AI Final Project
Project 4 : Glass Detection

1st Elkhan Imanov
Computer engineering (21.1)
elkhan.imanov.std@bhos.edu.az

2nd Nurlan Pirizade
Computer engineering (21.1)
nurlan.pirizade.std@bhos.edu.az

3rd Ilham Muradov
Computer engineering (21.2)
ilham.muradov.std@bhos.edu.az

I. Problem definition and Introduction

The task was to create diffetent CNN models that can correctly classify images of people into two categories: with glasses and not. The dataset from Kaggle was used for training and testing models. The dataset contains two subfolders: glass and no glass. Each subfolder contains slightly more than 2,000 images of people, wearing glasses and not. So, there is no significant class imbalance in the dataset used.

Fig. 1: Example Image

II. Data Preprocessing

The first stage was spliiting dataset into train, validation, and test. In other words, the following folder structure should be used:

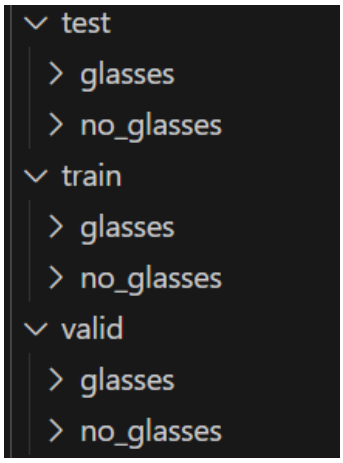

Fig. 2: Split structure

The proportion is: train - 0.7, validation - 0.2, test - 0.1. To split the data without leakage the script **"datasets / preprocessing.py"** was used. In that script the resizing of images was also implented. Before saving an image into the corresping folder, the image is resized to 224 x 224 pixels. This size was applied to each image.

III. Custom dataset and Data Augmentation

The **"dataset retrival.py"** script was used to extract data with labels. Then, the data was given to Data Loader to be divided into batches. One of the parameters of custom dataset function is transformer which will be applied to images. In our case, on fly augmentation was used. The augmentation which we used for train dataset was random rotation, random horizontal flipping, and random color jitter. However, for validation and testing the transformer only includes to tensor, resizing, and normalization.

Fig. 3: PyTorch transformer

The idea of using augmentation is to modify images randomly in the training stage. In this way, our model will learn better and be more robust to noises, rotations, or scaling while making predictions on unseen data. Since real-world data is not so clean, it is important to use this technique.

IV. Components

In the project, we used the following components in each combination.

A. Resnet18 and VGG16

Resnet18 and VGG16 CNN models(pretrained and not pretrained) were used. Both architectures contain a number of convolutional layers with pooling and dense layers. However, Resnet is generally more efficient as it has 10 times less parameters and uses skip connections to avoid vanishing gradient problem.

B. Cross Entropy loss function

In all machine learning problems, we have a loss function which is always minimized by finding gradients and then weights are updated. In our project Cross Entropy Loss was used. It measures the difference between the predicted

probability of the model and the actual (true) probability of the labels.

C. SGD and ADAM

Optimizers define the way how our loss function will be minimized and weights are updated in each iteration. The working principle are the same for all of them. Using back-propagation, the gradients are calculated and then parameters are updated. Adam is different from SGD in that it adapts learing rates for each parameter, it usually has faster convergence. Depending on the dataset and making experiments, it can be decided which one is better to use.

While training of the models learning rate of 0.01 and batch size of 32 was used in almost all cases except one.

V. Training and Validation

After preprocessing, splitting, and defining transformers for each split, we started training process with max 15 epochs for each model. To visualize the results on a graph tensorboard was used. Tensorboard was launched in a different terminal using the following command : **tensorboard –logdir runs/**. For the simplicity, the graphs of the same model but with different optimizers was combined in one window in the images below.

1. The first one is pretrained Resnet18 model with both optimizers(sgd and adam). Pink line - SGD, Yellow line - Adam.

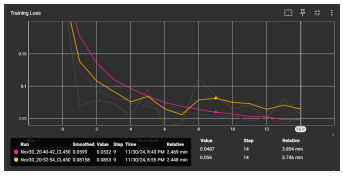

Fig. 4: Resnet pretrained trainig loss

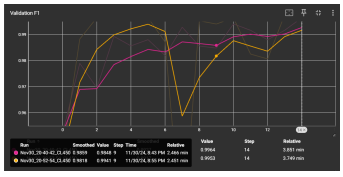

Fig. 5: Resnet pretrained validation F1

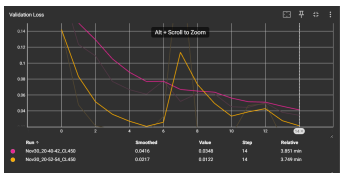

Fig. 6: Resnet pretrained validation loss

2. The second is Resnet18 with pretrained equals to False, which means without predefined weights. Gray line - SGD, Blue line - Adam.

