Application of CNN Models in Identification of Alzheimer Syndrome Based on MRI Segmentation Images

Marina Nikolaeva, Vladimir Bliznyukov

Abstract

Alzheimer syndrome is an increasing problem of today's population. This disease causes a significant brain damage and subsequently deprives individuals of performing daily tasks. In our project we would like to use MRI segmentation scans to explore the ways of identifying the extent of disease severity with the use of CNN models. This is a classification task.

1 Introduction

Alzheimer syndrome is a disease that causes brain and memory damage and gradual disability of performing daily activities. Nowadays, Alzheimer syndrome is becoming a common disease even within younger groups. Unfortunately, today there is no official cure for this syndrome. Thus, it is crucial to develop an automated way to detect this syndrome at early stages, which will allow for coherent treatment. Moreover, automating such diagnosis procedure will remove personal judgement making it more objective and give more people access to this system from anywhere in the world.

In this project, we explore CNN models that help identify Alzheimer syndrome based on brain scans. This will help to automate diagnosis process and pave the way to objective and accessible feedback for everyone.

GitHub link of the Project

2 Deliverable 1

2.1 Proposed solution

The dataset will be taken from Kaggle official page link. It consists of 4 groups of participants: Mild, Moderate, VeryMild and Healthy respectively. The first objective is to effectively preprocess (dilation, erosion, edge detection, filtering) the images for reasonable feature extraction.

The second objective is to specify the practical procedure of dealing with highly imbalanced datasets (patients are the minority group). This can be achieved via upsampling augmentation techniques, whose performance we would like to compare and test (Synthetic Minority Oversampling Technique or geometric augmentation with OpenCV, Albumentations). The effectiveness of augmentation diversity can be evaluated with SSIM index.

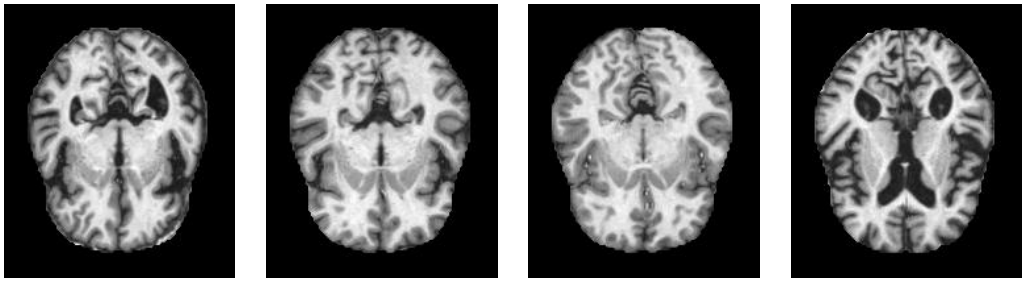
Lastly, we take an opportunity to compare performance of simple CNNs and VGG, ResNet, EfficientNet, DenseNet on the aforementioned classification task. This section would include building our own non-complex CNN (less than 20 layers) and working with tuning model layers (Batch, Dropout, Pooling) and hyper-parameters such as learning rate (or adaptive learning rate), early stopping, optimizers. Since it is a classification task, model performance would be evaluated on smoothed labels via Accuracy and AUC (TPR vs FPR) which is one of the most common metric in medical research.

By the end of this project we would expect to present the best working model with the benchmark on other configurations.

2.2 Alzheimer's Dataset

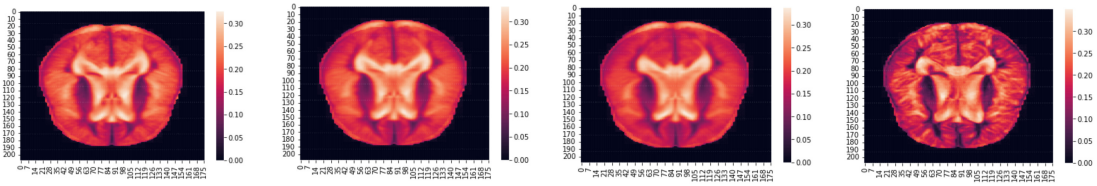
The data consists of MRI images. There are two files in dataset - Training and Testing both containing a total of around 5000 images each segregated into the severity of Alzheimer's. There are four classes:

