuring the accuracy of the model. It takes all the correct predictions and sum them together and divides by the total amount of guesses 2.4.

$$Accuracy = \frac{Number\ of\ Correct\ Prediction}{Total\ Number\ of\ Prediction} \tag{2.4}$$

Accuracy is not always the best evaluation and often miss details. This is especially the matter if there is a class-imbalanced dataset, or if misclassifications on one class have more severe consequences than misclassifications on another class.

Precision is used to measure how precise the model is at predicting one of the classes correctly. This is useful to see if the model is overpredicting it 2.5.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{2.5}$$

Recall is used to measure how accurate the model is to classify one of the classes correctly 2.6.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{2.6}$$

F1 is the harmonic mean of precision and recall. This is useful when the precision and recall is equally important 2.7.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.7}$$

Confusion matrix is a matrix that is used to show the performance of a model by showing all the predictions the model made on known test data. An example of a three classed confusion matrix is shown in figure 2.18. This is useful to see precisely how the model makes its predictions, and it gives a good overview of where the model excels and where it struggles. All the metrics explained above (F1, accuracy, precision, and recall) can be derived from the confusion matrix.

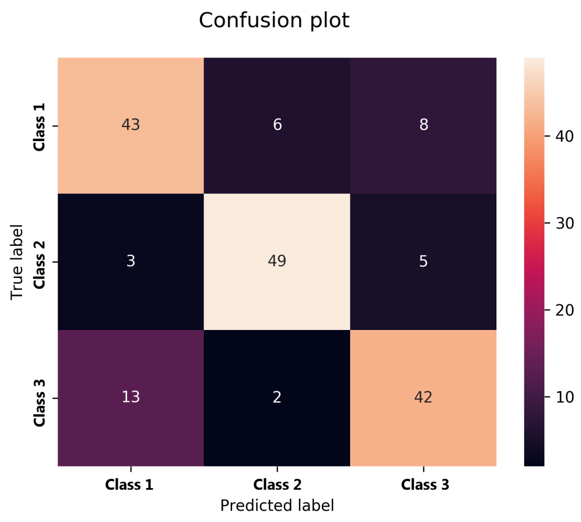


Figure 2.18: Example of a confusion matrix.

2.4.16 Generative Models

Generative Adversarial Network

Generative Adversarial Network (GAN) is a ML framework invented by Ian Goodfellow et. al [33]. GAN trains two neural networks simultaneously by plotting the two neural networks against each other, where one of the networks is called the generator and the other the discriminator. The generator generates fake data from a random input, while the discriminator distinguishes the real data from the generated fake data. The generators objective is to generate fake data that is good enough to deceive the discriminator. For each round of training, both the discriminator and generator learn by updating their weights through backpropagation. When the discriminators error rate is at 50%, the training should be done because the generator is then outputting data that is indistinguishable from the real data.

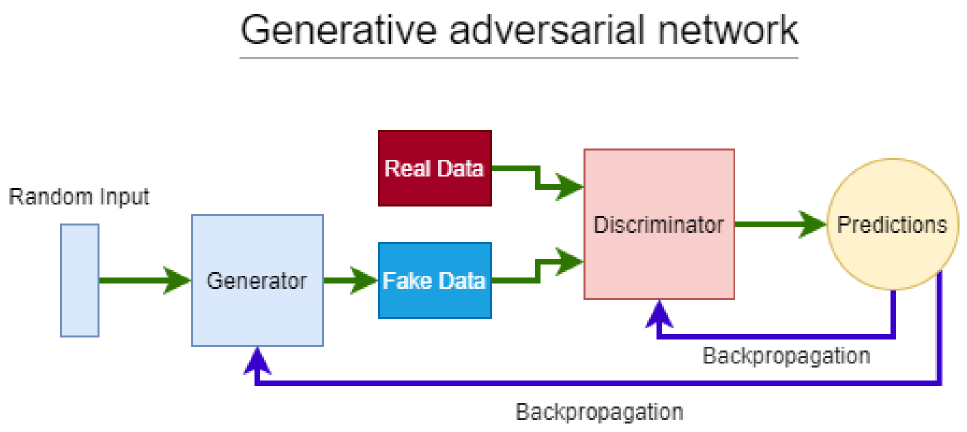


Figure 2.19: GAN flowchart.

GANs have proven to be useful in generating numerous kinds of data [34] [35], and it has been used a lot in upscaling/improving images with lower quality [36] [37]. However many GAN models suffer from non-convergence (when the model parameters oscillate and never converges), mode collapse (when a generator collapses and can only produce a limited number of new unique data samples before it repeats itself), and diminished gradient (when the discriminator is too good, and the generators gradient becomes too small). Unbalance between the training speed of the generator and discriminator can also cause the GAN model to overfit [38].

Variational Auto Encoder

An autoencoder consists of two networks, one encoding network, and one decoding network. The encoder network takes input data through its convolutional layers and converts it to a much more compact representation in the bottleneck (also called the latent vector). The decoder network uses the compact representation of the data as input and tries to recreate the original data. The autoencoder then compares the output result with the original input to calculate the reconstruction loss and updates the two models according to how good the recreation was. This makes autoencoders very good at reconstructing data, and have shown good results in compression and denoising applications. See figure 2.20 for a visual representation of an autoencoder.

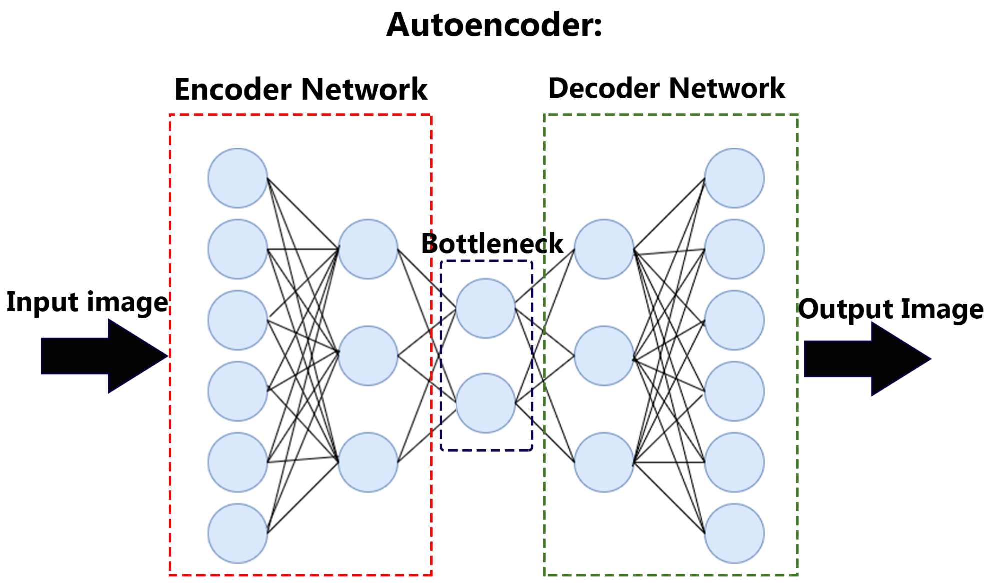


Figure 2.20: Illustration of an auto-encoder.

Variational Auto Encoder (VAE) has an encoding network that produces two vectors. One that represents the input data mean and one for the input data standard deviation, see figure 2.21. From these two, a sampled latent vector is made by taking a random sample from the standard deviation vector and adding the mean vector. This creates a new latent vector that closely resembles the latent vector of the original input, but it is a little different because the mean it is summed with the random sample from the standard deviation vector. The decoding network then decodes the sampled latent vector and outputs the new data [39].

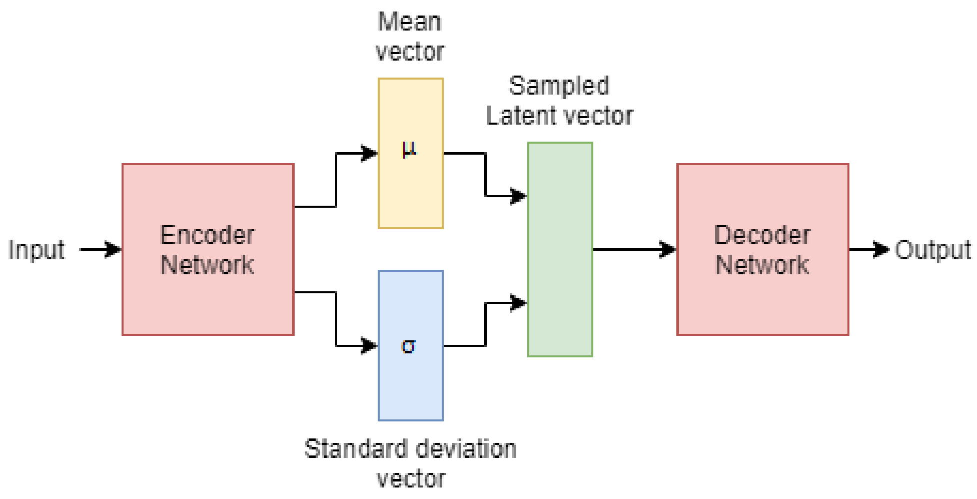


Figure 2.21: Illustration of a VAE.

α-GAN

When generating 3D images with a GAN, the complexity becomes much higher than with 2D images. This makes the 3D generation struggle with the mode collapse problem. The VAE, on the other hand, is free from the mode collapse, but it struggles with the output being blurry. To overcome the issues GAN and VAE have, α-GAN [40] is used. The α-GAN combines GAN and VAE by replacing the variational inference in the VAE with a discriminator network and then using both "random noise" and the encoder's output as input to train the generator. Because the generator now both reconstructs data from the encoder and generates data from the "random noise", it can be optimized using both the reconstruction loss and the discriminator loss hence avoiding the mode collapse problem. The α-GAN consists of four networks: a generator, an encoder, and two discriminators, see figure 2.22. One of the discriminators discriminates between the output of the encoder network and the "random noise" vector, and the other one discriminates between the data generated by the generator and the real data. The networks alternate between updating the parameters of the four network weights by minimizing the different loss functions.

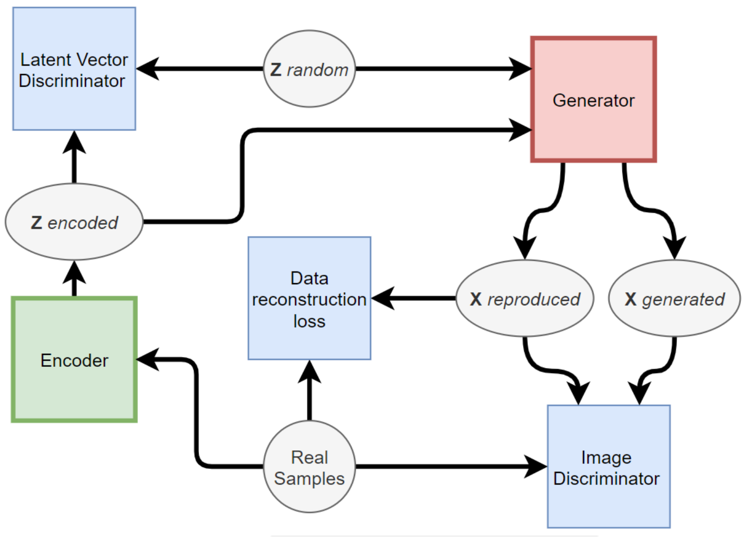


Figure 2.22: α-GAN structure. **X***reproduced*: samples reproduced by the generator from encodings produced by the encoder. **X***generated*: samples produced by the generator given a "random" vector. **Z***random*: samples from the latent-generating distribution (random noise). **Z***encoded*: vectors produced by the encoder given a real sample.

2.4.17 Federated Learning

Federated Learning (FL) is a machine learning technique that aims to resolve some of the concerns and restrictions about data and user privacy when accessing data for training machine learning algorithms. Federated learning trains an algorithm across multiple decentralized devices or servers that holds local data samples, without directly accessing the data. A centralized server maintains the global neural network and each device or server connected to this central server is given a copy to train on their own dataset. When the model has been trained locally for a number of iterations, the participating servers or devices will send their updated model back to the centralized server. The central server will then aggregate contributions from from all participating nodes, thus creating a new updated global neural network which can be shared with the participating nodes again.[41][42][43][44][45]

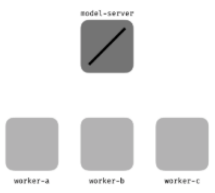
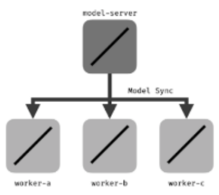
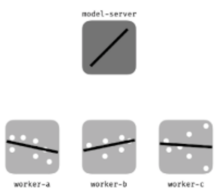
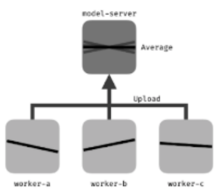
Step 1	Step 2	Step 3	Step 4
			
Central server chooses a statistical model to be trained	Central server transmits the initial model to several nodes	Nodes train the model locally with their own data	Central server pools model results and generate one global mode without accessing any data

Figure 2.23: General Federated Learning Process[46]

2.4.18 Federated Averaging

Federated Averaging is a function commonly used in federated learning implementations. The function is responsible for calculating new weights for the global model. Other approaches for generating the federated model exist, however these are not utilized in this thesis, but will be mentioned in the discussion chapter. The right side of the equation is estimating the weight parameters for clients based on loss values. On the left side of the equation each parameter is scaled and summed up component wise. [47]

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w). \tag{2.8}$$

w is the model parameters. ***K:*** is the total number of clients. ***k:*** is the index of the clients. ***nk:*** is the number of data samples available for client ***k***. ***n:*** is the total number of data samples. ***Pk:*** is the set if indexes of data on client ***k***.[47]

2.5 Software

This section will introduce and briefly explain the software, mainly the python libraries utilized in this thesis.

2.5.1 PyTorch

PyTorch is a platform for deep learning which was utilized for this thesis. PyTorch is an open-source platform, written in the Python programming language and centers around the use of tensors. This platform was chosen for several reasons, but mainly because this thesis builds on results and software from Simen Larsen's master thesis which utilized the same platform. [48]

2.5.2 PySyft

PySyft is an open-source FL framework for building secure and scalable models. PySyft is a hooked extension of PyTorch, thus complementing the use of PyTorch for this thesis. [49]

2.5.3 SciPy

Is an open-source python library for mathematics, science, and engineering. This library is mainly used in the thesis to perform multidimensional image processing when augmenting.

2.5.4 Docker

Docker as referred to in this thesis is a product that offers OS-level virtualization to deliver software in packages. The software packages are referred to as docker images in this thesis. An image is an instance of a system set up, this is useful for ensuring compatibility across platforms. This is particularly helpful when using a combination of software packages that might require a specific version of other packages to function properly. These docker images are run by a single operating system kernel and uses fewer resources than virtual machines. [50]

2.5.5 Nipype

Nipype is an open-source python project that provides an interface to many existing neuroimaging software and provides interaction between these software within a single workflow[51]. Nipype provides a dockerimage[52] with all the different packages (e.g., ANTS, SPM, FSL, FreeSurfer, Camino, MRtrix, MNE, AFNI, Slicer, DIPY) already installed. This makes it much easier to use and for others to reproduce the processing one applies on the MRI images.

2.6 Previous work on detecting AD-DLB-NC with machine learning

Larsens paper[23] on classifying AD, DLB, and NC with deep learning(DL) is the baseline for this study. In Larsen's paper, he proposes a DL framework where a custom DL model can be trained on a custom dataset. He proceeds to test his framework with his own model, SimenNet, on a dataset he made with MRI images of AD, DLB, and NC subject. Much of the work Larsen did with preprocessing, dataset balancing, and programming has been utilized in this thesis. The programs he wrote has been further extended and edited to implement new methods.

Chapter 3

Materials and method

3.1 Reproduce Larsen's results and Python Environment

To improve the results that Larsen [23] got in his thesis, the first step was to reproduce his results. A python environment was made to run the code in. All the missing/wrong versions of the different packages were found by running Larsen's code until failure and installing the right python package. A lot of the packages needed to be installed with specific older versions to be compatible with each other. For the full list of the installed python packages, see the enclosed file "requirements.txt" in appendix A.

3.2 Data

The data used in this thesis is T1 weighted MRI scans from the European Dementia with Lewy bodies (E-DLB) consortium and the Alzheimer's Disease Neuroimaging Initiative (ADNI) databases [53]. The E-DLB consortium is the only source of DLB subjects, and it contains 288 DLB, 146 AD, and 146 NC subjects. The ADNI database contains a number of AD and NC subjects which has been used to balance the dataset.

3.2.1 Preprocessing

Spatial Normalization

Spatial normalization was performed with the SPM12 software [54] that was included in the Nipype docker image. In addition to spatially normalize all the images, the software also normalizes the image intensity variations that are common in MRI images due to varying strength in the magnetic field.

Brain Extraction

The BET2 software [55] [56] was used to extract the brains from the MRI images. This software reported to give good results and high performance. The skull stripping process can also be adjusted with the fractional intensity threshold (frac) parameter. Higher frac values make the skull stripping more "aggressive" and removes more of the MRI image. Multiple frac values (0.4, 0.3, 0.2, 0.15, 0.05, 0, 0.25, 0.275, 0.265, 0.24) were tested on two random MRI images from the data and were manually inspected to find the optimal frac value for the skull stripping process. Frac value 0.25 was chosen because it removed the least amount of brain matter while still managing to remove most of the unwanted parts of the MRI image. There were still some parts around the eyes that did not get removed, but with higher values, the skull stripping started to remove more of the brain matter. After skull stripping the data with frac=0.25, ten random brains were inspected, and one of the brains had inferior results around the eyes (see figure 3.1). This brain needed a frac value of 0.4 to get rid of the eyes in the skull stripping process correctly. Therefore a dataset with frac = 0.4 was also created.

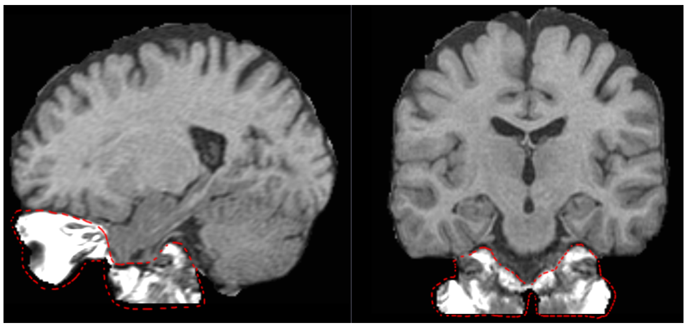


Figure 3.1: Poorly skullstripped brain with Frac=0.25.

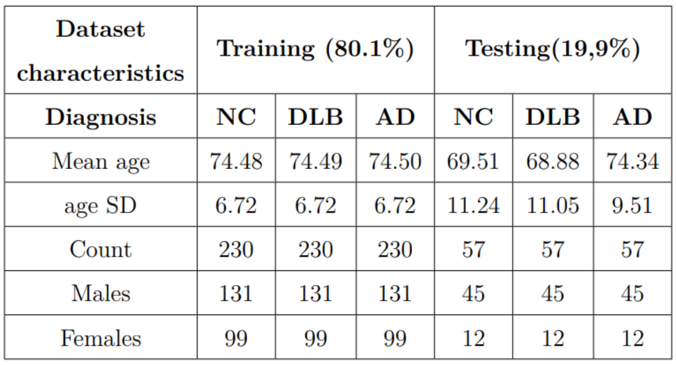
The BET2 also has some mutually exclusive options when running the skull stripping. The option "reduce bias" and "remove eyes" were tested.

The "remove eyes" option was used with the frac value 0,25 to make another dataset. The skull stripping process crashed on ten of the MRI images. These were supplemented from the frac = 0.4 data.

According to Popescu V, Battaglini M, Hoogstrate WS, et al.[57] the optimal parameters would be BET option "reduce bias" with frac = 0,2. The "reduce bias" option crashed the skull stripping process on 17 of the MRI images in the dataset. These were supplemented from the "frac = 0.25 remove eyes" data.

3.2.2 Dataset

The datasets are built on the work Simen Larsen did in his thesis. All the datasets use Larsen's age and gender matching so that patterns in the data that are not "dementia" related are reduced. For example, if all the AD brains were men, and all the DLB brains were women, the DL algorithm might learn false patterns related to the gender of the brains and not the disease. That is why the dataset is balanced, see table 3.2 for the dataset characteristics. One of the DLB subjects was discarded because the age was three standard deviations from the mean age of the DLB data. In total there are 861 MRI images in the dataset, 287 from each class.



Dataset characteristics	Training (80.1%)			Testing(19,9%)		
Diagnosis	**NC**	**DLB**	**AD**	**NC**	**DLB**	**AD**
Mean age	74.48	74.49	74.50	69.51	68.88	74.34
age SD	6.72	6.72	6.72	11.24	11.05	9.51
Count	230	230	230	57	57	57
Males	131	131	131	45	45	45
Females	99	99	99	12	12	12

Figure 3.2: Larsen's Dataset Characteristics [23] picture form Larsen's thesis is used with his consent.

The dataset structure is kept in all the new datasets that are made with different skull stripping values. All the subjects that are put into "testing" and "training" are also identical in all the datasets to keep the results consistent.

Six datasets have been used in total, and the different datasets are **Frac 0.5 (Larsen's dataset), with and without added upscaled GAN image** Larsen's dataset was copied from his thesis and used to generate and test the GAN images. **Frac 0.5 (Larsen's dataset), Resized to 64x64x64, with and without added GAN images**. **Frac 0.25 dataset**, **Frac 0.25 dataset Resized to 64x64x64**, **Frac 0.4 dataset**, **Frac 0.25 dataset with "remove eye" option**, **Frac 0.1 dataset with "reduce bias" option**, **Frac 0.2 dataset with "reduce bias" option**.

3.2.3 Federated learning Data Set

For the FL approach, two datasets are being used, one for each FL model. There are 230 of each class in the original training data, which means there are 115 brains to train in each FL dataset. Since the test data only contains 57 brains for each class, one of the FL datasets gets 29 of each brain type and the other one gets 28 in the test folder.

The MRI images from Larsen's dataset were randomly selected and added to either the FL1 or FL2 dataset. Because the splitting of Larsen's dataset is random, the FL1 and FL2 dataset does not have the same balance of age and gender, which can impact the generalization if they are trained separately.

3.3 Models

The used models were all modified to use 3D convolutional layers instead of 2D, as the 3D has shown to produce better results with MRI images [58]. The models tested was SimenNet, ResNet, DenseNet, as well as the federated models.

The **SimenNet** model was copied from Larsen's code and tested as it is.

The **ResNet** model used is from Zuppichini [59] implementation in PyTorch because it was scalable and easy to modify. The model was modified to use 3D convolutional layers and 3D images as input, and then tested with sizes of 18, 34, 50, 101, and 152. Extra fully connected layers were also added to the end of the model to see if it would increase performance (see figure 3.3). Both the fully connected layers and convolutional layers were tested with and without dropout. Dropout were always added to the model after BN [60], see figure 3.3 to see how it is added in the FC layers.

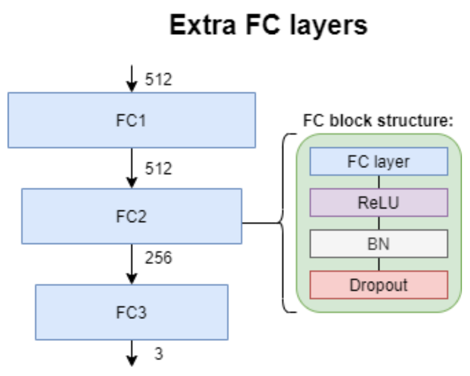


Figure 3.3: Extra Fully Connected Layers added to end of ResNet models.

The **DenseNet** was implemented from Aspris github[61], and modified to use 3D instead of 2D layers.

3.3.1 Federated Learning Models

Two different neural nets are constructed for the FL models. Both of these FL nets are based in large part on the SimenNet, and altered only when necessary to fit the federated setup. The nets include multiple convolutional and max-pooling layers, as well as some fully connected layers. The activation function used for both nets is the ReLU6 function.

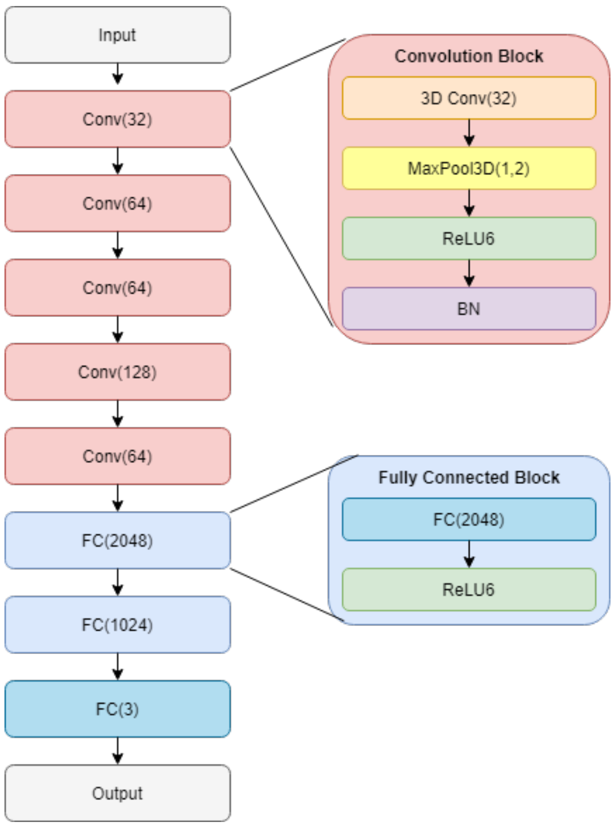


Figure 3.4: Federated Network Structure

This proposed network (3.4), is for testing the federated average function performance and effect. The first part of the network consists of five convolutional blocks, which includes convolution function, max-pooling function, ReLU6 function and batch normalization function. The latter part of the network consists of three fully connected blocks, which includes a fully connected layer and the ReLU6 activation function, except for the third block which only consists of the fully connected layer.

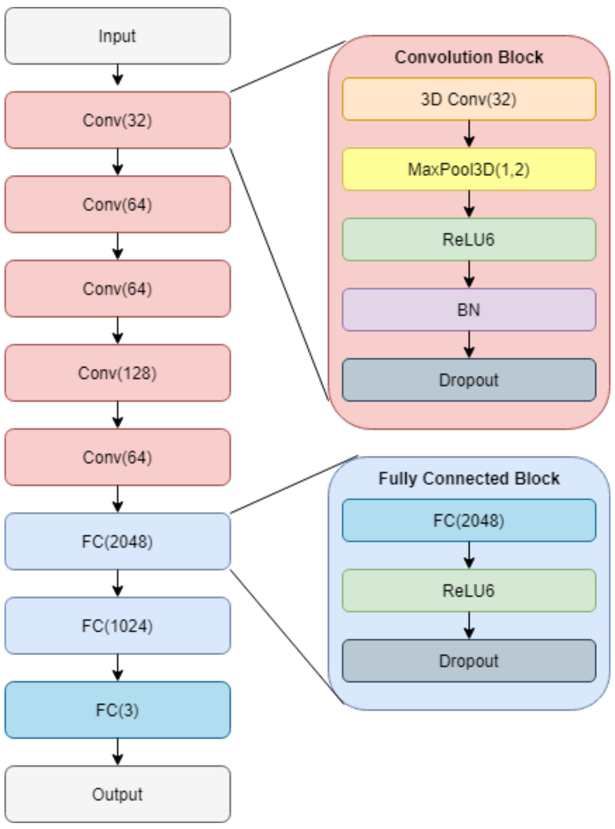


Figure 3.5: Asynchronous Federated Network Structure

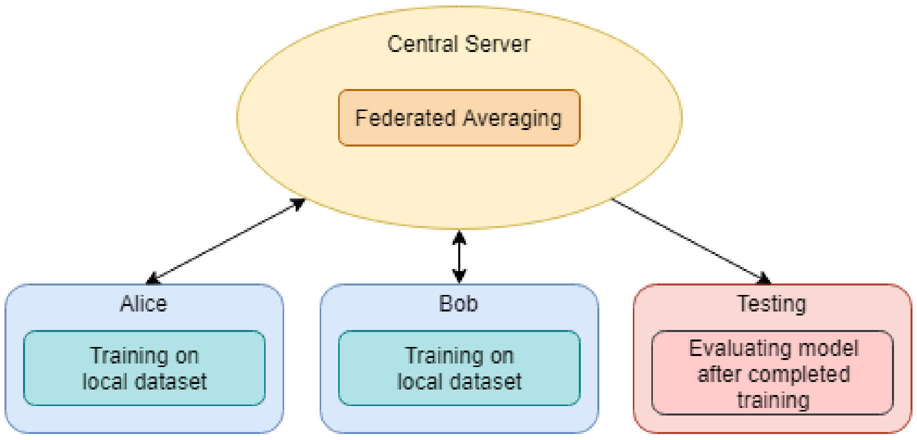

Central Server
Federated Averaging
Alice
Training on local dataset
Bob
Training on local dataset
Testing
Evaluating model after completed training

Figure 3.6: Asynchronous Federated system setup

The second proposed federated network (3.5), is for testing the federated setup in an asynchronous federated learning structure with a central server, worker and testing nodes (3.6). The reasoning behind the asynchronous setup is that it is based on the asynchronous MNIST websocket example detailed in chapter 3.5. An asynchronous structure will allow the nodes to contribute to the federated model more efficiently than a synchronous structure. This proposed network structure contains all the layers native to the first proposed network, however this network also includes some additional layers.

In both the convolutional and fully connected blocks, this network contains a dropout layer at the end. The dropout layers provide additional options to tweaking the model and can sometimes be useful as a conutermeasure to the overfitting problem.

3.4 Augmentation

Because the primary biomarker of dementia is atrophy throughout the brain, some augmentation techniques might make it harder to distinguish a healthy brain and one with dementia. Therefore multiple augmentation techniques were tested separately to single out any inferior ones. In the end, the best performing techniques are combined and tested.

When training a model, online augmentation is applied to the data. This is done to increase the variety of data when training and makes it harder for the model to overtrain. The augmentation was performed on all of the training data, which includes the data used in the validation. When verifying the model with the test images, no augmentations are applied. The probability of an MRI image getting augmented was manually set in the augmentation function. This is to limit the amount of augmentation applied to the dataset.

All the different augmentations were visually inspected to ensure that the code worked as expected, see figure 3.7

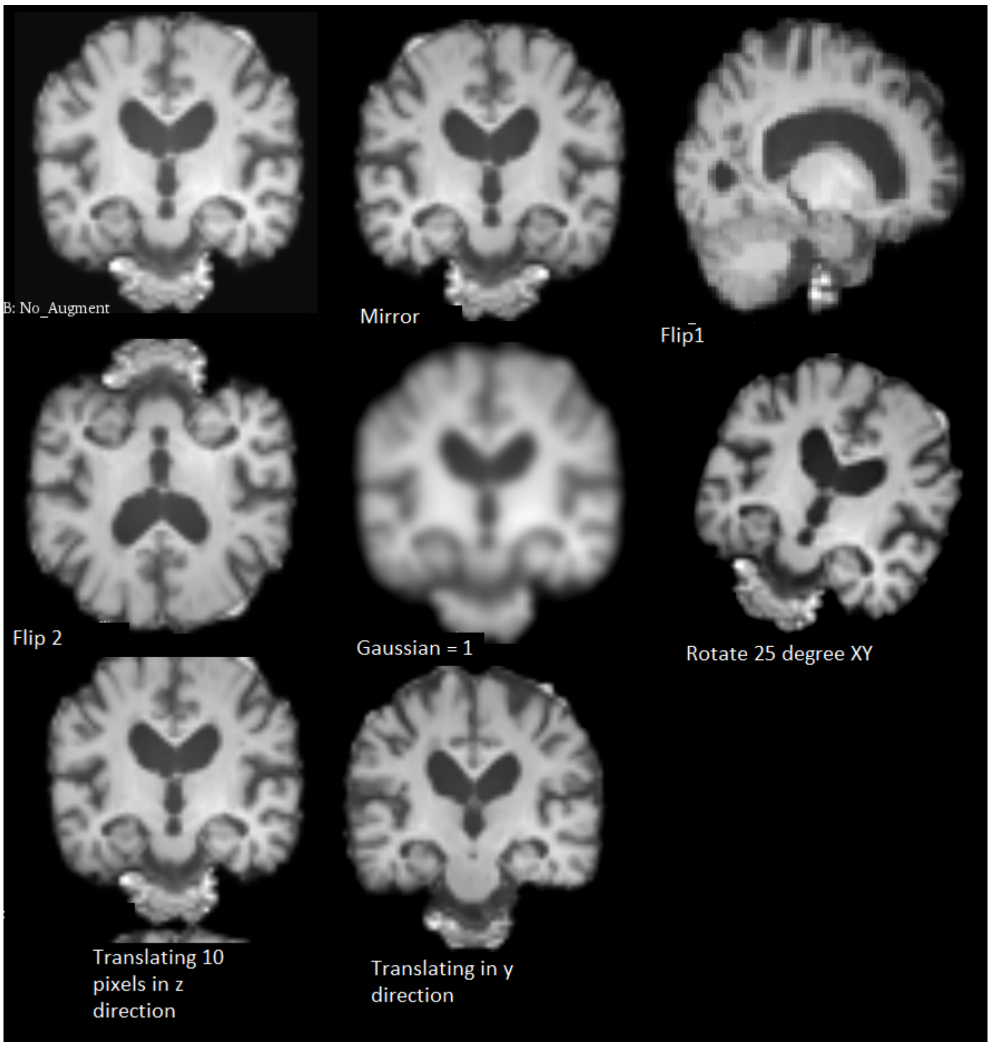

B: No_Augment
Mirror
Flip1
Flip 2
Gaussian = 1
Rotate 25 degree XY
Translating 10 pixels in z direction
Translating in y direction

Figure 3.7: Illustration of the different augmentations.

3.4.1 Simple Single Augmentation

Flipping and mirroring using the "flip" function in the NumPy library. The flips are left to right, upside down, and mirroring. The flipping has a 50% chance of being applied to an MRI image during training. Because a flip can only be applied once for every picture, higher probabilities seem useless.

Random rotating because the MRI data is in 3D, there are three different planes to rotate the data: XY, XZ, and YZ. All the different planes are tested one at the time, with varying rotation angles and probability. The probability for an image being rotated and the range of how many degrees to rotate is manually set in the augmentation function before training. The varying amount of rotation is done with a random function that gives a random number between two set values, e.g. a rotation with +-2 degrees can give these rotations: 1, 2 359 and 358 (negative values is 360 + the negative value). The images are rotated with the rotate function from the scipy.ndimage package.

Translation moves the brain around in the 3D space. The translations are done in all directions (right/left, up/down, in/out) between 1 to 4 pixels. With more than 4 pixels, the brain will start to "wrap" around to the other side. Each of the three directions has its own probability of being applied; this increases the total number of unique augmentations that can be done with the translations. The probability for an image being augmented and the probabilities for each direction are individually set in the augmentation function before training.

Gaussian Blur was applied to the data using the gaussian_filter method from SciPy. The probability for an image being applied with the Gaussian blur is manually set in the augmentation function before training. The Gaussian blur can also be applied with varying intensity levels by specifying the range in the augmentation function before training.

3.4.2 Simple Augmentation Combinations

When combining the different augmentation techniques, two different methods were used. The first method randomly applies multiple augmentation techniques on one image. The chance for each type of augmentation technique being applied is set individually in the augmentation function before training. This is to regulate which techniques are most likely to be applied. The techniques that performed better than others in the previous experiments have generally gotten higher probabilities of being applied than those who performed worse.

While the second method, called "exclusive augmentation", randomly applies only one augmentation technique to the image. In the exclusive augmentation, a random number is generated, and the different augmentations are given a number each. If the random number matches the number assigned to an augmentation technique, the image gets this augmentation applied. More numbers are assigned to the more desirable augmentation techniques to increase the likelihood of these augmentations being applied.

3.4.3 GAN

The model used is from the paper "Generation of 3D Brain MRI Using Auto-Encoding Generative Adversarial Networks"[62]. This model uses the α-GAN network structure to generate 3D MRI images. Results from the paper show that it outperforms other methods, see figure 3.8. All the code used to train the 3D-α-GAN model is in their

GitHub repository [63]. The data used to train the 3D-α-GAN models are from Larsen's dataset (Frac = 0.5) resized to 64x64x64. The Frac = 0.5 dataset was used because of none of the other datasets were made at the time of training the 3D-α-GAN. The resized data was used because of memory limitations on the GPUs when training. Because there were three different classes in Larsen's dataset, three different generators were trained. Transfer learning was not utilized because the pre-trained model, which was enclosed in the GitHub repository, used different preprocessing then the data in Larsen's dataset. Each class had 287 MRI images to train on, which should be enough. The hyperparametes used are the same as in the thesis [62] as they were shown to produce good results.

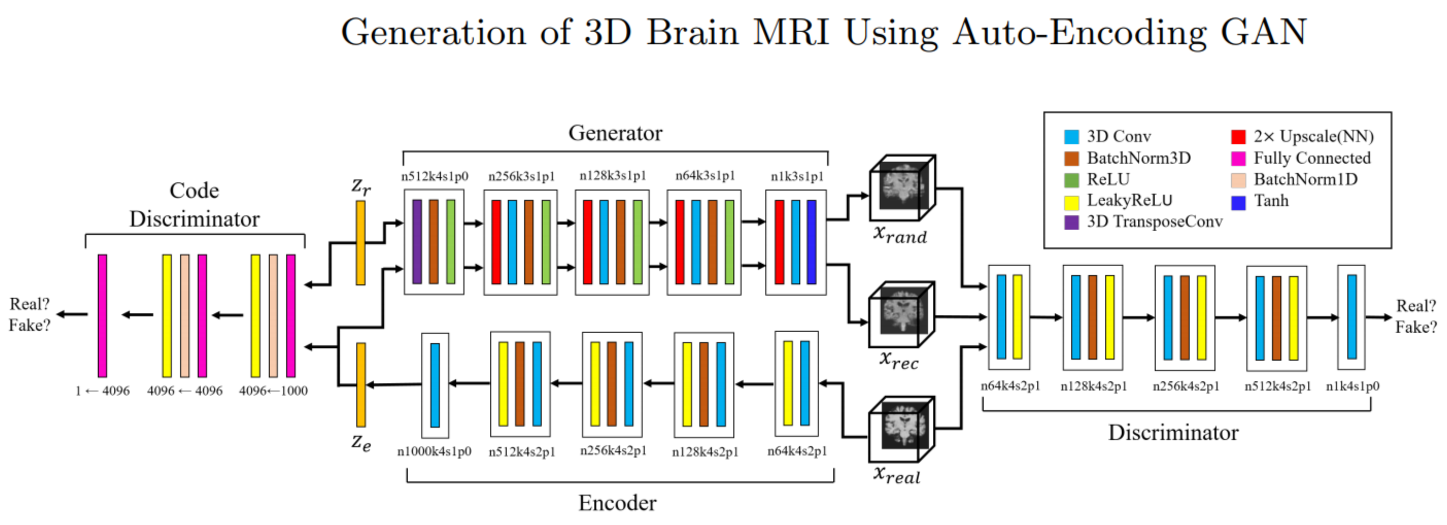
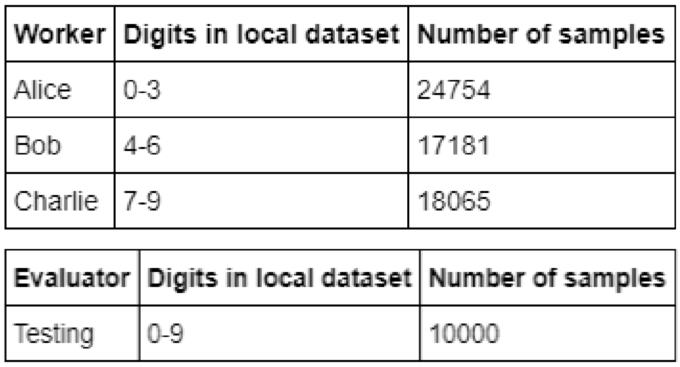


Figure 3.8: Detailed architecture of the model from [62]. *n = number of the channels, k = kernel size, s = stride size, and p = padding size. xrand is the generator output from random vectors zr and xrec is the output from encoded vectors ze.*

3.5 Existing Approaches/Baselines

3.5.1 Asynchronous federated learning on MNIST

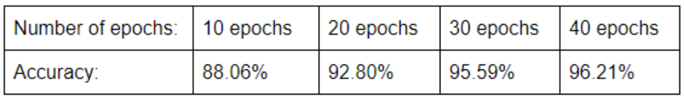
This example is run with code from the PySyft MNIST Websocket example[64]. The MNIST dataset is a set of handwritten numbers from 0-9, widely used for testing ML algorithms. This is done to get a baseline of what accuracy is achievable when training on a robust, large sample dataset using the PySyft FL-framwork. For this test setup we have three workers; Alice, Bob and Charlie, each holding a piece of the dataset. The evaluator holds testing data and tracks model performance. The structure of the setup is identical to (3.6), except there is an additional worker node; Charlie.



Worker	Digits in local dataset	Number of samples
Alice	0-3	24754
Bob	4-6	17181
Charlie	7-9	18065

Evaluator	Digits in local dataset	Number of samples
Testing	0-9	10000

Table 3.1: MNIST dataset Asynchronous FL table



Number of epochs:	10 epochs	20 epochs	30 epochs	40 epochs
Accuracy:	88.06%	92.80%	95.59%	96.21%

Table 3.2: MNIST dataset Asynchronous FL results table

Chapter 4

Experimental Evaluation / Results

This chapter contains the experiments and results presented in this thesis. Subchapter 4.1 - 4.9 lays out the experiments and results from the general approach of improving Simen Larsen's results, as well as general information about the dataset and augmentation methods. Subchapter 4.10 - 4.11 lays out the experiments and results from the federated learning approach.

4.1 Reproducing Simen Larsen's results

The algorithm ran with Larsen's start arguments for two days before the Bayesian optimization found the best hyperparameters to train the model. The best performing model had an accuracy of 72.5% which is more than expected, see figure 4.1 and 4.2 bellow for more details.

