HW1

Einan Regev - 059238717, Moshe Baker - 032090375

General

The assignment is to build a classifier neural network that decides whether a person in an image presented to it, is properly covered by a face mask (class 1) or not (class 0).

The given data is a collection of labeled images, split into a training sent and a data set.

We implemented the classifier using a convolutional neural network model.

Our model, when trained by the training set and tested on the test set achieved an F1 score of 97.2%.

1. Exploratory data analysis

(a) A small set of typical images is shown below:

(b) From looking at the data, it seems that all pictures contain faces, viewed from the front or side, with different levels of lighting, contrast, coloring, and resolution. The image sizes are spread over a large variation.
Base on these observations we conclude:
- There is no reason to try to train the model to identify faces (since all data contains faces)
- There is no need to train the model to identify different rotations (since all faces are in an upright position and shown from the front or angle)
- Augmentation using flip, blur and color removing may work.

2. Experiments

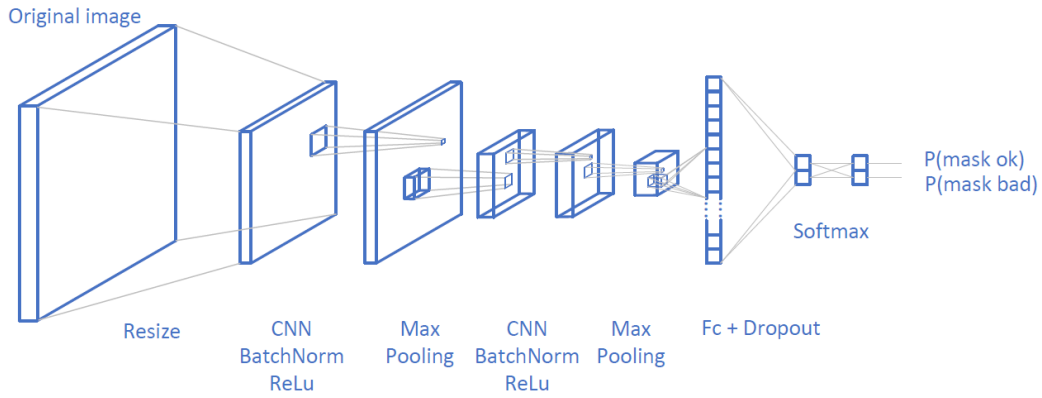
(a) We have implemented a pre-processor for augmentation that can be used as an option. The pre-processor code is in the file **augment.py**.
The pre-processor augments the training data set by reading all image files one by one, and generating augmented images from each one of the original images. The following 3 augmented set is generated from each image:
- The original image
- A H-flipped (left-right mirroring) version of the original image
- A Blurred (gaussian blur) version of the original image

After augmentation, the files from the augmented set are copied into a new folder at ./augmented/ so that the original data folder does not change. This allows changing the

training set between augmented and original in a simple manner. We have seen that using the augmented set improves the F1 score by **1-2 percentage points** in most experiments.

(b) Architecture: We have used a convolutional network architecture, which is largely based on the example code that was shown in the class. Our assumption is that due to the small number of classes (2) and the fact that the images are relatively homogenic the network should not be very deep, and the number of parameters should be kept low.

The following picture depicts the CNN model architecture:



Model parameters (total number of parameters for optimal configuration: 23362):

		Parameters
Sequential 1	Conv2d	In: 3 Out: 16 Kernel 5x5
	BatchNorm2d	16
	Activation	ReLu
	MaxPool2d	2
Sequential 2	Conv2d	In: 16 Out: 32 Kernel 5x5
	BatchNorm2d	32
	Activation	ReLu
	MaxPool2d	2
Dropout		0.6
Fc		
Activation+Loss	-Log+Softmax	

Model hyper-parameters:

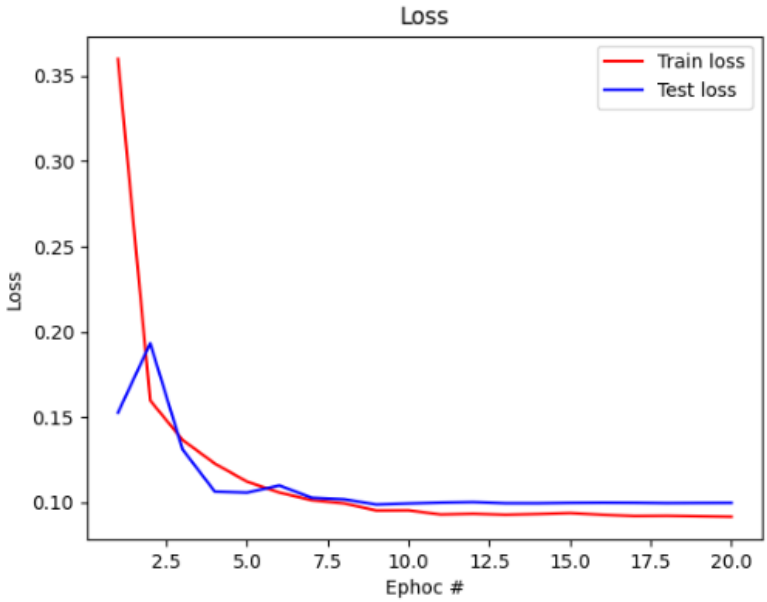
	Range tested	Final values used
Number of epochs	5-20	20
Augmentation	Used/Not used	Used
Batch size	64, 100, 200	100
Learning rate	0.001 – 0.05	0.01
Image dimension	32x32 – 128x128	48x48
Kernel size	3x3, 5x5, 7x7	5x5
Dropout rate	0.4, 0.6, 0.8	0.6
Optimizer	Adam, AdamW	Adam
Scheduler	LambdaLR	LambdaLR (lambda = 0.6)

(c) Loss function: The loss function used was NLLLoss (Negative log) which when combined with SoftMax is equivalent to minimizing cross entropy.