1. Mild Demented
2. Very Mild Demented
3. Non Demented
4. Moderate Demeneted



(a) Mild Demented (b) Very Mild Demented (c) Non Demented (d) Moderate Demeneted

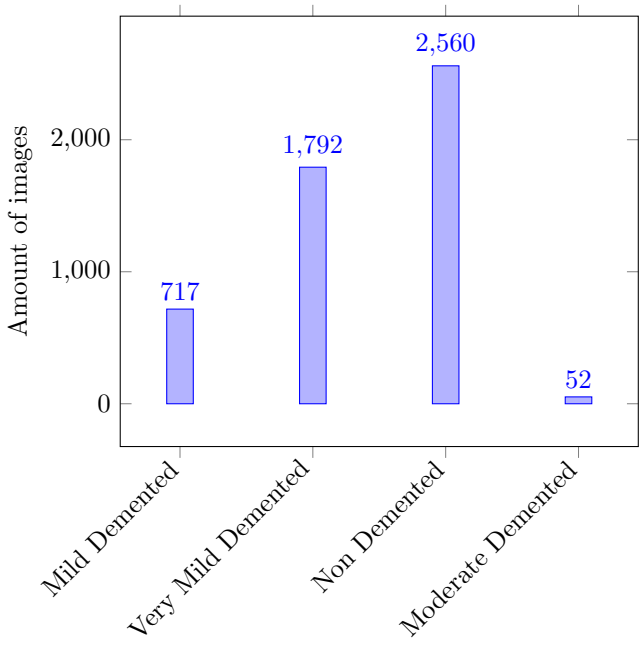
Figure 1: Alzheimer's disease severity classes



(a) Mild Demented (b) Very Mild Demented (c) Non Demented (d) Moderate Demeneted

Figure 2: Alzheimer's disease severity classes using heatmap

Mild Demented training set consists of 717 images, Very Mild Demented - 1792 images, Non Demented - 2560 images and Moderate Demented - 52 images. Each image has a resolution of 176×208 pixels.