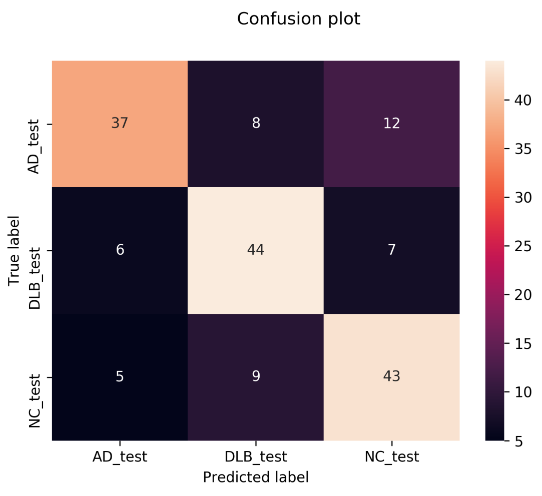
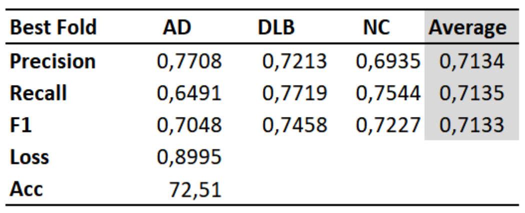


Figure 4.1: Confusion plot of reproduced model.



Best Fold	AD	DLB	NC	Average
Precision	0,7708	0,7213	0,6935	0,7134
Recall	0,6491	0,7719	0,7544	0,7135
F1	0,7048	0,7458	0,7227	0,7133
Loss	0,8995			
Acc	72,51			

Figure 4.2: Performance of reproduced model.

4.2 Overview

An overview of the structure for when the different experiments were conducted is present in figure 4.3.

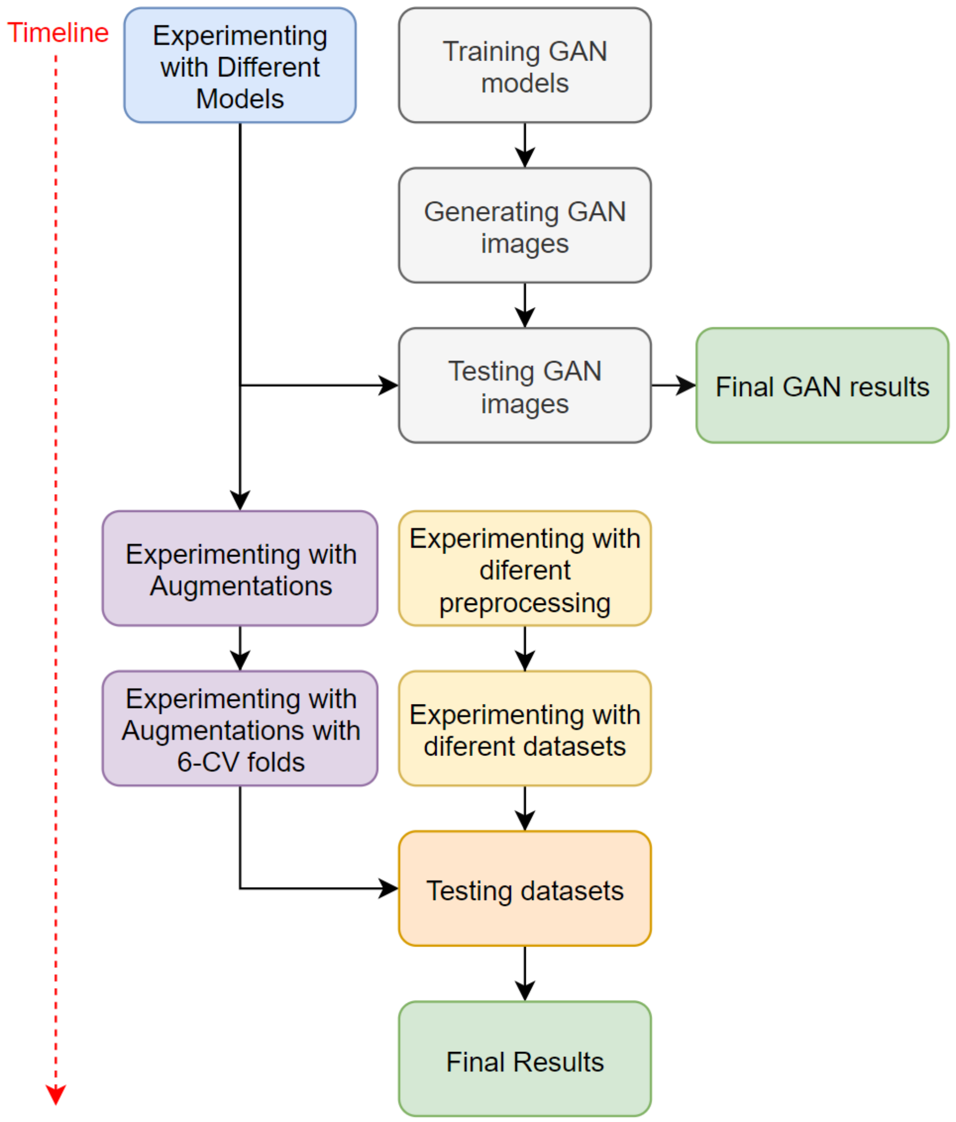

Timeline
Experimenting with Different Models
Training GAN models
Generating GAN images
Testing GAN images
Final GAN results
Experimenting with Augmentations
Experimenting with diferent preprocessing
Experimenting with Augmentations with 6-CV folds
Experimenting with diferent datasets
Testing datasets
Final Results

Figure 4.3: Structure with timeline when carrying out the different experiments.

4.3 Prepossessing

To visualize the preprocessing results, two different brains have been chosen. BrainA, which is a good MRI picture and reflects the preprocessing results for the majority of MRI images in the dataset, figure 4.5. BrainB, which is the problematic MRI image from figure 3.1. This MRI image shows where the prepossessing falls short. Unfortunately, the

MRI image does not work with the "reduce bias" option, see figure 4.6. The prepossessing process for a whole dataset took five days to complete when running on CPU.

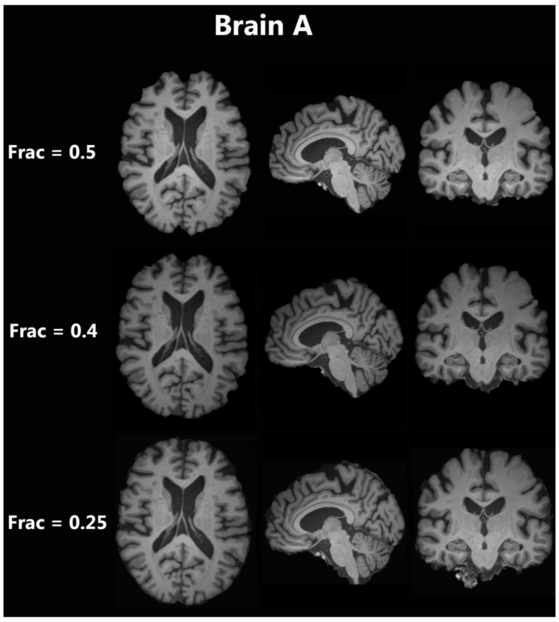


Figure 4.4: BrainA different frac preprocessing result.

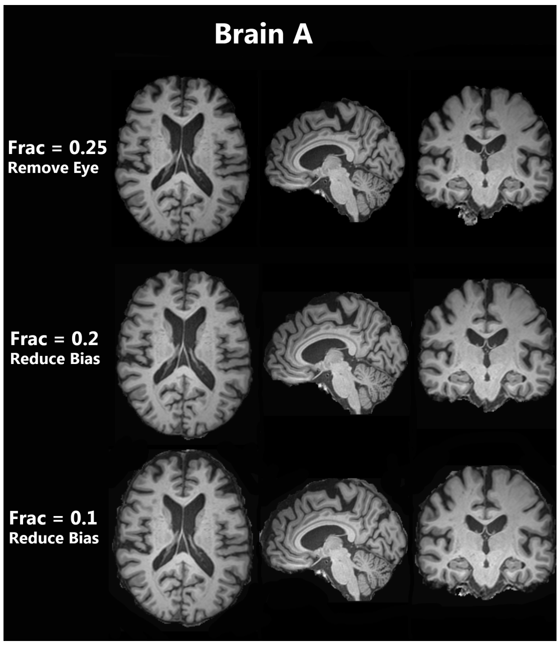


Figure 4.5: BrainA with extra options preprocessing result.

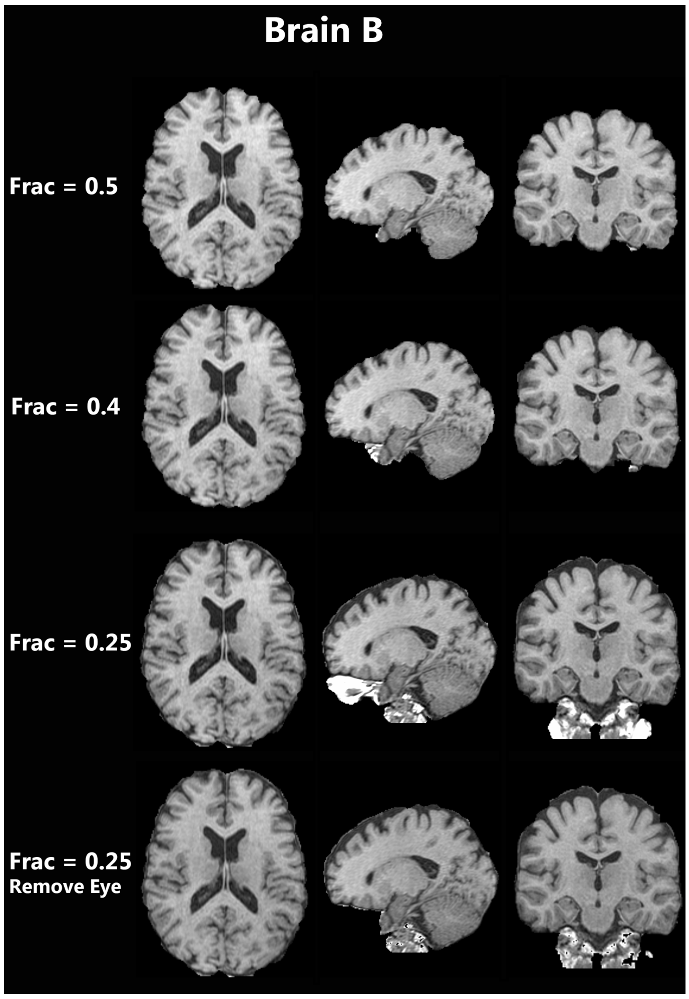


Figure 4.6: BrainB preprocessing result.

4.4 Generating MRI Images with GAN

The algorithm ran for 200000 epochs and it took approximately 84 hours to make a generative model for one type of brain on a nvidia tesla v100 PCIE 32gb GPU. See samples of the real and fake NC, AD and DLB MRI brains in figure 4.7 - 4.12.

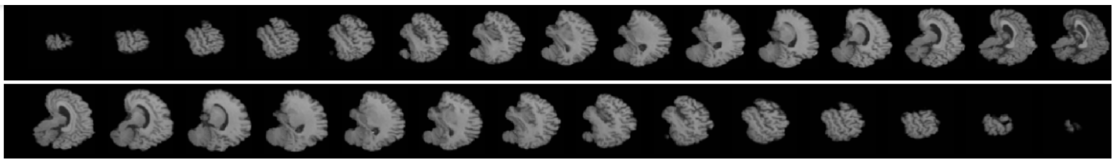


Figure 4.7: Sample of a **real.** NC brain

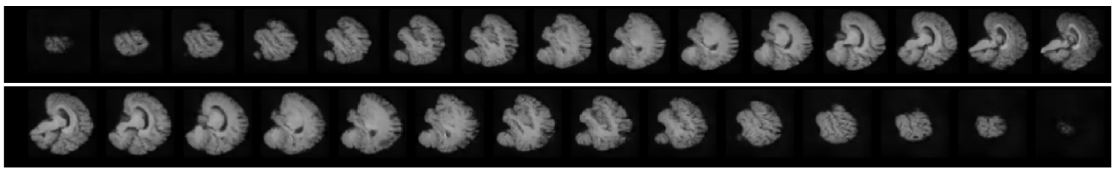


Figure 4.8: Sample of a **generated** NC brain.

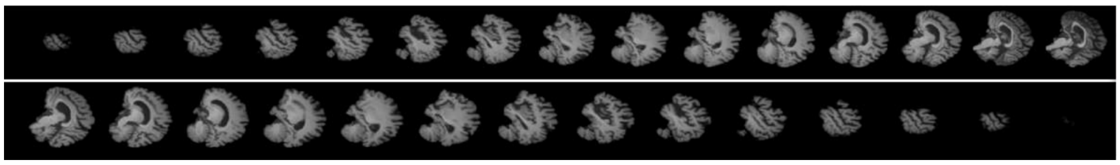


Figure 4.9: Sample of a **real** AD brain.

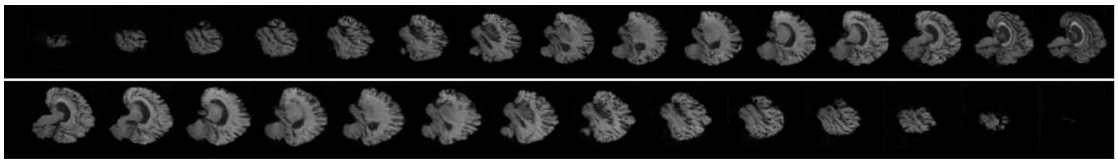


Figure 4.10: Sample of a **generated** AD brain.

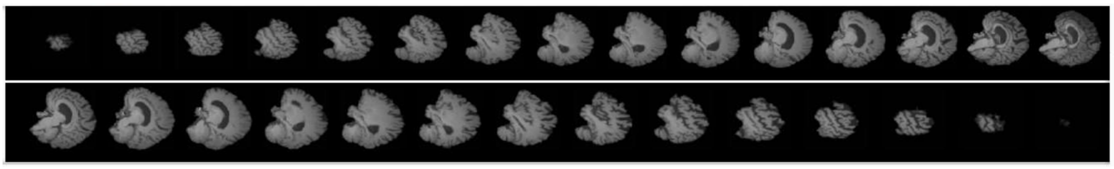


Figure 4.11: Sample of a **real** DLB brain.

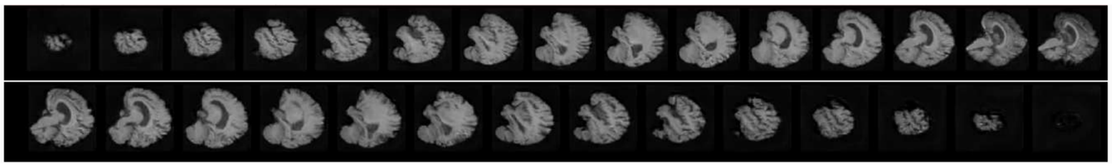


Figure 4.12: Sample of **generated** DLB brain.

The generated MRI images were visually inspected to make sure they look like brains with no obvious anomalies. To verify that the generated MRI images were different brains with DLB, AC, and NC diagnosis, the generated images were put in the test set of a dataset. Then a model that was trained with only real brains was used to classify the generated MRI images. If the results from this are similar to the results with the real MRI images, then it is a clear indicator that the generated brains reproduce their intended diagnosis well.

4.5 Experiment - ML Models

All the models were tested with the "frac = 0,25 resized 64x64x64 dataset" because the smaller resolution reduces the training time significantly. ResNet and DenseNet used the SGD optimizer with parameters: learning rate = 0.000995, momentum = 0.537, L2 weight decay=0.0549, nesterov=True. SimenNet used the Adam optimizer with parameters: learning rate =0.0000297, smoothing = 0.67, L2 weight decay=0.1552 and dropout = 0.2. These parameters were found by doing a quick Bayesian Optimisation with the ResNet18 model with a 2-fold CV.

Model Tested	Best Acc from 6 fold CV	Average Acc from 6 fold CV
DenseNet	67,83625793	NA
SimenNet	68,42105103	57,50487328
ResNet152	68,42105103	63,35282644
ResNet101	68,42105103	64,81481489
ResNet50	66,66666412	59,25926018
ResNet34	66,08187103	60,6237812
ResNet18	67,25146484	64,61988513
ResNet152 with 3 FC layers with dropout	68,42105103	61,40350978
ResNet101 with 3 FC layers with dropout	66,08187103	61,30604362
ResNet50 with 3 FC layers with dropout	66,08187103	61,11111132
ResNet34 with 3 FC layers with dropout	68,42105103	64,61988322
ResNet18 with 3 FC layers with dropout	72,51461792	66,8615996
ResNet152 with dropout in Conv layers	63,74269104	59,35672569
ResNet50 with dropout in Conv layers	65,49707794	60,03898621
ResNet18 with dropout in Conv layers	64,32748413	61,30604299
ResNet152 with 3 FC layers with dropout in both	64,32748413	54,19103305
ResNet101 with 3 FC layers with dropout in both	66,66666412	62,86549695
ResNet50 with 3 FC layers with dropout in both	67,83625793	62,7680308
ResNet34 with 3 FC layers with dropout in both	70,76023102	67,25146103
ResNet18 with 3 FC layers with dropout in both	71,92982483	67,83625793

Table 4.1: Results from testing different models.

For more detailed results, see appendix B.1 - B.2 were the results are reported with confusion plots, precision, recall and F1 scores.

4.6 Augmenting

The augmentation experiments were conducted in two phases. In the first phase, the augmentation techniques were tested by training the ResNet18 model once with hyper-parameters that were slightly adjusted from those used in the chapter 4.5 Experiment

- ML Models :learning rate=0.0004633, momentum=0.62, dampening=0, L2 weight decay=0.06, nesterov=True.

The different augmentations techniques were individually tested with varying parameters of probability and varying input range for the techniques that supported this (rotation and gaussian blur). Then the different augmentations techniques were combined and tested again with varying parameters of probability and varying input range. The result was reported with confusion plot, accuracy, loss, precision, recall, and F1 score. This method of testing was fast and gave reasonable indications of what worked and what did not, see appendix B.4 - B.14 for the results.

The best and most consistent augmentation from the first phase was the translation combined with a small rotation.

In phase two of the augmentation experiment, new hyperparameters were found with the Bayesian Optimization technique, using the ResNet18 model and the best and most consistent augmentation from the first phase while training, see table 4.2 and 4.3.

Iteration	Learning Rate	L2	Momentum	Average test Acc
1	0,0009807	0,1872	0,4756	69,88
2	**0,0009934**	**0,0549**	**0,6340**	**73,20**
3	1,46E-05	0,1373	0,9520	69,10
4	0,0003459	0,0695	0,7095	70,86
5	2,14E-05	0,0536	0,7153	65,79
6	0,0005519	0,0969	0,6541	70,86
7	0,0006567	0,1579	0,8224	69,49

Table 4.2: Bayesian optimization with augmenting.

Iterartinon 2	Acc Validation	Acc Test
Average Acc:	71,3 (0,7)	73,2 (3,57)

Table 4.3: Best iteration from the Bayesian optimization with augmenting. Values in brackets are the standard deviation.

The different augmentations were then tested again with the new hyperparameters with the ResNet18 model. This time the results were reported with a six-fold CV to get more reliable results to determine which of the augmentations performed best.

The different augmentations techniques were again individually tested, and then combined and tested, with varying parameters of probability and varying input range for the techniques that supported this (rotation and gaussian blur), see appendix B.15 - B.26.

The best result from the experiment was the combination of translating(Roll) and a small rotation in the XZ plane. The augmentation with the probability and range parameters was: "Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, Rotate +-6 degree XZ 80%". For detailed result see table highlighted in green in appendix B.24.

Baseline result with no Augmentations

See appendix B.3 for tables with the first results and appendix B.15 for the 6-CV results.

Rotation in XZ plane

See appendix B.4 for tables with the first results and appendix B.16 for the 6-CV results. Some of the augmentations were tested multiple times to see if the results were consistent like the "Rotate +-30 degree XZ axis, 90% augmented". The % augmented is to indicate the probability that an MRI image have to be augmented.

Rotation in XY plane

See appendix B.5 for tables with the first results and appendix B.18 for the 6-CV results.

Rotation in YZ plane

See appendix B.6 for tables with the first results and, appendix B.19 for the 6-CV results.

Translations

See appendix B.7 for tables with the first results and, appendix B.20 for the 6-CV results. "Roll" is used to indicate translations, because it is the name of the function in python and shorter to write than "Translation". The first "%" probability when translating is the probability for the MRI image to translated at all, and the second "%" probability is for each direction (left/right, up/down, and back/forth).

Mirroring/Flip(0)

See appendix B.8 for tables with the first results and, appendix B.21 for the 6-CV results.

Flip Left Right/Flip(1)

See appendix B.9 for tables with results. This augmentation was not tested with 6-CV because it performed poorly in the first test.

Flip Upside Down/Flip(2)

See appendix B.10 for tables with results. This augmentation was not tested with 6-CV because it performed poorly in the first test.

Gaussian Filter

See appendix B.11 for tables with the first results and, appendix B.22 for the 6-CV results.

Combinations of Different Augmentations

See appendix B.12 - B.14 for tables with the first experiment results. See appendix B.23 - B.26 for tables with the 6-CV results.

4.6.1 Augmenting with GAN

The augmentations with GAN used the Frac0.5 dataset because the GAN model was trained with this dataset. When comparing the GAN results, the comparisons are done with the Frac = 0.5 dataset and not the Frac = 0.25 dataset that the rest of the augmentations use. This is because the GAN was trained before the frac = 0.25 dataset was created.

GAN with the Frac = 0.5 Dataset Resized to 64x64x64

230 GAN images were generated for each of the classes and added to the dataset training data. None of the generated data was added to the test folder in the dataset to keep this folder "clean".

The first experiments were done by training a ResNet18 model once on the dataset and report the results (tables in appendix B.27). The second experiment, combined the GAN supplemented dataset with augmentations, see tables in appendix B.28.

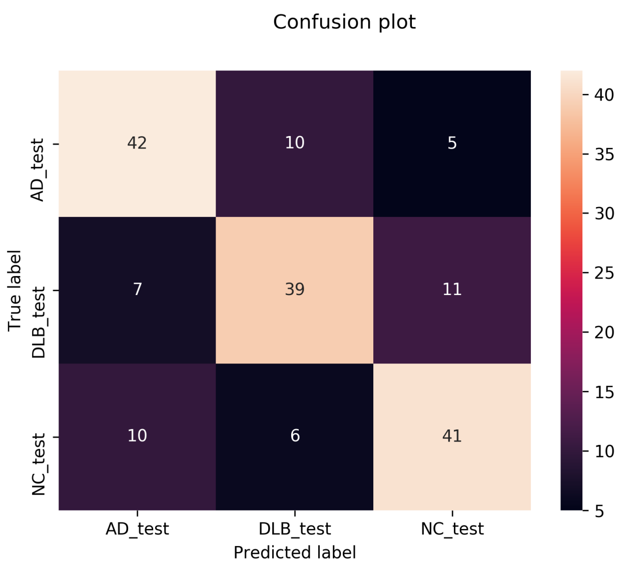
The GAN supplemented dataset was also trained with six-fold CV to get more reliable results, see figure in appendix B.44 for experiments with no augmentation, and figure in appendix B.46 for experiments with augmentation.

The resized frac = 0.5 dataset without the supplemented GAN images was trained with six-fold CV both with and without augmentations to be compared with the GAN supplemented experiments (see figures in appendix B.45 B.47).

6-fold CV Average results, with Standard Deviation(SD)				
	Validation Acc	SD	Test Acc	SD
GAN 25%	78,06	1,66	67,06	3,23
GAN 50%	82,32	3,11	67,15	4,21
GAN 50% w/Augmentation	82,32	1,58	67,15	3,19
Standard Dataset	66,71	4,21	63,55	2,02
Standard Dataset w/Augmentation	67,71	3,05	60,04	2,36

Table 4.4: 6-CV Augmentation Results with Frac=0.5 with and without GAN Images. GAN 25% had 115 added generated MRI images in each class in the training data, GAN 50% had 230. w/augmention = *Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, Rotate +-6 degree XY 90%.*

To get a better intuition of how the GAN images impacts the ResNet model, the best result from appendix B.44, which is fold 5, is displayed here with extra evaluation metrics, see figure 4.13.



Best Fold	AD	DLB	NC	Average
Precision	0,7119	0,7091	0,7193	0,7134
Recall	0,7368	0,6842	0,7193	0,7135
F1	0,7241	0,6964	0,7193	0,7133
Loss	0,8559			
Acc	71,35			

Figure 4.13: Best Result from appendix B.44 with extra evaluation metrics.

GAN Upscaled to 157x189x156 with Frac0.5 Dataset

Because the datasets with the original size (157x189x156) provided better results than the smaller resized (64x64x64) dataset, an experiment was conducted to upscale the GAN generated images and supply them to the original dataset. To upscale the GAN images, the "numpy.resize" function was used. This was used because it was fast and easy and needed no additional software, and there was no need to make extra slides as the GAN images contained the same amount of slides that the original sized pictures. See figure 4.14 for a comparison of the upscaled generated brain, original-sized generated brain, and a real brain.

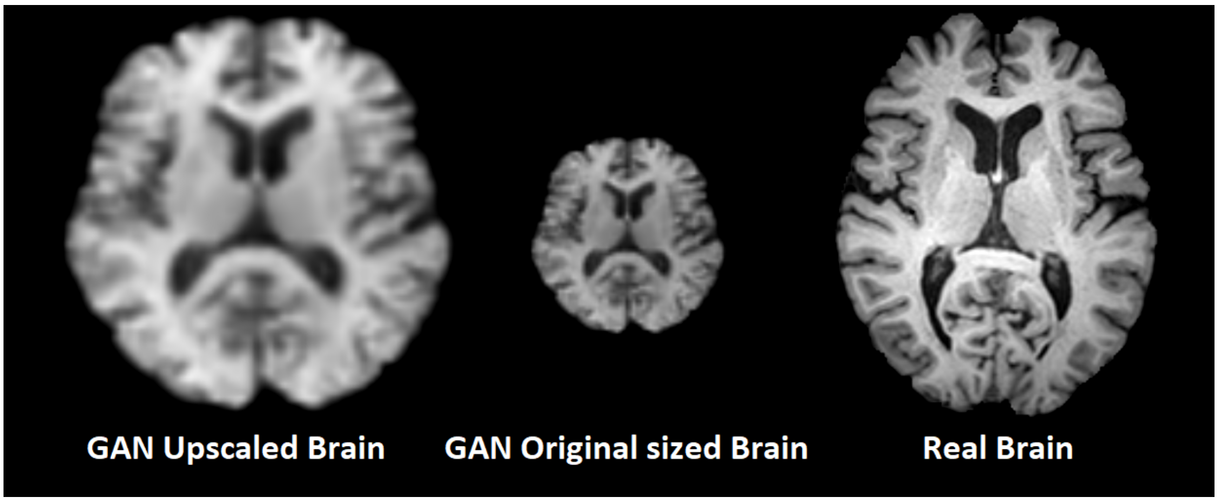


Figure 4.14: Comparison of Upscaled generated brain, original sized brain and a real brain.

The experiments with the Upscaled GAN images were trained with ResNet34 with 3 fully connected layers with dropout. The first experiment used multiple augmentations: translation with 95% chance of being augmented and a 66% chance for each direction, and rotating +-30 degrees in XY and XZ plane with 50% chance of being augmented. For Loss and accuracy plots see figures in appendix B.48 and B.49

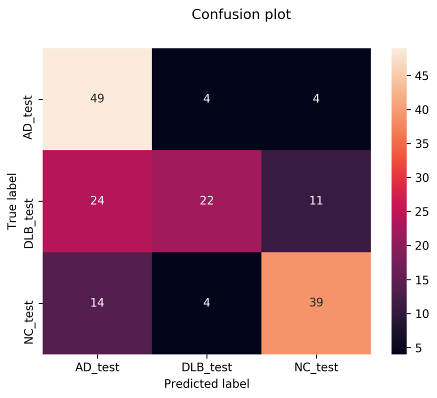


Figure 4.15: Confusion plot for upscaled GAN test.

	AD	**DLB**	**NC**
Precision	0,563	0,733	0,722
Recall	0,860	0,386	0,684
F1	0,681	0,506	0,703
Loss	0,979		
Acc	**64,327**		

Figure 4.16: Performance of upscaled GAN test.

The second experiment used only one augmentation: translation with 95% chance of being augmented and a 66% chance for each direction and rotating in. For Loss and accuracy plots see figures 4.17 and 4.18

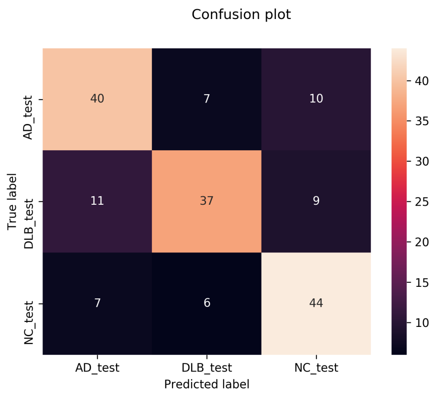


Figure 4.17: Confusion plot for upscaled GAN test.

	AD	**DLB**	**NC**
Precision	0,690	0,740	0,698
Recall	0,702	0,649	0,772
F1	0,696	0,692	0,733
Loss	0,771		
Acc	**70,760**		

Figure 4.18: Performance of upscaled GAN test.

Only GAN images used in training

To test how the GAN images would perform when training a model without any real brains in the dataset, a ResNet18 model was used. The model was then tested on a dataset of real brains to see how the model performed, see figure 4.19 - 4.22.

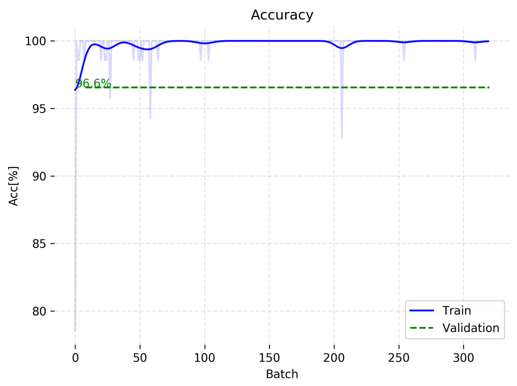


Figure 4.19: Accuracy plot for training only GAN.

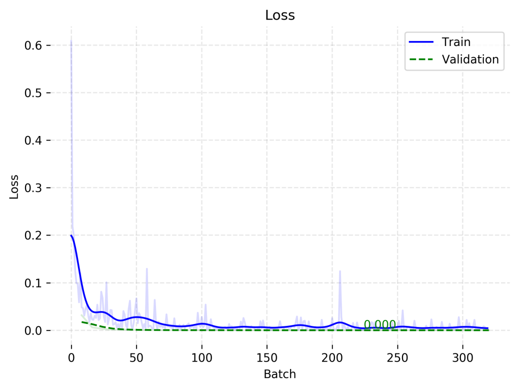


Figure 4.20: Loss plot for training only GAN.

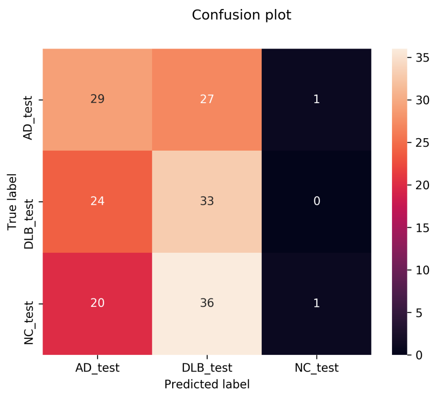
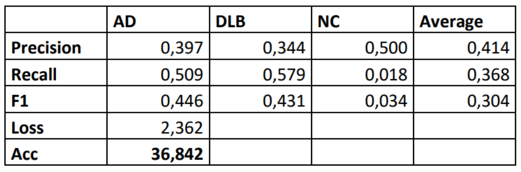


Figure 4.21: Confusion plot for only GAN test.

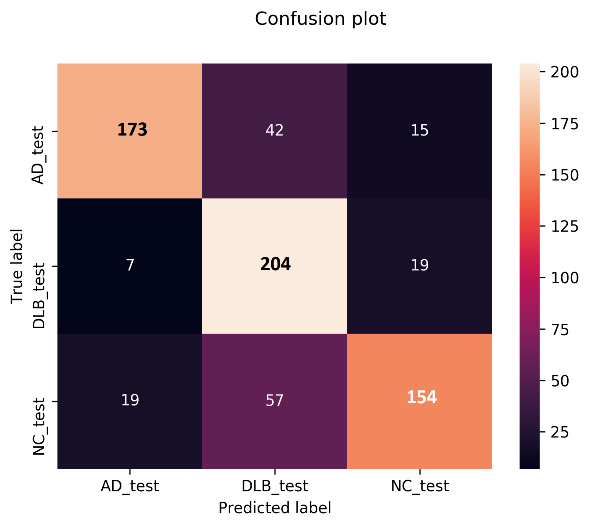


	AD	DLB	NC	Average
Precision	0,397	0,344	0,500	0,414
Recall	0,509	0,579	0,018	0,368
F1	0,446	0,431	0,034	0,304
Loss	2,362			
Acc	**36,842**			

Figure 4.22: Performance of only GAN test.

Verifying the GAN images

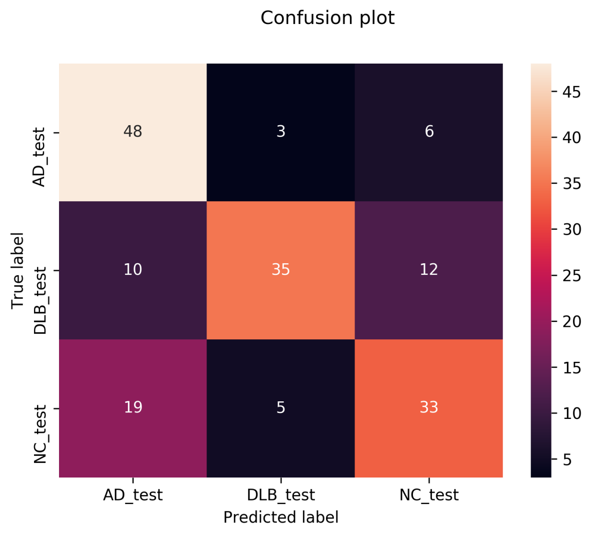
To verify that the GAN images are different brains with DLB, AC, and NC diagnosis, the best model from the six-fold CV from table in appendix B.45(which is Fold 6) which were trained on only real brains were used to classify all the generated images.



Test GAN	AD	DLB	NC	Average
Precision	0,8693	0,6733	0,8191	0,7873
Recall	0,7522	0,8870	0,6696	0,7696
F1	0,8065	0,7655	0,7368	0,7696
Loss	0,5349			
Acc	76,9565			

Figure 4.23: Model trained on real MRI images, classify the Generated GAN images.

For context the model metrics of Fold 6 from table in appendix B.45 is displayed here to show how the same model performed on the real data (see, figure 4.24).

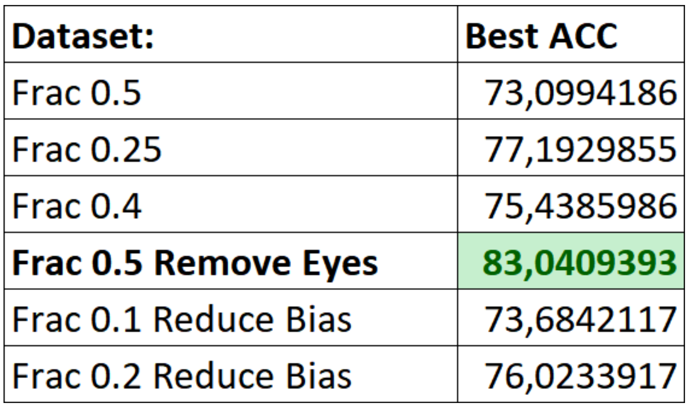


Best Fold	AD	DLB	NC	Average
Precision	0,6234	0,8140	0,6471	0,6948
Recall	0,8421	0,6140	0,5789	0,6784
F1	0,7164	0,7000	0,6111	0,6758
Loss	0,8745			
Acc	67,8363			

Figure 4.24: Detailed metrics of Fold 6 from table B.45 when tested on real data.

4.7 Datasets

Experiments to find the best dataset was conducted. The different datasets were tested with varying amounts of augmentations, and the ones showing promising results were tested further. The hyperparameters used was: learningrate=0.0009956, momentum=0.537948, L2 weight decay=0.0549, nesterov=True. For a quick overview of the best results of the different datasets, see table 4.5.



Dataset:	**Best ACC**
Frac 0.5	73,0994186
Frac 0.25	77,1929855
Frac 0.4	75,4385986
Frac 0.5 Remove Eyes	**83,0409393**
Frac 0.1 Reduce Bias	73,6842117
Frac 0.2 Reduce Bias	76,0233917

Table 4.5: Single Best ACC for every dataset

Frac0.5 Dataset (Simens Dataset)

See tables in appendix B.29 for the results. This dataset is the only with GAN as augmentation.

Frac0.25 Dataset

See tables in appendix B.30 for the results.

Frac0.4 Dataset

See tables in appendix B.31 for the results.

Frac0.25 Remove Eyes Dataset

See tables in appendix B.32, and B.33 for the results. Because this dataset had the best results, further experiments were conducted with different, hyperparameters, LeakyRelu, varying rotations and varying chances for different rotations, and with more significant translations see tables in appendix B.36 - B.43.

Frac0.1 Reduce Bias Dataset

See tables in appendix B.34 for the results.

Frac0.2 Reduce Bias Dataset

See tables in appendix B.35 for the results.

4.8 Final Evaluation of Three Class Classification

The final evaluation uses the results from the model, augmentation, and dataset experiments to train a model with the Bayesian optimization algorithm. The Bayesian optimization algorithm finds the best hyperparameters (learning rate, L2 regularisation, and momentum) for the model to use when training, thus finding the best performing model.

Iteration	Learning Rate	L2	Momentum	**Average test Acc**
1	0,0009807	0,1872	0,4756	74,85
2	0,0009934	0,0549	0,6340	76,61
3	1,46E-05	0,1373	0,9520	75,83
4	0,0003459	0,0695	0,7095	77,58
5	2,14E-05	0,0536	0,7153	74,37
6	0,0005519	0,0969	0,6541	77,49
7	0,0006567	0,1579	0,8224	71,44
8	0,0002560	0,0140	0,6622	77,39
9	0,0003936	0,1244	0,8799	70,57
10	6,59E-06	0,0090	0,7687	69,40
11	**0,00016394**	**0,0001**	**0,6606**	**78,65**
12	1,17E-06	0,0001	0,3000	45,42
13	0,0005424	0,0624	0,5890	76,61
14	0,0003027	0,0101	0,6300	77,68
15	4,00E-05	0,0515	0,6749	75,24

Table 4.6: Results from the Bayesian optimization algorithm with the ResNet34 with three FC blocks, augmentation(roll and rotate +-6 degree) and the "Frac=0.25 Remove Eye" dataset

6 CV results	Validation Acc	Test Acc
Average Acc	74,64 (5,4)	78,65 (3,1)

Table 4.7: Best six-fold CV result from the Bayesian optimization using the ResNet34 model with three FC blocks, augmentation(roll and rotate +-6 degree) and the "Frac=0.25 Remove Eye" dataset. Values in brackets are the standard deviation

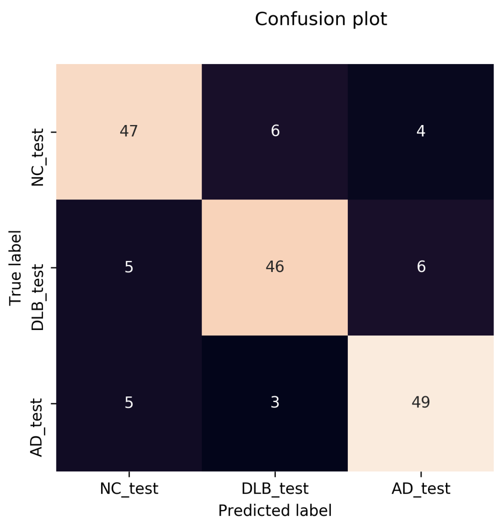
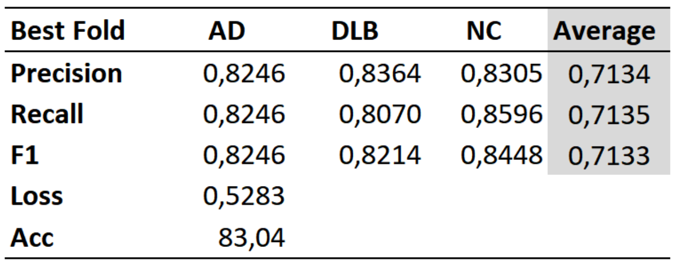


Figure 4.25: Confusion plot of Fold 6 from table 4.7



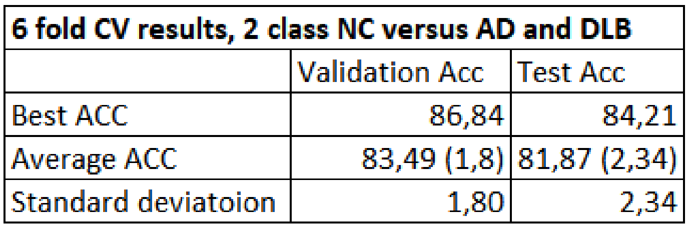
Best Fold	AD	DLB	NC	Average
Precision	0,8246	0,8364	0,8305	0,7134
Recall	0,8246	0,8070	0,8596	0,7135
F1	0,8246	0,8214	0,8448	0,7133
Loss	0,5283			
Acc	83,04			

Figure 4.26: Evaluation metrics from fold 6 from table 4.7. *Loss and Accuracy graph see appendix* B.52

4.9 Two Class Classification

Experiments with a two-class classification problem were conducted to see if the DL model would perform better with fewer classes. The classes tested was NC versus dementia(DLB and AD), and DLB versus AD. AD versus NC was also tested to compare this thesis to other state of the art methods. The experiments used hyperparameters, wich were manually found when conducting experiments to find the best dataset. (Learning rate=0.0009956, momentum=0.537948, L2 weight decay=0.0549, nesterov=True). The hyperparameters found in table 4.6 was tested once, but they provided worse results and was therefore discarded.