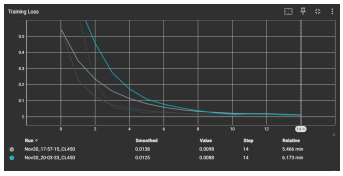

Fig. 7: Resnet training loss

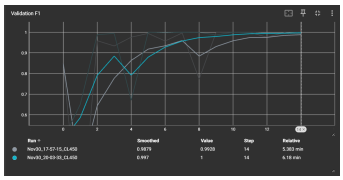

Fig. 8: Resnet validation F1

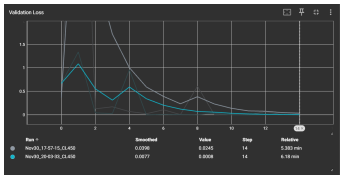

Fig. 9: Resnet validation loss

3. VGG16 pretrained model with both optimizers. Purple line - SGD, Green line - Adam.

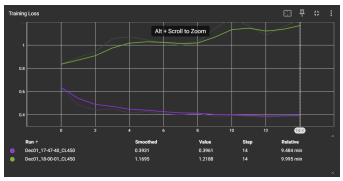

Fig. 10: VGG pretrained training loss

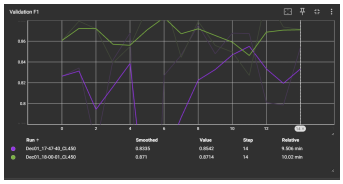

Fig. 11: VGG pretrained validation F1

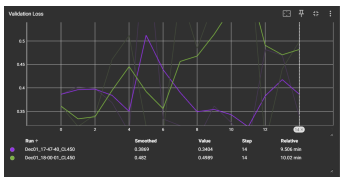

Fig. 12: VGG pretrained validation loss

4. VGG16 model with pretrained False. Orange line - SGD, Blue line - Adam.

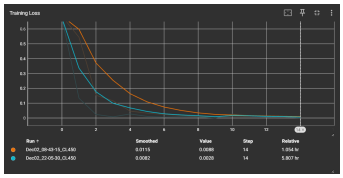

Fig. 13: VGG training loss

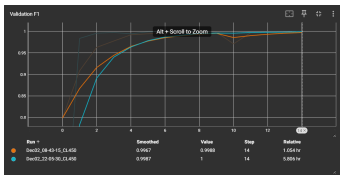

Fig. 14: VGG validation f1

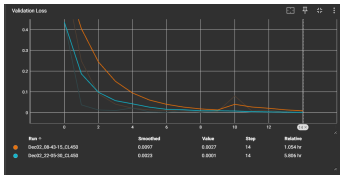

Fig. 15: VGG validation loss

The loss and f1 score for VGG with Adam optimizer initially was very poor. So, after some expirementations we came up with optimal learning rate of 0.0001 and batch size of 128. Lower learning rate in this case helped to prevent overshooting of optimal weights and bigger batch size, we assume, resulted in better generalization.

As it can be seen, all combinations of Resnet and VGG architecture with both optimizers performed great results on validation data set with high f1 scores. Also, low validation loss with high f1 score indicates that model is confident about its predictions. Since model gives probability to each class, it is crucial to consider both.

Also, VGG16 with pretrained weights gave little lower results compared to others, but generally pretrained models should give better results than its not pretrained version. So, we think that this is due to hyperparameters and with more time spent on fine tuning it is possible to find optimal hyperparameters and increase the results.

VI. Test Results

After training of each model, its accuracy was calculated on test split, with ”test.py” script. The results are following:

Model	Optimizer	Loss	Accuracy F1	Classwise accuracy
Resnet	SGD	CE	0.98	Class 0 : 1 ; Class 1 : 0.97
Resnet	Adam	CE	0.99	Class 0 : 0.99 ; Class 1 : 1
Resnet, pre	SGD	CE	0.99	Class 0 : 0.98 ; Class 1 : 1
Resnet, pre	Adam	CE	0.98	Class 0 : 0.97 ; Class 1 : 1
VGG	SGD	CE	0.99	Class 0 : 0.99 ; Class 1 : 1
VGG	Adam	CE	1	Class 0 : 1 ; Class 1 : 1
VGG, pre	SGD	CE	0.85	Class 0 : 0.7 ; Class 1 : 0.98
VGG, pre	Adam	CE	0.82	Class 0 : 0.76 ; Class 1 : 0.87

Fig. 16: Test results of the models

The results for all models with different optimizers are quite high. One of the reasons of such high results is the number of classes(2). Also, another reason is good data for training. If we look at the dataset, we will see that it is not hard to differentiate between them, and the data is quite clean. However, in real-world problems, time and complexity involved in preprocessing stage is much more. So, with the right approach to this project it is possible to achieve high accuracy.