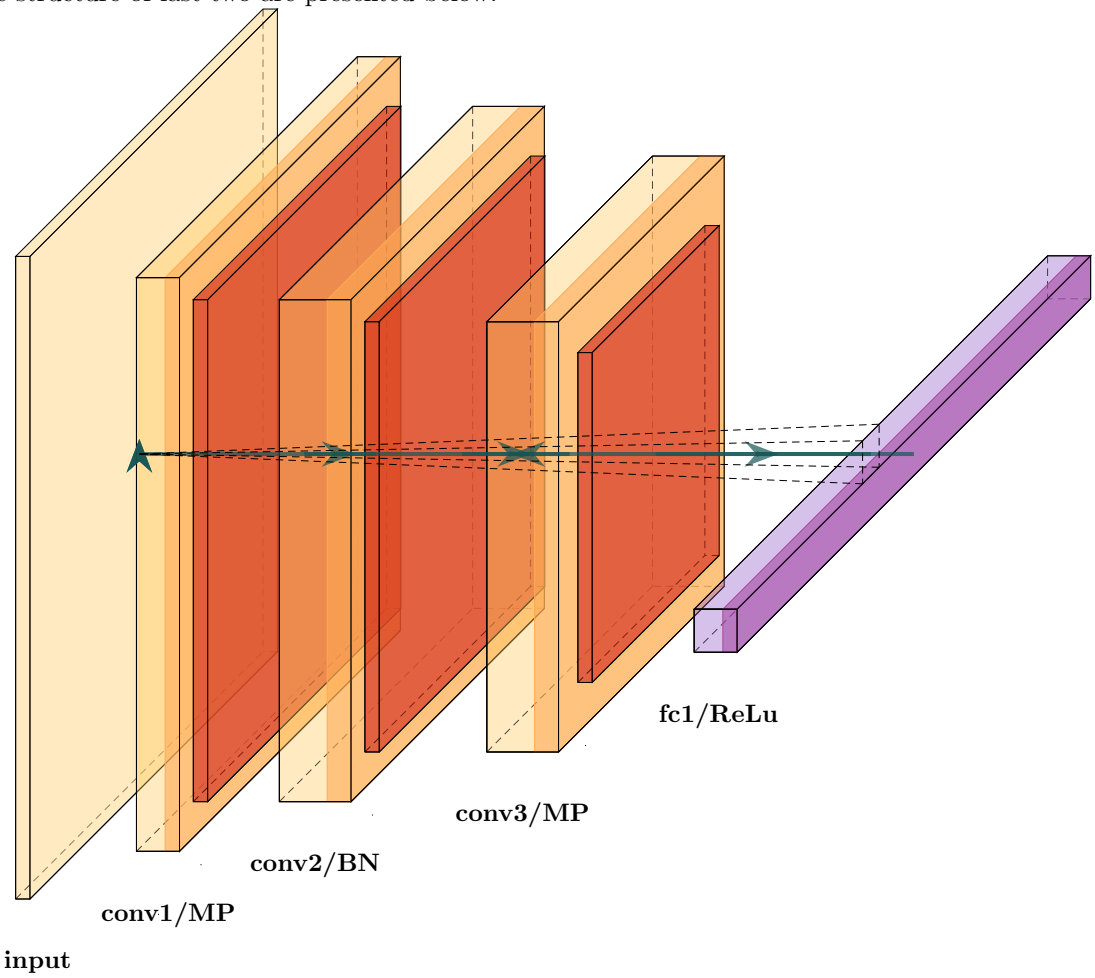


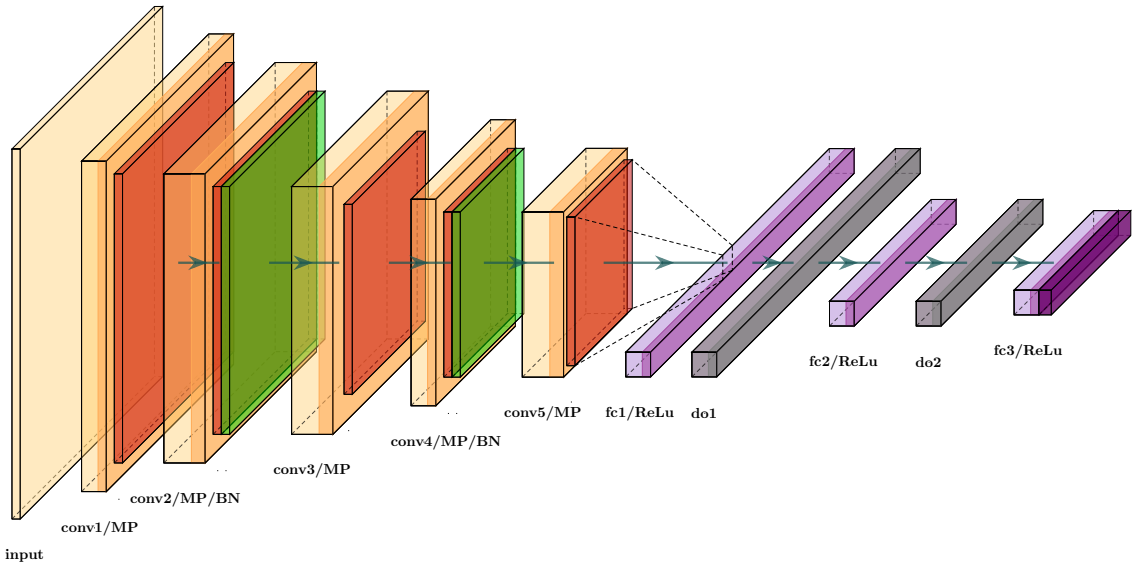
2.2.1 Preprocessing

As initial step for preprocessing we chose PyTorch transform function with 3 steps: Resize (for faster processing), ToTensor, Normalize.After that we loaded data from train and test (pre-splitted by Kaggle). As

2.3 Models

For our first iteration we wanted to look into models performance on the dataset. We chose pretrained state of art models: VGG, DenseNet, ResNet and created our own simple CNNs, namely, 3 and 5 layer CNNs. The structure of last two are presented below:





We run the models on the dataset for 15 epochs and evaluated after every epoch on the test set. Overall, each model was run 3 times for each of the optimizers. For the metric we use accuracy and Area Under Curve, which is a typical evaluation technique for medical data (accounts for TPR and FPR).