NC versus DLB and AD



6 fold CV results, 2 class NC versus AD and DLB		
	Validation Acc	Test Acc
Best ACC	86,84	84,21
Average ACC	83,49 (1,8)	81,87 (2,34)
Standard deviatoion	1,80	2,34

Figure 4.27: Six-fold CV result from experiment with 2 class classification NC versus DLB and AD

	AD	DLB	Average
Average recall	0,8655	0,7251	0,7953
Average precision	0,8648	0,7375	0,8012

Figure 4.28: Average recall and precision from the Six-fold CV result 4.27 from experiment with 2 class classification NC versus DLB and AD

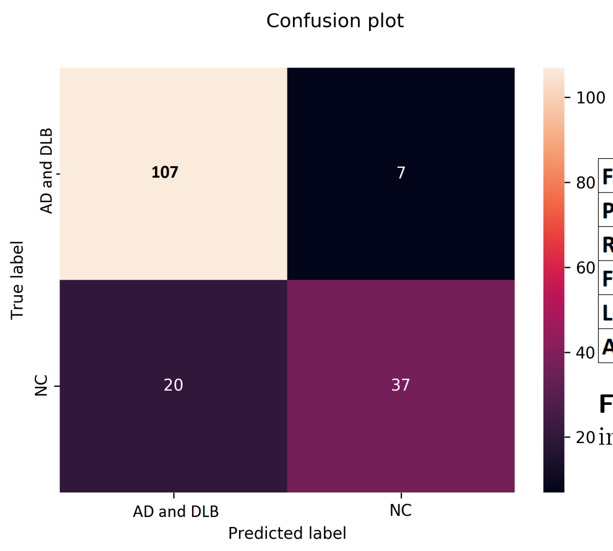


Figure 4.29: Confusion plot of fold 6 from 4.27

Fold 6	AD and DLB	NC
Precision	0,8425	0,8409
Recall	0,9386	0,6491
F1	0,8880	0,7327
Loss	0,3817	
Acc	**84,2105**	

Figure 4.30: Performance of model in fold 6 from 4.27. *Loss and accuracy graph see appendix*B.53

A model was also trained on a dateset with NC vs AD subjects so that the model easier can be compaired with other related works in chapter five, see appendix B.55 for results.

AD versus DLB

6 fold CV results, 2 class AD vs DLB		
	Validation Acc	Test Acc
Best ACC	81,58	90,35
Average ACC	77,4 (2,7)	87,28 (1,95)

Figure 4.31: Six-fold CV result from experiment with 2 class classification DLB versus AD

	AD	DLB	Average
Average recall	0,8684	0,8772	0,8728
Average precision	0,8765	0,8700	0,8732

Figure 4.32: Average recall and precision from the Six-fold CV result 4.31 from experiment with 2 class classification DLB versus AD

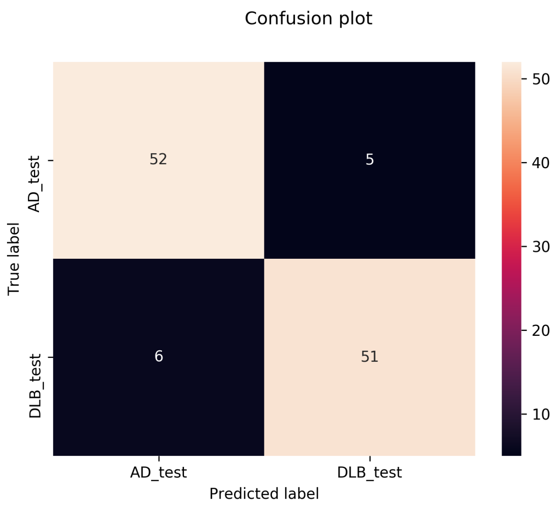
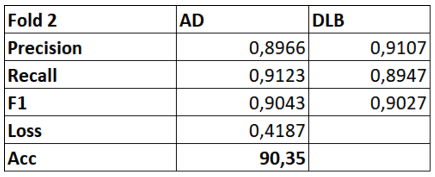


Figure 4.33: Confusion plot of fold 2 from 4.31



Fold 2	**AD**	**DLB**
Precision	0,8966	0,9107
Recall	0,9123	0,8947
F1	0,9043	0,9027
Loss	0,4187	
Acc	**90,35**	

Figure 4.34: Performance of model in fold 2 from 4.31. *Loss and accuracy graph see appendix* B.54

4.10 Federated Learning Experimental Setup

This section contains the FL experimental setup and specifies the datasets selected including benchmarking of these.

4.10.1 Federated learning Dataset Benchmarking

Benchmarking models were trained on the federated datasets; FL1 and FL2 on two different servers. The models were trained with the same parameters except for batch size. The model trained on the GPU with less memory has half the batch size for training and validation witch is 7 instead of 14.

Results for the FL1 dataset model trained on the Tesla-v100-32GB card is 61.9% accuracy. see confusion plot below:

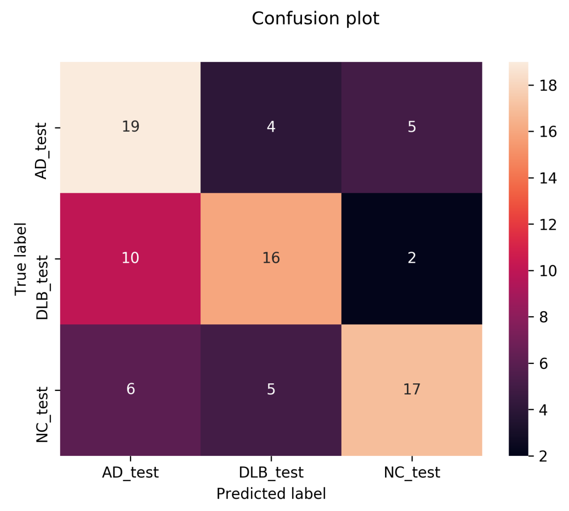
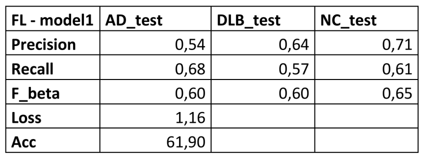


Figure 4.35: Confusion plot of model trained on FL1



FL - model1	AD_test	DLB_test	NC_test
Precision	0,54	0,64	0,71
Recall	0,68	0,57	0,61
F_beta	0,60	0,60	0,65
Loss	1,16		
Acc	61,90		

Figure 4.36: Performance of model trained on FL1

Results for the FL2 dataset model trained on the Tesla-P100-16GB card is 55.17% accuracy. see confusion plot below:

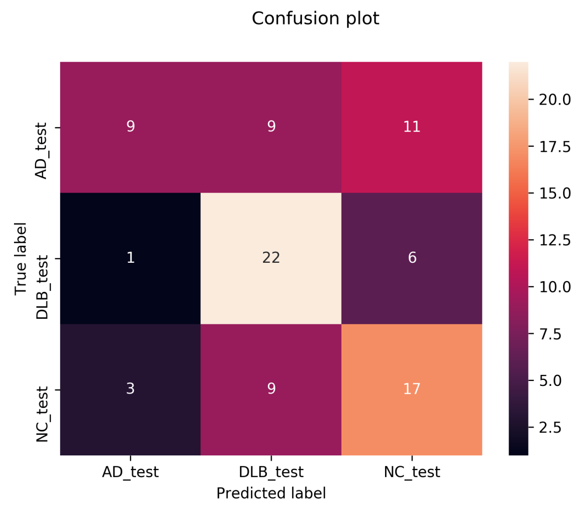
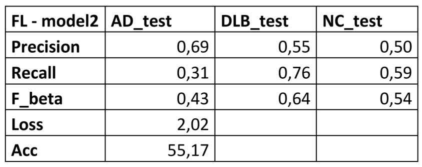


Figure 4.37: Confusion plot of model trained on FL2



FL - model2	AD_test	DLB_test	NC_test
Precision	0,69	0,55	0,50
Recall	0,31	0,76	0,59
F_beta	0,43	0,64	0,54
Loss	2,02		
Acc	55,17		

Figure 4.38: Performance of model trained on FL2

4.10.2 Federated Learning experiment using Federated Averaging

These experiments were conducted by training two sets of models using the FL datasets. Models are trained separately and later used to generate a federated average model. The models are tested for accuracy and loss for comparison between the best locally trained model and the federated version. Sets of models are trained with the same parameters and all use the Adam optimizer. The models are trained without the use of dropout layers in the network as this will most likely reduce the accuracy of a federated average model due to increased randomness.

4.10.3 Asynchronous Federated Learning experiment using Federated Averaging

The asynchronous FL experiment is structured as shown in figure 3.6. This setup is an attempt to implement a closer to real-world scenario structure of the FL training. Two websocket clients; Alice and Bob, serve as the nodes in the network. There is also a third websocket client; Trainer, used for evaluation of the model when training is complete. The central server is a websocket server which receives model parameters from the workers and applies the federated average function to generate a new model which is distributed back to the workers or evaluated by the testing worker if training is complete. All models use the Adam optimizer.

4.11 Federated Learning Experiment Results

This section contains the results of the FL experiments.

4.11.1 Federated Average experiment result

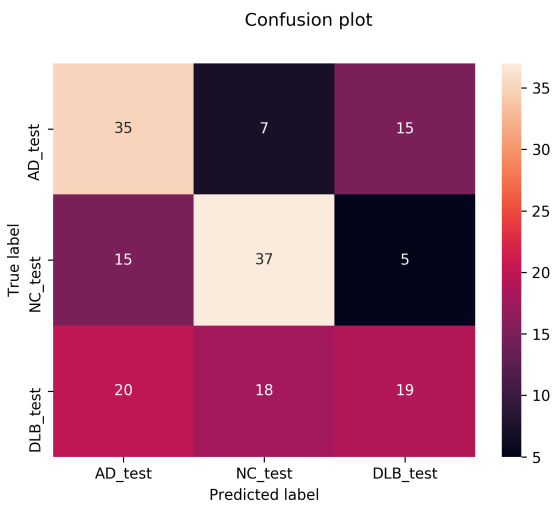
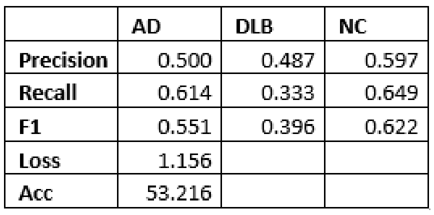


Figure 4.39: Confusion plot of best local model, set 1, trained with the Adam optimizer



	AD	DLB	NC
Precision	0.500	0.487	0.597
Recall	0.614	0.333	0.649
F1	0.551	0.396	0.622
Loss	1.156		
Acc	53.216		

Figure 4.40: Performance of best local model, set 1, trained with the Adam optimizer

Loss and accuracy plot for this model can be found here:(B.56)(B.57)

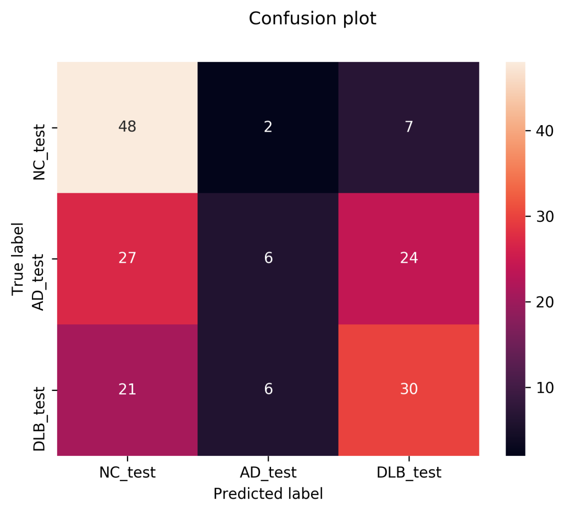
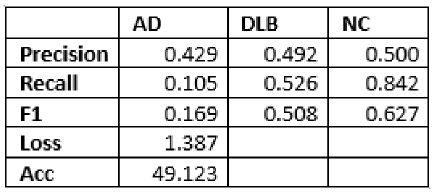


Figure 4.41: Confusion plot of the federated model, set 1, trained with the Adam optimizer



	AD	DLB	NC
Precision	0.429	0.492	0.500
Recall	0.105	0.526	0.842
F1	0.169	0.508	0.627
Loss	1.387		
Acc	49.123		

Figure 4.42: Performance of the federated model, set 1, trained with the Adam optimizer

Loss and accuracy plot for this model can be found here:(B.58)(B.59)

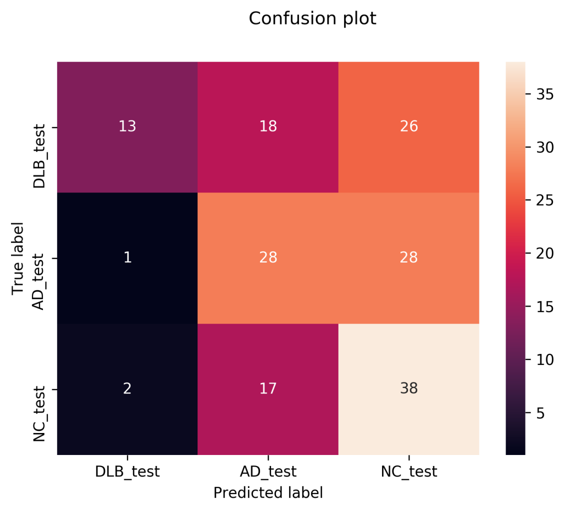
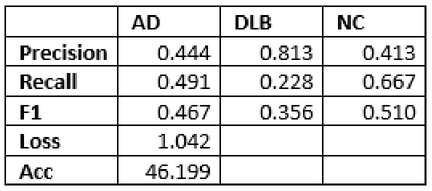


Figure 4.43: Confusion plot of best local model, set 2, trained with the Adam optimizer



	AD	DLB	NC
Precision	0.444	0.813	0.413
Recall	0.491	0.228	0.667
F1	0.467	0.356	0.510
Loss	1.042		
Acc	46.199		

Figure 4.44: Performance of best local model, set 2, trained with the Adam optimizer

Loss and accuracy plot for this model can be found here:(B.60)(B.61)

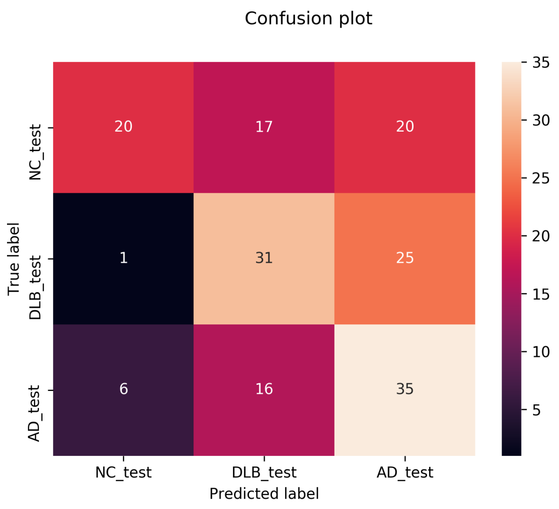
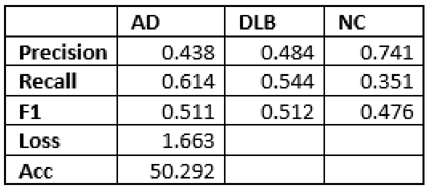


Figure 4.45: Confusion plot of the federated model, set 2, trained with the Adam optimizer



	AD	DLB	NC
Precision	0.438	0.484	0.741
Recall	0.614	0.544	0.351
F1	0.511	0.512	0.476
Loss	1.663		
Acc	50.292		

Figure 4.46: Performance of the federated model, set 2, trained with the Adam optimizer

Loss and accuracy plot for this model can be found here:(B.62)(B.63)

	Loss	Diff Loss (X)	Acc %	Diff Acc %
Best local model set 1	1.156	-	53.216	-
Federated model set 1	1.387	1.200	49.123	-8.332
Best local model set 2	1.042	-	46.199	-
Federated model set 2	1.663	1.596	50.292	8.138

Table 4.8: Results summary table from models set 1 and 2, normal and federated model.

4.11.2 Asynchronous Federated Learning experiment results

Experiment	Alice Accuracy %	Alice Average Loss	Bob Accuracy %	Bob Average Loss	Federated Accuracy %	Federated Average Loss
1	29.89	0.166	39.08	0.164	27.59	0.165
2	34.48	0.167	43.68	0.163	32.18	0.167
3	34.48	0.166	31.03	0.168	25.29	0.167
4	33.33	0.169	29.89	0.164	35.63	0.164
5	36.78	0.089	29.89	0.089	33.33	0.088
6	35.63	0.092	34.48	0.090	34.48	0.091
7	29.89	0.092	33.33	0.094	35.63	0.092
8	29.89	0.094	33.33	0.093	-	-

Table 4.9: Results summary table from asynchronous federated learning experiment.

Chapter 5

Discussion

5.1 Preprocessing and Datasets

It is difficult to see which skull stripping results that are best by inspecting the brains from the figures 4.4 - 4.6, feedback from a radiologist on which brains looks the best would have been useful. The "Frac0.2 Reduce Bias" looks like the best option because it removes very little brain matter and removes most of the unwanted parts. The "Frac0.2 Reduce Bias" was also reported to be the best option to use with the BET2 software in this thesis [57]. However, this was not the case when training a ResNet model on different datasets. The "Frac0.25 Remove Eye" dataset outperformed the rest, see figure 4.1. The "Frac0.25 Remove Eye" is successful at removing unwanted parts, but it struggles with removing parts of the necks, which is very clear in figure 4.6. Nevertheless, the "Frac0.25 Remove Eye" performed the best and was used in the final evaluation.

One possible justification for why the "Frac0.25 Remove Eye" performs better than the "Frac0.2 Reduce Bias" is: because the "Frac0.25 Remove Eye" removes a bit more of the brain, it has more space to move when it is augmented without it causing the MRI image to wrap around itself. Alternatively, the extra bits of the neck in the MRI images help the model figure out that a brain is rotated or not when the rotation augmentation is applied. [57] also reported that with the use of extra "neck removing" software in addition to the BET2 with the "Frac = 0.1 and reduce bias" option, they were able to produce a little better results than with only the BET2.

5.1.1 Federated Learning Data Set

As mentioned in subchapter 3.2.3, the federated learning dataset is divided in a manner which might not be optimal. Since the splitting of Simen Larsen's dataset is random, the

FL1 and FL2 datasets does not have the same balance of age and gender as the original dataset, which may impact the generalization of the result. This might especially be noteworthy if the datasets are used to train separate models. The main justification for not balancing the federated datasets is that both FL1 and FL2 are utilized in the training process for all of the models in the FL experiments. In addition, it is not unreasonable to assume that unbalanced datasets occur in a real world setting where medical centers contribute in a federated learning process.

5.2 Models

The best model to use on the "Frac0.25 resized dataset" was the ResNet18, as it provided the best average result on all the tests except for the test without any extra fully connected layers or dropout, where it was second-best after ResNet101. However, the results of this test do not give the absolute correct answer, only a guideline, because there were many flaws with the testing of the different models. As an example, all the models used the same hyperparameters and this could mean some models got a huge advantage. With testing models in the future, every model should be run with the Bayesian Optimization algorithm to ensure all models use the best hyperparameters. The main lesson from the test was that the pattern from table 4.1 shows that the shallower models performed better with dropout than their deeper counterparts. This can be caused by the deeper models runing through the dropout function many more times than the shallower models, and the deeper models, therefore, lose too many vital nodes. It is not given that the best model on the resized dataset is the best model to use on the fullsized dataset, which was shown to be the case when comparing tables from appendix B.36 and appendix B.40. Where the ResNet34 model was shown to give the best results.

5.3 Augmentations

It can be argued that the way the augmentation experiments are setup is limiting the scope of the testing to "The best single augmentations used in combination" when the best combination might be a combination of two suboptimal single augmentation techniques. This is most likely not the case here because only two techniques were discarded for performing too bad during the single augmentation testing, which was the filp(1) and flip(2) option. Moreover, the ones that were left were extensively tested to make sure no good combinations were left out. However, with limited time, some assumptions were made when testing. For example, if the augmentation technique did not work well with rotation in the XY plane, it would probably not work much better with the rotations in the XZ and YZ planes. Throughout the testing of the different augmentation techniques, see appendix B.4 - B.26, it was clear that most of the augmentations work well. The majority of the models trained with augmentations provide better test accuracy than the models without any augmentations. What seems to be the problem with the different augmentation techniques is to find how much of it to apply to the data when training. With too much augmentation, the accuracy starts to decrease; this is especially evident in the deeper models, see tables in appendix B.25. With too little augmentation, nothing drastic happens, the accuracy stays the same as without augmentation. When the amount of augmentation is just right, the accuracy increases, see tables in appendix B.24. In the first table, the test accuracy is even higher than the validation accuracy. This is most likely because the subjects in the validation results are augmented, while the test results are not.

5.4 GAN

5.4.1 GAN

The generated brains look very real, and it is very hard to differentiate between real and fake ones, see figures 4.7 - 4.12. It is impossible to say for sure with just visual inspection from these figures if the generated brain has the correct disease, as this is not even possible with real brains.

The tests with the generated images supplied with the real data show improvement in accuracy both in validation and testing, see table 4.13 (For detailed metrics, see appendix B.44 - B.47). The improvement in the validation is most likely due to the model finding something in the generated data that easily separates the different generated classes. This was further confirmed when training a model with only the generated data, see figures 4.19 - 4.22. The way the model loss drops to 0 after only a few batches signifies that the computer sees something in the data that is not clear to see for a human. Furthermore, when the model was tested on real MRI images, the model was barely better than random guessing, which means the model probably focuses on some small details specific to the classes in the generated brains. As an example, if the NC generated class had lighter black as the background color, and the other two classes had a darker black as the background, the model would learn to classify the NC class by only looking at the background and not the whole brain. The dataset with 50% added GAN images scored just 0.09% more than the dataset with 25% added, suggesting that adding more GAN images than 50% will probably not increase the test accuracy that much.

When doing the test the other way around to verify that the generated images actually replicated the different diseases with good precision, a model trained on only real MRI images was used to classify all the generated images. Results from this show that the generated images replicated the different diseases with good precision, see figure 4.23. The results might be somewhat biased because the generators used the same images for training as the model used in figure 4.23. The results might have differed if the model was trained on images that the generator had never seen before.

The improvement in the testing accuracy is most likely caused by better generalization with the additional GAN images, which provides the model with more variety in the testing data. However, since the generator had access to all the data while training, it might introduce a data leakage. Because a GAN trained with all the data might have an advantage over a GAN trained with the same amount of data but had a separate test set. When training the GAN images in the future, only the training dataset should be used.

5.4.2 Upscaled GAN

Comparing the results from the tabels in appendix B.29, it is clear to see that without the upscaled GAN images, the model performs better. The upscaling seems only to pollute the real data when training. This is probably caused by the upscaled generated images being too blurry/low quality, see figure 4.14.

5.5 Final Evaluation

5.5.1 Clasification of AD-DLB-NC

There have been few studies using MRI-based differential diagnoses of AD, DLB, and NC. This might be because DLB MRI images are sparse and less accessible than the AD and NC MRI images. The results of the reports with the three-class classification problem can be seen in figure 5.1.

Three-Class Problem: NC versus AD versus DLB

	Accuracy[%]	Averag precision	Average recall	Size of dataset
[65]	73	0.78	0.73	48
[66]	87 (8)	0.88	0.87	109
[23]	67.2 (3.85)	0.73	0.72	861
This thesis	78.65 (3.14)	0.786	0.788	861

Table 5.1: Three-Class Problem: NC versus AD versus DLB. Values in brackets are the standard deviation. The [66] results are the average accuracy of a ten-fold CV, while ours and [23] is the average accuracy of a six-fold CV. The [65] are reporting the accuracy of a single model.

Both [66] [65] had significantly smaller datasets than ours and [23], comparisons between the first-mentioned should be made with reservation as the results might be biased due to unbalanced data. The result of this thesis classification improved last year's results [23] with more than 10 percent. While not being as accurate as [66] in the three classification problem, this thesis method got some redemption when comparing the two-class classification problems NC versus DLB and NC, and DLB versus AD(see table 5.2 and 5.3)

Two-Class Problem: NC versus AD and DLB

	Accuracy[%]	Average precision	Average recall	Size of dataset
[66]	98 (4)	0.985	0.986	109
This thesis	81.87 (2.34)	0.801	0.795	861

Table 5.2: Results of this thesis and related work. Two-Class Problem: NC versus AD and DLB. Values in brackets are the standard deviation. The [66] results is the average accuracy of a ten-fold CV, while ours is the average accuracy of a six-fold CV.

Two-Class Problem: AD versus DLB

	Accuracy[%]	Average precision	Average recall	Size of dataset
[66]	74 (16)	0.625	0.73	73 (57 AD and 16 DLB)
This thesis	87.28 (1.95)	0.87649	0.8728	574

Table 5.3: Two-Class Problem: AD versus DLB. Values in brackets are the standard deviation. The [66] is the average accuracy of a ten-fold CV, while ours is the average accuracy of a six-fold CV.

Opedals method scores almost perfect on NC vs DLB and AD while this thesis struggled which might be because the dataset was not balanced as before because of twice the amount of data in the DLB and AD class. In the AD versus DLB Opedals method falls a little short. With this thesis having a 13% better accuracy. Which shows that the deep learning method is superior in differentiating between DLB and AD.

5.5.2 State of the art

Most other works involving the classification of dementia is mainly focused on AC, MCI and NC. This thesis focused mainly on the three-class problem, but some experiments were done with the 2 class classification to make it easier to compare this work with others.

A resent paper [67] discusses the many problems with reproducible evaluation of different methods related to the classification of AC, MCI and NC. In table 1 in [67] all the "state of the art methods" is analyzed, and data leakage problems are listed in those papers where it is found or suspected. In [67] data leakage is defined as when the test set has been "leaked" into the training set. The methods compared in table 5.4 are the papers with the best-reported results with no reported or suspected data leakages. Most of the results are two-class classification problems as most of the Multi-class classification results either were suspected of data leakage, or had a clear data leakage in them. All the approaches to the classification uses CNNs except for the support-vector machine (SVM).

Study	**Performance**					**Approach**
	AD vs NC	sMCI vs pMCI	MCI vs CN	AD vs MCI	Multi-class	
[68]	-	-	-	-	ACC=0.93[1,2]	2D slice
[69]	ACC=0.91	ACC=0.78[1]	-	-	-	3D patches
[70]	ACC=0.90	-	-	-	-	3D subject
[71]	BA=0.90	-	BA=0.73	BA=0.83	-	ROI-based
[72]	ACC=0.76	-	ACC=0.75	ACC=0.76	-	3D subject
[67]	BA=0.85 (0.04)	BA=0.73 (0.03)	-	-	-	3D subject
[67]	BA=0.88 (0.03)	BA=0.78 (0.07)	-	-	-	3D ROI-based
[67]	BA=0.88 (0.02)	BA=0.70 (0.02)	-	-	-	SVM
Ours	BA=0.83 (0.02)	-	-	-	BA=0.79(0.03) [3]	3D subject

Table 5.4: State of the art comparison. ACC: accuracy; BA: balanced accuracy. Values in round brackets are the standard deviation. MCI: Mild Cognitive Impairment; sMCI: MCI subjects that will remain stable; pMCI: MCI subjects that will progress to AD;

[1] Use of accuracy on a severely imbalanced dataset where one ore more classes is less than half of the other, leading to an over-optimistic estimation of performance.

[2] The classes in the multi-class clasification is: Non-demented, Very mild, Mild, and

Moderate
[3] The classes in the multi-class clasification is: NC vs AD vs DLB

The model used in this thesis might be at an disadvantage when compairing the two-class clasification problems, as the model is deeper and optimized for the three-class clasification problem.

5.6 Federated Learning

There are several lessons to be learned and points worth discussing from the federated learning approach in this thesis. The federated learning branch of machine learning is very much in a pioneer stage when this report is being written. The field is developing rapidly and frameworks upon which federated systems are implemented are improving daily. The next subchapters will discuss some of the main takeaways, including the results from conducted experiments.

5.6.1 Federated Model Generation Method

The federated model generation method/function used in this thesis is the federated average function described in the background chapter. This function is not very complex and is almost certainly not the best choice for many applications. However, the FL field is very young and a limited number of methods have been properly established. The low complexity of the federated average function, and the lack of established options justified it for selection in this thesis. In an event where more time and resources were available, the optimal outcome would be to test several different federated methods and compare to find the option best suited for 3D imaging and MRI applications. Figure 5.1 is a comparison table of federated learning methods, some of which were published after work on this thesis started.

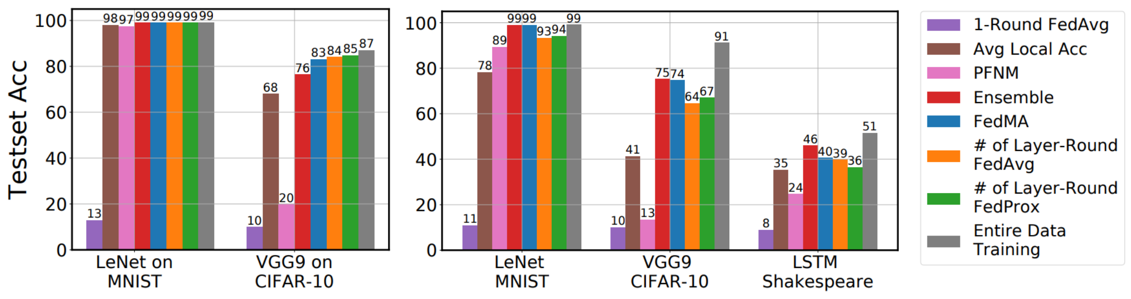


Figure 5.1: Federated Learning methods compared on various datasets. Graph from thesis in public domain with permission[73]

5.6.2 Federated Learning Framework/Software Choice

Most of the code written for the federated learning part of this project is based on the open-source FL framework; PySyft[49], which is an extension of the PyTorch[48] framework. Simen Larsen's code is in large part written on the PyTorch framework, which is a big reason why PySyft was selected. The PySyft framework also seemed fairly well documented and included extensive tutorials, as well as being free to use and alter because of the open-source format.

As the project had been going for a while and more of the PySyft framework was explored, it became clear that the framework was less polished than advertised. Some of the promoted functionality was more a work in progress than working product, and thus complicated the work related to this thesis. Some workarounds had to be implemented and a few manual changes were made to the PySyft library files in order to complete the experiments. These changes are not desirable as replication of experiments will be very difficult. This is also something that is reflected in the results to some extent. Upon reflection, other alternatives such as the Tensorflow[74] framework might have been a better alternative.

5.6.3 Network Structure, Optimizer and Parameter Choices

The network structures of the federated models in this thesis are all based in large part on Simen Larsen's SimenNet. Changes have only been made from the SimenNet structure when necessary to adapt to the federated learning format. The federated model nets were kept similar to SimenNet in part to save time, as hopefully less experimentation would be needed. Another reason for building on the SimenNet structure was that this net already had achieved fairly good accuracy levels on dementia classification using the same dataset.

The batch normalization in the fully connected layers from SimenNet were removed in all of the federated nets. This was done because these layers interfered too much in the learning process and flattened learning progress. The dropout layers in the some of the nets were removed to simplify the models. In the asynchronous federated learning experiments the dropout layers were included and adjusted to various degrees to tweak the model.

The Adam optimizer was used in all of the FL experiments in this thesis*(excluding the FL MNIST example which utilizes SGD).* The SGD optimizer was experimented with in early iterations of some of the FL models, however this did not yield promising results and was scraped early on. The Adam optimizer yielded better results in all instances and is also the optimizer used in Simen Larsen's model.

Parameters in the FL nets were experimented with extensively. Some of the notable parameter adjustments were various levels of dropout between layers, batch size adjustments, learning rate adjustments and changes in number of local training rounds before federating the model weights. All of these adjustments were made to improve the performance the the federated models.

5.6.4 Federated Learning and Privacy

The subject of privacy and protection of data was not directly included in the experimental part of this thesis. A fundamental part of the federated learning structure is protection of data and privacy through the local storage of datasets. Apart from the locally stored data principle, this thesis does not dive deep into the subject. The justification for not focusing more on for instance encryption techniques or similar methods to improve security and privacy, is the time constraint the project is under. Security and privacy should be researched and experimented with further given the opportunity. Suggested research paper[75] by Tien-Dung Cao, Tram Truong-Huu, Hien Tran and Khanh Tran for more details on privacy-preservation and federated learning.

5.6.5 Federated Learning Experiment Results

The results from the federated learning experiments conducted were in many ways a mixed bag, and did not necessarily live up to all expectations. However, there are probably some lessons to be learned and important takeaways from the final results.

The results from the federated average experiment(4.8), shows that training an FL model on 3d images is definitely achievable, although these results did not significantly improve the accuracy of the models. The federated model set 1 fell approximately 8 percent in accuracy from the best locally trained model. The federated model set 2 rose approximately 8 percent in accuracy from the best locally trained model. Seen together these results on average evens out to about the same accuracy as the best locally trained models. It has to be noted that the accuracy might have increased further if the setup had been run for several iterations. It is also notable that the federated models have higher loss values than the best locally trained models.

The results from the asynchronous federated learning experiments(4.9) did not achieve significant accuracy in any of the conducted tests. The average accuracy from the federated model tests came out to be approximately 1/3. This is essentially the same accuracy one would get by rolling a dice in a three class classification problem. The suspected main reason for not being able to produce better models in this setup is related to problems in the PySyft[49] framework. Implementing the structure in an asynchronous manner might also have complicated the process somewhat unnecessarily. Although it is disappointing that better results were not achieved with this setup, it is important to remember that PySyft[49] is an open-source project in a rapidly developing field.

Chapter 6

Conclusion and Future Directions

6.1 Conclusion

6.1.1 GAN and improving the existing classifier

- The preprocessing of the data had a significant impact on how well the models performed. The best preprocessing to use while training was the proposed preprocessing method with the following brain extraction parameters "Frac = 0.25 and Remove Eye = True".
- The model architectures tested all provided decent results. The ResNet34 with three extra FC layers was chosen to be used in the final evaluation because of its good performance and faster training.
- The GAN experiment was somewhat a success. Memory limitations when training the GAN resulted in the generator only producing brains with limited resolution. When the dataset of real brains was resized to match the GAN brains and trained together, the model accuracy increased. This indicates that GAN can be used to supply 3D brains with different types of dementia. The end result did not include GAN images as the higher resolution in the original datasets provided better results than the lower ones.
- The final evaluation used the Bayesian optimization technique to search for the best hyperparameters (learning rate, L2 regularisation, and momentum) while training with the Resnet34 model, the "Frac = 0.25 and Remove Eye = True" dataset and with the best augmentation. The final result got a six-fold CV average accuracy of 78.65% with standard deviation of 3.14%, and a best singular model had an accuracy of 83.04%.

- The 2-class classification problem displayed poor results compared to other state of the art methods when classifying between NC and dementia, but when classifying between AD and DLB. This thesis model outperformed other methods significantly, having a average accuracy of 87.28% and a standard deviation of 1.95.

6.1.2 Federated Learning

- The federated average experiment(4.8) showed promising results which suggest large scale federated learning on 3d MRI images is achievable. Increased iterations might improve the accuracy of the models further.
- The asynchronous federated learning experiment(4.9) did not achieve significant results in terms of accuracy of the models. This is for the most part attributed to the utilized framework; PySyft[49], which had major shortcomings as of the writing of this thesis. The setup would be migrated to an alternative framework like for instance Tensorflow[74], had it not been for the time constraint.

6.2 Future Directions

6.2.1 GAN

Getting the current generated MRI images verified by a radiologist would be interesting to get a definite evaluation of the 3D-α-GAN model.

Training the 3D-α-GAN model with the full-sized dataset and generate new full-sized images. As GANs, in general, have shown good results in producing high-resolution images[36] [37] it should be possible to do it with brains as well. And if a radiologist were to verify the generated images, the full-sized brains would give more reliable results as the lower resolution makes it much harder to spot differences.

6.2.2 Visualizing the Model with Grad-CAM

Because there has only been reported a few biomarkers that can be used in differentiating between AD and DLB, it would be exciting to see what parts of the brain the DL model looks at when discriminating these two diagnoses. The model could maybe give insight to new biomarkers that can be used in future research.

6.2.3 Federated Learning

There is a lot of potential for the future of federated learning in relation to distribution and sharing of medical data and protection of patient privacy. Further research and experimentation should be done in relation to federated learning methods as mentioned in the discussions chapter. The security and privacy protecting aspects of federated learning should definitely be investigated further. It is also probable that the asynchronous federated setup experiment in this thesis would yield better results if migrated to a different federated learning platform. The field of federated learning is very young and rapidly developing. Use cases and performance will undoubtedly improve in the foreseeable future.

List of Figures

List of Tables

-

Appendix A

Appendix A

A.1 requirements.txt

File with the required python packages to run this code on.

A.2 fit.py

This file takes a dictionary as input and executes a model evaluation, until it reaches maximum amount of epochs or runs out of patience. Also the augmentation function is here.

A.3 Main_setup.py

File used for executing experiments. Dictionary with different hyperparameters is made here and sent to fit.py.

A.4 system_resources.py

File that contains functions used in fit.py, Main_setup.py, and test.py. All models are stored in here as well.

A.5 test.py

File containing setup for testing a model on the test set.

A.6 data_resources.py

A file containing code for creating specific metadata needed for the create_dataset function.

A.7 NormalizeSkullStripPipeline.py

Code for running prepossessing.

A.8 TestingAllFoldsInCVfold.py

Code used to test all folds inside a k-fold CV run.

A.9 upscaleGANimages.py

Code used to upscale GAN images.

A.10 Make_new_dataset_from_Simens_balance.py

Code used to make new datasets with Simens dataset balance.

A.11 AD_dataset.py, DLB_dataset.py, NC_dataset.py

Code used to load our dataset into the GAN training.

A.12 federatedAverage.py

File containing code for loading two models and calculating a new federated model.