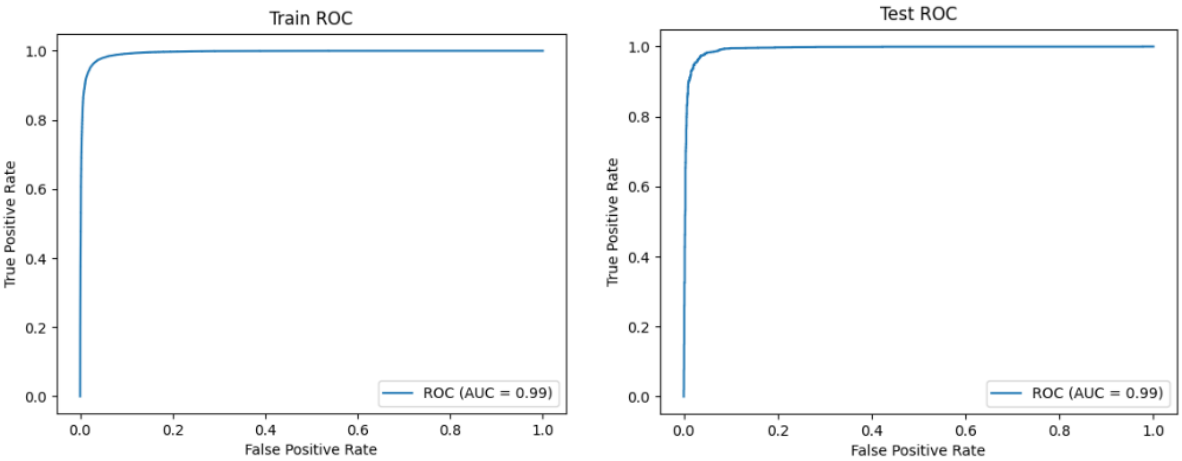
(d) Optimizer: We used the Adam optimizer with an adaptive learning rate that decreases by a factor of 0.6 after each epoch.

(e) Regularization was achieved through Batch normalization (after each convolutional layer), max pooling (between sequential layers) and by using dropout prior to the final fully connected layer. Also the use of augmentation (as explained in section 2 above) can be considered a type of regularization.

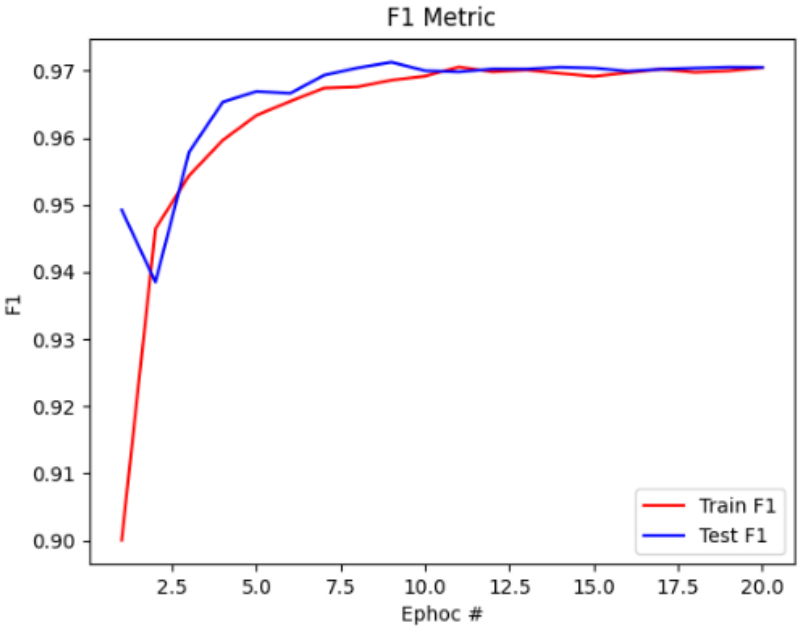
(f) Train and test loss progress as a function of training time can be seen in the figure below.



(g) Train and test ROC and AUC graph can be seen in the figure below.



(h) Train and test F1 graph can be seen in the figure below.



(i) Conclusions:

After running various combinations of the hyper-parameters, we have set the parameter values as shown in the hyper-parameters table in section (b) above. We have observed that using augmentation is adding a significant performance gain (about 1%-2% increase in F1).

Using an image size of more than 48x48 did not improve performance, and adding another sequential layer had no impact o performance therefore we have stayed with a compact architecture of 2 sequential CNN layers only, with a total of 23362 parameters.

We have set the number of epochs to 20 and tracked loss and F1 in each epoch. The final model saved after training was the model that achieved the best F1 during the 20 epochs (and not the last epoch model)

The final score (F1) on test data is slightly over 97% which seems like a good performance for such a compact network.