2.3.1 Results

Here is a preliminary table. The images were not augmented for now, and unbalanced distribution is kept. On the confusion matrices below, you can see that the best performing plain configuration is DenseNet for now. Anyway, we need take into account further model tuning.

	cnn3	cnn5	densenet	vgg	resnet
adam	56.22% (0.577)	64.19% (0.65)	77.17% (0.761)	56.68% (0.585)	65.36% (0.665)
sgd	56.37% (0.566)	71.85% (0.701)	66.69% (0.644)	76.23% (0.746)	71.62% (0.713)
rmsprop	55.9% (0.561)	59.97% (0.599)	76.08% (0.81)	58.64% (0.579)	62.31% (0.644)

Table 1: Results for CNNs Accuracy+AUC

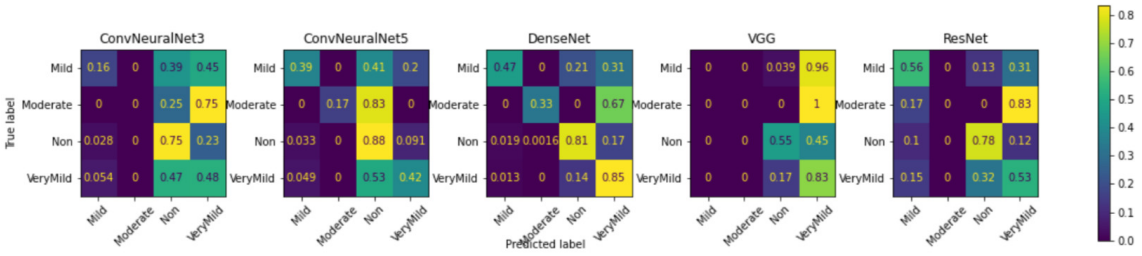


Figure 3: Confusion percentage matrices for set of models with adam optimizer

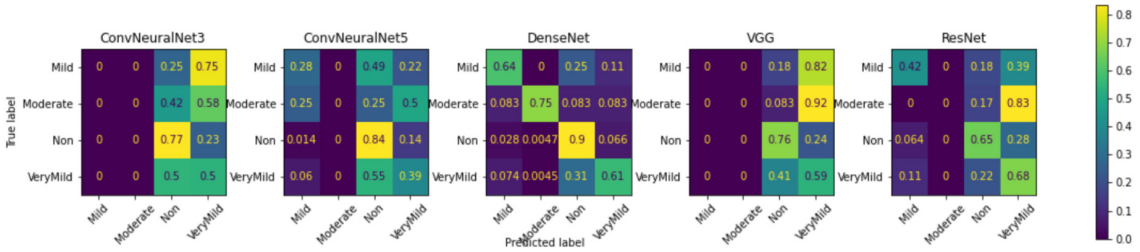


Figure 4: Confusion percentage matrices for set of models with rmsprop optimizer

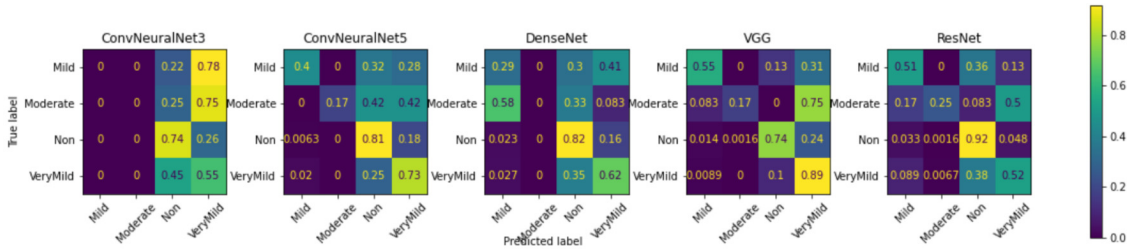


Figure 5: Confusion percentage matrices for set of models with sgd optimizer

Figure 6: Plots for RMSProp optimizer

2.4 Completed tasks

- Looked into the dataset

- Created simple CNNs
- Completed code for training
- Completed code for model evaluation
- Looked into pixel-wise variance of each class

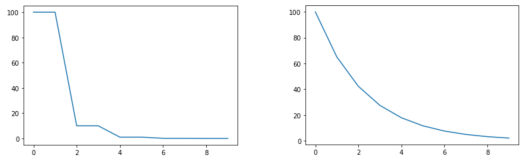
2.5 Future tasks

- Try to apply contrast, thresholding on the target images. In this way we would highlight main image feature
- Use weighted loss for unbalanced data on best model for each optimizer
- Use more epochs, but apply early stopping, since, as one can see on the plots, some models do not improve over time.
- Try out different learning rates (Step, exponential)
- Try weighted loss on the best model (DenseNet)
- Split train into train and validation