A.13 start_websocket_server.py

Part of the asynchronous federated learning setup. Starts websocket clients on set ports. *(Requires customized PySyft version)*

A.14 run_websocket_server.py

Part of the asynchronous federated learning setup. File which contains code for the websocket server. *(Requires customized PySyft version)*

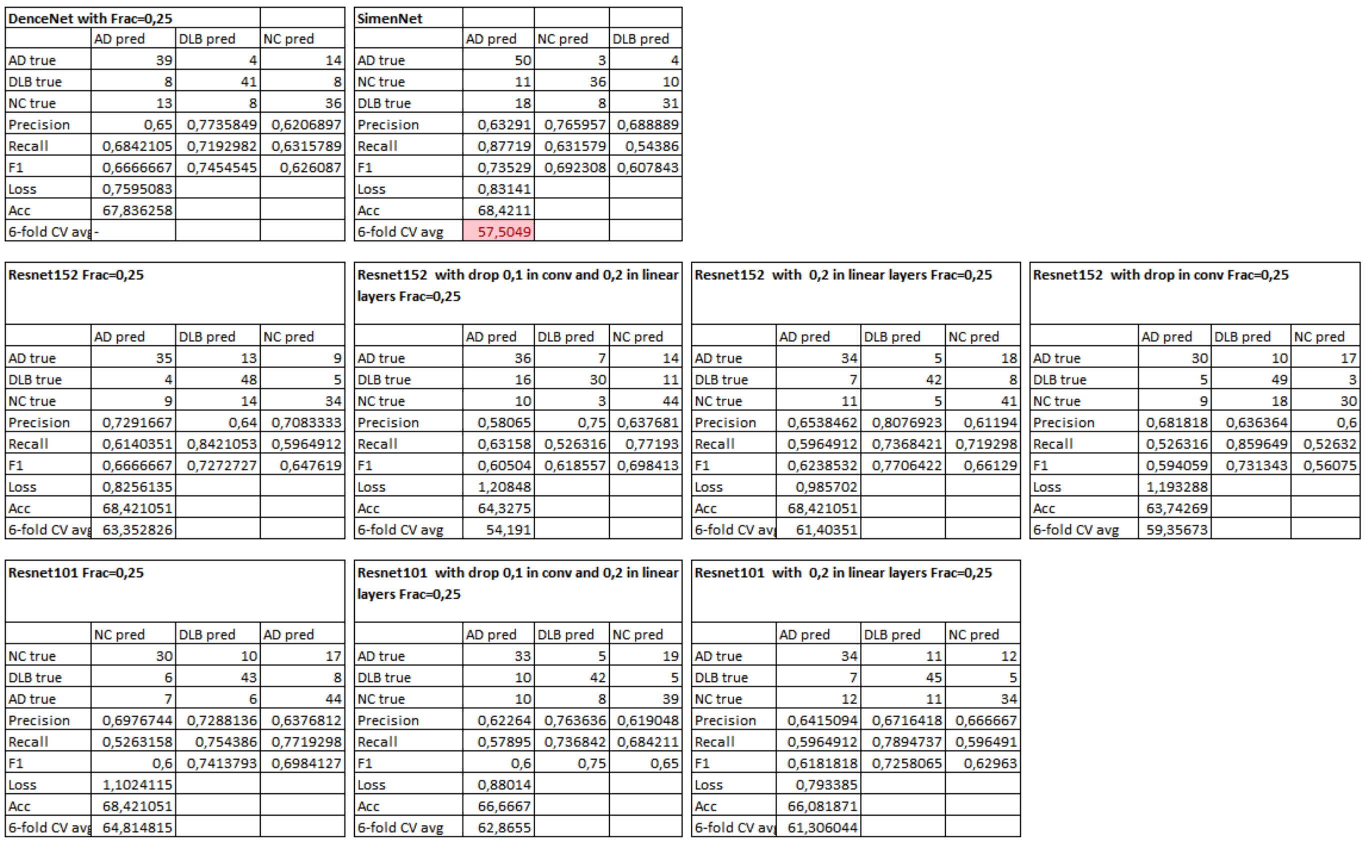
A.15 run_websocket_client.py

Part of the asynchronous federated learning setup. Starts the federated learning process. *(Requires customized PySyft version)* Appendix A

-

Appendix B

Appendix B



DenceNet with Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	39	4	14
DLB true	8	41	8
NC true	13	8	36
Precision	0,65	0,7735849	0,6206897
Recall	0,6842105	0,7192982	0,6315789
F1	0,6666667	0,7454545	0,626087
Loss	0,7595083		
Acc	67,836258		
6-fold CV avg	-		

SimenNet			
	AD pred	NC pred	DLB pred
AD true	50	3	4
NC true	11	36	10
DLB true	18	8	31
Precision	0,63291	0,765957	0,688889
Recall	0,87719	0,631579	0,54386
F1	0,73529	0,692308	0,607843
Loss	0,83141		
Acc	68,4211		
6-fold CV avg	57,5049		

Resnet152 Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	35	13	9
DLB true	4	48	5
NC true	9	14	34
Precision	0,7291667	0,64	0,7083333
Recall	0,6140351	0,8421053	0,5964912
F1	0,6666667	0,7272727	0,647619
Loss	0,8256135		
Acc	68,421051		
6-fold CV avg	63,352826		

Resnet152 with drop 0,1 in conv and 0,2 in linear layers Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	36	7	14
DLB true	16	30	11
NC true	10	3	44
Precision	0,58065	0,75	0,637681
Recall	0,63158	0,526316	0,77193
F1	0,60504	0,618557	0,698413
Loss	1,20848		
Acc	64,3275		
6-fold CV avg	54,191		

Resnet152 with 0,2 in linear layers Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	34	5	18
DLB true	7	42	8
NC true	11	5	41
Precision	0,6538462	0,8076923	0,61194
Recall	0,5964912	0,7368421	0,719298
F1	0,6238532	0,7706422	0,66129
Loss	0,985702		
Acc	68,421051		
6-fold CV av	61,40351		

Resnet152 with drop in conv Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	30	10	17
DLB true	5	49	3
NC true	9	18	30
Precision	0,681818	0,636364	0,6
Recall	0,526316	0,859649	0,52632
F1	0,594059	0,731343	0,56075
Loss	1,193288		
Acc	63,74269		
6-fold CV avg	59,35673		

Resnet101 Frac=0,25			
	NC pred	DLB pred	AD pred
NC true	30	10	17
DLB true	6	43	8
AD true	7	6	44
Precision	0,6976744	0,7288136	0,6376812
Recall	0,5263158	0,754386	0,7719298
F1	0,6	0,7413793	0,6984127
Loss	1,1024115		
Acc	68,421051		
6-fold CV avg	64,814815		

Resnet101 with drop 0,1 in conv and 0,2 in linear layers Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	33	5	19
DLB true	10	42	5
NC true	10	8	39
Precision	0,62264	0,763636	0,619048
Recall	0,57895	0,736842	0,684211
F1	0,6	0,75	0,65
Loss	0,88014		
Acc	66,6667		
6-fold CV avg	62,8655		

Resnet101 with 0,2 in linear layers Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	34	11	12
DLB true	7	45	5
NC true	12	11	34
Precision	0,6415094	0,6716418	0,666667
Recall	0,5964912	0,7894737	0,596491
F1	0,6181818	0,7258065	0,62963
Loss	0,793385		
Acc	66,081871		
6-fold CV av	61,306044		

Table B.1: Results from experiment with different models

Resnet50 Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	35	5	17
DLB true	6	42	9
NC true	7	13	37
Precision	0,7291667	0,7	0,5873016
Recall	0,6140351	0,7368421	0,6491228
F1	0,6666667	0,7179487	0,6166667
Loss	0,9434584		
Acc	66,666664		
6-fold CV avg	59,25926		

Resnet50 with drop 0,1 in conv and 0,2 in linear layers Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	34	5	18
DLB true	10	40	7
NC true	8	7	42
Precision	0,65385	0,769231	0,626866
Recall	0,59649	0,701754	0,736842
F1	0,62385	0,733945	0,677419
Loss	0,87347		
Acc	67,8363		
6-fold CV avg	62,768		

Resnet50 with 0,2 in linear layers Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	35	8	14
DLB true	9	37	11
NC true	8	8	41
Precision	0,6730769	0,6981132	0,621212
Recall	0,6140351	0,6491228	0,719298
F1	0,6422018	0,6727273	0,666667
Loss	1,0607615		
Acc	66,081871		
6-fold CV avg	61,111111		

Resnet50 with drop in conv Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	33	9	15
DLB true	6	45	6
NC true	13	10	34
Precision	0,634615	0,703125	0,61818
Recall	0,578947	0,789474	0,59649
F1	0,605505	0,743802	0,60714
Loss	0,867344		
Acc	65,49708		
6-fold CV avg	60,03899		

Resnet34 Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	41	2	14
DLB true	9	36	12
NC true	10	11	36
Precision	0,6833333	0,7346939	0,5806452
Recall	0,7192982	0,6315789	0,6315789
F1	0,7008547	0,6792453	0,605042
Loss	0,83795		
Acc	66,081871		
6-fold CV avg	60,623781		

Resnet34 with drop 0,1 in conv and 0,2 in linear layers, Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	45	5	7
DLB true	7	37	13
NC true	10	8	39
Precision	0,72581	0,74	0,661017
Recall	0,78947	0,649123	0,684211
F1	0,7563	0,691589	0,672414
Loss	0,73705		
Acc	70,7602		
6-fold CV avg	67,2515		

Resnet34 with 0,2 in linear layers Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	48	4	5
DLB true	12	33	12
NC true	13	8	36
Precision	0,6575342	0,7333333	0,679245
Recall	0,8421053	0,5789474	0,631579
F1	0,7384615	0,6470588	0,654545
Loss	0,8126327		
Acc	68,421051		
6-fold CV avg	64,619883		

Resnet18			
	AD pred	NC pred	DLB pred
AD true	33	11	13
NC true	8	40	9
DLB true	7	8	42
Precision	0,6875	0,6779661	0,65625
Recall	0,5789474	0,7017544	0,7368421
F1	0,6285714	0,6896552	0,6942149
Loss	0,7760452		
Acc	67,251465		
6-fold CV avg	64,619885		

Resnet18 with drop 0,1 in conv and 0,2 in linear layers, Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	44	4	9
DLB true	10	36	11
NC true	5	9	43
Precision	0,74576	0,734694	0,68254
Recall	0,77193	0,631579	0,754386
F1	0,75862	0,679245	0,716667
Loss	0,69824		
Acc	71,9298		
6-fold CV avg	67,8363		

Resnet18 with 0,2 in linear layers Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	42	7	8
DLB true	11	42	4
NC true	8	9	40
Precision	0,6885246	0,7241379	0,769231
Recall	0,7368421	0,7368421	0,701754
F1	0,7118644	0,7304348	0,733945
Loss	0,7552613		
Acc	72,514618		
6-fold CV avg	66,8616		

Resnet18 with drop in conv Frac=0,25			
	AD pred	DLB pred	NC pred
AD true	34	7	16
DLB true	8	41	8
NC true	7	15	35
Precision	0,693878	0,650794	0,59322
Recall	0,596491	0,719298	0,61404
F1	0,641509	0,683333	0,60345
Loss	0,936865		
Acc	64,32748		
6-fold CV avg	61,30604		

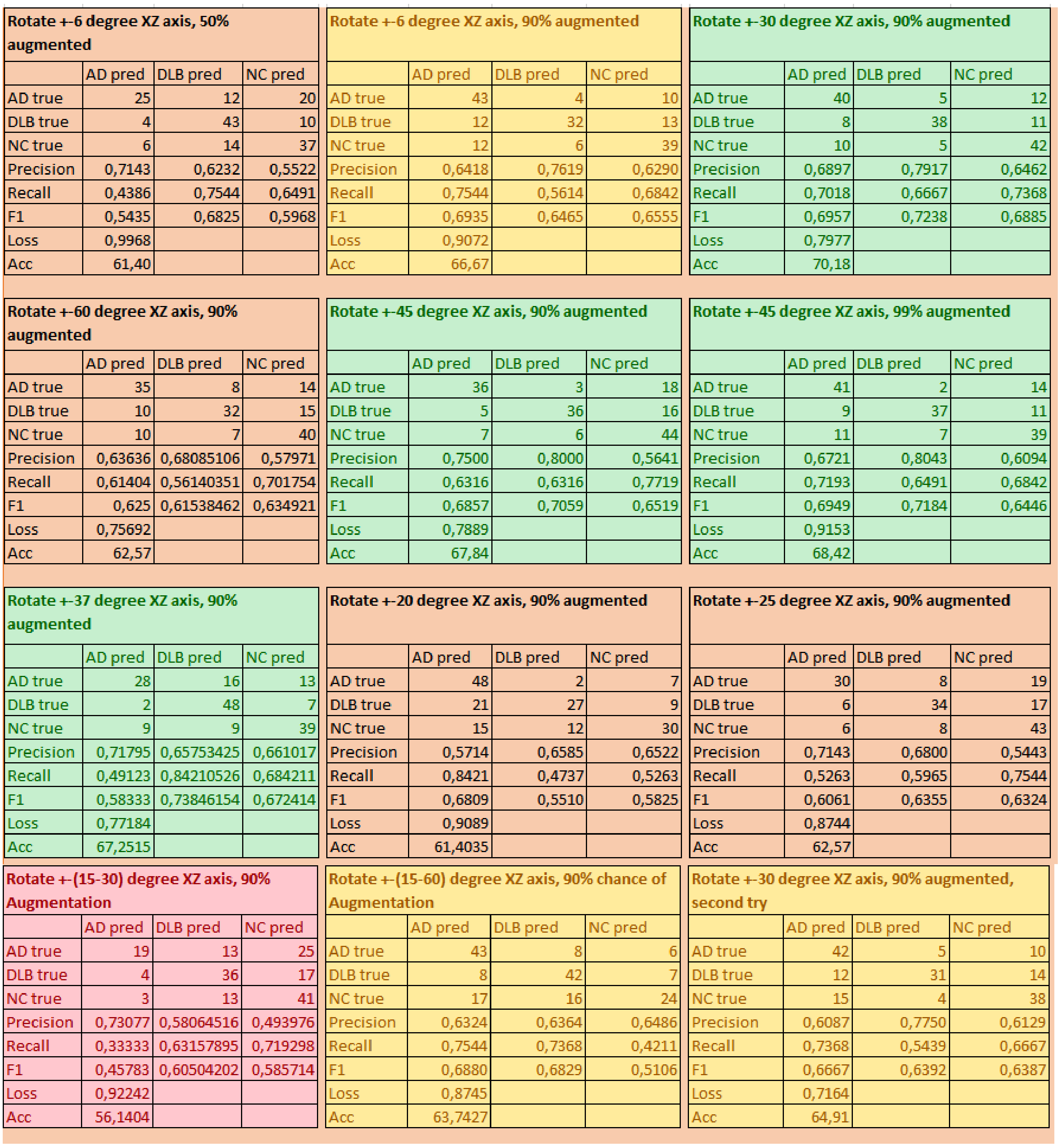
Table B.2: Results from experiment with different models

No Augmentation			
	AD pred	DLB pred	NC pred
AD true	38	11	8
DLB true	14	32	11
NC true	13	10	34
Precision	0,5846	0,6038	0,6415
Recall	0,6667	0,5614	0,5965
F1	0,6230	0,5818	0,6182
Loss	0,8525		
Acc	60,819		

No Augmentation			
	AD pred	DLB pred	NC pred
AD true	42	3	12
DLB true	16	24	17
NC true	12	6	39
Precision	0,6000	0,7273	0,5735
Recall	0,7368	0,4211	0,6842
F1	0,6614	0,5333	0,6240
Loss	0,9697		
Acc	61,404		

No Augmentation			
	AD pred	DLB pred	NC pred
AD true	29	11	17
DLB true	4	31	22
NC true	6	7	44
Precision	0,7436	0,6327	0,5301
Recall	0,5088	0,5439	0,7719
F1	0,6042	0,5849	0,6286
Loss	0,8462		
Acc	60,819		

Table B.3: Results from test without augmentation



Rotate +-6 degree XZ axis, 50% augmented	AD pred	DLB pred	NC pred
AD true	25	12	20
DLB true	4	43	10
NC true	6	14	37
Precision	0,7143	0,6232	0,5522
Recall	0,4386	0,7544	0,6491
F1	0,5435	0,6825	0,5968
Loss	0,9968		
Acc	61,40		

Rotate +-6 degree XZ axis, 90% augmented	AD pred	DLB pred	NC pred
AD true	43	4	10
DLB true	12	32	13
NC true	12	6	39
Precision	0,6418	0,7619	0,6290
Recall	0,7544	0,5614	0,6842
F1	0,6935	0,6465	0,6555
Loss	0,9072		
Acc	66,67		

Rotate +-30 degree XZ axis, 90% augmented	AD pred	DLB pred	NC pred
AD true	40	5	12
DLB true	8	38	11
NC true	10	5	42
Precision	0,6897	0,7917	0,6462
Recall	0,7018	0,6667	0,7368
F1	0,6957	0,7238	0,6885
Loss	0,7977		
Acc	70,18		

Rotate +-60 degree XZ axis, 90% augmented	AD pred	DLB pred	NC pred
AD true	35	8	14
DLB true	10	32	15
NC true	10	7	40
Precision	0,63636	0,68085106	0,57971
Recall	0,61404	0,56140351	0,701754
F1	0,625	0,61538462	0,634921
Loss	0,75692		
Acc	62,57		

Rotate +-45 degree XZ axis, 90% augmented	AD pred	DLB pred	NC pred
AD true	36	3	18
DLB true	5	36	16
NC true	7	6	44
Precision	0,7500	0,8000	0,5641
Recall	0,6316	0,6316	0,7719
F1	0,6857	0,7059	0,6519
Loss	0,7889		
Acc	67,84		

Rotate +-45 degree XZ axis, 99% augmented	AD pred	DLB pred	NC pred
AD true	41	2	14
DLB true	9	37	11
NC true	11	7	39
Precision	0,6721	0,8043	0,6094
Recall	0,7193	0,6491	0,6842
F1	0,6949	0,7184	0,6446
Loss	0,9153		
Acc	68,42		

Rotate +-37 degree XZ axis, 90% augmented	AD pred	DLB pred	NC pred
AD true	28	16	13
DLB true	2	48	7
NC true	9	9	39
Precision	0,71795	0,65753425	0,661017
Recall	0,49123	0,84210526	0,684211
F1	0,58333	0,73846154	0,672414
Loss	0,77184		
Acc	67,2515		

Rotate +-20 degree XZ axis, 90% augmented	AD pred	DLB pred	NC pred
AD true	48	2	7
DLB true	21	27	9
NC true	15	12	30
Precision	0,5714	0,6585	0,6522
Recall	0,8421	0,4737	0,5263
F1	0,6809	0,5510	0,5825
Loss	0,9089		
Acc	61,4035		

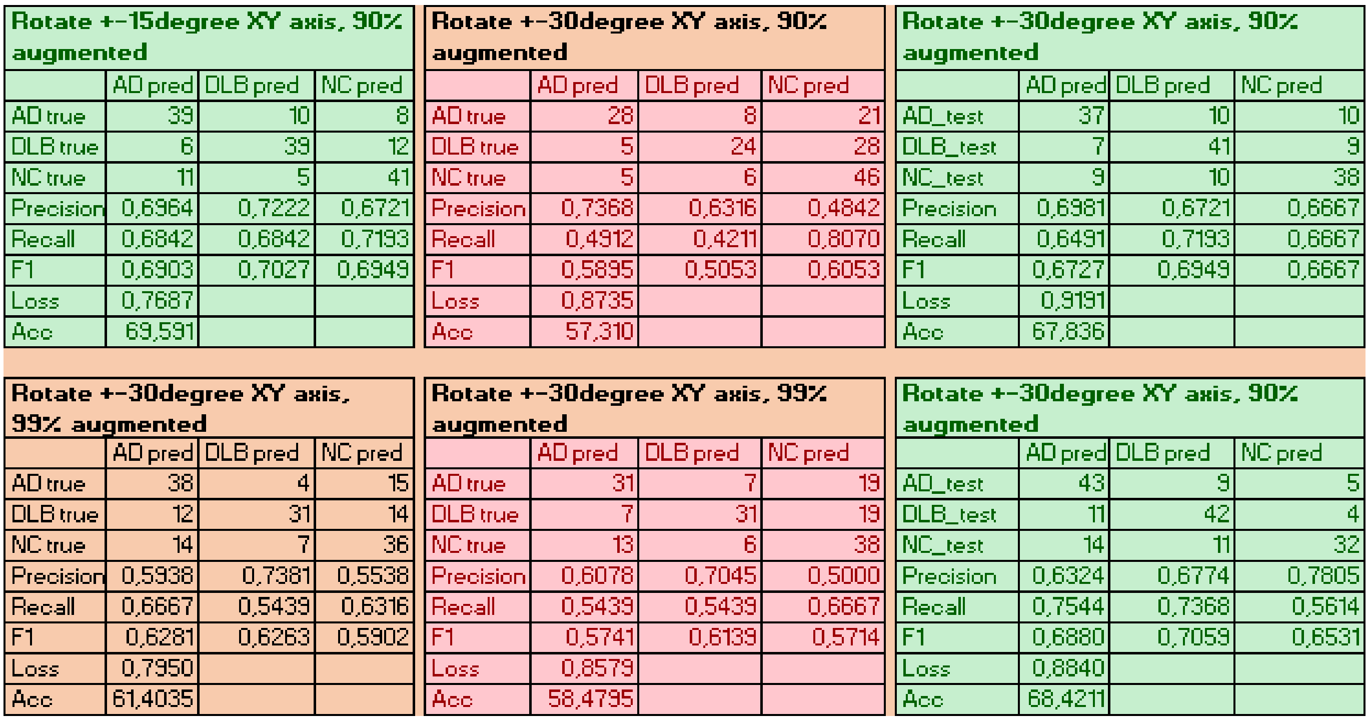
Rotate +-25 degree XZ axis, 90% augmented	AD pred	DLB pred	NC pred
AD true	30	8	19
DLB true	6	34	17
NC true	6	8	43
Precision	0,7143	0,6800	0,5443
Recall	0,5263	0,5965	0,7544
F1	0,6061	0,6355	0,6324
Loss	0,8744		
Acc	62,57		

Rotate +-(15-30) degree XZ axis, 90% Augmentation	AD pred	DLB pred	NC pred
AD true	19	13	25
DLB true	4	36	17
NC true	3	13	41
Precision	0,73077	0,58064516	0,493976
Recall	0,33333	0,63157895	0,719298
F1	0,45783	0,60504202	0,585714
Loss	0,92242		
Acc	56,1404		

Rotate +-(15-60) degree XZ axis, 90% chance of Augmentation	AD pred	DLB pred	NC pred
AD true	43	8	6
DLB true	8	42	7
NC true	17	16	24
Precision	0,6324	0,6364	0,6486
Recall	0,7544	0,7368	0,4211
F1	0,6880	0,6829	0,5106
Loss	0,8745		
Acc	63,7427		

Rotate +-30 degree XZ axis, 90% augmented, second try	AD pred	DLB pred	NC pred
AD true	42	5	10
DLB true	12	31	14
NC true	15	4	38
Precision	0,6087	0,7750	0,6129
Recall	0,7368	0,5439	0,6667
F1	0,6667	0,6392	0,6387
Loss	0,7164		
Acc	64,91		

Table B.4: Augmentation Results from Rotate in XZ direction



Rotate +-15degree XY axis, 90% augmented

	AD pred	DLB pred	NC pred
AD true	39	10	8
DLB true	6	39	12
NC true	11	5	41
Precision	0,6964	0,7222	0,6721
Recall	0,6842	0,6842	0,7193
F1	0,6903	0,7027	0,6949
Loss	0,7687		
Acc	69,591		

Rotate +-30degree XY axis, 90% augmented

	AD pred	DLB pred	NC pred
AD true	28	8	21
DLB true	5	24	28
NC true	5	6	46
Precision	0,7368	0,6316	0,4842
Recall	0,4912	0,4211	0,8070
F1	0,5895	0,5053	0,6053
Loss	0,8735		
Acc	57,310		

Rotate +-30degree XY axis, 90% augmented

	AD pred	DLB pred	NC pred
AD_test	37	10	10
DLB_test	7	41	9
NC_test	9	10	38
Precision	0,6981	0,6721	0,6667
Recall	0,6491	0,7193	0,6667
F1	0,6727	0,6949	0,6667
Loss	0,9191		
Acc	67,836		

Rotate +-30degree XY axis, 99% augmented

	AD pred	DLB pred	NC pred
AD true	38	4	15
DLB true	12	31	14
NC true	14	7	36
Precision	0,5938	0,7381	0,5538
Recall	0,6667	0,5439	0,6316
F1	0,6281	0,6263	0,5902
Loss	0,7950		
Acc	61,4035		

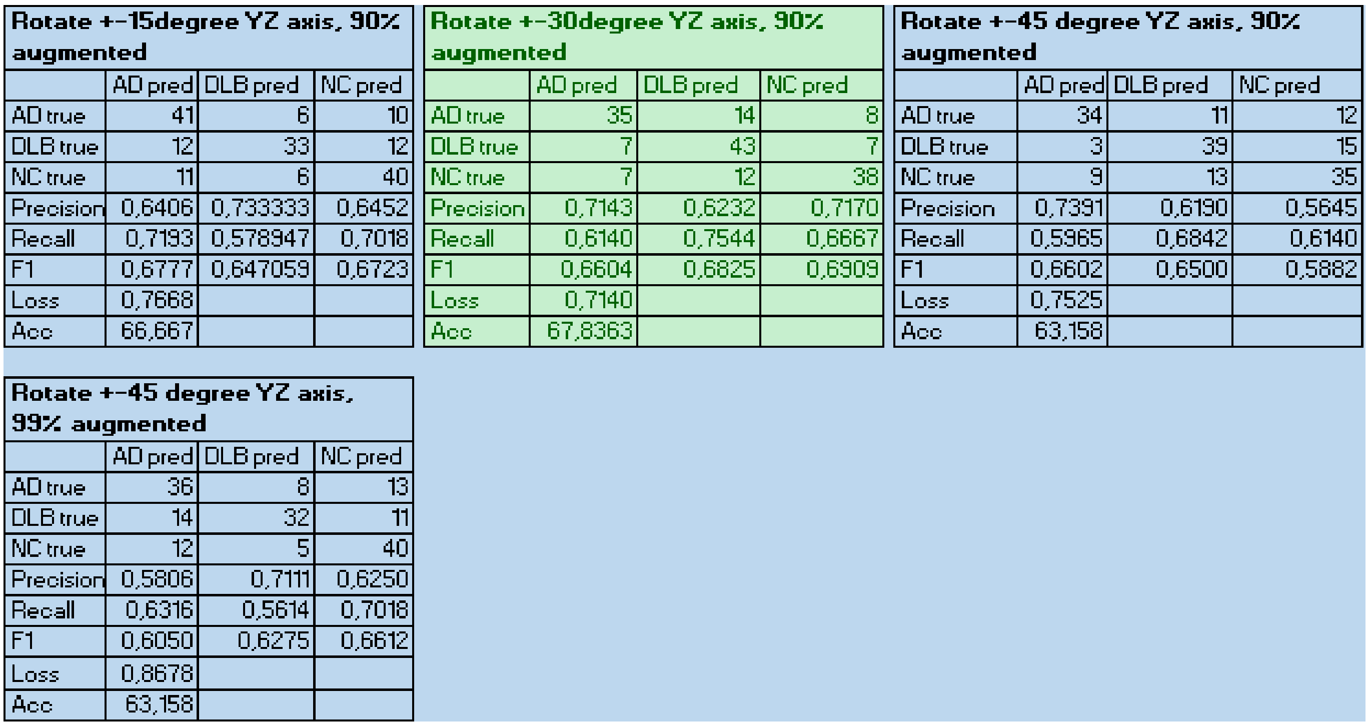
Rotate +-30degree XY axis, 99% augmented

	AD pred	DLB pred	NC pred
AD true	31	7	19
DLB true	7	31	19
NC true	13	6	38
Precision	0,6078	0,7045	0,5000
Recall	0,5439	0,5439	0,6667
F1	0,5741	0,6139	0,5714
Loss	0,8579		
Acc	58,4795		

Rotate +-30degree XY axis, 90% augmented

	AD pred	DLB pred	NC pred
AD_test	43	9	5
DLB_test	11	42	4
NC_test	14	11	32
Precision	0,6324	0,6774	0,7805
Recall	0,7544	0,7368	0,5614
F1	0,6880	0,7059	0,6531
Loss	0,8840		
Acc	68,4211		

Table B.5: Augmentation Results from Rotate in XY direction



Rotate +-15degree YZ axis, 90% augmented

	AD pred	DLB pred	NC pred
AD true	41	6	10
DLB true	12	33	12
NC true	11	6	40
Precision	0,6406	0,733333	0,6452
Recall	0,7193	0,578947	0,7018
F1	0,6777	0,647059	0,6723
Loss	0,7668		
Acc	66,667		

Rotate +-30degree YZ axis, 90% augmented

	AD pred	DLB pred	NC pred
AD true	35	14	8
DLB true	7	43	7
NC true	7	12	38
Precision	0,7143	0,6232	0,7170
Recall	0,6140	0,7544	0,6667
F1	0,6604	0,6825	0,6909
Loss	0,7140		
Acc	67,8363		

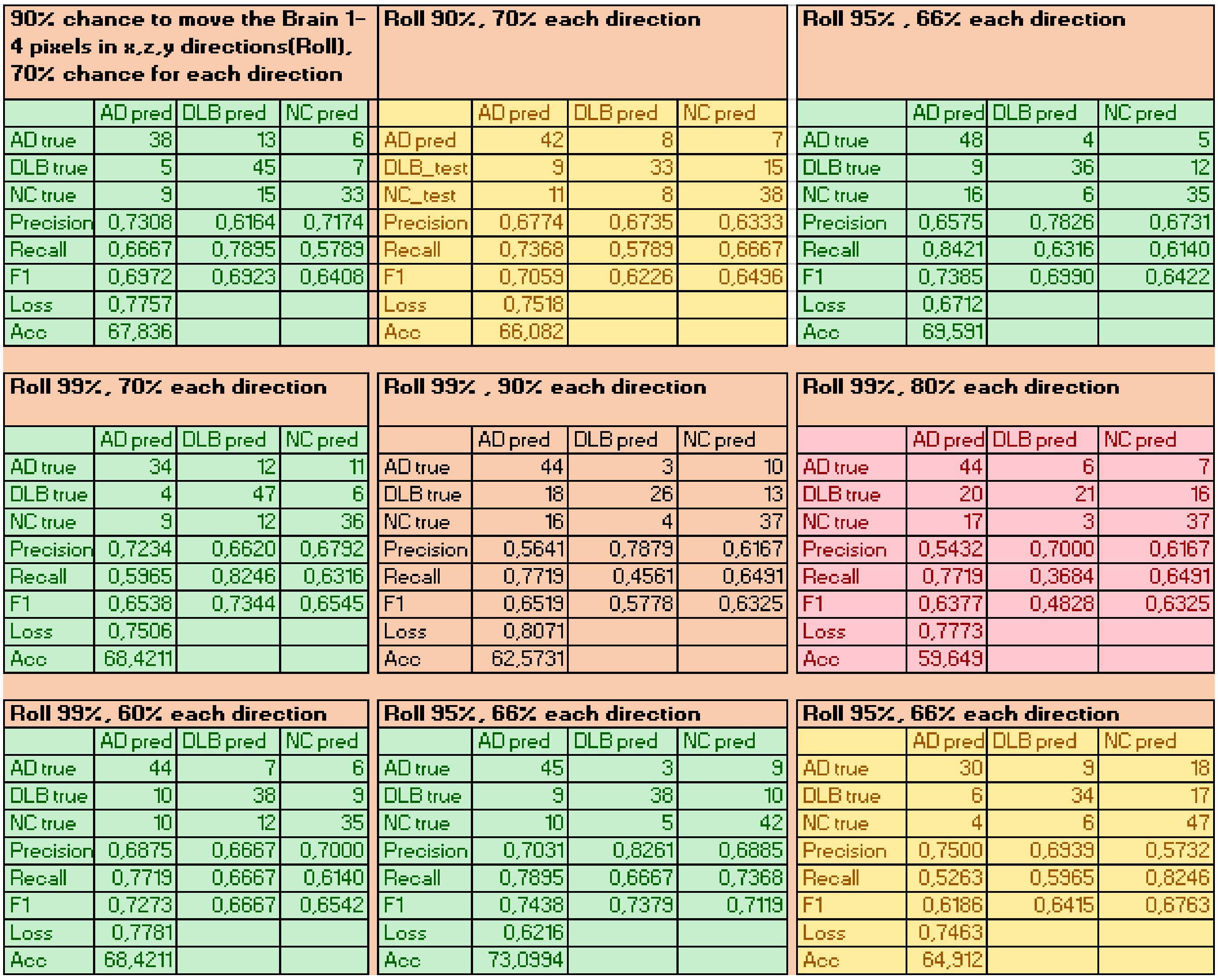
Rotate +-45 degree YZ axis, 90% augmented

	AD pred	DLB pred	NC pred
AD true	34	11	12
DLB true	3	39	15
NC true	9	13	35
Precision	0,7391	0,6190	0,5645
Recall	0,5965	0,6842	0,6140
F1	0,6602	0,6500	0,5882
Loss	0,7525		
Acc	63,158		

Rotate +-45 degree YZ axis, 99% augmented

	AD pred	DLB pred	NC pred
AD true	36	8	13
DLB true	14	32	11
NC true	12	5	40
Precision	0,5806	0,7111	0,6250
Recall	0,6316	0,5614	0,7018
F1	0,6050	0,6275	0,6612
Loss	0,8678		
Acc	63,158		

Table B.6: Augmentation Results from Rotate in YZ direction



90% chance to move the Brain 1-4 pixels in x,z,y directions(Roll), 70% chance for each direction

	AD pred	DLB pred	NC pred
AD true	38	13	6
DLB true	5	45	7
NC true	9	15	33
Precision	0,7308	0,6164	0,7174
Recall	0,6667	0,7895	0,5789
F1	0,6972	0,6923	0,6408
Loss	0,7757		
Acc	67,836		

Roll 90%, 70% each direction

	AD pred	DLB pred	NC pred
AD pred	42	8	7
DLB_test	9	33	15
NC_test	11	8	38
Precision	0,6774	0,6735	0,6333
Recall	0,7368	0,5789	0,6667
F1	0,7059	0,6226	0,6496
Loss	0,7518		
Acc	66,082		

Roll 95% , 66% each direction

	AD pred	DLB pred	NC pred
AD true	48	4	5
DLB true	9	36	12
NC true	16	6	35
Precision	0,6575	0,7826	0,6731
Recall	0,8421	0,6316	0,6140
F1	0,7385	0,6990	0,6422
Loss	0,6712		
Acc	69,591		

Roll 99%, 70% each direction

	AD pred	DLB pred	NC pred
AD true	34	12	11
DLB true	4	47	6
NC true	9	12	36
Precision	0,7234	0,6620	0,6792
Recall	0,5965	0,8246	0,6316
F1	0,6538	0,7344	0,6545
Loss	0,7506		
Acc	68,4211		

Roll 99% , 90% each direction

	AD pred	DLB pred	NC pred
AD true	44	3	10
DLB true	18	26	13
NC true	16	4	37
Precision	0,5641	0,7879	0,6167
Recall	0,7719	0,4561	0,6491
F1	0,6519	0,5778	0,6325
Loss	0,8071		
Acc	62,5731		

Roll 99%, 80% each direction

	AD pred	DLB pred	NC pred
AD true	44	6	7
DLB true	20	21	16
NC true	17	3	37
Precision	0,5432	0,7000	0,6167
Recall	0,7719	0,3684	0,6491
F1	0,6377	0,4828	0,6325
Loss	0,7773		
Acc	59,649		

Roll 99%, 60% each direction

	AD pred	DLB pred	NC pred
AD true	44	7	6
DLB true	10	38	9
NC true	10	12	35
Precision	0,6875	0,6667	0,7000
Recall	0,7719	0,6667	0,6140
F1	0,7273	0,6667	0,6542
Loss	0,7781		
Acc	68,4211		

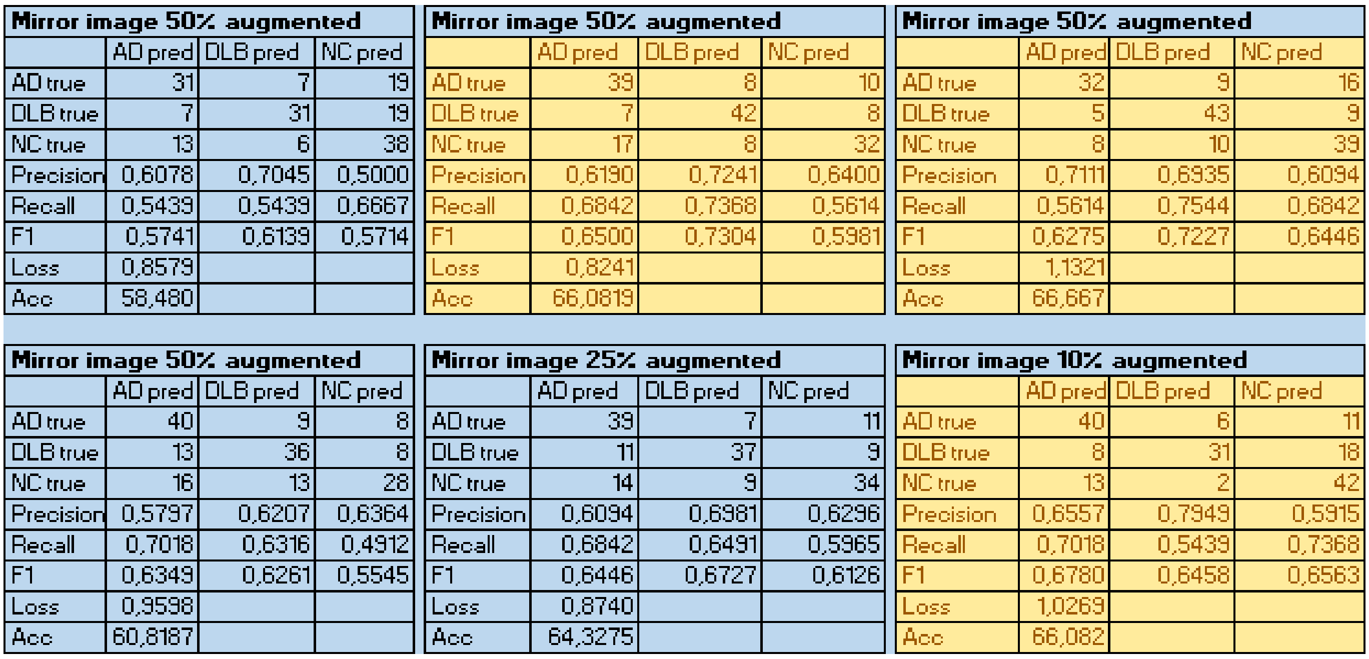
Roll 95%, 66% each direction

	AD pred	DLB pred	NC pred
AD true	45	3	9
DLB true	9	38	10
NC true	10	5	42
Precision	0,7031	0,8261	0,6885
Recall	0,7895	0,6667	0,7368
F1	0,7438	0,7379	0,7119
Loss	0,6216		
Acc	73,0994		

Roll 95%, 66% each direction

	AD pred	DLB pred	NC pred
AD true	30	9	18
DLB true	6	34	17
NC true	4	6	47
Precision	0,7500	0,6939	0,5732
Recall	0,5263	0,5965	0,8246
F1	0,6186	0,6415	0,6763
Loss	0,7463		
Acc	64,912		

Table B.7: Augmentation Results from Translating/Roll



Mirror image 50% augmented	AD pred	DLB pred	NC pred
AD true	31	7	19
DLB true	7	31	19
NC true	13	6	38
Precision	0,6078	0,7045	0,5000
Recall	0,5439	0,5439	0,6667
F1	0,5741	0,6139	0,5714
Loss	0,8579		
Acc	58,480		

Mirror image 50% augmented	AD pred	DLB pred	NC pred
AD true	39	8	10
DLB true	7	42	8
NC true	17	8	32
Precision	0,6190	0,7241	0,6400
Recall	0,6842	0,7368	0,5614
F1	0,6500	0,7304	0,5981
Loss	0,8241		
Acc	66,0819		

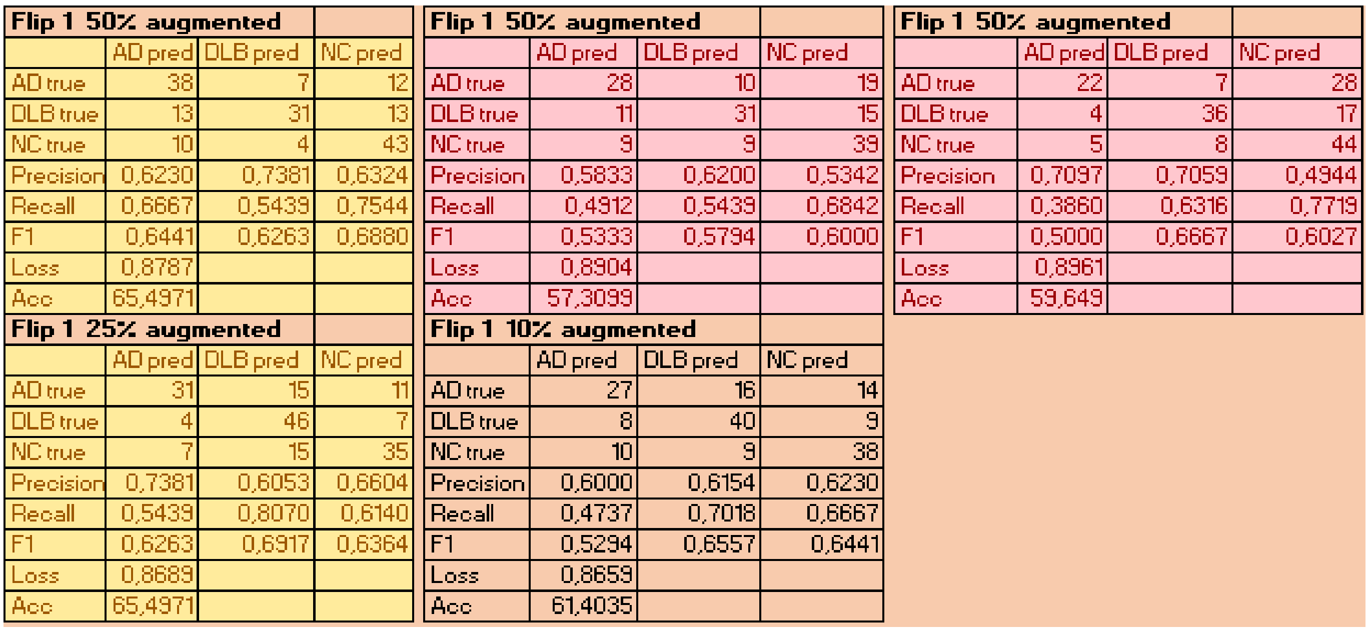
Mirror image 50% augmented	AD pred	DLB pred	NC pred
AD true	32	9	16
DLB true	5	43	9
NC true	8	10	39
Precision	0,7111	0,6935	0,6094
Recall	0,5614	0,7544	0,6842
F1	0,6275	0,7227	0,6446
Loss	1,1321		
Acc	66,667		

Mirror image 50% augmented	AD pred	DLB pred	NC pred
AD true	40	9	8
DLB true	13	36	8
NC true	16	13	28
Precision	0,5797	0,6207	0,6364
Recall	0,7018	0,6316	0,4912
F1	0,6349	0,6261	0,5545
Loss	0,9598		
Acc	60,8187		

Mirror image 25% augmented	AD pred	DLB pred	NC pred
AD true	39	7	11
DLB true	11	37	9
NC true	14	9	34
Precision	0,6094	0,6981	0,6296
Recall	0,6842	0,6491	0,5965
F1	0,6446	0,6727	0,6126
Loss	0,8740		
Acc	64,3275		

Mirror image 10% augmented	AD pred	DLB pred	NC pred
AD true	40	6	11
DLB true	8	31	18
NC true	13	2	42
Precision	0,6557	0,7949	0,5915
Recall	0,7018	0,5439	0,7368
F1	0,6780	0,6458	0,6563
Loss	1,0269		
Acc	66,082		

Table B.8: Augmentation Results from Mirroring(Flip(0))



Flip 1 50% augmented	AD pred	DLB pred	NC pred
AD true	38	7	12
DLB true	13	31	13
NC true	10	4	43
Precision	0,6230	0,7381	0,6324
Recall	0,6667	0,5439	0,7544
F1	0,6441	0,6263	0,6880
Loss	0,8787		
Acc	65,4971		