3 Deliverable 2

3.1 Adaptive Learning Rate

For our implementation of learning rates we decided to start with a simple LR approach, namely, StepLR and LambdaLR (not exponential). The main objective was to test which type of LR decay



(a) Step learning rate (b) Lambda learning rate

Figure 7: Learning rates representation

works better - smooth or steep. StepLR definitely decays faster, and allows the model not to "jump" inside the loss function. On the other hand, with such scheduler the model is less and less sensitive to any changes (slow learner). LambdaLR is mostly the opposite of the StepLR, that is why we decided to compare them. The implementation idea was taken from link

3.2 Early Stopping

We implemented early stopping, which stops the training, if there is no val improvement for more than 10 epochs.

After stopping, the best model is returned.

3.3 Weighted Loss

Since our classes a highly unbalanced and augmentation is the task for the next iteration. We wanted to see the impact of adding weights to our classes. Weights are splits of 1, where the class of least population has the highest coefficient. In this way, model's loss is higher if it mistakes the minority class.

This is important, because we deal with the medical data, and precision is vital for illness diagnosis.

3.4 Split Train and Val

We split train set into train and validation with the 80/20 rule. Thus 20% of images are in the validation set. dataloaders and train/evaluation for the PyTorch implementations were changed accordingly.

In the next section, we want to extract main image features though thresholding. Contrast was not used, as it will create even more noise to the image.

For the preprocessing we first blur the image with gaussian blur, then change image according the threshold algorithm (OTSU or AdaptiveGaussian) and after that we erode.