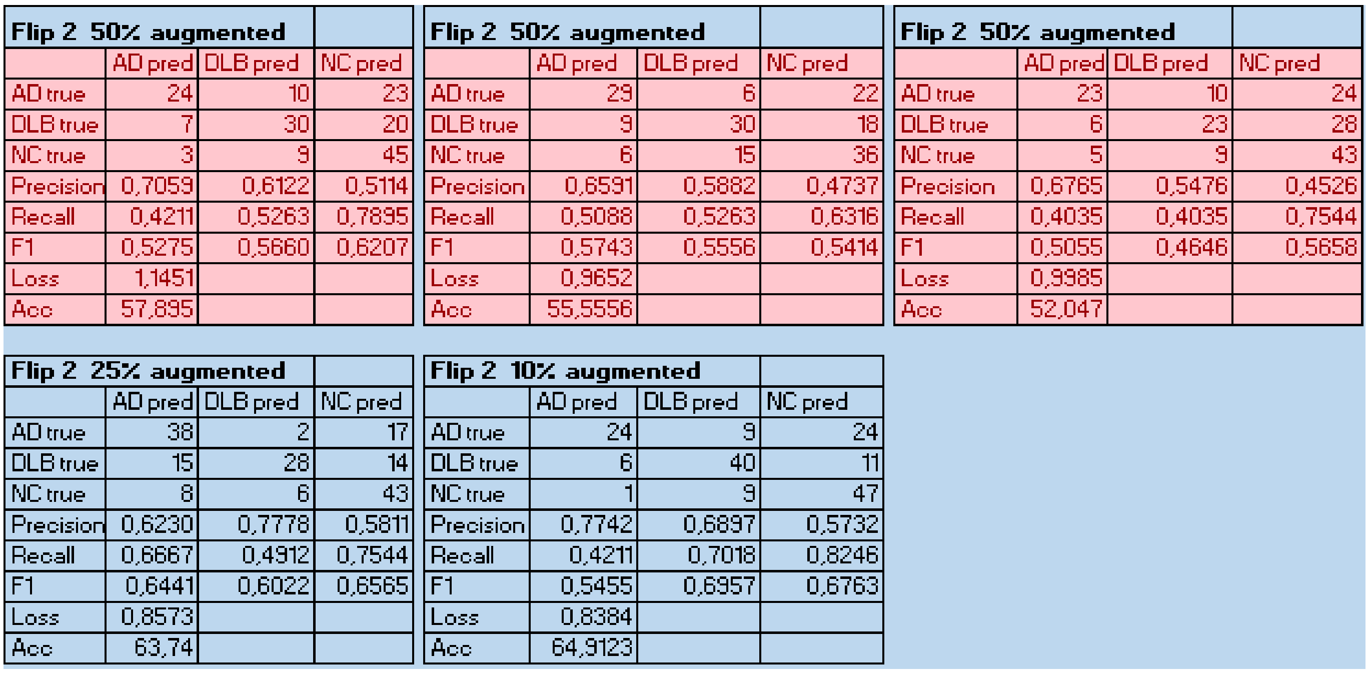
Flip 1 50% augmented	AD pred	DLB pred	NC pred
AD true	28	10	19
DLB true	11	31	15
NC true	9	9	39
Precision	0,5833	0,6200	0,5342
Recall	0,4912	0,5439	0,6842
F1	0,5333	0,5794	0,6000
Loss	0,8904		
Acc	57,3099		

Flip 1 50% augmented	AD pred	DLB pred	NC pred
AD true	22	7	28
DLB true	4	36	17
NC true	5	8	44
Precision	0,7097	0,7059	0,4944
Recall	0,3860	0,6316	0,7719
F1	0,5000	0,6667	0,6027
Loss	0,8961		
Acc	59,649		

Flip 1 25% augmented	AD pred	DLB pred	NC pred
AD true	31	15	11
DLB true	4	46	7
NC true	7	15	35
Precision	0,7381	0,6053	0,6604
Recall	0,5439	0,8070	0,6140
F1	0,6263	0,6917	0,6364
Loss	0,8689		
Acc	65,4971		

Flip 1 10% augmented	AD pred	DLB pred	NC pred
AD true	27	16	14
DLB true	8	40	9
NC true	10	9	38
Precision	0,6000	0,6154	0,6230
Recall	0,4737	0,7018	0,6667
F1	0,5294	0,6557	0,6441
Loss	0,8659		
Acc	61,4035		

Table B.9: Augmentation Results from flipping left/right (Flip(1))



Flip 2 50% augmented			
	AD pred	DLB pred	NC pred
AD true	24	10	23
DLB true	7	30	20
NC true	3	9	45
Precision	0,7059	0,6122	0,5114
Recall	0,4211	0,5263	0,7895
F1	0,5275	0,5660	0,6207
Loss	1,1451		
Acc	57,895		

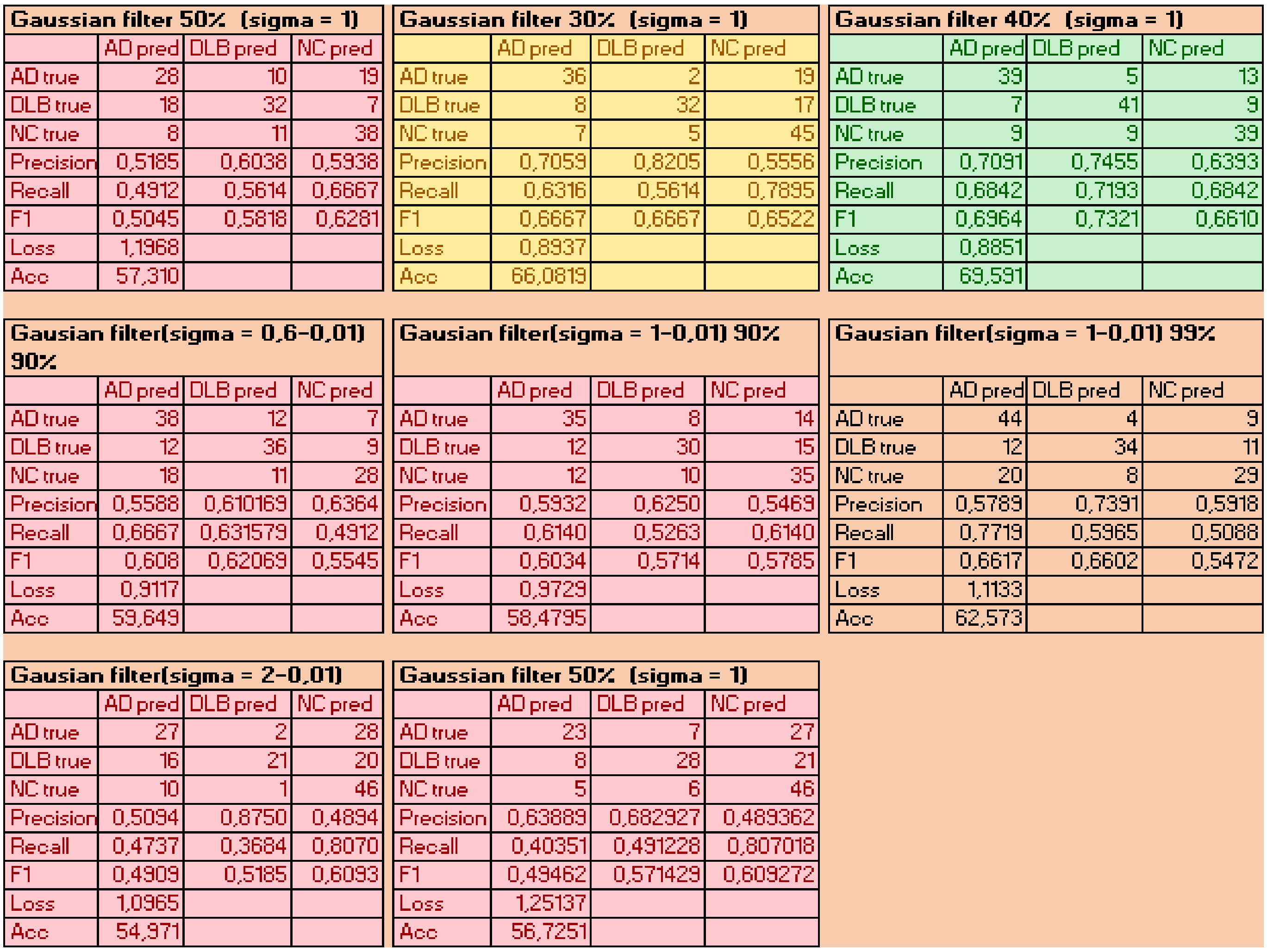
Flip 2 50% augmented			
	AD pred	DLB pred	NC pred
AD true	29	6	22
DLB true	9	30	18
NC true	6	15	36
Precision	0,6591	0,5882	0,4737
Recall	0,5088	0,5263	0,6316
F1	0,5743	0,5556	0,5414
Loss	0,9652		
Acc	55,5556		

Flip 2 50% augmented			
	AD pred	DLB pred	NC pred
AD true	23	10	24
DLB true	6	23	28
NC true	5	9	43
Precision	0,6765	0,5476	0,4526
Recall	0,4035	0,4035	0,7544
F1	0,5055	0,4646	0,5658
Loss	0,9985		
Acc	52,047		

Flip 2 25% augmented			
	AD pred	DLB pred	NC pred
AD true	38	2	17
DLB true	15	28	14
NC true	8	6	43
Precision	0,6230	0,7778	0,5811
Recall	0,6667	0,4912	0,7544
F1	0,6441	0,6022	0,6565
Loss	0,8573		
Acc	63,74		

Flip 2 10% augmented			
	AD pred	DLB pred	NC pred
AD true	24	9	24
DLB true	6	40	11
NC true	1	9	47
Precision	0,7742	0,6897	0,5732
Recall	0,4211	0,7018	0,8246
F1	0,5455	0,6957	0,6763
Loss	0,8384		
Acc	64,9123		

Table B.10: Augmentation Results from flipping upside/down (Flip(2))



Gaussian filter 50% (sigma = 1)			
	AD pred	DLB pred	NC pred
AD true	28	10	19
DLB true	18	32	7
NC true	8	11	38
Precision	0,5185	0,6038	0,5938
Recall	0,4912	0,5614	0,6667
F1	0,5045	0,5818	0,6281
Loss	1,1968		
Acc	57,310		

Gaussian filter 30% (sigma = 1)			
	AD pred	DLB pred	NC pred
AD true	36	2	19
DLB true	8	32	17
NC true	7	5	45
Precision	0,7059	0,8205	0,5556
Recall	0,6316	0,5614	0,7895
F1	0,6667	0,6667	0,6522
Loss	0,8937		
Acc	66,0819		

Gaussian filter 40% (sigma = 1)			
	AD pred	DLB pred	NC pred
AD true	39	5	13
DLB true	7	41	9
NC true	9	9	39
Precision	0,7091	0,7455	0,6393
Recall	0,6842	0,7193	0,6842
F1	0,6964	0,7321	0,6610
Loss	0,8851		
Acc	69,591		

Gausian filter(sigma = 0,6-0,01) 90%			
	AD pred	DLB pred	NC pred
AD true	38	12	7
DLB true	12	36	9
NC true	18	11	28
Precision	0,5588	0,610169	0,6364
Recall	0,6667	0,631579	0,4912
F1	0,608	0,62069	0,5545
Loss	0,9117		
Acc	59,649		

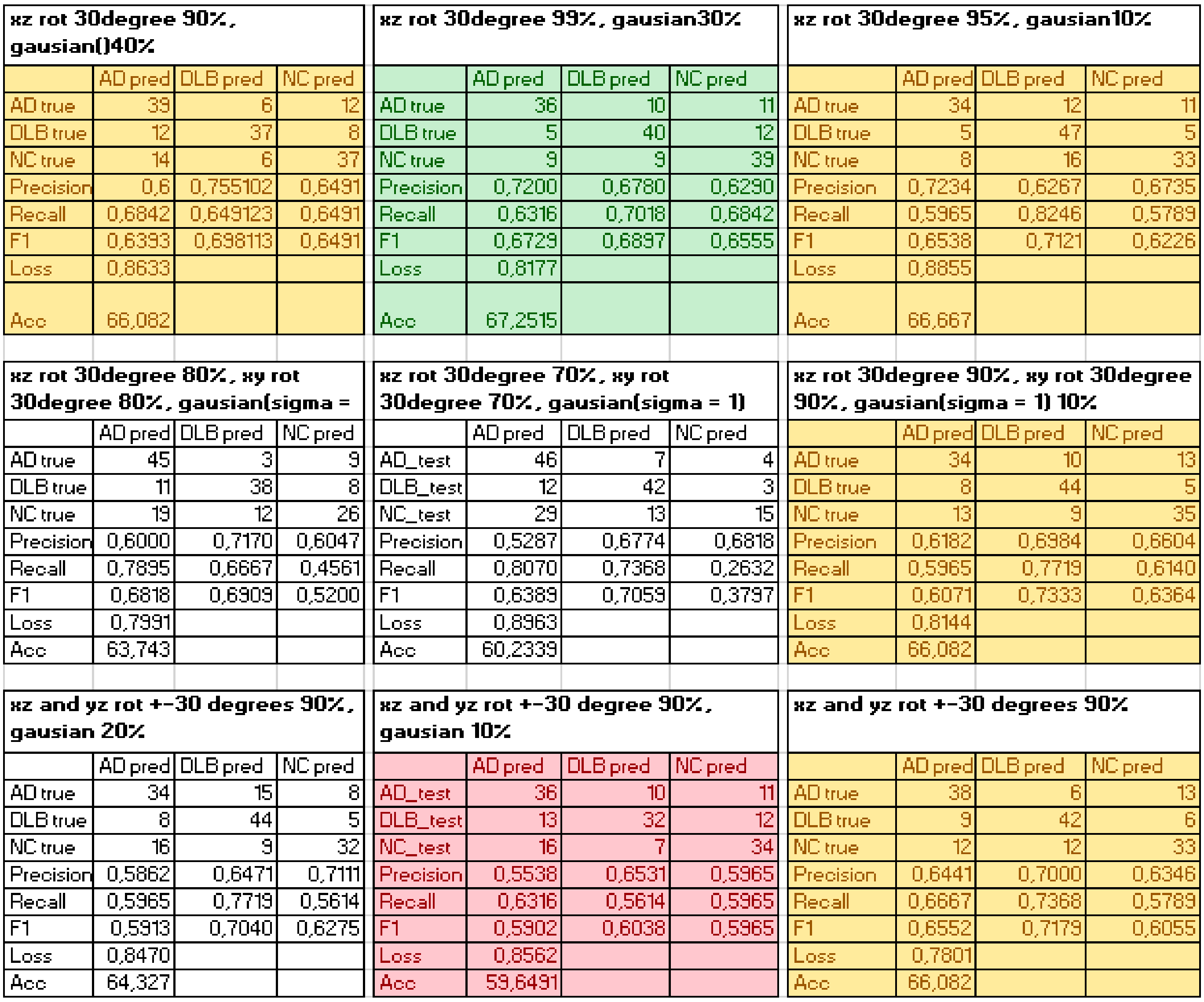
Gausian filter(sigma = 1-0,01) 90%			
	AD pred	DLB pred	NC pred
AD true	35	8	14
DLB true	12	30	15
NC true	12	10	35
Precision	0,5932	0,6250	0,5469
Recall	0,6140	0,5263	0,6140
F1	0,6034	0,5714	0,5785
Loss	0,9729		
Acc	58,4795		

Gausian filter(sigma = 1-0,01) 99%			
	AD pred	DLB pred	NC pred
AD true	44	4	9
DLB true	12	34	11
NC true	20	8	29
Precision	0,5789	0,7391	0,5918
Recall	0,7719	0,5965	0,5088
F1	0,6617	0,6602	0,5472
Loss	1,1133		
Acc	62,573		

Gausian filter(sigma = 2-0,01)			
	AD pred	DLB pred	NC pred
AD true	27	2	28
DLB true	16	21	20
NC true	10	1	46
Precision	0,5094	0,8750	0,4894
Recall	0,4737	0,3684	0,8070
F1	0,4909	0,5185	0,6093
Loss	1,0965		
Acc	54,971		

Gaussian filter 50% (sigma = 1)			
	AD pred	DLB pred	NC pred
AD true	23	7	27
DLB true	8	28	21
NC true	5	6	46
Precision	0,63889	0,682927	0,489362
Recall	0,40351	0,491228	0,807018
F1	0,49462	0,571429	0,609272
Loss	1,25137		
Acc	56,7251		

Table B.11: Augmentation Results from Gaussian Blur



xz rot 30degree 90%, gausian()40%

	AD pred	DLB pred	NC pred
AD true	39	6	12
DLB true	12	37	8
NC true	14	6	37
Precision	0,6	0,755102	0,6491
Recall	0,6842	0,649123	0,6491
F1	0,6393	0,698113	0,6491
Loss	0,8633		
Acc	66,082		

xz rot 30degree 99%, gausian30%

	AD pred	DLB pred	NC pred
AD true	36	10	11
DLB true	5	40	12
NC true	9	9	39
Precision	0,7200	0,6780	0,6290
Recall	0,6316	0,7018	0,6842
F1	0,6729	0,6897	0,6555
Loss	0,8177		
Acc	67,2515		

xz rot 30degree 95%, gausian10%

	AD pred	DLB pred	NC pred
AD true	34	12	11
DLB true	5	47	5
NC true	8	16	33
Precision	0,7234	0,6267	0,6735
Recall	0,5965	0,8246	0,5789
F1	0,6538	0,7121	0,6226
Loss	0,8855		
Acc	66,667		

xz rot 30degree 80%, xy rot 30degree 80%, gausian(sigma =

	AD pred	DLB pred	NC pred
AD true	45	3	9
DLB true	11	38	8
NC true	19	12	26
Precision	0,6000	0,7170	0,6047
Recall	0,7895	0,6667	0,4561
F1	0,6818	0,6909	0,5200
Loss	0,7991		
Acc	63,743		

xz rot 30degree 70%, xy rot 30degree 70%, gausian(sigma = 1)

	AD pred	DLB pred	NC pred
AD_test	46	7	4
DLB_test	12	42	3
NC_test	29	13	15
Precision	0,5287	0,6774	0,6818
Recall	0,8070	0,7368	0,2632
F1	0,6389	0,7059	0,3797
Loss	0,8963		
Acc	60,2339		

xz rot 30degree 90%, xy rot 30degree 90%, gausian(sigma = 1) 10%

	AD pred	DLB pred	NC pred
AD true	34	10	13
DLB true	8	44	5
NC true	13	9	35
Precision	0,6182	0,6984	0,6604
Recall	0,5965	0,7719	0,6140
F1	0,6071	0,7333	0,6364
Loss	0,8144		
Acc	66,082		

xz and yz rot +-30 degrees 90%, gausian 20%

	AD pred	DLB pred	NC pred
AD true	34	15	8
DLB true	8	44	5
NC true	16	9	32
Precision	0,5862	0,6471	0,7111
Recall	0,5965	0,7719	0,5614
F1	0,5913	0,7040	0,6275
Loss	0,8470		
Acc	64,327		

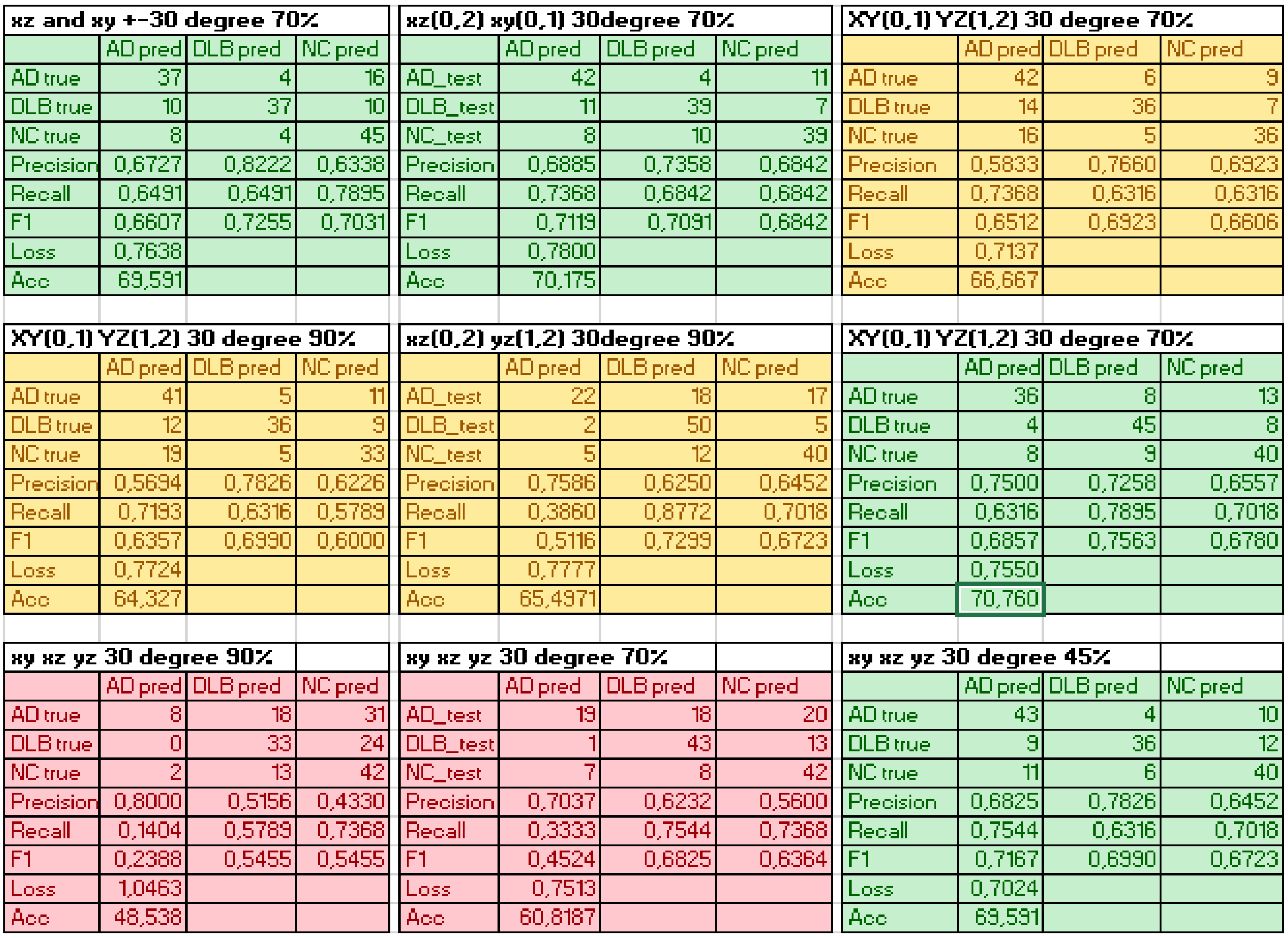
xz and yz rot +-30 degree 90%, gausian 10%

	AD pred	DLB pred	NC pred
AD_test	36	10	11
DLB_test	13	32	12
NC_test	16	7	34
Precision	0,5538	0,6531	0,5965
Recall	0,6316	0,5614	0,5965
F1	0,5902	0,6038	0,5965
Loss	0,8562		
Acc	59,6491		

xz and yz rot +-30 degrees 90%

	AD pred	DLB pred	NC pred
AD true	38	6	13
DLB true	9	42	6
NC true	12	12	33
Precision	0,6441	0,7000	0,6346
Recall	0,6667	0,7368	0,5789
F1	0,6552	0,7179	0,6055
Loss	0,7801		
Acc	66,082		

Table B.12: Augmentation Results from different combinations 1



xz and xy +-30 degree 70%			
	AD pred	DLB pred	NC pred
AD true	37	4	16
DLB true	10	37	10
NC true	8	4	45
Precision	0,6727	0,8222	0,6338
Recall	0,6491	0,6491	0,7895
F1	0,6607	0,7255	0,7031
Loss	0,7638		
Acc	69,591		

xz(0,2) xy(0,1) 30degree 70%			
	AD pred	DLB pred	NC pred
AD_test	42	4	11
DLB_test	11	39	7
NC_test	8	10	39
Precision	0,6885	0,7358	0,6842
Recall	0,7368	0,6842	0,6842
F1	0,7119	0,7091	0,6842
Loss	0,7800		
Acc	70,175		

XY(0,1) YZ(1,2) 30 degree 70%			
	AD pred	DLB pred	NC pred
AD true	42	6	9
DLB true	14	36	7
NC true	16	5	36
Precision	0,5833	0,7660	0,6923
Recall	0,7368	0,6316	0,6316
F1	0,6512	0,6923	0,6606
Loss	0,7137		
Acc	66,667		

XY(0,1) YZ(1,2) 30 degree 90%			
	AD pred	DLB pred	NC pred
AD true	41	5	11
DLB true	12	36	9
NC true	19	5	33
Precision	0,5694	0,7826	0,6226
Recall	0,7193	0,6316	0,5789
F1	0,6357	0,6990	0,6000
Loss	0,7724		
Acc	64,327		

xz(0,2) yz(1,2) 30degree 90%			
	AD pred	DLB pred	NC pred
AD_test	22	18	17
DLB_test	2	50	5
NC_test	5	12	40
Precision	0,7586	0,6250	0,6452
Recall	0,3860	0,8772	0,7018
F1	0,5116	0,7299	0,6723
Loss	0,7777		
Acc	65,4971		

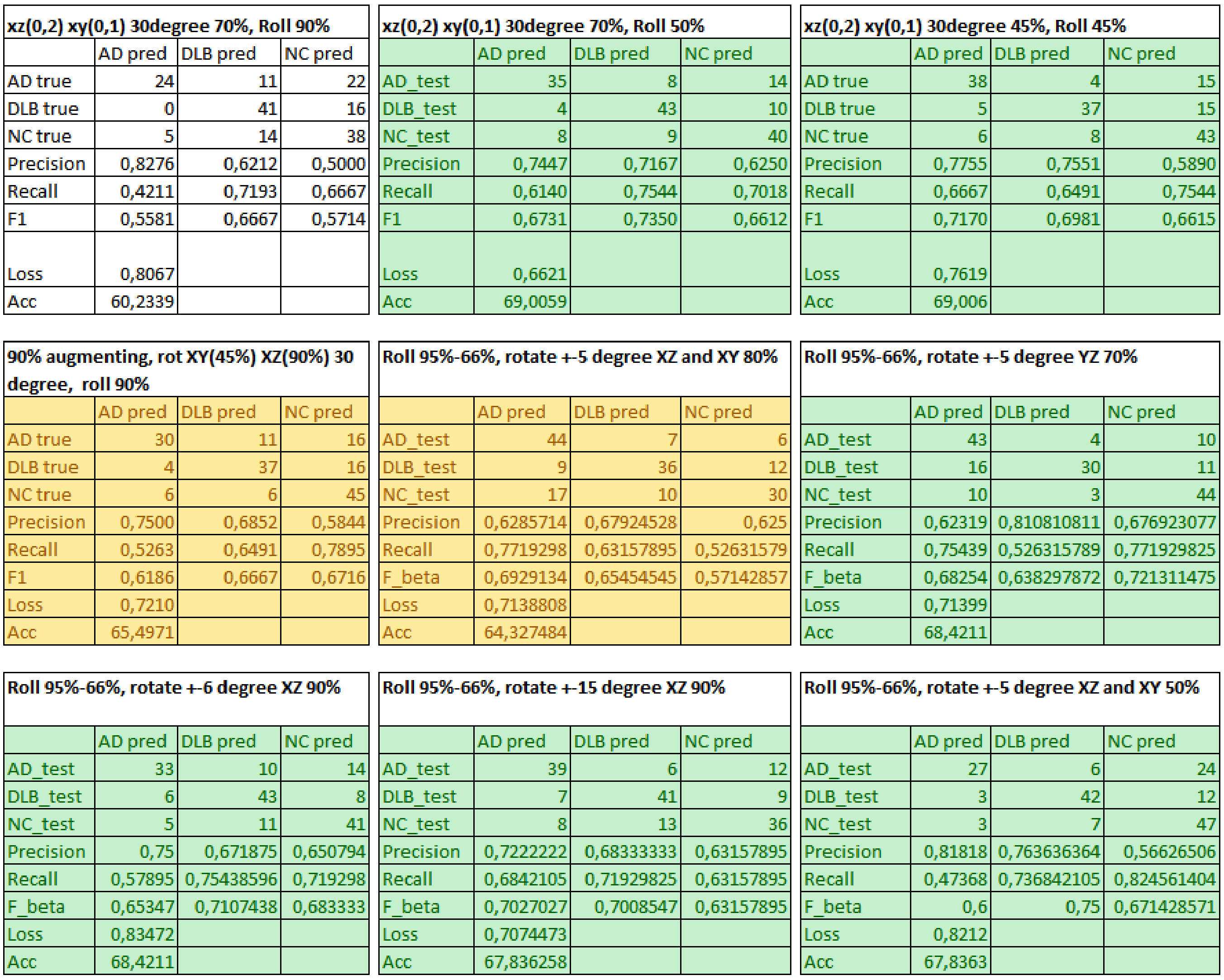
XY(0,1) YZ(1,2) 30 degree 70%			
	AD pred	DLB pred	NC pred
AD true	36	8	13
DLB true	4	45	8
NC true	8	9	40
Precision	0,7500	0,7258	0,6557
Recall	0,6316	0,7895	0,7018
F1	0,6857	0,7563	0,6780
Loss	0,7550		
Acc	70,760		

xy xz yz 30 degree 90%			
	AD pred	DLB pred	NC pred
AD true	8	18	31
DLB true	0	33	24
NC true	2	13	42
Precision	0,8000	0,5156	0,4330
Recall	0,1404	0,5789	0,7368
F1	0,2388	0,5455	0,5455
Loss	1,0463		
Acc	48,538		

xy xz yz 30 degree 70%			
	AD pred	DLB pred	NC pred
AD_test	19	18	20
DLB_test	1	43	13
NC_test	7	8	42
Precision	0,7037	0,6232	0,5600
Recall	0,3333	0,7544	0,7368
F1	0,4524	0,6825	0,6364
Loss	0,7513		
Acc	60,8187		

xy xz yz 30 degree 45%			
	AD pred	DLB pred	NC pred
AD true	43	4	10
DLB true	9	36	12
NC true	11	6	40
Precision	0,6825	0,7826	0,6452
Recall	0,7544	0,6316	0,7018
F1	0,7167	0,6990	0,6723
Loss	0,7024		
Acc	69,591		

Table B.13: Augmentation Results from different combinations 2



xz(0,2) xy(0,1) 30degree 70%, Roll 90%	AD pred	DLB pred	NC pred
AD true	24	11	22
DLB true	0	41	16
NC true	5	14	38
Precision	0,8276	0,6212	0,5000
Recall	0,4211	0,7193	0,6667
F1	0,5581	0,6667	0,5714
Loss	0,8067		
Acc	60,2339		

xz(0,2) xy(0,1) 30degree 70%, Roll 50%	AD pred	DLB pred	NC pred
AD_test	35	8	14
DLB_test	4	43	10
NC_test	8	9	40
Precision	0,7447	0,7167	0,6250
Recall	0,6140	0,7544	0,7018
F1	0,6731	0,7350	0,6612
Loss	0,6621		
Acc	69,0059		

xz(0,2) xy(0,1) 30degree 45%, Roll 45%	AD pred	DLB pred	NC pred
AD true	38	4	15
DLB true	5	37	15
NC true	6	8	43
Precision	0,7755	0,7551	0,5890
Recall	0,6667	0,6491	0,7544
F1	0,7170	0,6981	0,6615
Loss	0,7619		
Acc	69,006		

90% augmenting, rot XY(45%) XZ(90%) 30 degree, roll 90%	AD pred	DLB pred	NC pred
AD true	30	11	16
DLB true	4	37	16
NC true	6	6	45
Precision	0,7500	0,6852	0,5844
Recall	0,5263	0,6491	0,7895
F1	0,6186	0,6667	0,6716
Loss	0,7210		
Acc	65,4971		

Roll 95%-66%, rotate +-5 degree XZ and XY 80%	AD pred	DLB pred	NC pred
AD_test	44	7	6
DLB_test	9	36	12
NC_test	17	10	30
Precision	0,6285714	0,67924528	0,625
Recall	0,7719298	0,63157895	0,52631579
F_beta	0,6929134	0,65454545	0,57142857
Loss	0,7138808		
Acc	64,327484		

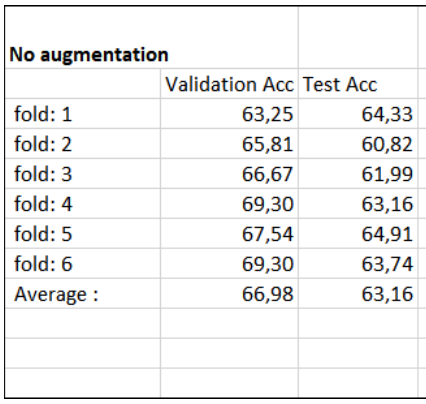
Roll 95%-66%, rotate +-5 degree YZ 70%	AD pred	DLB pred	NC pred
AD_test	43	4	10
DLB_test	16	30	11
NC_test	10	3	44
Precision	0,62319	0,810810811	0,676923077
Recall	0,75439	0,526315789	0,771929825
F_beta	0,68254	0,638297872	0,721311475
Loss	0,71399		
Acc	68,4211		

Roll 95%-66%, rotate +-6 degree XZ 90%	AD pred	DLB pred	NC pred
AD_test	33	10	14
DLB_test	6	43	8
NC_test	5	11	41
Precision	0,75	0,671875	0,650794
Recall	0,57895	0,75438596	0,719298
F_beta	0,65347	0,7107438	0,683333
Loss	0,83472		
Acc	68,4211		

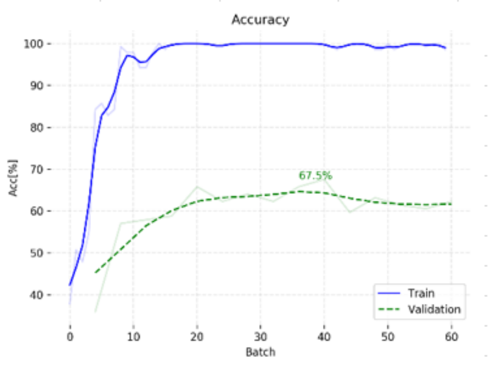
Roll 95%-66%, rotate +-15 degree XZ 90%	AD pred	DLB pred	NC pred
AD_test	39	6	12
DLB_test	7	41	9
NC_test	8	13	36
Precision	0,7222222	0,68333333	0,63157895
Recall	0,6842105	0,71929825	0,63157895
F_beta	0,7027027	0,7008547	0,63157895
Loss	0,7074473		
Acc	67,836258		

Roll 95%-66%, rotate +-5 degree XZ and XY 50%	AD pred	DLB pred	NC pred
AD_test	27	6	24
DLB_test	3	42	12
NC_test	3	7	47
Precision	0,81818	0,763636364	0,56626506
Recall	0,47368	0,736842105	0,824561404
F_beta	0,6	0,75	0,671428571
Loss	0,8212		
Acc	67,8363		

Table B.14: Augmentation Results from different combinations 3



No augmentation	Validation Acc	Test Acc
fold: 1	63,25	64,33
fold: 2	65,81	60,82
fold: 3	66,67	61,99
fold: 4	69,30	63,16
fold: 5	67,54	64,91
fold: 6	69,30	63,74
Average :	66,98	63,16



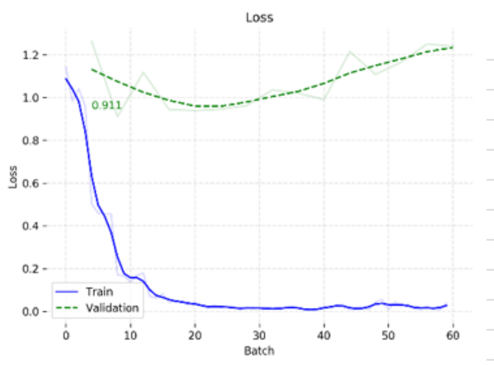
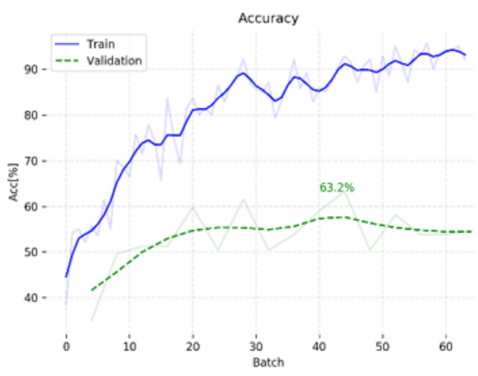
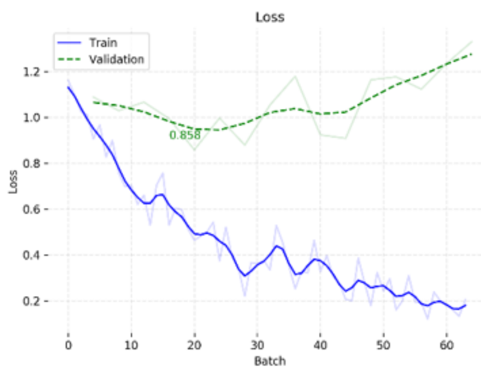


Table B.15: 6-CV Augmentation Results with no Augmentation

Rotate +- 15,0 degree in XZ plane 90% chance

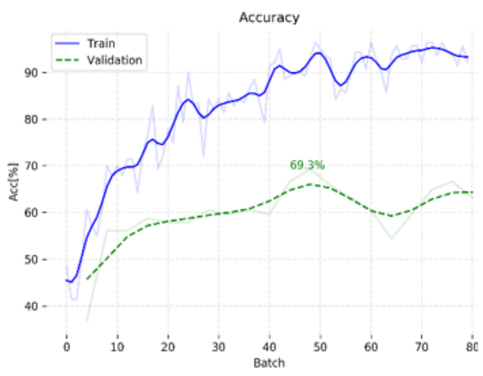
	Validation Acc	Test Acc
fold: 1	61,54	66,67
fold: 2	63,25	65,50
fold: 3	68,42	63,16
fold: 4	72,81	63,74
fold: 5	68,42	64,33
fold: 6	71,93	61,40
Average :	67,73	64,13

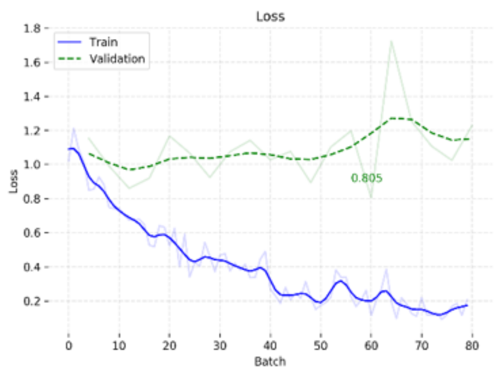




Rotate +- 15 degree in XZ plane 90% chance

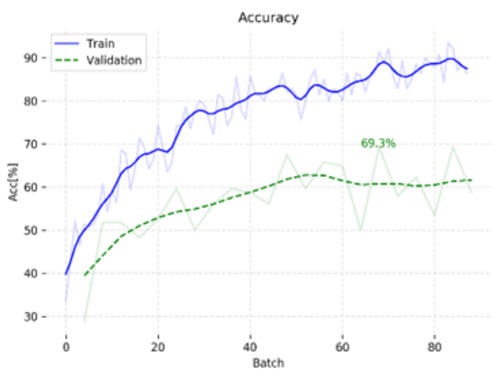
	Validation Acc	Test Acc
fold: 1	64,10	67,84
fold: 2	61,54	63,74
fold: 3	69,30	68,42
fold: 4	69,30	56,73
fold: 5	70,18	66,08
fold: 6	74,56	65,50
Average :	68,16	64,72

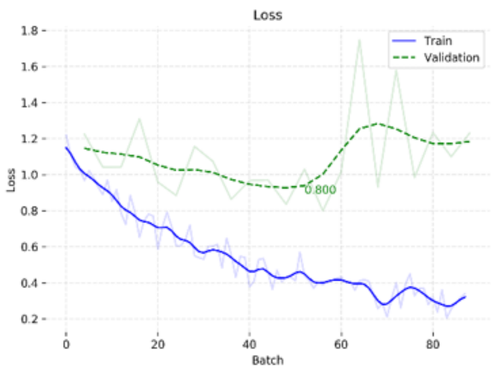




Rotate +- (15,0 to 30,0) degree in XZ plane 90% chance

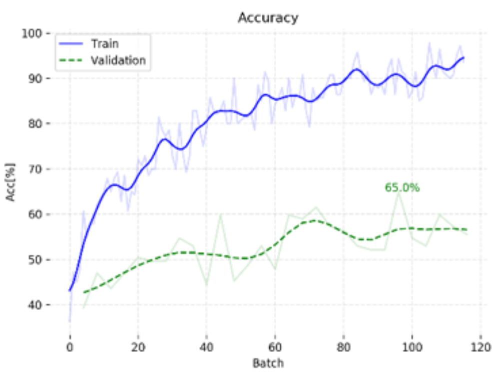
	Validation Acc	Test Acc
fold: 1	61,54	64,33
fold: 2	59,83	60,23
fold: 3	65,79	61,99
fold: 4	59,65	54,97
fold: 5	67,54	61,40
fold: 6	69,30	64,91
Average :	63,94	61,31





Rotate +- (15 to 30) degree in XZ plane 90% chance

	Validation Acc	Test Acc
fold: 1	59,83	62,57
fold: 2	64,96	66,08
fold: 3	64,04	47,37
fold: 4	66,67	64,33
fold: 5	66,67	61,40
fold: 6	64,91	56,73
Average :	64,51	59,75



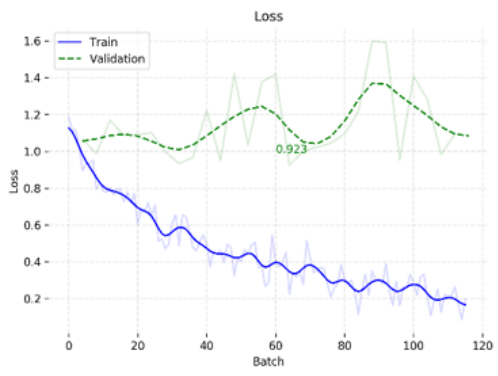
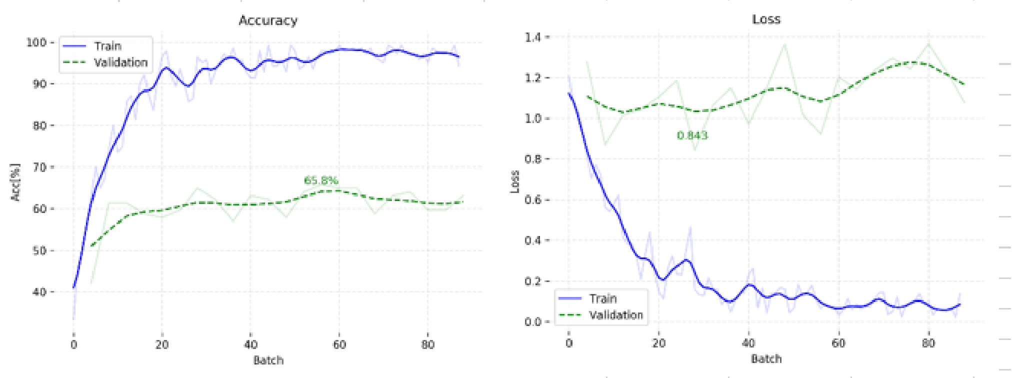


Table B.16: 6-CV Augmentation Results for Rotation in the XZ plane

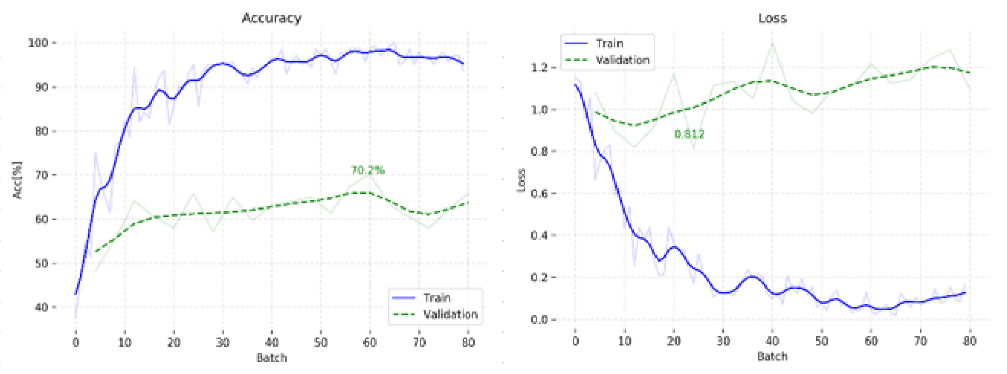
Rotate +- 6,0 degree in XZ plane 90% chance

	Validation Acc	Test Acc
fold: 1	58,97	66,08
fold: 2	64,10	60,23
fold: 3	64,04	58,48
fold: 4	67,54	63,16
fold: 5	65,79	64,33
fold: 6	70,18	60,82
Average :	65,10	62,18



Rotate +- 6 degree in XZ plane 90% chance

	Validation Acc	Test Acc
fold: 1	64,96	69,59
fold: 2	68,38	63,74
fold: 3	66,67	65,50
fold: 4	67,54	60,23
fold: 5	70,18	67,25
fold: 6	65,79	63,16
Average :	67,25	64,91



Rotate +- 45 degree in XZ plane 90% chance

	Validation Acc	Test Acc
fold: 1	61,54	66,67
fold: 2	59,83	64,33
fold: 3	68,42	65,50
fold: 4	68,42	69,01
fold: 5	66,67	60,82
fold: 6	66,67	69,59
Average :	65,26	65,98

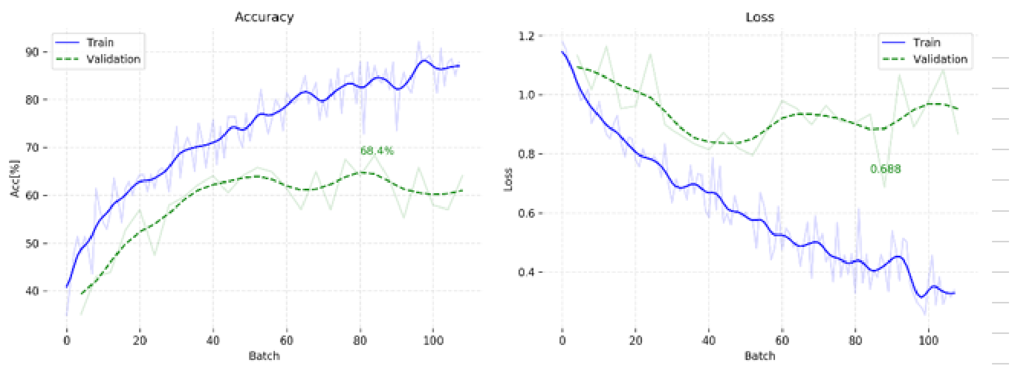
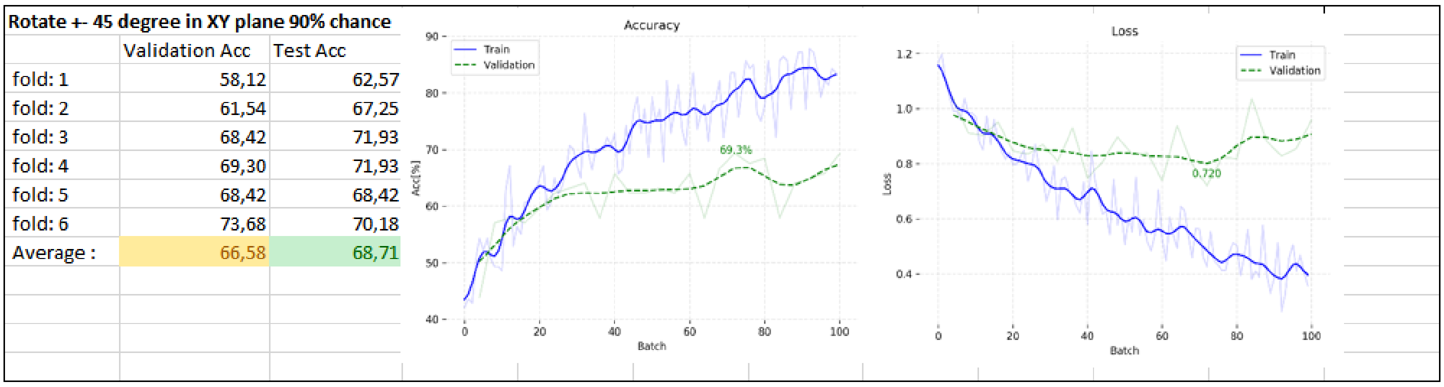
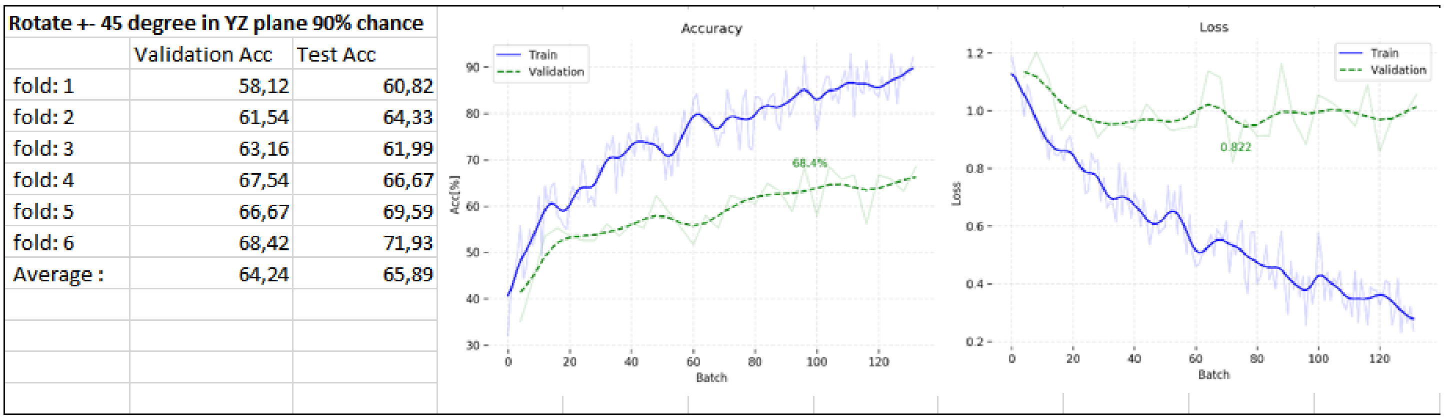


Table B.17: 6-CV Augmentation Results for Rotation in the XZ plane



Rotate +- 45 degree in XY plane 90% chance		
	Validation Acc	Test Acc
fold: 1	58,12	62,57
fold: 2	61,54	67,25
fold: 3	68,42	71,93
fold: 4	69,30	71,93
fold: 5	68,42	68,42
fold: 6	73,68	70,18
Average :	66,58	68,71

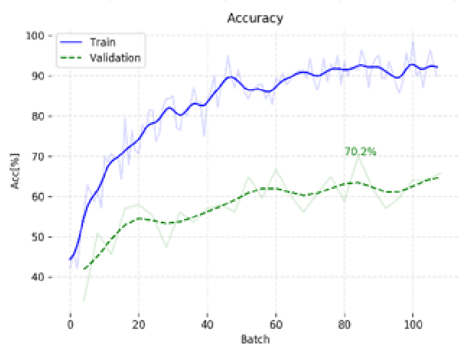
Table B.18: 6-CV Augmentation Results for Rotation in the XY plane

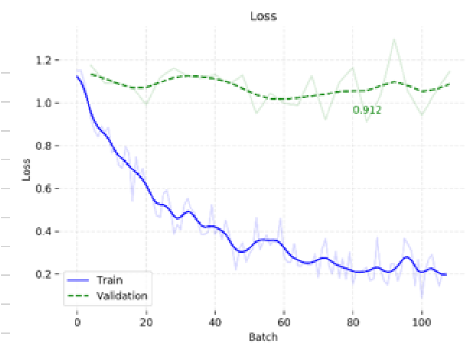


Rotate +- 45 degree in YZ plane 90% chance		
	Validation Acc	Test Acc
fold: 1	58,12	60,82
fold: 2	61,54	64,33
fold: 3	63,16	61,99
fold: 4	67,54	66,67
fold: 5	66,67	69,59
fold: 6	68,42	71,93
Average :	64,24	65,89

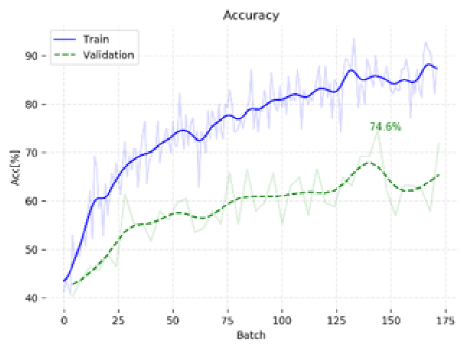
Table B.19: 6-CV Augmentation Results for Rotation in the YZ plane

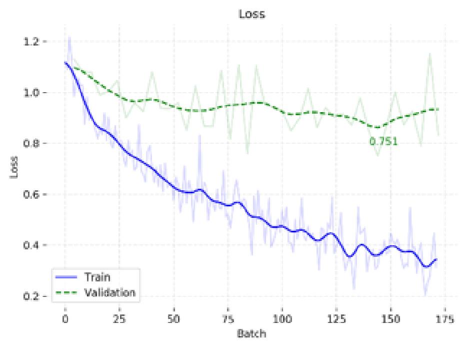
Roll 1-4 pixels in each direction, 95% chance of translation and 33% change for each direction		
	Validation Acc	Test Acc
fold: 1	63,25	70,18
fold: 2	63,25	63,16
fold: 3	70,18	61,40
fold: 4	65,79	66,08
fold: 5	66,67	63,16
fold: 6	70,18	69,01
Average :	66,55	65,50



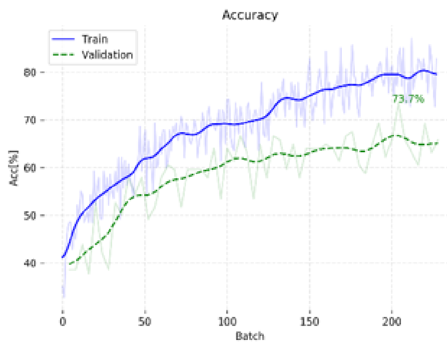


Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction		
	Validation Acc	Test Acc
fold: 1	67,52	59,06
fold: 2	67,52	67,84
fold: 3	69,30	66,67
fold: 4	68,42	70,18
fold: 5	74,56	71,93
fold: 6	67,54	69,01
Average :	69,14	67,45





Roll 1-8 pixels in left/right and back/forth , and Roll 1-4 pixels in up/down direction, 95% chance of translation and 66% change for each direction		
	Validation Acc	Test Acc
fold: 1	58,12	65,50
fold: 2	64,96	66,08
fold: 3	65,79	65,50
fold: 4	70,18	71,93
fold: 5	61,40	57,89
fold: 6	73,68	71,35
Average :	65,69	66,37



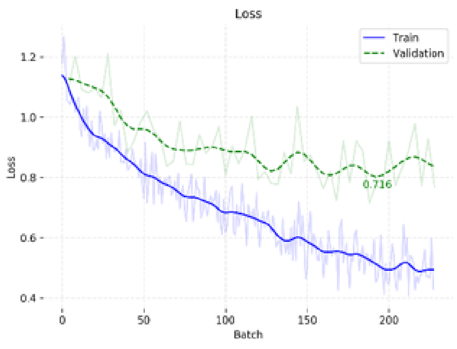
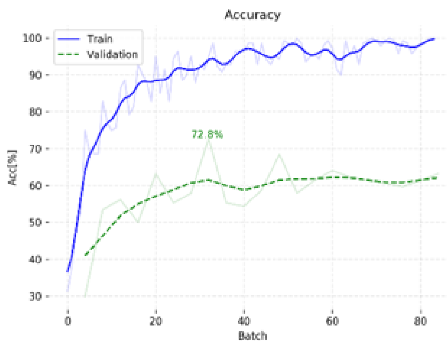


Table B.20: 6-CV Augmentation Results for Translations

Mirroring 50%		
	Validation Acc	Test Acc
fold: 1	61,54	64,91
fold: 2	60,68	63,16
fold: 3	70,18	66,08
fold: 4	61,40	65,50
fold: 5	71,05	68,42
fold: 6	72,81	68,42
Average :	66,28	66,08



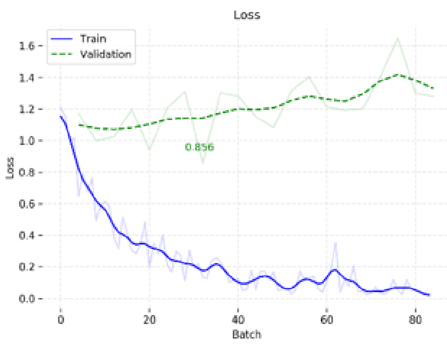


Table B.21: 6-CV Augmentation Results for Mirroring

Gaussian Blur (0,1-1,0) varying intensety 90% chance		
	Validation Acc	Test Acc
fold: 1	58,97	58,48
fold: 2	64,10	64,91
fold: 3	68,42	63,16
fold: 4	67,54	61,99
fold: 5	64,91	64,33
fold: 6	64,04	56,73
Average :	64,66	61,60

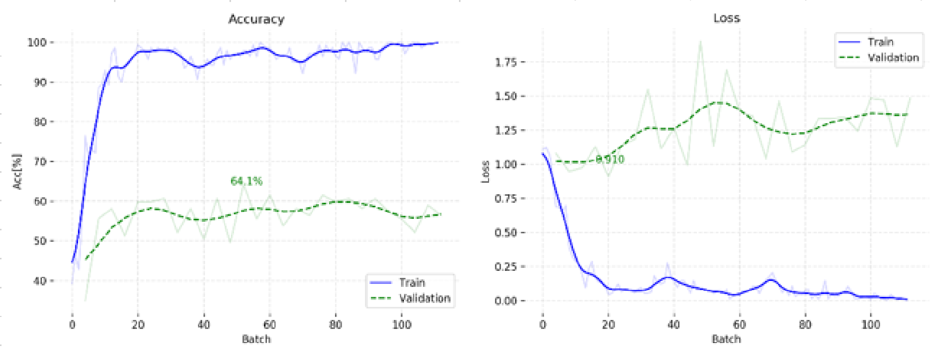
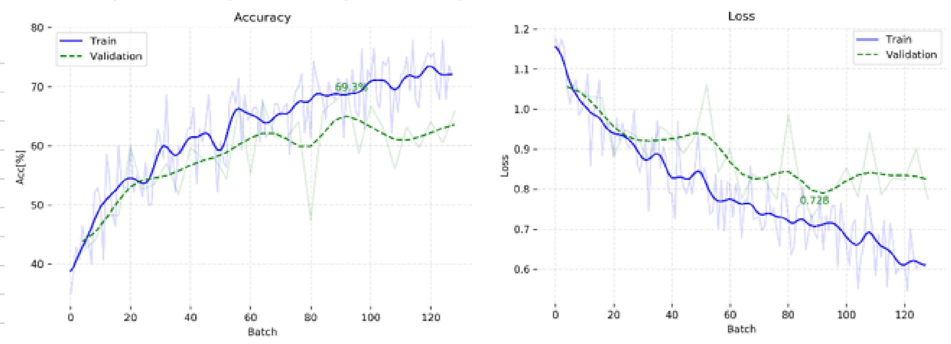
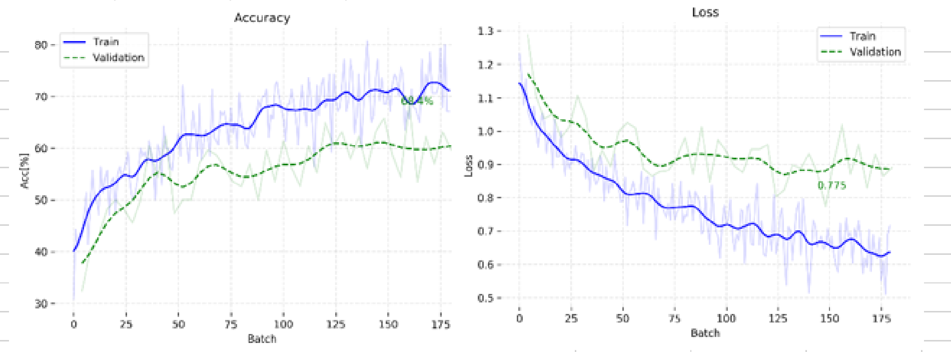


Table B.22: 6-CV Augmentation Results for Gaussian Blur

Rotate +- 45 degree in XZ and XY plane 70% chance each		
	Validation Acc	Test Acc
fold: 1	58,12	65,50
fold: 2	61,54	64,91
fold: 3	64,91	67,25
fold: 4	69,30	67,84
fold: 5	63,16	69,59
fold: 6	67,54	64,91
Average :	64,10	66,67



Rotate +- 45 degree in XZ , XY and YZ plane 50% chance each		
	Validation Acc	Test Acc
fold: 1	57,26	64,91
fold: 2	63,25	65,50
fold: 3	68,42	70,18
fold: 4	68,42	60,82
fold: 5	63,16	60,82
fold: 6	64,91	67,25
Average :	64,24	64,91



Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, and Rotate +-30 degree XZ and XY, 70% change each		
	Validation Acc	Test Acc
fold: 1	64,96	63,16
fold: 2	54,70	50,88
fold: 3	67,54	63,74
fold: 4	68,42	63,16
fold: 5	64,04	59,65
fold: 6	71,05	71,93
Average :	65,12	62,09

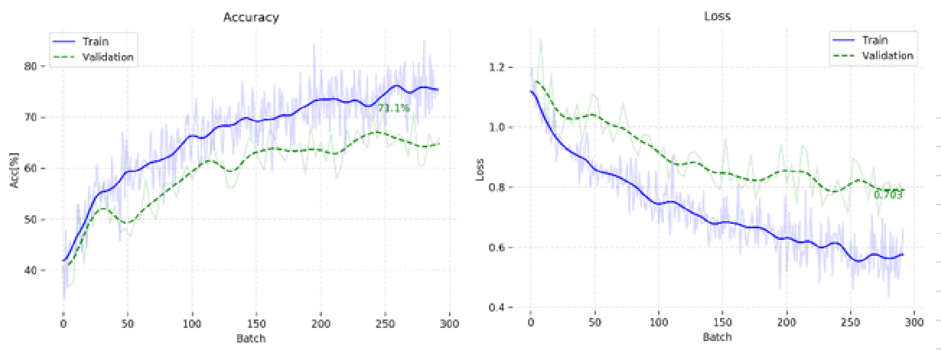
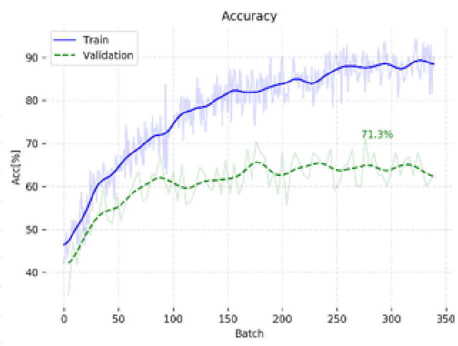
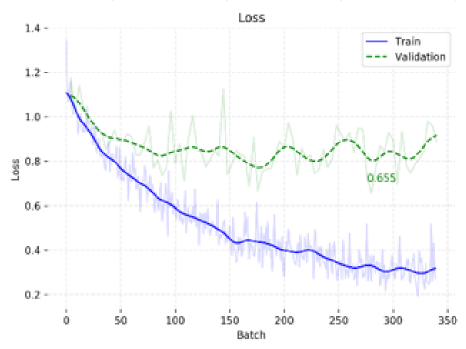


Table B.23: 6-CV Augmentation Results from Different Combinations

Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, Rotate +-6 degree XZ 80%

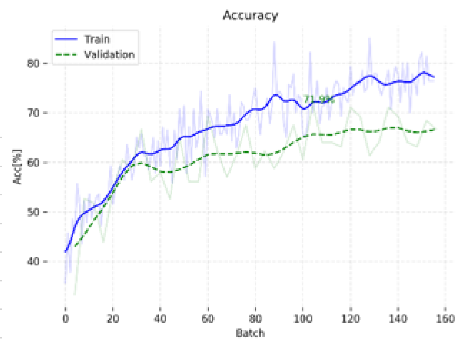
	Validation Acc	Test Acc
fold: 1	72,17	71,93
fold: 2	71,30	74,27
fold: 3	70,43	74,85
fold: 4	72,17	70,76
fold: 5	71,30	73,68
fold: 6	70,43	73,68
Average :	71,30	73,20

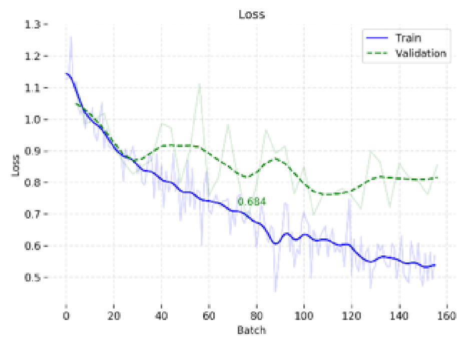




Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, Rotate +-6 degree XY 80%, Mirror 50%

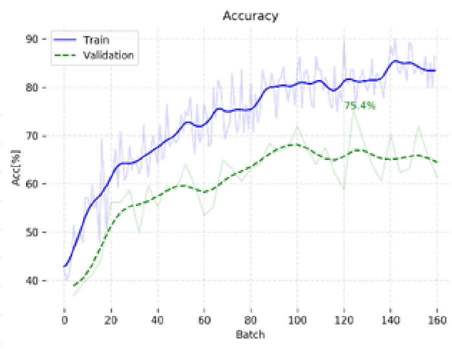
	Validation Acc	Test Acc
fold: 1	61,54	69,59
fold: 2	67,52	69,01
fold: 3	67,54	66,08
fold: 4	71,93	73,10
fold: 5	68,42	74,27
fold: 6	69,30	71,35
Average :	67,71	70,57





Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, Rotate +-6 degree XY 80%

	Validation Acc	Test Acc
fold: 1	65,81196594	66,66666412
fold: 2	66,66667175	67,25146484
fold: 3	70,17543793	70,17543793
fold: 4	73,68421173	68,42105103
fold: 5	71,05263519	65,49707794
fold: 6	75,43859863	70,17543793
Average :	70,47158686	68,03118896



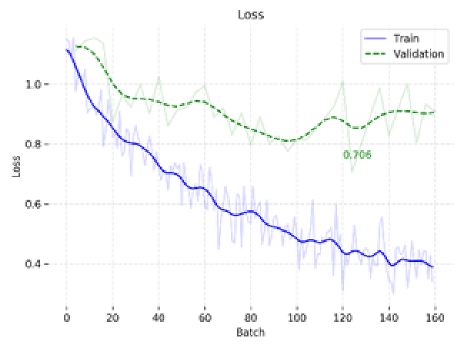
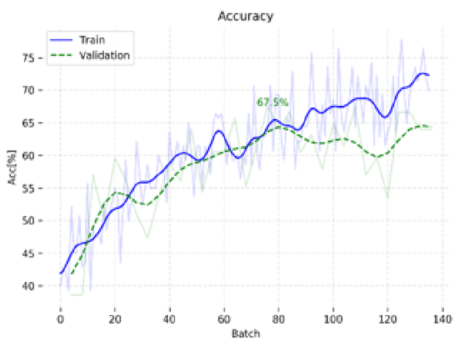
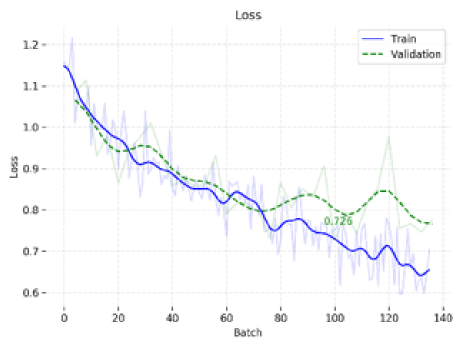


Table B.24: 6-CV Augmentation Results from Different Combinations

Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, Rotate +-45 degree XY 80%

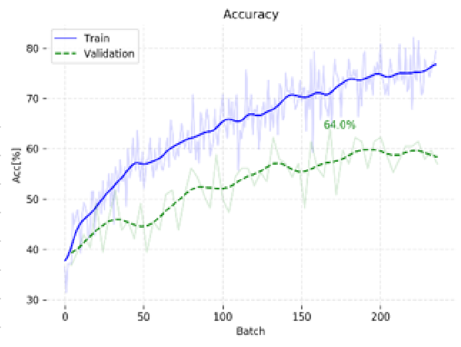
	Validation Acc	Test Acc
fold: 1	58,97436142	63,74269104
fold: 2	61,53846359	66,08187103
fold: 3	65,78947449	58,47953415
fold: 4	67,54386139	66,08187103
fold: 5	63,15789413	47,95321655
fold: 6	58,77193069	65,49707794
Average :	62,62933095	61,30604362





Testing with ResNet101, Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, Rotate +-45 degree XY 80%

	Validation Acc	Test Acc
fold: 1	56,41025925	49,12280655
fold: 2	57,26496124	56,14035034
fold: 3	64,91228485	49,70760345
fold: 4	71,05263519	59,06432724
fold: 5	58,77193069	42,10526276
fold: 6	64,03508759	61,98830414
Average :	62,07452647	53,02144241



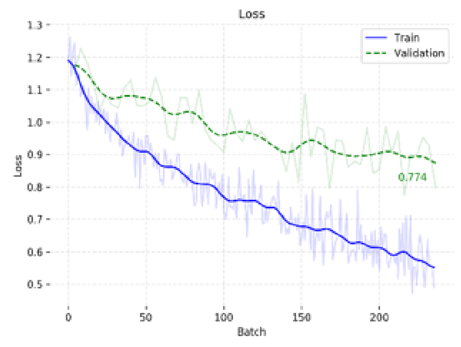
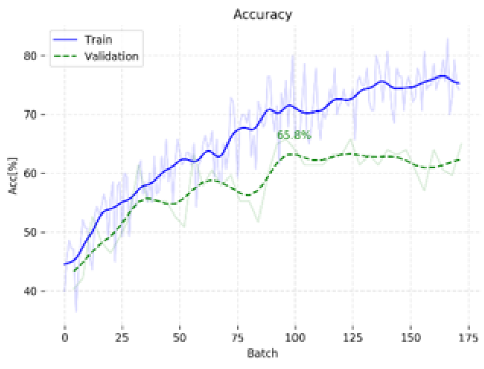
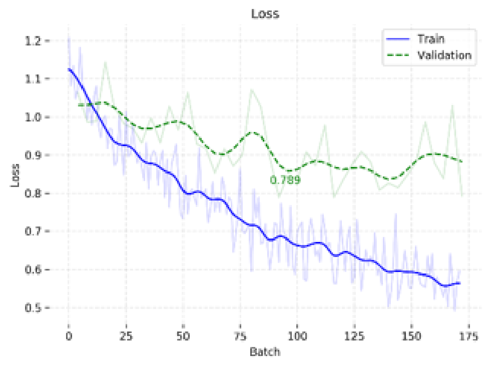


Table B.25: 6-CV Augmentation Results from Different Combinations

Exclusive Augmentation: Roll 1-4 pixels in each direction 60% chance of translation and 66% change for each direction, Rotate +-6 degree XY,XZ,YZ 10% each, Mirror 10%

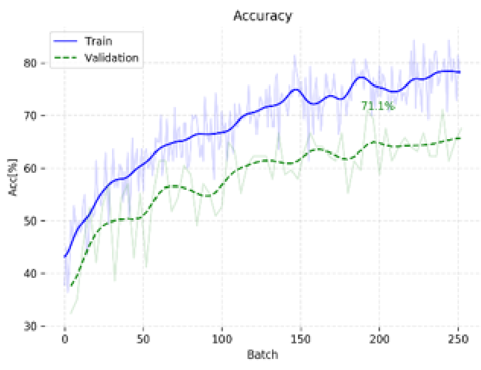
	Validation Acc	Test Acc
fold: 1	64,10256958	67,25146484
fold: 2	63,24786758	66,08187103
fold: 3	65,78947449	68,42105103
fold: 4	71,92982483	66,66666412
fold: 5	65,78947449	71,34503174
fold: 6	76,31578827	68,42105103
Average :	67,86249987	68,03118896





Exclusive Augmentation: Roll 1-4 pixels in each direction 60% chance of translation and 66% change for each direction, Rotate +-10 degree XY,XZ,YZ 10% each, Mirror 10%

	Validation Acc	Test Acc
fold: 1	57,26496124	60,81871414
fold: 2	62,39316559	63,74269104
fold: 3	65,78947449	56,72514725
fold: 4	71,05263519	67,25146484
fold: 5	64,03508759	67,83625793
fold: 6	71,05263519	71,92982483
Average :	65,26465988	64,71735001



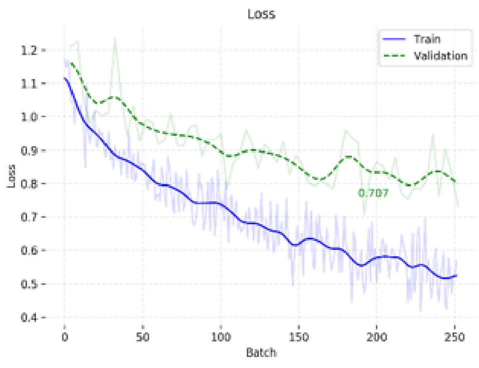
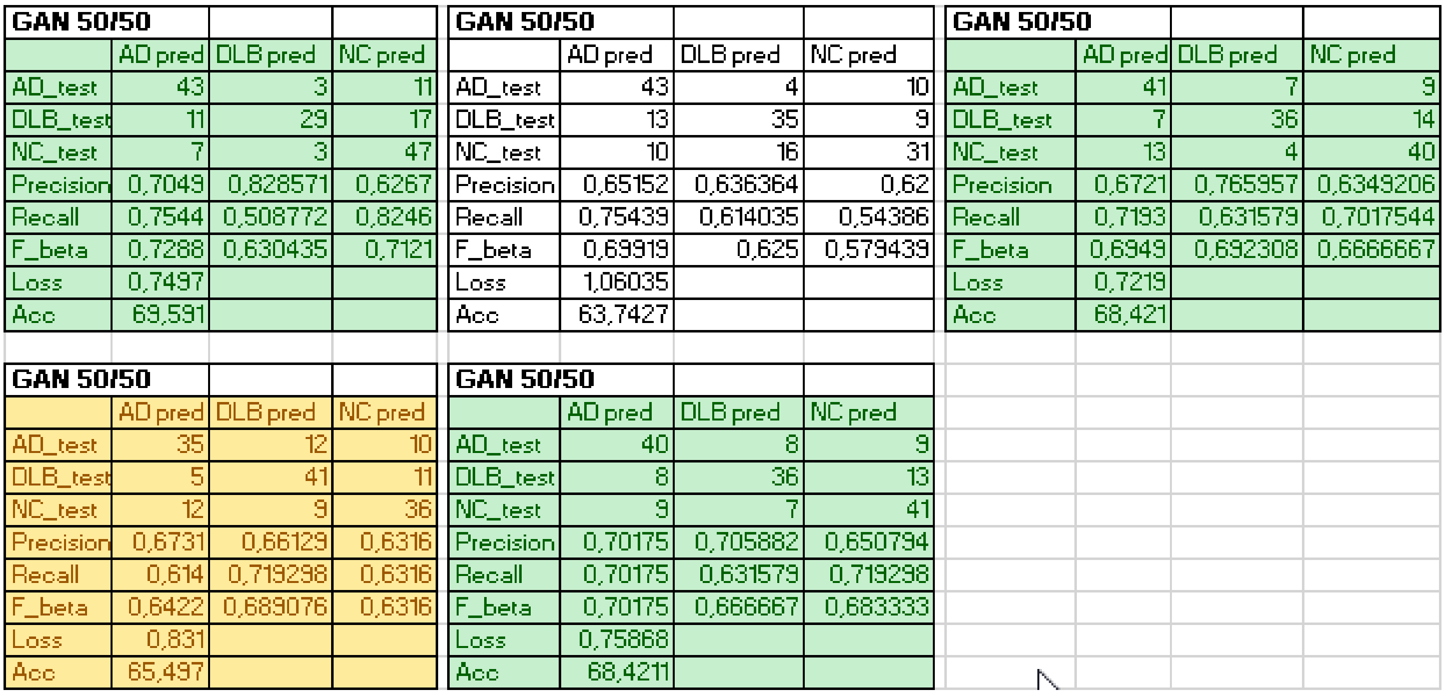


Table B.26: 6-CV Augmentation Results from Different Combinations



GAN 50/50			
	AD pred	DLB pred	NC pred
AD_test	43	3	11
DLB_test	11	29	17
NC_test	7	3	47
Precision	0,7049	0,828571	0,6267
Recall	0,7544	0,508772	0,8246
F_beta	0,7288	0,630435	0,7121
Loss	0,7497		
Acc	69,591		

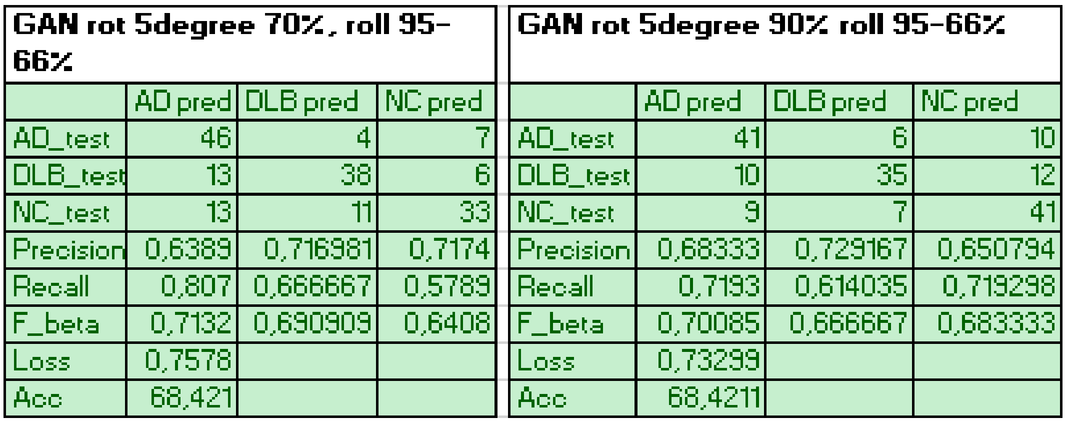
GAN 50/50			
	AD pred	DLB pred	NC pred
AD_test	43	4	10
DLB_test	13	35	9
NC_test	10	16	31
Precision	0,65152	0,636364	0,62
Recall	0,75439	0,614035	0,54386
F_beta	0,69919	0,625	0,579439
Loss	1,06035		
Acc	63,7427		

GAN 50/50			
	AD pred	DLB pred	NC pred
AD_test	41	7	9
DLB_test	7	36	14
NC_test	13	4	40
Precision	0,6721	0,765957	0,6349206
Recall	0,7193	0,631579	0,7017544
F_beta	0,6949	0,692308	0,6666667
Loss	0,7219		
Acc	68,421		

GAN 50/50			
	AD pred	DLB pred	NC pred
AD_test	35	12	10
DLB_test	5	41	11
NC_test	12	9	36
Precision	0,6731	0,66129	0,6316
Recall	0,614	0,719298	0,6316
F_beta	0,6422	0,689076	0,6316
Loss	0,831		
Acc	65,497		

GAN 50/50			
	AD pred	DLB pred	NC pred
AD_test	40	8	9
DLB_test	8	36	13
NC_test	9	7	41
Precision	0,70175	0,705882	0,650794
Recall	0,70175	0,631579	0,719298
F_beta	0,70175	0,666667	0,683333
Loss	0,75868		
Acc	68,4211		

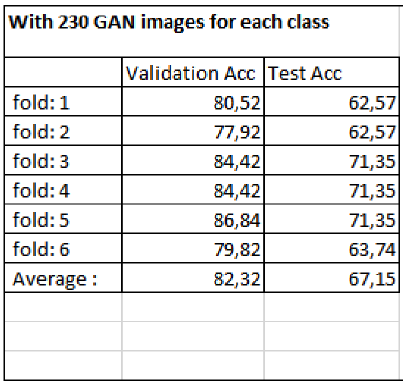
Table B.27: Augmentation Results with GAN



GAN rot 5degree 70%, roll 95-66%			
	AD pred	DLB pred	NC pred
AD_test	46	4	7
DLB_test	13	38	6
NC_test	13	11	33
Precision	0,6389	0,716981	0,7174
Recall	0,807	0,666667	0,5789
F_beta	0,7132	0,690909	0,6408
Loss	0,7578		
Acc	68,421		

GAN rot 5degree 90% roll 95-66%			
	AD pred	DLB pred	NC pred
AD_test	41	6	10
DLB_test	10	35	12
NC_test	9	7	41
Precision	0,68333	0,729167	0,650794
Recall	0,7193	0,614035	0,719298
F_beta	0,70085	0,666667	0,683333
Loss	0,73299		
Acc	68,4211		

Table B.28: Augmentation Results with GAN



With 230 GAN images for each class		
	Validation Acc	Test Acc
fold: 1	80,52	62,57
fold: 2	77,92	62,57
fold: 3	84,42	71,35
fold: 4	84,42	71,35
fold: 5	86,84	71,35
fold: 6	79,82	63,74
Average :	82,32	67,15

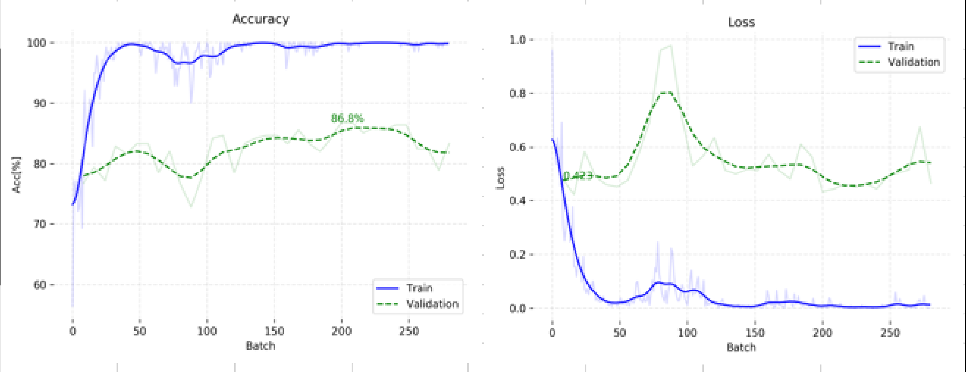
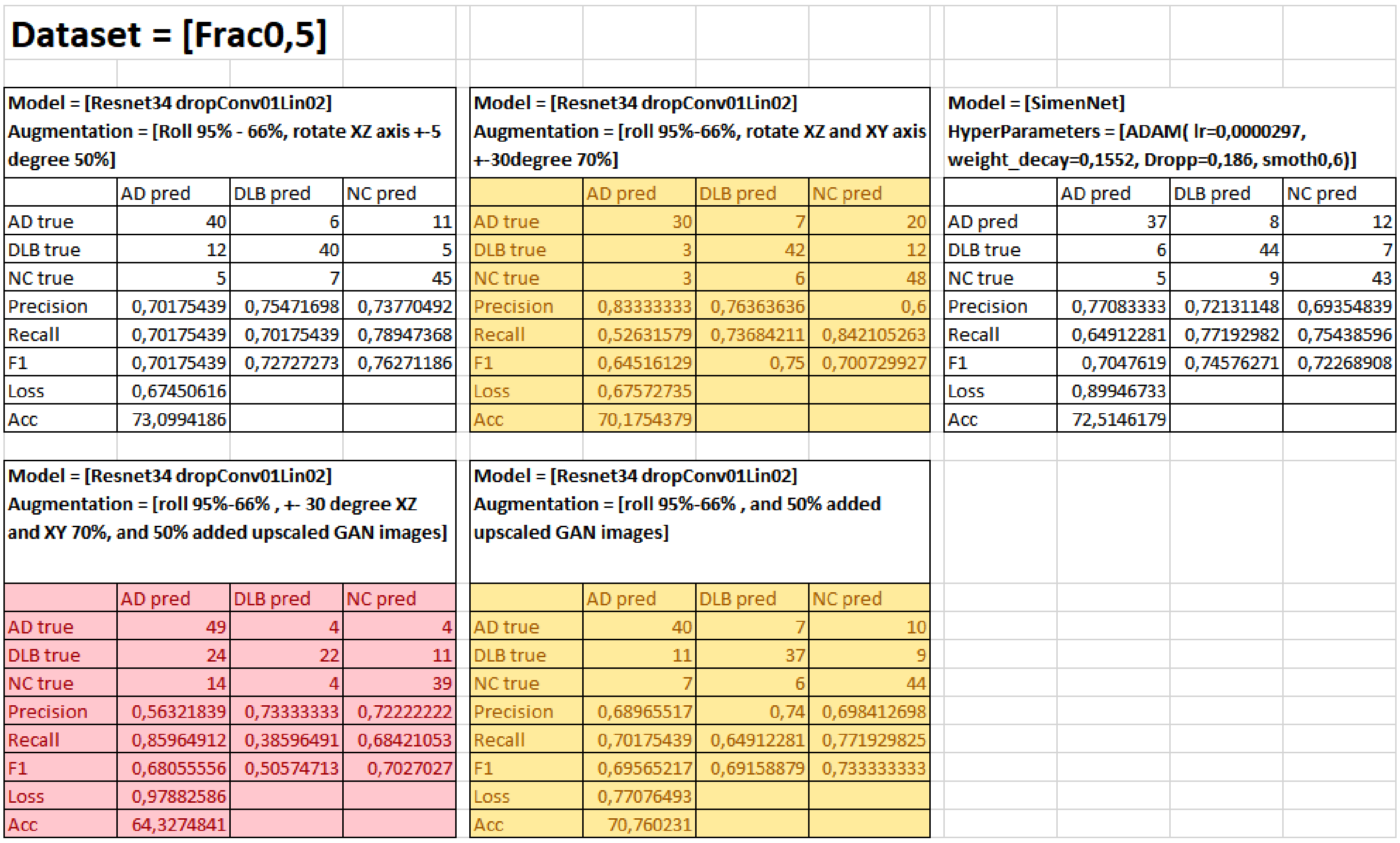