3.5 Otsu Thresholding

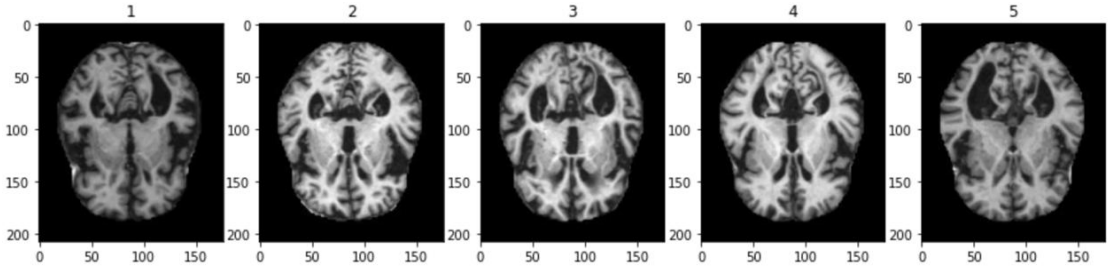


Figure 8: Mild Demented original images

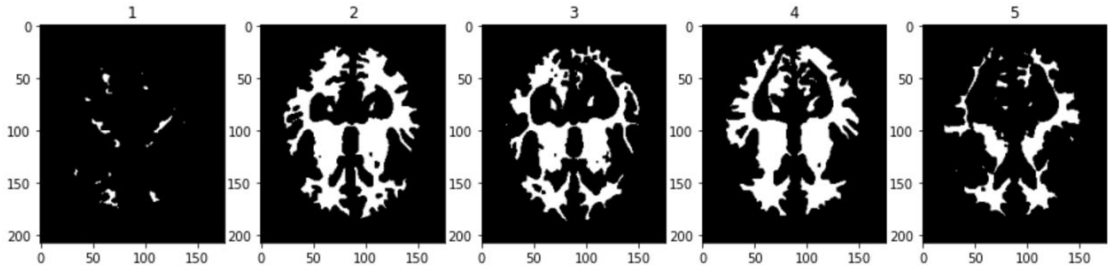


Figure 9: Mild Demented processed images

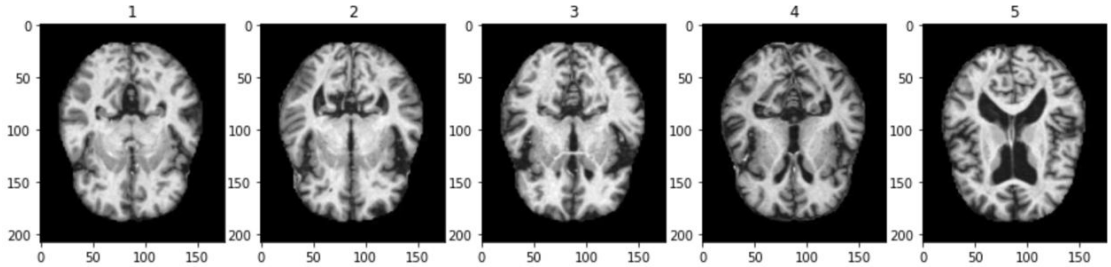


Figure 10: Very Mild Demented original images

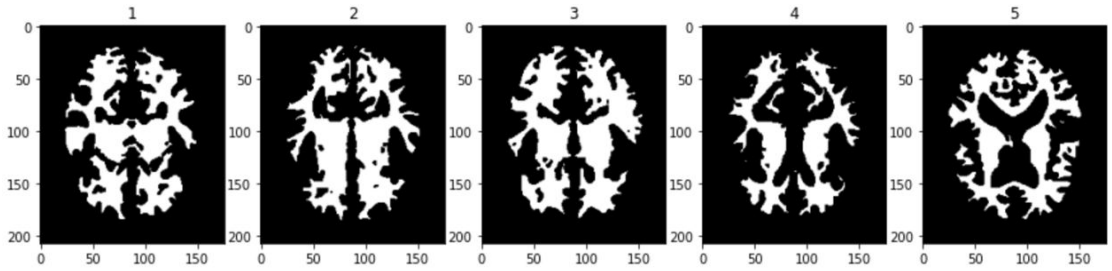


Figure 11: Very Mild Demented processed images

As you can see, Otsu algorithm produces quite nice results, however, it lacks adaptivenes for dimmed images: they appear to be total black and will break the task logic. To tackle this issue, we decided to try adaptive threshold.