Table B.44: 6-CV Augmentation Results with 50%GAN Images



Dataset = [Frac0,5]

Model = [Resnet34 dropConv01Lin02]
Augmentation = [Roll 95% - 66%, rotate XZ axis +-5 degree 50%]

	AD pred	DLB pred	NC pred
AD true	40	6	11
DLB true	12	40	5
NC true	5	7	45
Precision	0,70175439	0,75471698	0,73770492
Recall	0,70175439	0,70175439	0,78947368
F1	0,70175439	0,72727273	0,76271186
Loss	0,67450616		
Acc	73,0994186		

Model = [Resnet34 dropConv01Lin02]
Augmentation = [roll 95%-66%, rotate XZ and XY axis +-30degree 70%]

	AD pred	DLB pred	NC pred
AD true	30	7	20
DLB true	3	42	12
NC true	3	6	48
Precision	0,83333333	0,76363636	0,6
Recall	0,52631579	0,73684211	0,842105263
F1	0,64516129	0,75	0,700729927
Loss	0,67572735		
Acc	70,1754379		

Model = [SimenNet]
HyperParameters = [ADAM(lr=0,0000297, weight_decay=0,1552, Dropp=0,186, smoth0,6)]

	AD pred	DLB pred	NC pred
AD pred	37	8	12
DLB true	6	44	7
NC true	5	9	43
Precision	0,77083333	0,72131148	0,69354839
Recall	0,64912281	0,77192982	0,75438596
F1	0,7047619	0,74576271	0,72268908
Loss	0,89946733		
Acc	72,5146179		

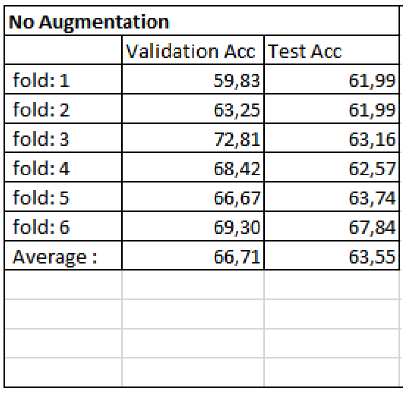
Model = [Resnet34 dropConv01Lin02]
Augmentation = [roll 95%-66% , +- 30 degree XZ and XY 70%, and 50% added upscaled GAN images]

	AD pred	DLB pred	NC pred
AD true	49	4	4
DLB true	24	22	11
NC true	14	4	39
Precision	0,56321839	0,73333333	0,72222222
Recall	0,85964912	0,38596491	0,68421053
F1	0,68055556	0,50574713	0,7027027
Loss	0,97882586		
Acc	64,3274841		

Model = [Resnet34 dropConv01Lin02]
Augmentation = [roll 95%-66% , and 50% added upscaled GAN images]

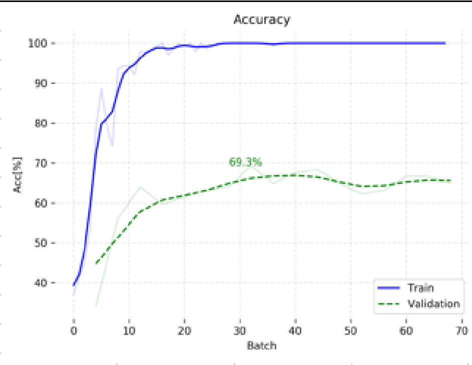
	AD pred	DLB pred	NC pred
AD true	40	7	10
DLB true	11	37	9
NC true	7	6	44
Precision	0,68965517	0,74	0,698412698
Recall	0,70175439	0,64912281	0,771929825
F1	0,69565217	0,69158879	0,733333333
Loss	0,77076493		
Acc	70,760231		

Table B.29: Frac0.5 Dataset with augmentations



No Augmentation

	Validation Acc	Test Acc
fold: 1	59,83	61,99
fold: 2	63,25	61,99
fold: 3	72,81	63,16
fold: 4	68,42	62,57
fold: 5	66,67	63,74
fold: 6	69,30	67,84
Average :	66,71	63,55



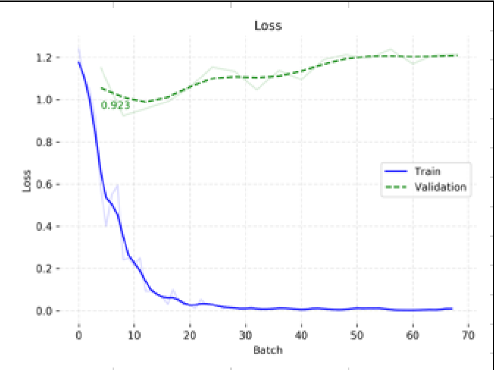
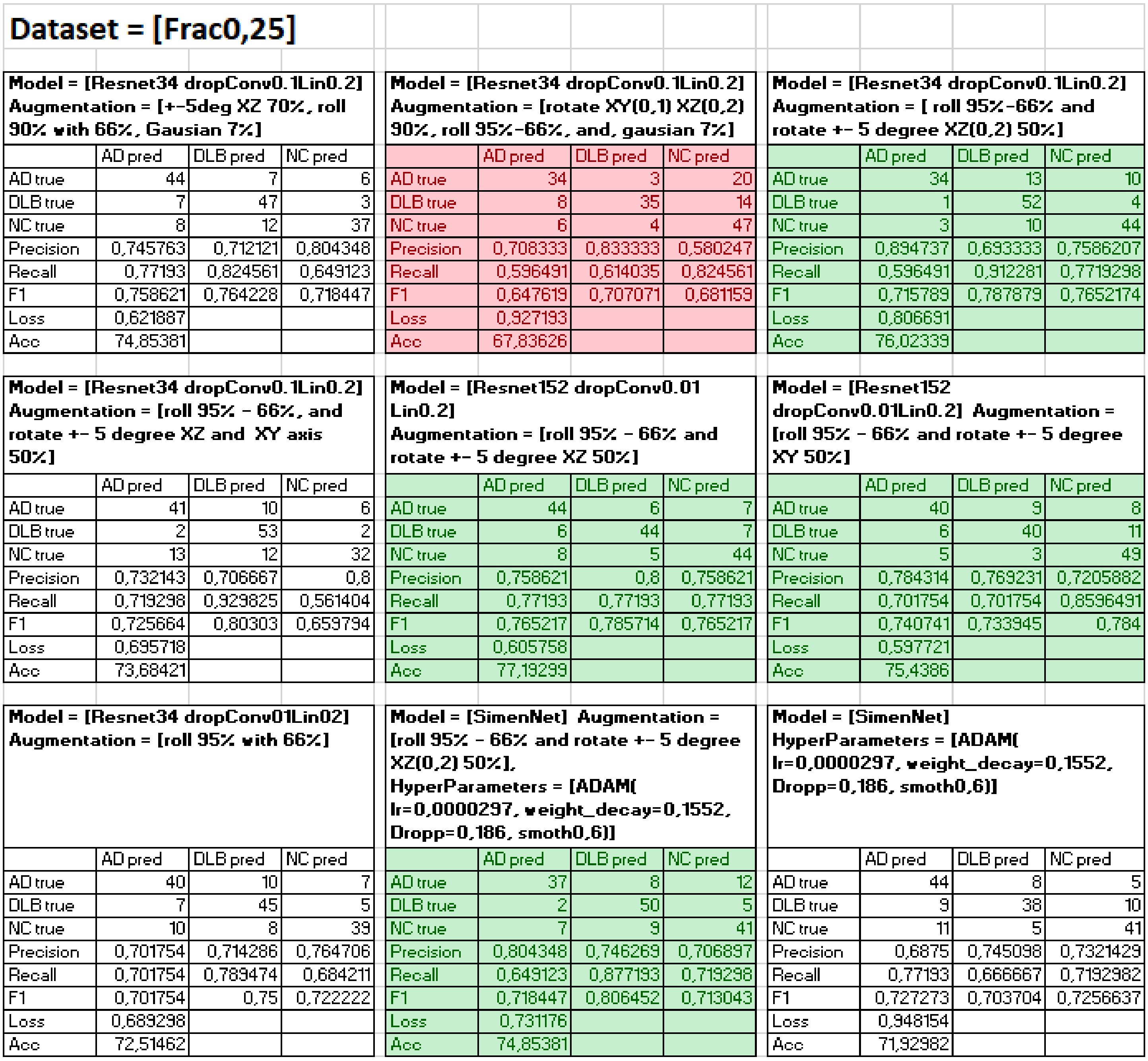


Table B.45: 6-CV Result Frac=0.5 dataset, No Augmentation



Dataset = [Frac0,25]

Model = [Resnet34 dropConv0.1Lin0.2] Augmentation = [+-5deg XZ 70%, roll 90% with 66%, Gausian 7%]

	AD pred	DLB pred	NC pred
AD true	44	7	6
DLB true	7	47	3
NC true	8	12	37
Precision	0,745763	0,712121	0,804348
Recall	0,77193	0,824561	0,649123
F1	0,758621	0,764228	0,718447
Loss	0,621887		
Acc	74,85381		

Model = [Resnet34 dropConv0.1Lin0.2] Augmentation = [rotate XY(0,1) XZ(0,2) 90%, roll 95%-66%, and, gausian 7%]

	AD pred	DLB pred	NC pred
AD true	34	3	20
DLB true	8	35	14
NC true	6	4	47
Precision	0,708333	0,833333	0,580247
Recall	0,596491	0,614035	0,824561
F1	0,647619	0,707071	0,681159
Loss	0,927193		
Acc	67,83626		

Model = [Resnet34 dropConv0.1Lin0.2] Augmentation = [roll 95%-66% and rotate +- 5 degree XZ(0,2) 50%]

	AD pred	DLB pred	NC pred
AD true	34	13	10
DLB true	1	52	4
NC true	3	10	44
Precision	0,894737	0,693333	0,7586207
Recall	0,596491	0,912281	0,7719298
F1	0,715789	0,787879	0,7652174
Loss	0,806691		
Acc	76,02339		

Model = [Resnet34 dropConv0.1Lin0.2] Augmentation = [roll 95% - 66%, and rotate +- 5 degree XZ and XY axis 50%]

	AD pred	DLB pred	NC pred
AD true	41	10	6
DLB true	2	53	2
NC true	13	12	32
Precision	0,732143	0,706667	0,8
Recall	0,719298	0,929825	0,561404
F1	0,725664	0,80303	0,659794
Loss	0,695718		
Acc	73,68421		

Model = [Resnet152 dropConv0.01 Lin0.2] Augmentation = [roll 95% - 66% and rotate +- 5 degree XZ 50%]

	AD pred	DLB pred	NC pred
AD true	44	6	7
DLB true	6	44	7
NC true	8	5	44
Precision	0,758621	0,8	0,758621
Recall	0,77193	0,77193	0,77193
F1	0,765217	0,785714	0,765217
Loss	0,605758		
Acc	77,19299		

Model = [Resnet152 dropConv0.01Lin0.2] Augmentation = [roll 95% - 66% and rotate +- 5 degree XY 50%]

	AD pred	DLB pred	NC pred
AD true	40	9	8
DLB true	6	40	11
NC true	5	3	49
Precision	0,784314	0,769231	0,7205882
Recall	0,701754	0,701754	0,8596491
F1	0,740741	0,733945	0,784
Loss	0,597721		
Acc	75,4386		

Model = [Resnet34 dropConv01Lin02] Augmentation = [roll 95% with 66%]

	AD pred	DLB pred	NC pred
AD true	40	10	7
DLB true	7	45	5
NC true	10	8	39
Precision	0,701754	0,714286	0,764706
Recall	0,701754	0,789474	0,684211
F1	0,701754	0,75	0,722222
Loss	0,689298		
Acc	72,51462		

Model = [SimenNet] Augmentation = [roll 95% - 66% and rotate +- 5 degree XZ(0,2) 50%], HyperParameters = [ADAM(lr=0,0000297, weight_decay=0,1552, Dropp=0,186, smoth0,6)]

	AD pred	DLB pred	NC pred
AD true	37	8	12
DLB true	2	50	5
NC true	7	9	41
Precision	0,804348	0,746269	0,706897
Recall	0,649123	0,877193	0,719298
F1	0,718447	0,806452	0,713043
Loss	0,731176		
Acc	74,85381		

Model = [SimenNet] HyperParameters = [ADAM(lr=0,0000297, weight_decay=0,1552, Dropp=0,186, smoth0,6)]

	AD pred	DLB pred	NC pred
AD true	44	8	5
DLB true	9	38	10
NC true	11	5	41
Precision	0,6875	0,745098	0,7321429
Recall	0,77193	0,666667	0,7192982
F1	0,727273	0,703704	0,7256637
Loss	0,948154		
Acc	71,92982		

Table B.30: Frac0.25 Dataset with augmentations

Model = [Resnet34 dropConv01Lin02] Augmentation = [roll 95%-66%, rotate XZ and XY axis +-30degree 70%]			
	AD pred	DLB pred	NC pred
AD true	41	9	7
DLB true	6	42	9
NC true	7	4	46
Precision	0,7592593	0,7636364	0,7419355
Recall	0,7192982	0,7368421	0,8070175
F1	0,7387387	0,75	0,7731092
Loss	0,6041034		
Acc	75,438599		

Model = [Resnet34 dropConv01Lin02] Augmentation = [roll 95%-66%]			
	AD pred	DLB pred	NC pred
AD true	42	8	7
DLB true	6	45	6
NC true	9	10	38
Precision	0,7368421	0,7142857	0,74509804
Recall	0,7368421	0,7894737	0,66666667
F1	0,7368421	0,75	0,7037037
Loss	0,6698055		
Acc	73,099419		

Table B.31: Frac0.4 Dataset with augmentations

Dataset = [Frac0,25 Eye Removal]

Model = [Resnet34 dropConv01Lin02]			
	AD pred	DLB pred	NC pred
AD true	41	6	10
DLB true	8	34	15
NC true	8	4	45
Precision	0,71929825	0,77272727	0,64285714
Recall	0,71929825	0,59649123	0,78947368
F1	0,71929825	0,67326733	0,70866142
Loss	0,80671223		
Acc	70,1754379		

Model = [Resnet34 dropConv01Lin02] Augmentation = [XZ +-6 degree 50%, Roll 95%-66% each direction]			
	AD pred	DLB pred	NC pred
AD true	43	6	8
DLB true	3	49	5
NC true	13	2	42
Precision	0,72881356	0,85964912	0,76363636
Recall	0,75438596	0,85964912	0,73684211
F1	0,74137931	0,85964912	0,75
Loss	0,58335911		
Acc	78,3625717		

Model = [Resnet50 dropConv01Lin02], Augmentation = [XZ +-6degree 50%, Roll 95%-66% each direction]			
	AD pred	DLB pred	NC pred
AD pred	43	3	11
DLB true	7	43	7
NC true	10	1	46
Precision	0,71666667	0,91489362	0,71875
Recall	0,75438596	0,75438596	0,807017544
F1	0,73504274	0,82692308	0,760330579
Loss	0,67090586		
Acc	77,1929855		

Model = [Resnet34 dropConv01Lin02], Augmentation = [XZ axsis +-6,0 degree 70%, Roll 95%-66% each direction]			
Best Fold	AD pred	DLB pred	NC pred
AD true	45	4	8
DLB true	6	44	7
NC true	4	3	50
Precision	0,81818182	0,8627451	0,76923077
Recall	0,78947368	0,77192982	0,87719298
F1	0,80357143	0,81481481	0,81967213
Loss	0,45309733		
Acc	81,2865524		

Model = [Resnet34 dropConv01Lin02], Augmentation = [XZ axsis +-6,0 degree 90%, Roll 95%-66% each direction]			
	AD pred	DLB pred	NC pred
AD true	51	3	3
DLB true	7	47	3
NC true	8	5	44
Precision	0,7727	0,8545	0,8800
Recall	0,8947	0,8246	0,7719
F1	0,8293	0,8393	0,8224
Loss	0,4890		
Acc	83,0409		

Model = [Resnet34 dropConv01Lin02], Augmentation = [XZ axsis +-6,0 degree 98%, Roll 95%-66% each direction]			
	AD pred	DLB pred	NC pred
AD true	45	6	6
DLB true	10	47	0
NC true	7	3	47
Precision	0,72580645	0,83928571	0,886792453
Recall	0,78947368	0,8245614	0,824561404
F1	0,75630252	0,83185841	0,854545455
Loss	0,52965451		
Acc	81,2865524		

Table B.32: Frac0.25 Remove Eyes Dataset with augmentations

With 230 GAN images for each class, Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, Rotate +-6 degree XY 90%

	Validation Acc	Test Acc
fold: 1	81,39	64,33
fold: 2	80,52	67,84
fold: 3	82,68	63,16
fold: 4	83,98	66,08
fold: 5	84,65	57,89
fold: 6	80,70	66,08
Average :	82,32	64,23

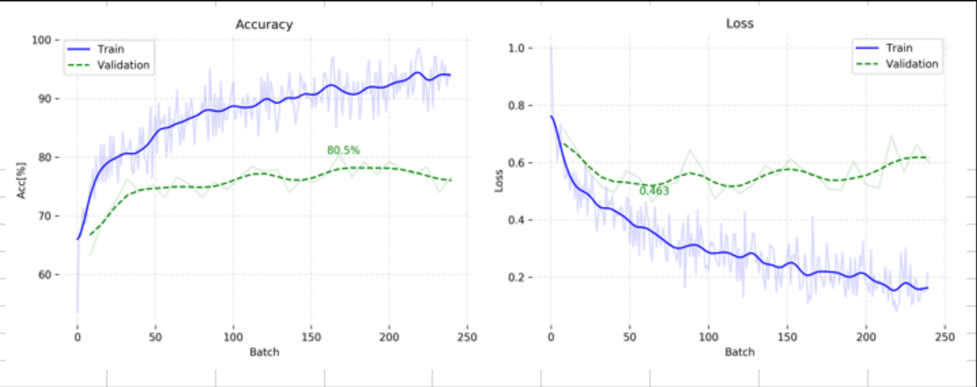
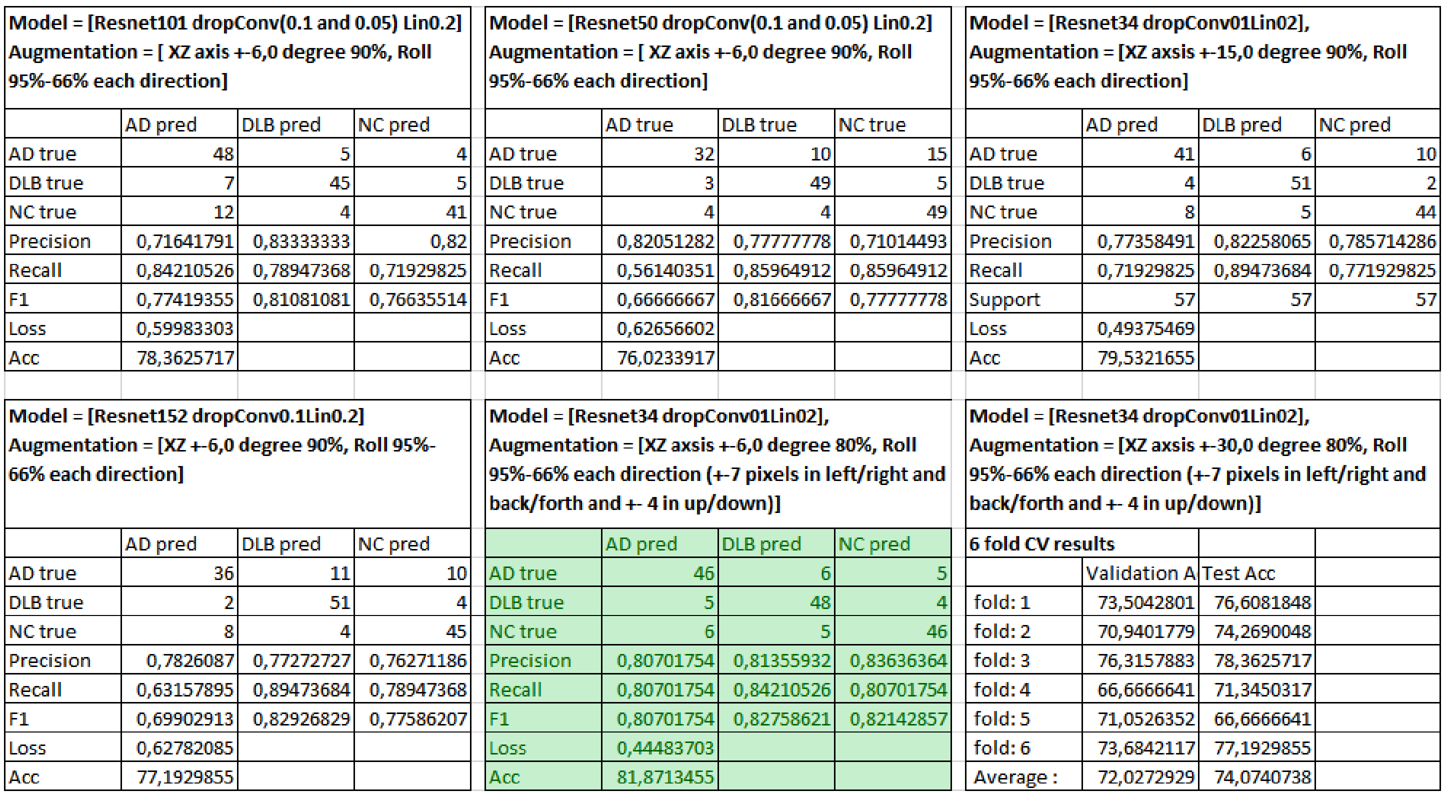


Table B.46: 6-CV Augmentation Results with 50%GAN Images, Roll and Rotate Augmentations



Model = [Resnet101 dropConv(0.1 and 0.05) Lin0.2] Augmentation = [XZ axis +-6,0 degree 90%, Roll 95%-66% each direction]

	AD pred	DLB pred	NC pred
AD true	48	5	4
DLB true	7	45	5
NC true	12	4	41
Precision	0,71641791	0,83333333	0,82
Recall	0,84210526	0,78947368	0,71929825
F1	0,77419355	0,81081081	0,76635514
Loss	0,59983303		
Acc	78,3625717		

Model = [Resnet50 dropConv(0.1 and 0.05) Lin0.2] Augmentation = [XZ axis +-6,0 degree 90%, Roll 95%-66% each direction]

	AD true	DLB true	NC true
AD true	32	10	15
DLB true	3	49	5
NC true	4	4	49
Precision	0,82051282	0,77777778	0,71014493
Recall	0,56140351	0,85964912	0,85964912
F1	0,66666667	0,81666667	0,77777778
Loss	0,62656602		
Acc	76,0233917		

Model = [Resnet34 dropConv01Lin02], Augmentation = [XZ axsis +-15,0 degree 90%, Roll 95%-66% each direction]

	AD pred	DLB pred	NC pred
AD true	41	6	10
DLB true	4	51	2
NC true	8	5	44
Precision	0,77358491	0,82258065	0,785714286
Recall	0,71929825	0,89473684	0,771929825
Support	57	57	57
Loss	0,49375469		
Acc	79,5321655		

Model = [Resnet152 dropConv0.1Lin0.2] Augmentation = [XZ +-6,0 degree 90%, Roll 95%-66% each direction]

	AD pred	DLB pred	NC pred
AD true	36	11	10
DLB true	2	51	4
NC true	8	4	45
Precision	0,7826087	0,77272727	0,76271186
Recall	0,63157895	0,89473684	0,78947368
F1	0,69902913	0,82926829	0,77586207
Loss	0,62782085		
Acc	77,1929855		

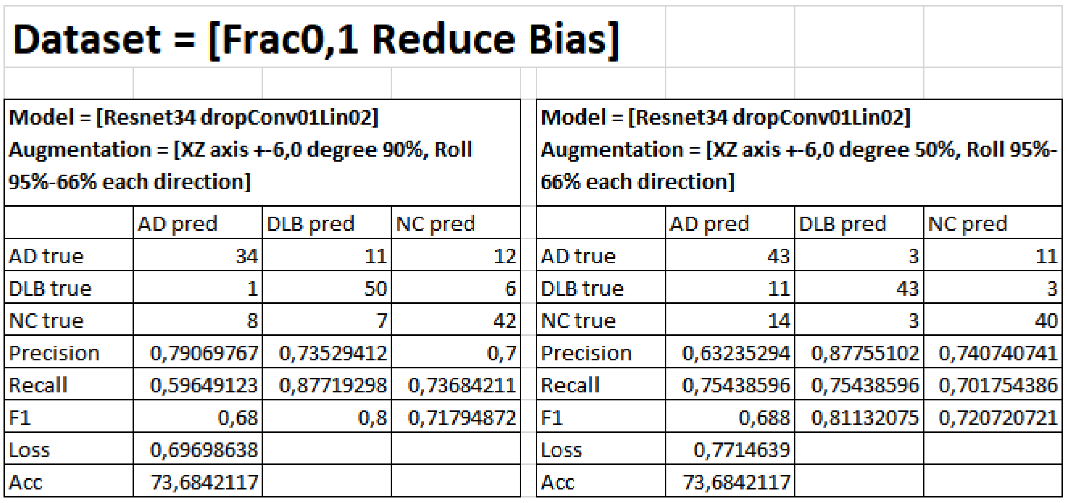
Model = [Resnet34 dropConv01Lin02], Augmentation = [XZ axsis +-6,0 degree 80%, Roll 95%-66% each direction (+-7 pixels in left/right and back/forth and +- 4 in up/down)]

	AD pred	DLB pred	NC pred
AD true	46	6	5
DLB true	5	48	4
NC true	6	5	46
Precision	0,80701754	0,81355932	0,83636364
Recall	0,80701754	0,84210526	0,80701754
F1	0,80701754	0,82758621	0,82142857
Loss	0,44483703		
Acc	81,8713455		

Model = [Resnet34 dropConv01Lin02], Augmentation = [XZ axsis +-30,0 degree 80%, Roll 95%-66% each direction (+-7 pixels in left/right and back/forth and +- 4 in up/down)]

6 fold CV results		
	Validation A	Test Acc
fold: 1	73,5042801	76,6081848
fold: 2	70,9401779	74,2690048
fold: 3	76,3157883	78,3625717
fold: 4	66,6666641	71,3450317
fold: 5	71,0526352	66,6666641
fold: 6	73,6842117	77,1929855
Average :	72,0272929	74,0740738

Table B.33: Frac0.25 Remove Eyes Dataset with augmentations



Dataset = [Frac0,1 Reduce Bias]

Model = [Resnet34 dropConv01Lin02] Augmentation = [XZ axis +-6,0 degree 90%, Roll 95%-66% each direction]

	AD pred	DLB pred	NC pred
AD true	34	11	12
DLB true	1	50	6
NC true	8	7	42
Precision	0,79069767	0,73529412	0,7
Recall	0,59649123	0,87719298	0,73684211
F1	0,68	0,8	0,71794872
Loss	0,69698638		
Acc	73,6842117		

Model = [Resnet34 dropConv01Lin02] Augmentation = [XZ axis +-6,0 degree 50%, Roll 95%-66% each direction]

	AD pred	DLB pred	NC pred
AD true	43	3	11
DLB true	11	43	3
NC true	14	3	40
Precision	0,63235294	0,87755102	0,740740741
Recall	0,75438596	0,75438596	0,701754386
F1	0,688	0,81132075	0,720720721
Loss	0,7714639		
Acc	73,6842117		

Table B.34: Frac0.1 Reduce Bias Dataset with augmentations

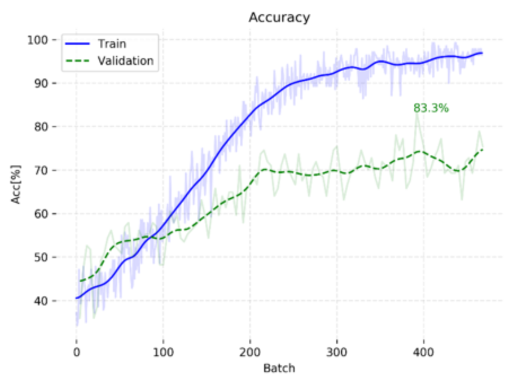
Dataset = [Frac0,2 Reduce Bias]

Dataset = [Frac=0,2 Reduce Bias], Model=[Resnet34 dropConv01Lin02], Augmentation = [Rotate XZ +-6,0 degree 90%, Roll 95%-66% each direction]

Best Fold	AD pred	DLB pred	NC pred
AD true	49	4	4
DLB true	10	40	7
NC true	7	9	41
Precision	0,74242424	0,75471698	0,78846154
Recall	0,85964912	0,70175439	0,71929825
F1	0,79674797	0,72727273	0,75229358
Loss	0,74746777		
Acc	76,0233917		

6 fold CV results

	Validation	Test Acc
fold: 1	74,36	74,27
fold: 2	70,09	73,10
fold: 3	68,42	67,84
fold: 4	83,33	76,02
fold: 5	71,93	69,59
fold: 6	73,68	66,08
Average :	73,64	71,15



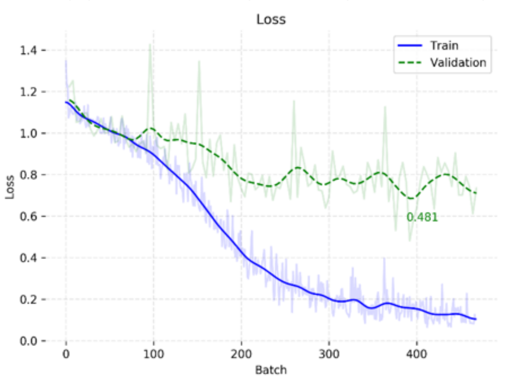
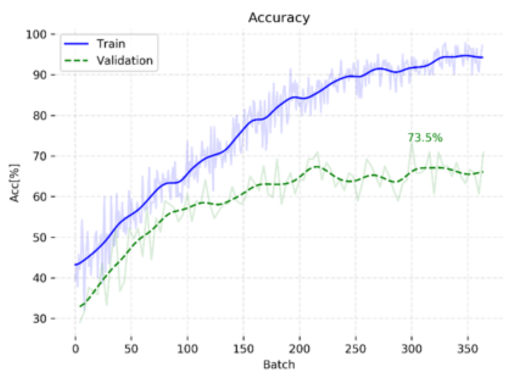


Table B.35: Frac0.2 Reduce Bias Dataset with augmentations

Frac0,25 with eye removal, Resnet18 dropConv01Lin02, +-6,0 degree 90%, Roll 95%-66% each direction

	AD pred	DLB pred	NC pred
AD true	51	3	3
DLB true	6	47	4
NC true	13	4	40
Precision	0,7286	0,8704	0,8511
Recall	0,8947	0,8246	0,7018
F1	0,8031	0,8468	0,7692
Loss	0,6854		
Acc	80,7018		

6 fold CV results		
	Validation Acc	Test Acc
fold: 1	67,52	60,82
fold: 2	73,50	80,12
fold: 3	71,93	80,12
fold: 4	71,05	65,50
fold: 5	73,68	78,95
fold: 6	74,56	76,02
Average :	72,04	73,59



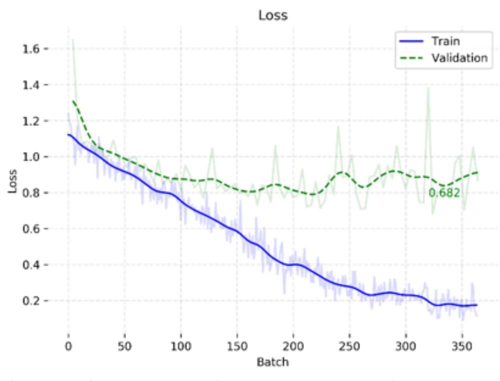


Table B.36: Experiments from the Frac0.25 Remove Eye Dataset: ResNet18

Dataset = [Frac0,25 with eye removal], Model = [Resnet34 dropConv01Lin02], Augmentation = [XZ axsis +-6,0 degree 90%, Roll 95%-66% each direction] With LeakyRelu

6 fold CV results

	Validation Acc	Test Acc
fold: 1	69,23077393	75,438599
fold: 2	72,64957428	73,099419
fold: 3	73,68421173	76,023392
fold: 4	69,29824829	80,701752
fold: 5	71,92982483	74,853806
fold: 6	70,17543793	63,157894
Average :	71,16134516	73,879143

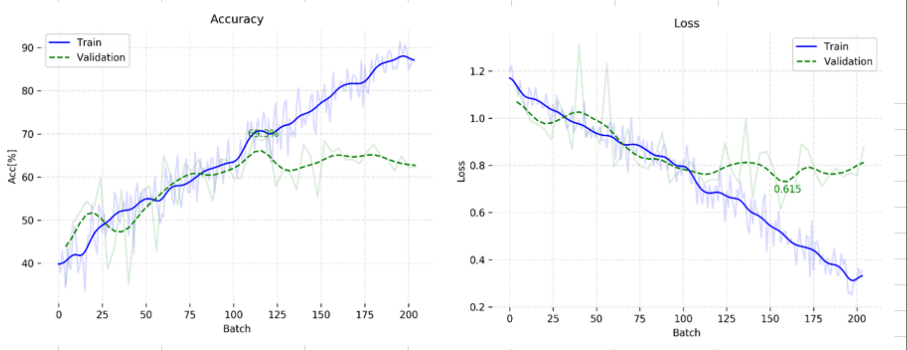


Table B.37: Experiments from the Frac0.25 Remove Eye Dataset: ResNet34 with LeakyReLU

Frac0,25 with eye removal, Resnet34 dropConv01Lin02, +-6,0 degree 90%, Roll 95%-66% each direction, 20% for mirroring,

With theese hyper param: optim,SGD(model,parameters(), lr=0,00034588581370567563, momentum=0,6666591635414372, dampening=0, weight_decay=0,047059700188148056, nesterov=True)

6 fold CV results

	Validation Acc	Test Acc
fold: 1	69,23077393	80,701752
fold: 2	72,64957428	80,116959
fold: 3	73,68421173	76,023392
fold: 4	76,31578827	80,701752
fold: 5	74,56140137	75,438599
fold: 6	78,94737244	76,608185
Average :	74,23152033	78,265106

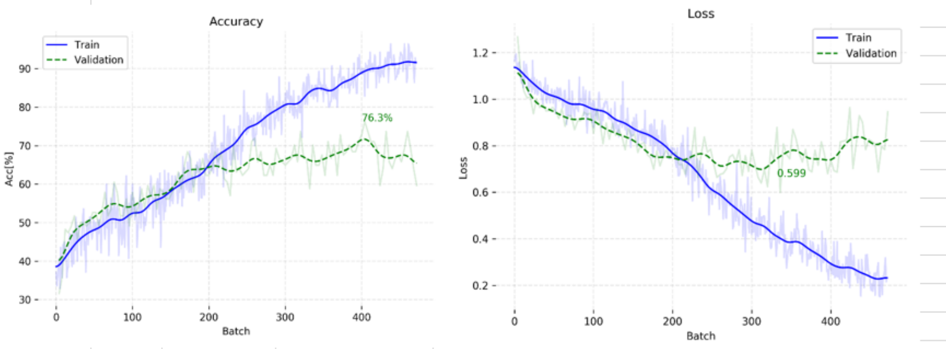


Table B.38: Experiments from the Frac0.25 Remove Eye Dataset: ResNet34 with Different Hyperparameters

Frac 0,5 dataset, Roll 1-4 pixels in each direction, 95% chance of translation and 66% change for each direction, Rotate +-6 degree XY 90%

	Validation Acc	Test Acc
fold: 1	61,54	56,73
fold: 2	66,67	61,40
fold: 3	68,42	64,33
fold: 4	69,30	59,06
fold: 5	71,05	59,65
fold: 6	69,30	59,06
Average :	67,71	60,04

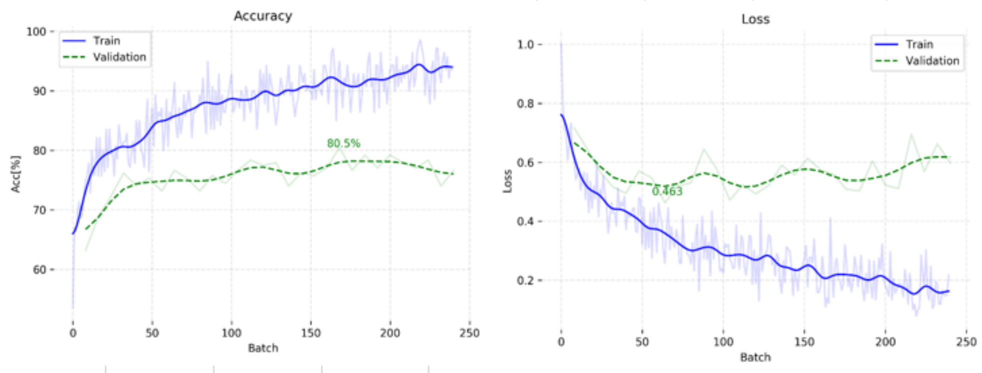
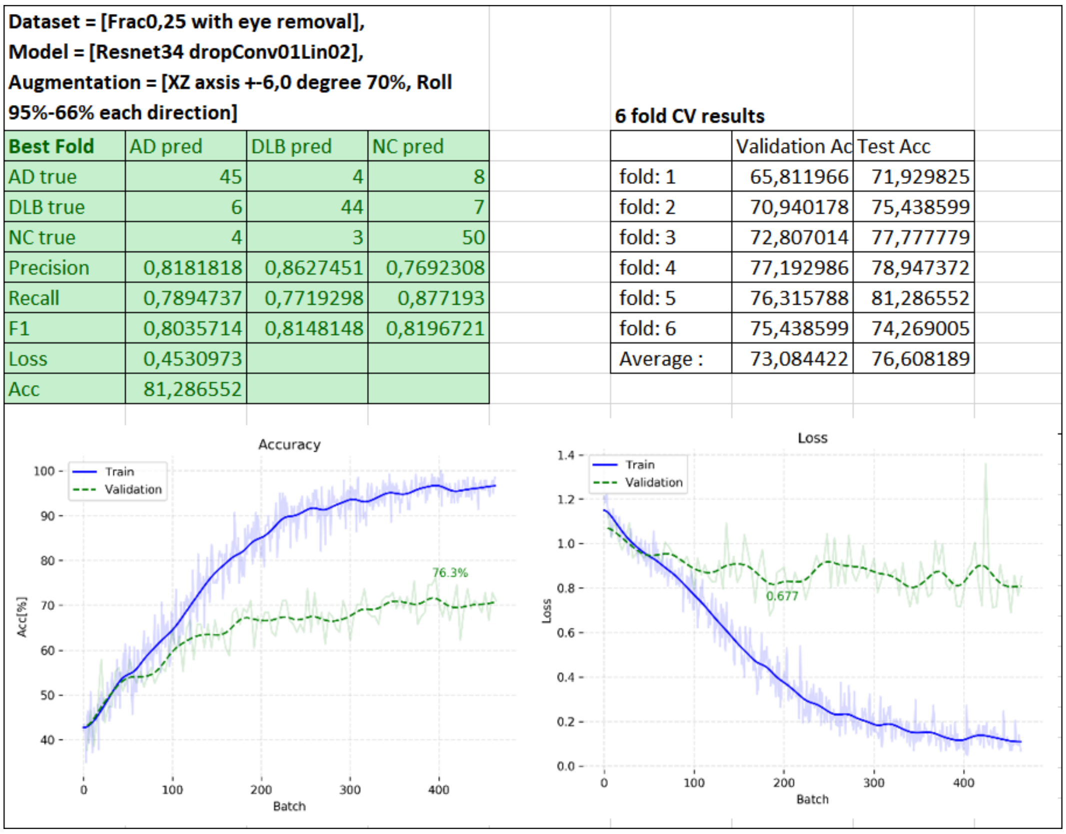


Table B.47: 6-CV Result Frac=0.5 dataset, With Roll and Rotate Augmentations



Dataset = [Frac0,25 with eye removal],
Model = [Resnet34 dropConv01Lin02],
Augmentation = [XZ axsis +-6,0 degree 70%, Roll 95%-66% each direction]

Best Fold	AD pred	DLB pred	NC pred
AD true	45	4	8
DLB true	6	44	7
NC true	4	3	50
Precision	0,8181818	0,8627451	0,7692308
Recall	0,7894737	0,7719298	0,877193
F1	0,8035714	0,8148148	0,8196721
Loss	0,4530973		
Acc	81,286552		

6 fold CV results

	Validation Ac	Test Acc
fold: 1	65,811966	71,929825
fold: 2	70,940178	75,438599
fold: 3	72,807014	77,777779
fold: 4	77,192986	78,947372
fold: 5	76,315788	81,286552
fold: 6	75,438599	74,269005
Average :	73,084422	76,608189

Table B.39: Experiments from the Frac0.25 Remove Eye Dataset: ResNet34 with Different Rotation, chance=70

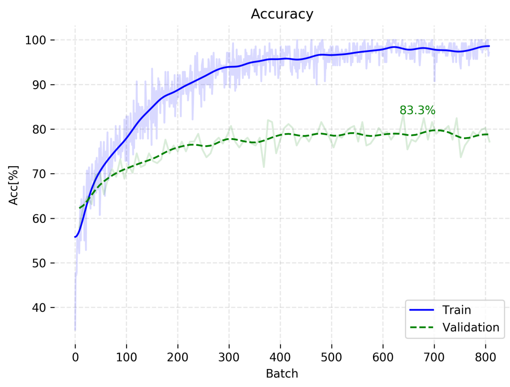


Table B.48: Accuracy plot for training upscaled GAN Experiment1

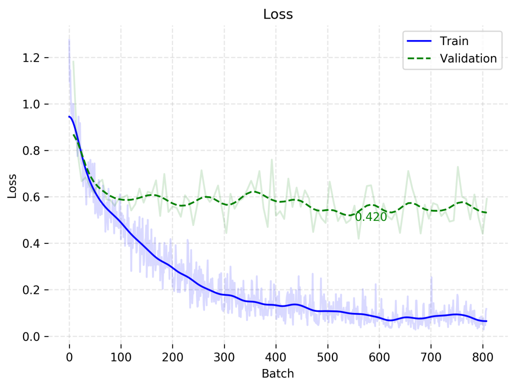
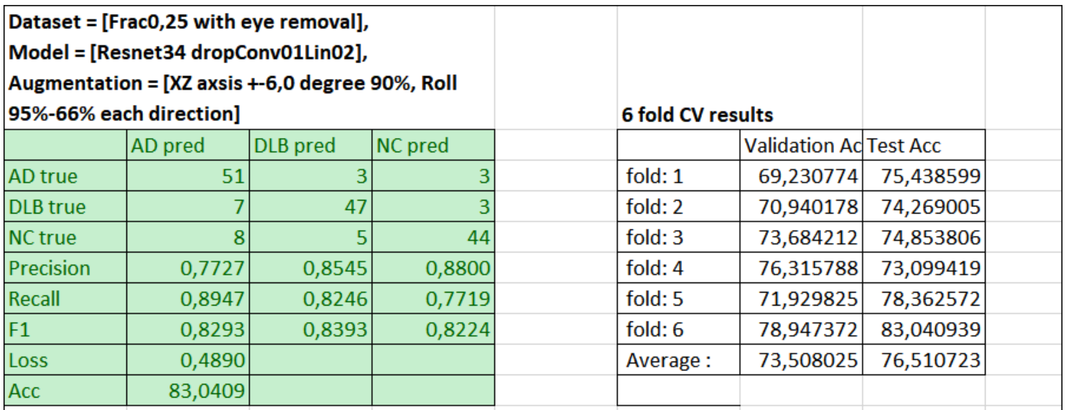


Table B.49: Loss plot for training upscaled GAN Experiment1



Dataset = [Frac0,25 with eye removal],
Model = [Resnet34 dropConv01Lin02],
Augmentation = [XZ axsis +-6,0 degree 90%, Roll 95%-66% each direction]

	AD pred	DLB pred	NC pred
AD true	51	3	3
DLB true	7	47	3
NC true	8	5	44
Precision	0,7727	0,8545	0,8800
Recall	0,8947	0,8246	0,7719
F1	0,8293	0,8393	0,8224
Loss	0,4890		
Acc	83,0409		

6 fold CV results

	Validation Ac	Test Acc
fold: 1	69,230774	75,438599
fold: 2	70,940178	74,269005
fold: 3	73,684212	74,853806
fold: 4	76,315788	73,099419
fold: 5	71,929825	78,362572
fold: 6	78,947372	83,040939
Average :	73,508025	76,510723

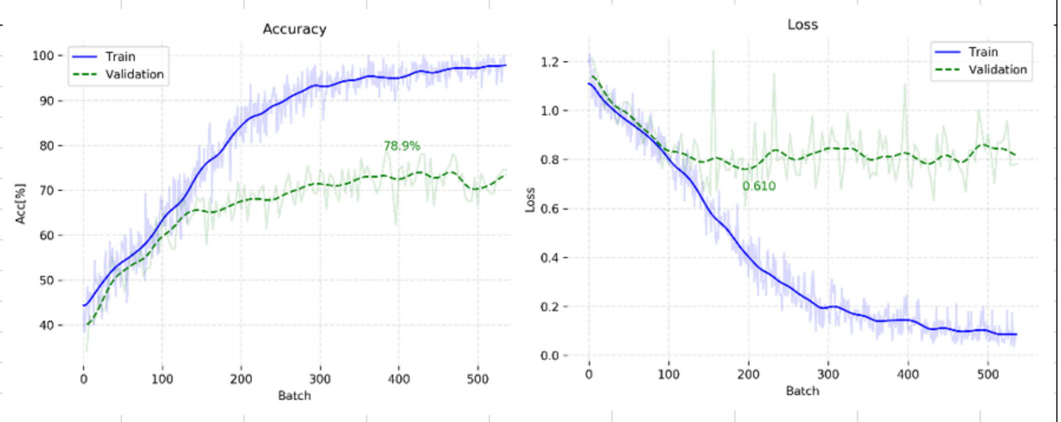


Table B.40: Experiments from the Frac0.25 Remove Eye Dataset: ResNet34 with Different Rotation, chance=90

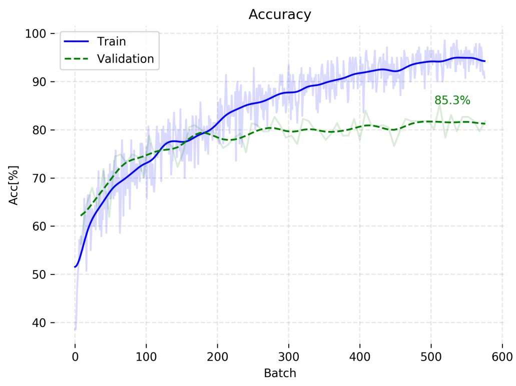


Table B.50: Accuracy plot for training upscaled GAN Experiment2

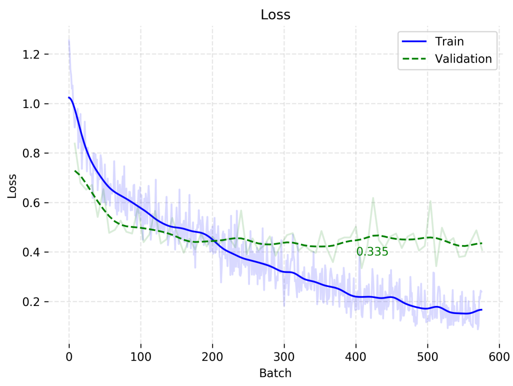
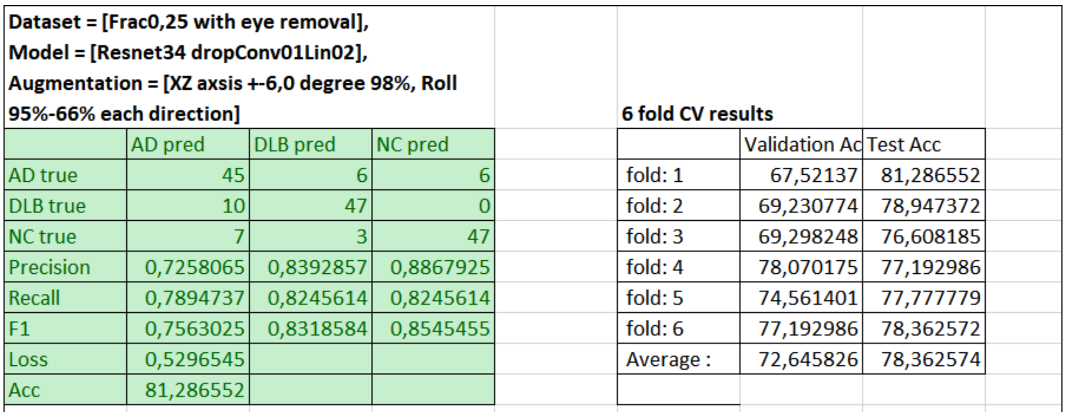


Table B.51: Loss plot for training upscaled GAN Experiment2



Dataset = [Frac0,25 with eye removal], Model = [Resnet34 dropConv01Lin02], Augmentation = [XZ axsis +-6,0 degree 98%, Roll 95%-66% each direction]

	AD pred	DLB pred	NC pred
AD true	45	6	6
DLB true	10	47	0
NC true	7	3	47
Precision	0,7258065	0,8392857	0,8867925
Recall	0,7894737	0,8245614	0,8245614
F1	0,7563025	0,8318584	0,8545455
Loss	0,5296545		
Acc	81,286552		

6 fold CV results

	Validation Ac	Test Acc
fold: 1	67,52137	81,286552
fold: 2	69,230774	78,947372
fold: 3	69,298248	76,608185
fold: 4	78,070175	77,192986
fold: 5	74,561401	77,777779
fold: 6	77,192986	78,362572
Average :	72,645826	78,362574

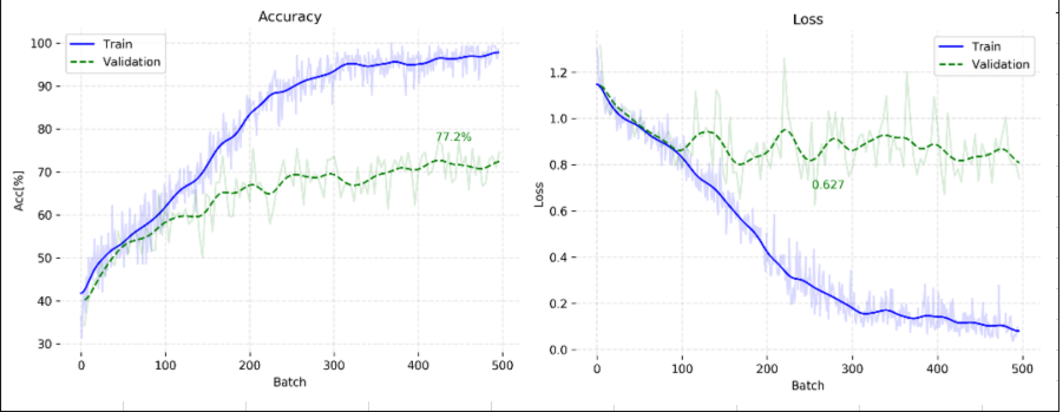
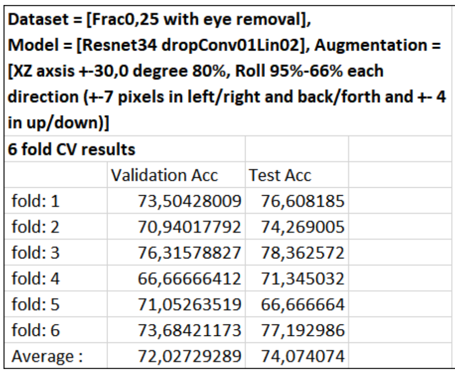


Table B.41: Experiments from the Frac0.25 Remove Eye Dataset: ResNet34 with Different Rotation, chance=98



Dataset = [Frac0,25 with eye removal], Model = [Resnet34 dropConv01Lin02], Augmentation = [XZ axsis +-30,0 degree 80%, Roll 95%-66% each direction (+-7 pixels in left/right and back/forth and +- 4 in up/down)]

6 fold CV results

	Validation Acc	Test Acc
fold: 1	73,50428009	76,608185
fold: 2	70,94017792	74,269005
fold: 3	76,31578827	78,362572
fold: 4	66,66666412	71,345032
fold: 5	71,05263519	66,666664
fold: 6	73,68421173	77,192986
Average :	72,02729289	74,074074

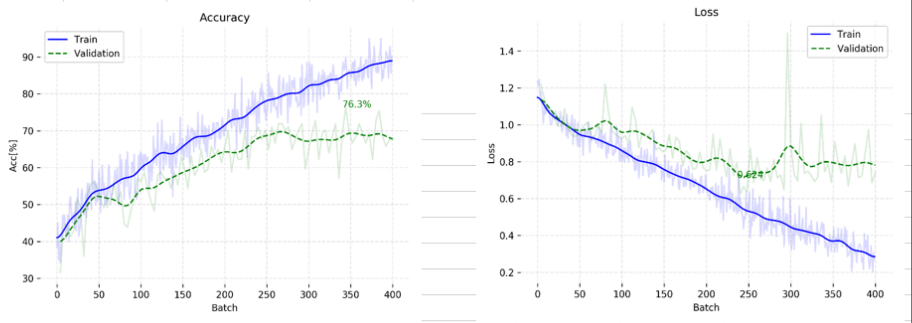
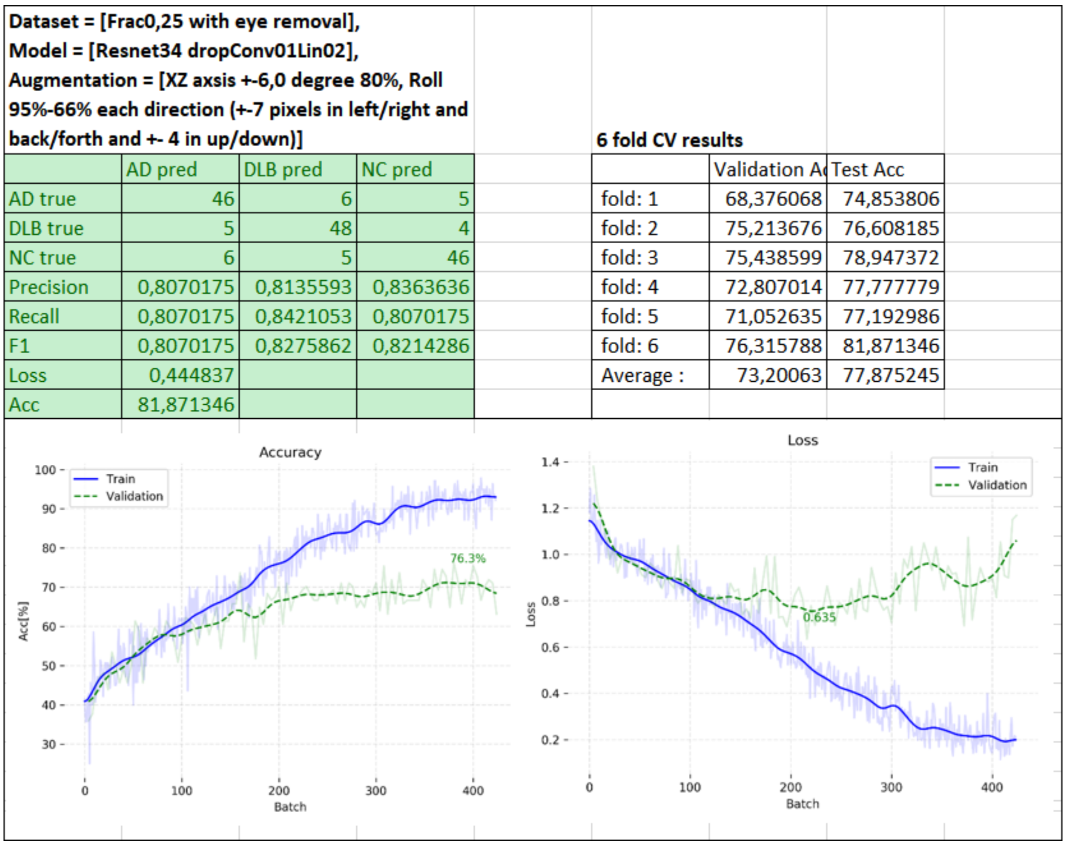


Table B.42: Experiments from the Frac0.25 Remove Eye Dataset: ResNet34 with Different Rotation=+-30 Degree, and Bigger Translations



Dataset = [Frac0,25 with eye removal],
Model = [Resnet34 dropConv01Lin02],
Augmentation = [XZ axsis +-6,0 degree 80%, Roll 95%-66% each direction (+-7 pixels in left/right and back/forth and +- 4 in up/down)]

	AD pred	DLB pred	NC pred
AD true	46	6	5
DLB true	5	48	4
NC true	6	5	46
Precision	0,8070175	0,8135593	0,8363636
Recall	0,8070175	0,8421053	0,8070175
F1	0,8070175	0,8275862	0,8214286
Loss	0,444837		
Acc	81,871346		

6 fold CV results

	Validation Ac	Test Acc
fold: 1	68,376068	74,853806
fold: 2	75,213676	76,608185
fold: 3	75,438599	78,947372
fold: 4	72,807014	77,777779
fold: 5	71,052635	77,192986
fold: 6	76,315788	81,871346
Average :	73,20063	77,875245

Table B.43: Experiments from the Frac0.25 Remove Eye Dataset: ResNet34 with Different Rotation=+-6 Degree, and Bigger Translations

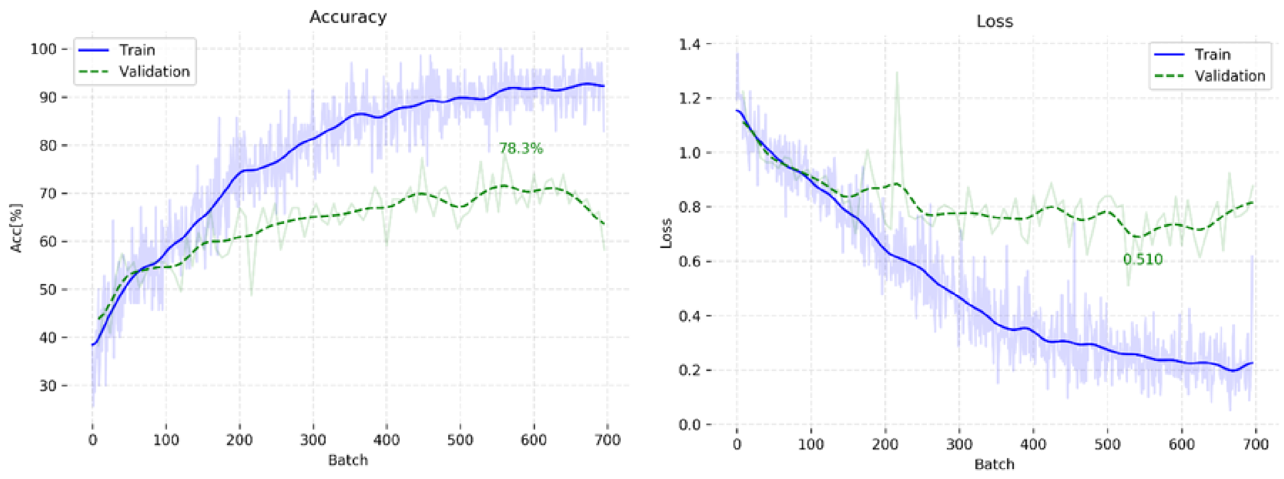


Table B.52: Accuracy and Loss graphs from the training of fold 6 from table 4.7

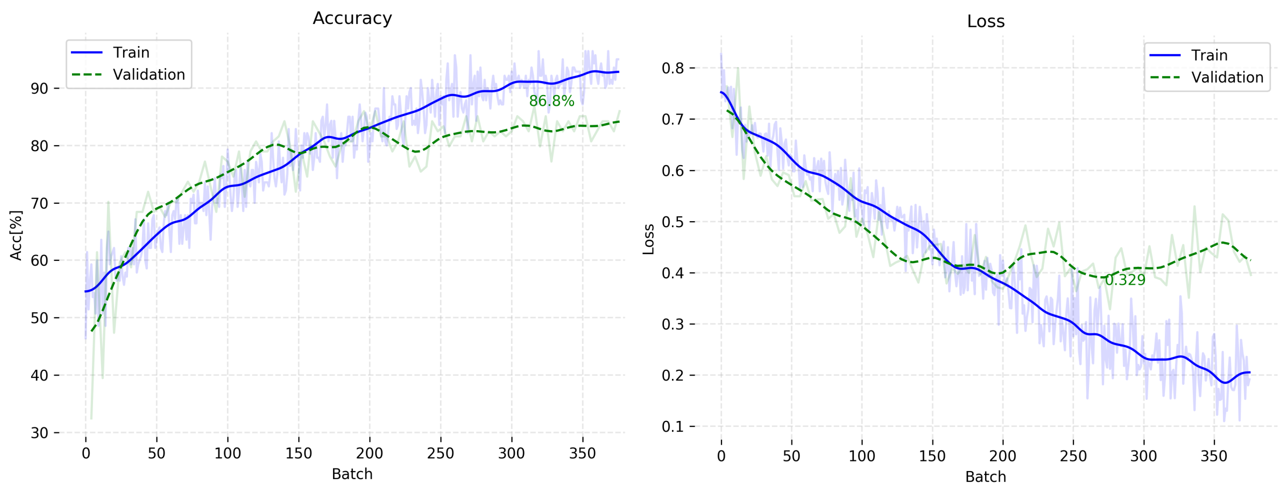


Table B.53: Accuracy and Loss graphs from the training of fold 6 from table 4.27

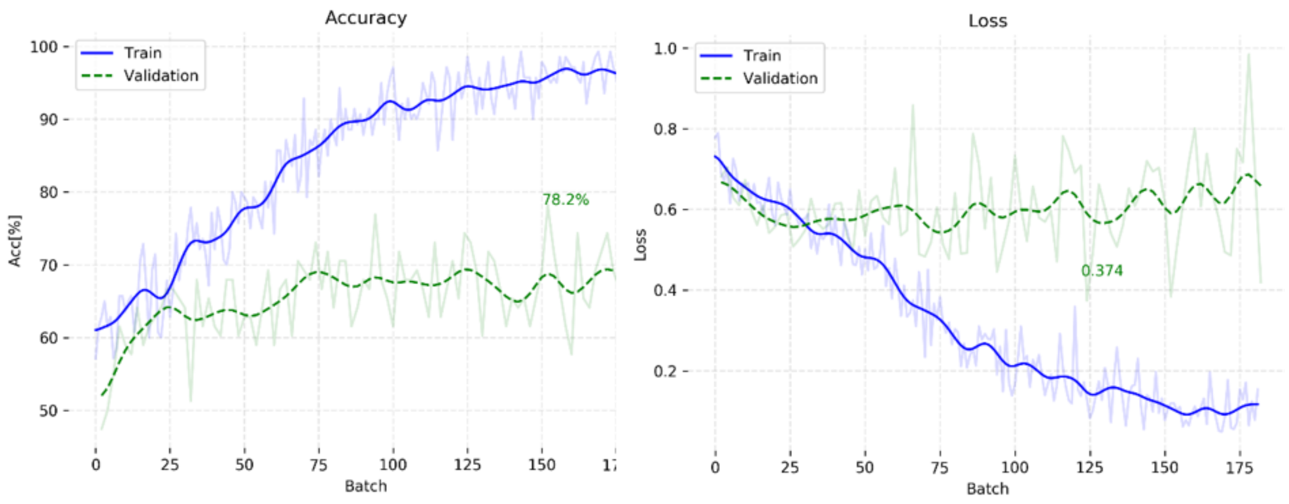


Table B.54: Accuracy and Loss graphs from the training of fold 2 from table 4.31

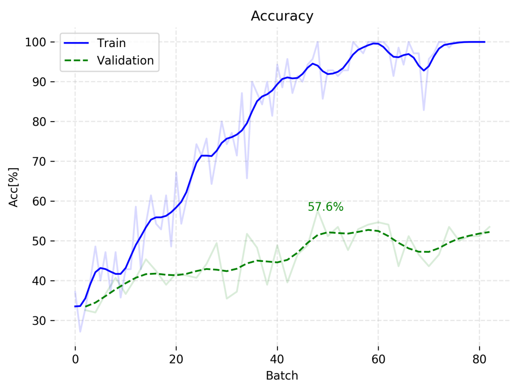


Table B.56: Accuracy plot of best local model, set 1, trained with the Adam optimizer

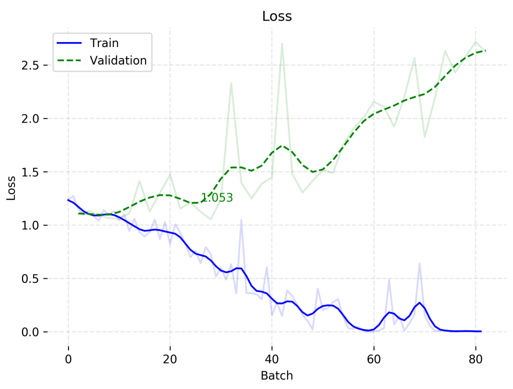


Table B.57: Loss plot of best local model, set 1, trained with the Adam optimizer

6 fold CV results, 2 class AD vs NC		
	Validation Ac	Test Acc
fold: 1	79,49	85,09
fold: 2	80,77	84,21
fold: 3	71,05	80,70
fold: 4	80,26	78,95
fold: 5	89,47	85,09
fold: 6	80,26	83,33
Average :	80,22	82,89

Confusion plot and metics from results in fold 5		
	AD pred	**Ncpred**
AD true	51	6
NC true	11	46
Precision	0,8226	0,8846
Recall	0,8947	0,8070
F1	0,8571	0,8440
Loss	0,3991	
Acc	85,09	

	AD	**NC**	**Average**
Average recall	0,8509	0,8070	0,8289
Average precision	0,8161	0,8466	0,8313

Table B.55: Experiments with training proposed model on NC vs AD data, so it can be compared to other results easier

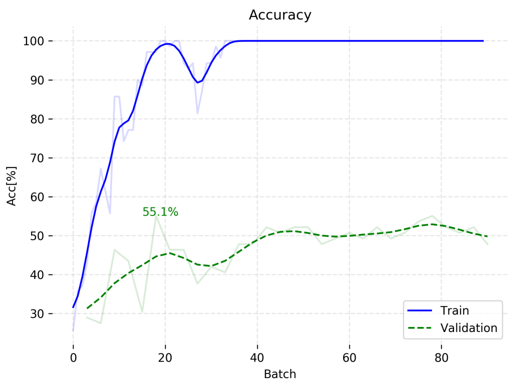


Table B.58: Accuracy plot of the federated model, set 1, trained with the Adam optimizer

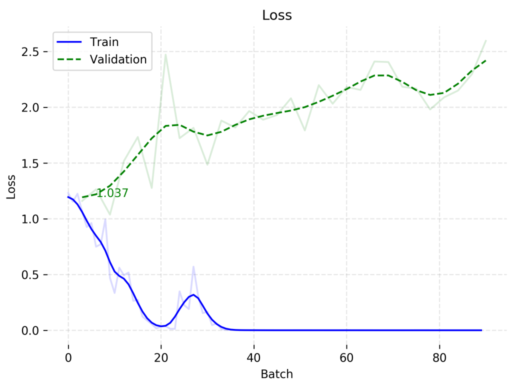


Table B.59: Loss plot of the federated model, set 1, trained with the Adam optimizer

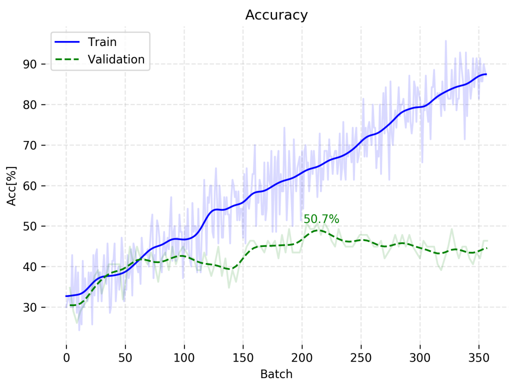


Table B.60: Accuracy plot of best local model, set 2, trained with the Adam optimizer

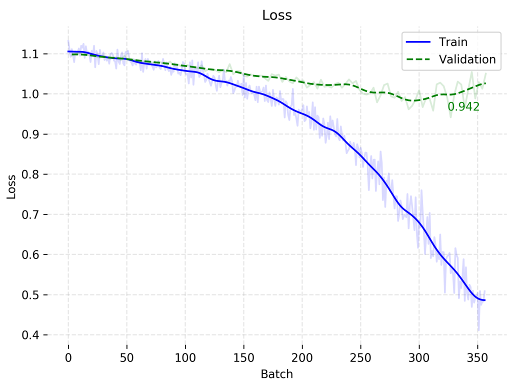


Table B.61: Loss plot of best local model, set 2, trained with the Adam optimizer

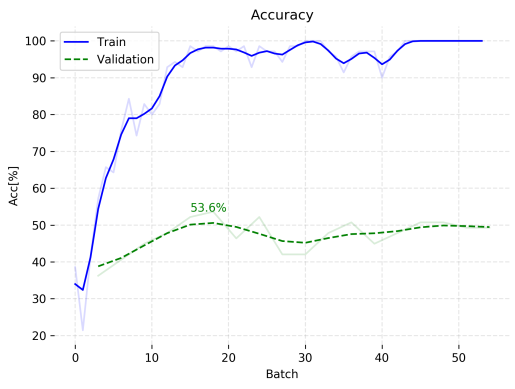


Table B.62: Accuracy plot of the federated model, set 2, trained with the Adam optimizer

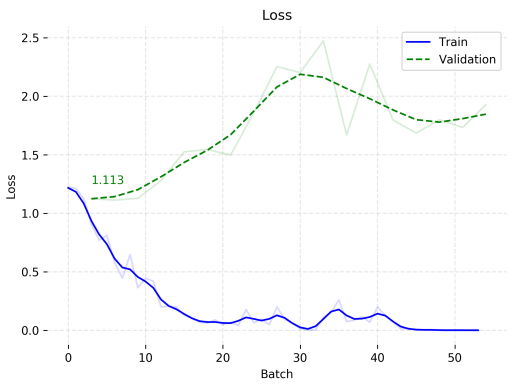


Table B.63: Loss plot of the federated model, set 2, trained with the Adam optimizer

Appendix B

Bibliography

[1] Engineering National Academies of Sciences, Medicine, et al. *Improving diagnosis in health care*. National Academies Press, 2015.

[2] Liam Peytona Mana Azarm-Daigle, Craig Kuziemsky. A Review of Cross Organizational Healthcare Data Sharing. URL https://www.sciencedirect.com/science/article/pii/S1877050915024989.

[3] Zeynettin Akkus Timothy L. Kline Bradley J. Erickson, Panagiotis Korfiatis. Machine learning for medical imaging. URL https://pubs.rsna.org/doi/10.1148/rg.2017160130.

[4] National Institute on Aging. What is Dementia? Symptoms, Types, and diagnosis. URL https://www.nia.nih.gov/health/what-dementia-symptoms-types-and-diagnosis.

[5] The World Health Organization. Dementia, 2020. URL https://www.who.int/news-room/fact-sheets/detail/dementia.

[6] Alzheimer's Association. Vascular dementia, date: 2020-07-13. URL https://www.alz.org/alzheimers-dementia/what-is-dementia/types-of-dementia/vascular-dementia.

[7] Alz.org. Lewy body dementia, . URL https://www.alz.org/alzheimers-dementia/what-is-dementia/types-of-dementia/lewy-body-dementia.

[8] Alz.org. What Is Dementia?, . URL https://www.alz.org/alzheimers-dementia/what-is-dementia.

[9]

[10] National Institute on aging. Alzheimer's Disease Fact Sheet. URL https://www.nia.nih.gov/health/alzheimers-disease-fact-sheet.

[11] BruceBlaus. Alzheimersdisease.jpg, date: 2020-07-08, license: Creative commons attribution-share alike 4.0 international. URL https://commons.wikimedia.org/wiki/File:Alzheimers_Disease.jpg.

[12] alzheimers.org.uk. What Is Alzheimer's Disease? URL https://www.alzheimers.org.uk/about-dementia/types-dementia/alzheimers-disease.

[13] Knut Engedal Store Norske Leksikon. Demens med lewylegemer. URL https://sml.snl.no/demens_med_lewylegemer.

[14] National Institute on Aging. What is Lewy Body Dementia. URL https://www.nia.nih.gov/health/what-lewy-body-dementia.

[15] Suraj Rajan. Lewy bodies(alpha synuclein inclusions)1.jpg, date: 2020-07-08, license: Creative commons attribution-share alike 4.0 international. URL https://commons.wikimedia.org/wiki/File:Lewy_bodies_(alpha_synuclein_inclusions)_1.jpg.

[16] National Health Service. Dementia with Lewy bodies. URL https://www.nhs.uk/conditions/dementia-with-lewy-bodies/.

[17] National Institute of Biomedical Imaging and Bioengineering. Magnetic Resonance Imaging (MRI. URL https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri.

[18] Marilyn S Albert, Steven T DeKosky, Dennis Dickson, Bruno Dubois, Howard H Feldman, Nick C Fox, Anthony Gamst, David M Holtzman, William J Jagust, Ronald C Petersen, et al. The diagnosis of mild cognitive impairment due to alzheimer's disease: recommendations from the national institute on aging-alzheimer's association workgroups on diagnostic guidelines for alzheimer's disease. *Alzheimer's & dementia*, 7(3):270–279, 2011.

[19] Laura Bonanni, Astrid Thomas, and Marco Onofrj. Diagnosis and management of dementia with lewy bodies: Third report of the dlb consortium. *Neurology*, 66(9): 1455–1455, 2006. ISSN 0028-3878. doi: 10.1212/01.wnl.0000224698.67660.45. URL https://n.neurology.org/content/66/9/1455.1.

[20] EJ Burton, R Barber, EB Mukaetova-Ladinska, J Robson, RH Perry, E Jaros, RN Kalaria, and JT O'brien. Medial temporal lobe atrophy on mri differentiates alzheimer's disease from dementia with lewy bodies and vascular cognitive impairment: a prospective study with pathological verification of diagnosis. *Brain*, 132(1): 195–203, 2009.

[21] Federica Agosta, Sebastiano Galantucci, and Massimo Filippi. Advanced magnetic resonance imaging of neurodegenerative diseases. *Neurological sciences*, 38(1):41–51, 2017.

[22] Elijah Mak, Li Su, Guy B Williams, Rosie Watson, Michael Firbank, Andrew Blamire, and John O'Brien. Differential atrophy of hippocampal subfields: a comparative study of dementia with lewy bodies and alzheimer disease. *The American Journal of Geriatric Psychiatry*, 24(2):136–143, 2016.

[23] Simen Norrheim Larsen. Data-assisted differential diagnosis of dementia by deep neural networks, 2019. URL `http://hdl.handle.net/11250/2620347`.

[24] SH Shabbeer Basha, Shiv Ram Dubey, Viswanath Pulabaigari, and Snehasis Mukherjee. Impact of fully connected layers on performance of convolutional neural networks for image classification. *Neurocomputing*, 378:112–119, 2020.

[25] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

[26] Sebastian Ruder. An overview of gradient descent optimization algorithms, date: 2020-07-14. URL `https://arxiv.org/pdf/1609.04747.pdf`.

[27] Jimmy Lei Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *Published as a conference paper at ICLR 2015*, 2015.

[28] DanB. Rectified linear units (relu) in deep learning, date: 2020-07-14. URL `https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning`.

[29] Anas Al-Masri. What are overfitting and underfitting in machine learning?, date: 2020-07-14. URL `https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690`.

[30] Gringer. Overfitting svg, date: 2020-07-14, license: Creative commons attribution 3.0 unported. URL `https://en.wikipedia.org/wiki/File:Overfitting_svg.svg`.

[31] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[34] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[35] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 613–621. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6194-generating-videos-with-scene-dynamics.pdf.

[36] Kevin Schawinski, Ce Zhang, Hantian Zhang, Lucas Fowler, and Gokula Krishnan Santhanam. Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *Monthly Notices of the Royal Astronomical Society: Letters*, 467(1):L110–L114, 2017.

[37] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *The European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[38] GAN — Why it is so hard to train Generative Adversarial Networks, https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b, (Accessed: 25/06/2020).

[39] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[40] Mihaela Rosca, Balaji Lakshminarayanan, David Warde-Farley, and Shakir Mohamed. Variational approaches for auto-encoding generative adversarial networks. *arXiv preprint arXiv:1706.04987*, 2017.

[41] Wolfgang Grieskamp Dzmitry Huba Alex Ingerman Vladimir Ivanov Chloe Kiddon Jakub Konecny Stefano Mazzocchi H. Brendan McMahan Timon Van Overveldt David Petrou Daniel Ramage Jason Roselander Keith Bonawitz, Hubert Eichner. Towards federated learning at scale: System design. 2019.

[42] Felix X. Yu Ananda Theertha Suresh Dave Bacon Jakub Konecny, H. Brendan McMahan. Federated learning: Strategies for improving communication efficiency. 2017.

[43] Daniel Ramage Jakub Konecny, H. Brendan McMahan. Federated optimization: Distributed optimization beyond the datacenter. 2015.

[44] Marc Tommasi Paul Vanhaesebrouck, Aurelien Bellet. Decentralized collaborative learning of personalized models over networks. 2017.

[45] Brendan McMahan and Daniel Ramage. Federated Learning: Collaborative Machine Learning without Centralized Training Data. URL https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[46] Jeromemetronome. Federated learning general process in central orchestrator setup, date: 2020-07-13, license: Creative commons attribution-share alike 4.0 international. URL https://en.wikipedia.org/wiki/Federated_learning#/media/File:Federated_learning_process_central_case.png.

[47] Daniel Ramage Seth Hampson Blaise Aguera y Arcas H. Brendan McMahan, Eider Moore. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629v3*, 2017.

[48] PyTorch. PyTorch. URL https://pytorch.org/.

[49] Open Minded. PySyft. URL https://github.com/OpenMined/PySyft/.

[50] Inc Red Hat. Docker. URL https://opensource.com/resources/what-docker/.

[51] Krzysztof Gorgolewski, Christopher D Burns, Cindee Madison, Dav Clark, Yaroslav O Halchenko, Michael L Waskom, and Satrajit S Ghosh. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Frontiers in neuroinformatics*, 5:13, 2011.

[52] Nipype. Nipype Dockerimage. URL https://hub.docker.com/r/nipype/nipype/.

[53] Alzheimer's Disease Neuroimaging Initiative(ADNI) databases, http://adni.loni.usc.edu/data-samples/access-data/, (Accessed: 13/07/2020).

[54] SPM12 - Statistical Parametric Mapping, https://www.fil.ion.ucl.ac.uk/spm/software/spm12/, (Accessed: 14/06/2020).

[55] Stephen M Smith. Fast robust automated brain extraction. *Human brain mapping*, 17(3):143–155, 2002.

[56] Mark Jenkinson, Mickael Pechaud, Stephen Smith, et al. Bet2: Mr-based estimation of brain, skull and scalp surfaces. In *Eleventh annual meeting of the organization for human brain mapping*, volume 17, page 167. Toronto., 2005.

[57] Valeriu Popescu, Marco Battaglini, WS Hoogstrate, Sander CJ Verfaillie, IC Sluimer, Ronald A van Schijndel, Bob W van Dijk, Keith S Cover, Dirk L Knol, Mark Jenkinson, et al. Optimizing parameter choice for fsl-brain extraction tool (bet) on 3d t1 images in multiple sclerosis. *Neuroimage*, 61(4):1484–1494, 2012.

[58] Adrien Payan and Giovanni Montana. Predicting alzheimer's disease: a neuroimaging study with 3d convolutional neural networks. *arXiv preprint arXiv:1502.02506*, 2015.

[59] Francesco Saverio Zuppichini — Implementing ResNet in Pytorch, https://github.com/francescosaveriozuppichini/resnet, (Accessed: 04/07/2020).

[60] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2682–2690, 2019.

[61] Marios Aspris — Simple Implementation of Densely Connected Convolutional Networks in PyTorch, https://github.com/con-mi/deep-learning-projects/blob/master/simple_implementation_of_densely_connected_neural_networks.ipynb, (Accessed: 04/07/2020).

[62] Gihyun Kwon, Chihye Han, and Dae-shik Kim. Generation of 3d brain mri using auto-encoding generative adversarial networks. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 118–126. Springer, 2019.

[63] Generation of 3d brain mri using auto-encoding generative adversarial networks.

[64] OpenMinded. Websockets mnist example. URL https://github.com/OpenMined/PySyft/tree/master/examples/tutorials/advanced/websockets_mnist.

[65] Akihiko Wada, Kohei Tsuruta, Ryusuke Irie, Koji Kamagata, Tomoko Maekawa, Shohei Fujita, Saori Koshino, Kanako Kumamaru, Michimasa Suzuki, Atsushi Nakanishi, et al. Differentiating alzheimer's disease from dementia with lewy bodies using a deep learning technique based on structural brain connectivity. *Magnetic Resonance in Medical Sciences*, 18(3):219, 2019.

[66] Ketil Oppedal, Trygve Eftestøl, Kjersti Engan, Mona K Beyer, and Dag Aarsland. Classifying dementia using local binary patterns from different regions in magnetic resonance images. *International journal of biomedical imaging*, 2015, 2015.

[67] Junhao Wen, Elina Thibeau-Sutre, Mauricio Diaz-Melo, Jorge Samper-González, Alexandre Routier, Simona Bottani, Didier Dormont, Stanley Durrleman, Ninon

Burgos, Olivier Colliot, et al. Convolutional neural networks for classification of alzheimer's disease: Overview and reproducible evaluation. *Medical Image Analysis*, page 101694, 2020.

[68] Jyoti Islam and Yanqing Zhang. Brain mri analysis for alzheimer's disease diagnosis using an ensemble system of deep convolutional neural networks. *Brain informatics*, 5(2):2, 2018.

[69] Mingxia Liu, Jun Zhang, Ehsan Adeli, and Dinggang Shen. Landmark-based deep multi-instance learning for brain disease diagnosis. *Medical image analysis*, 43: 157–168, 2018.

[70] Karl Bäckström, Mahmood Nazari, Irene Yu-Hua Gu, and Asgeir Store Jakola. An efficient 3d deep convolutional network for alzheimer's disease diagnosis using mr images. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 149–153. IEEE, 2018.

[71] Karim Aderghal, Alexander Khvostikov, Andrei Krylov, Jenny Benois-Pineau, Karim Afdel, and Gwenaelle Catheline. Classification of alzheimer disease on imaging modalities with deep cnns using cross-modal transfer learning. In *2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS)*, pages 345–350. IEEE, 2018.

[72] Upul Senanayake, Arcot Sowmya, and Laughlin Dawes. Deep fusion pipeline for mild cognitive impairment diagnosis. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 1394–1997. IEEE, 2018.

[73] Yuekai Sun Dimitris Papailiopoulos Yasaman Khazaeni Hongyi Wang, Mikhail Yurochkin. Federated learning with matched averaging. *Published as a conference paper at ICLR 2020*, 2020.

[74] Tensorflow. Federated learning, date: 2020-07-13. URL https://www.tensorflow.org/federated/federated_learning.

[75] Hien Tran Khanh Tran Tien-Dung Cao, Tram Truong-Huu. A federated learning framework for privacy-preserving and parallel training. *arXiv:2001.09782v2*, 2020.