3.6 Adaptive Gaussian Thresholding

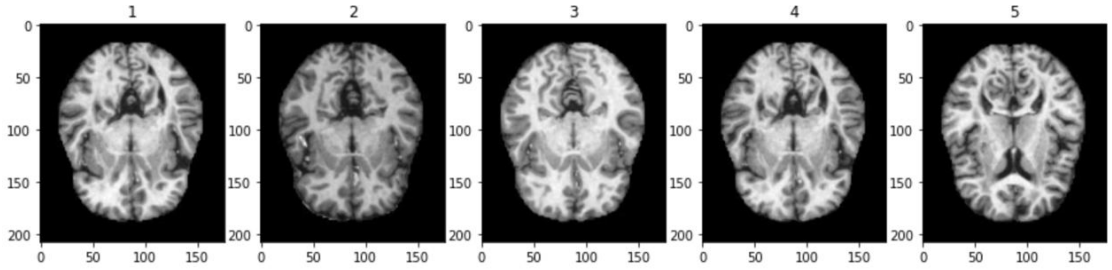


Figure 12: Mild Demented original images

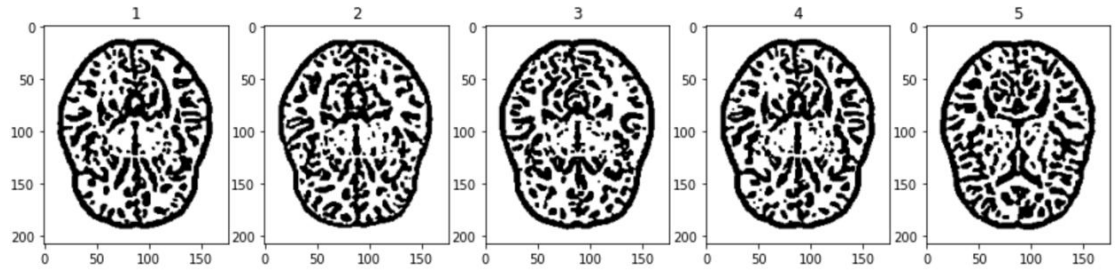


Figure 13: Mild Demented processed images

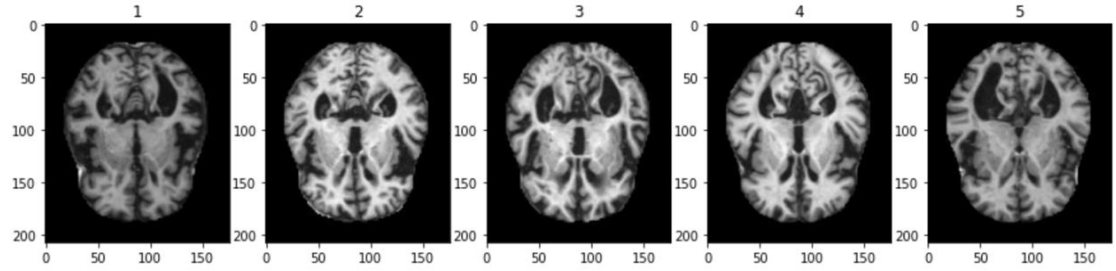


Figure 14: Very Mild Demented original images

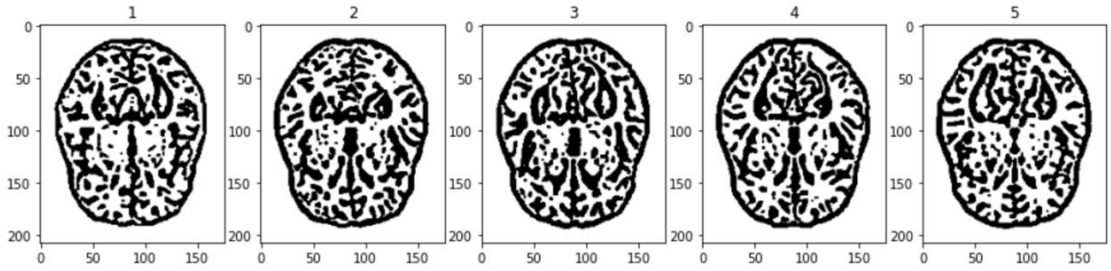
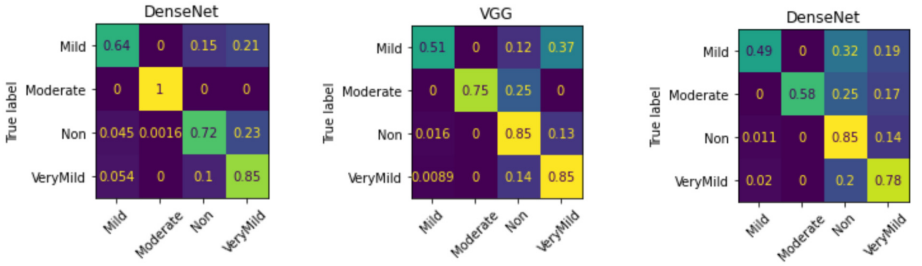


Figure 15: Very Mild Demented processed images

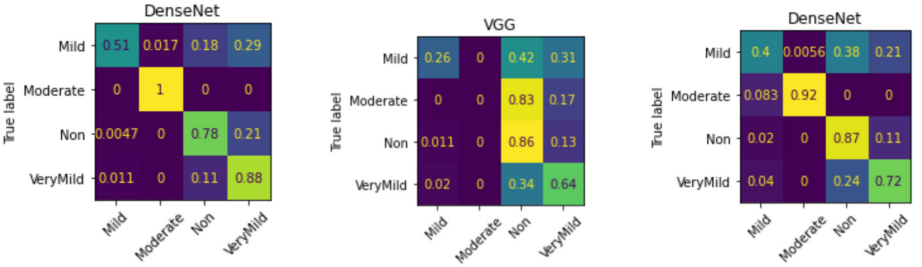
Adaptive thresholding is invariant to the image brightness conditions, it outlines the main features. Yet it still creates some noise. As we said, Otsu is more preferable, but we need to find the way to deal with low light images.

3.7 Training results



(a) DenseNet+Adam+WL (b) VGG+SGD+WL (c) DenseNet+rmsprop+WL

Figure 16: Confusion matrix for best models with StepLR (WL-weighted loss)



(a) DenseNet+Adam+WL (b) VGG+SGD+WL (c) DenseNet+rmsprop+WL

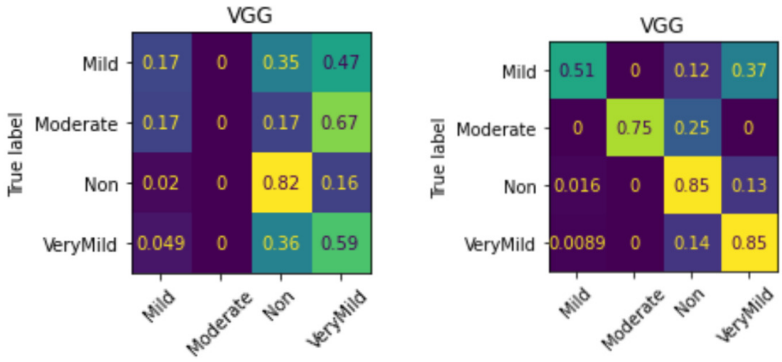
Figure 17: Confusion matrix for best models with LambdaLR (WL-weighted loss)

	StepLR	LambdaLR
densenet+adam+weighted loss	75.84% (0.854)	77.87% (0.85)
vgg+sgd+weighted loss	80.3% (0.829)	69.27% (0.652)
densenet+rmsprop+weighted loss	77.17% (0.787)	75.29% (0.81)

From the table above we can see, that our modifications have a positive impact on model performance (Accuracy, AUC) (**Dense+Adam** (+0.7%,+9%), **VGG+SGD** (+3.8%, 8.3%), **Dense+Rmsprop** (+1.09%, -4%)). Moreover,we can see that the Moderate prediction accuracy improved drastically (up to 100% in DenseNet).

If we compare learning rates, VGG performs much better with LambdaLR and others with StepLR.

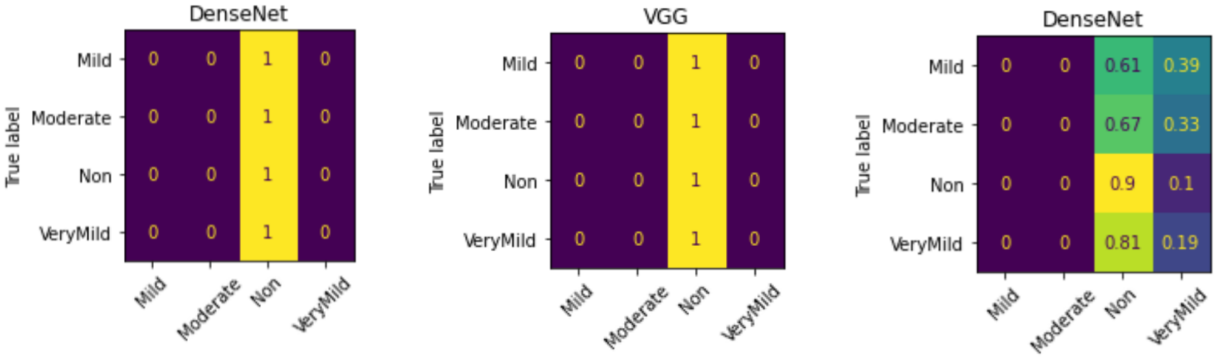
Perhaps, this is due to different model architecture (need to check that). For example, for DenseNet we initializated lr with 0.005, but for VGG this LR value lead to lower performance, but if we keep it at 0.01 at first - the result is stable and better.



(a) VGG + lr=0.005 (b) VGG + lr=0.01

Figure 18: VGG performance with different LR

Unfortunately, when we apply thresholding the accuracy drops, it is clear that GaussianThresholding did not give the desired image feature extraction, thus, we need to look into better way for the next delivery. The results are presented below:



(a) Dense+Adam+Thresholding (b) VGG+Thresholding (c) Dense+Rms+Thresholding

Figure 19: Thresholding applied

3.8 Completed tasks

- Split train into train and validation - *Marina*
- Applied thresholding to highlight main image features - *Vladimir*
- Used weighted loss for unbalanced data on best model for each optimizer - *Vladimir*
- Used more epochs, plus apply early stopping -*Marina*
- Tried out different learning rates (Step, Lambda) - *Marina, Vladimir*
- Created report - *Marina, Vladimir*

3.9 Future tasks

- Try to solve low light images problem
- Use augmentation to balance datasets (SMOTE or Grid/Elastic Distortion)
- Try to use InceptionV3 or EfficientNet
- Try to add dropout, batch normalization and pooling layers between FC(..., 1000) and FC(1000, num_classes) in pre-trained models