Analysis & Data Presentation Laboratory – 094295
Igor Drozdov 317336451
Shir Lissak 203507207

COVID-19: Face Mask Recognition

Homework 2

The following report will present a comprehensive discussion about the process of predicting whether a person in a frame is wearing a mask correctly.

GitHub Repository

https://github.com/IgorDroz/COVID-19-Face-Mask-Recognition

EDA

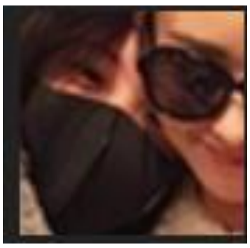
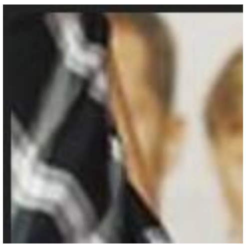
<u>Raw Data</u>

The dataset consists of jpg files of faces of people from variety of ethnics, ages and genders and different angles. The jpg files are in different sizes (avg ~128x128). Some of the frames hold distortions:


Some of the frames present a mask doesn't cover the face properly:



Some frames are ambivalent or uncertain:

The test set consist of 6092 frames:

- 3295 frames with mask (1)

- 2797 frames of faces without a mask (0)

The Training set consists of 18,259 frames:

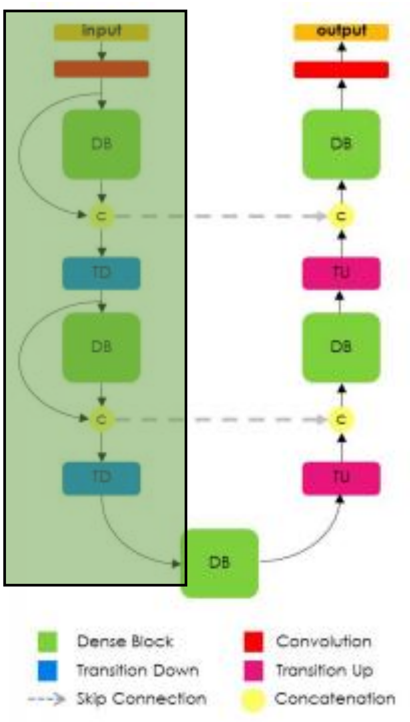
- 9947 frames with mask (1)
- 8311 frames of faces without a mask (0)

Experiments

We implemented two different models.

<u>Experiment 1 – self-implemented “Tiramisu” based model [1]</u>

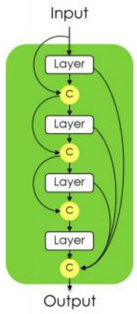
We have got an inspiration from this [1] article presents a U-Net model with a backbone of densenet that originally used to tackle segmentation tasks. We implemented only the encoder of the U net and added a fully connected layer and then to a sigmoid layer.



Architecture
Input, $m = 3$
3×3 Convolution, $m = 48$
DB (4 layers) + TD, $m = 112$
DB (5 layers) + TD, $m = 192$
DB (7 layers) + TD, $m = 304$
DB (10 layers) + TD, $m = 464$
DB (12 layers) + TD, $m = 656$
DB (15 layers), $m = 896$
TU + DB (12 layers), $m = 1088$
TU + DB (10 layers), $m = 816$
TU + DB (7 layers), $m = 578$
TU + DB (5 layers), $m = 384$
TU + DB (4 layers), $m = 256$
1×1 Convolution, $m = c$
Softmax

Figure 1 - Tiramisu model with frame on the implemented part. The last layer of the framed part was connected to a fully connected layeer and then to a sigmoid layer.

On each dense block (DB) we performed the following architecture:



- Data loading and preprocessing (data augmentation - if any) –
    - resizing all frames to 64x64
    - augmentation – rotation
    - built data loader that for local interface
    - performed data normalization
- Architecture (describe fully and/or refer to original paper and code) – as described above
- Loss Function – BCELoss (binary cross entropy)
- Optimizers – Adam (also tried SGD)
- Regularization – batch normalization
- Adaptive learning rate

Experiment 2 – GitHub densely connected CNN [2]

In our second experiment we used a GitHub code that implements this [2] article. A classic dense net CNN with the following architecture:

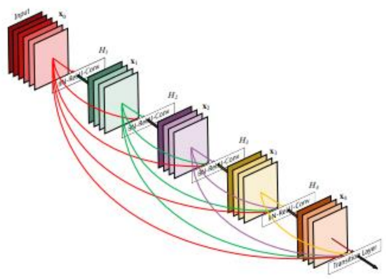


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

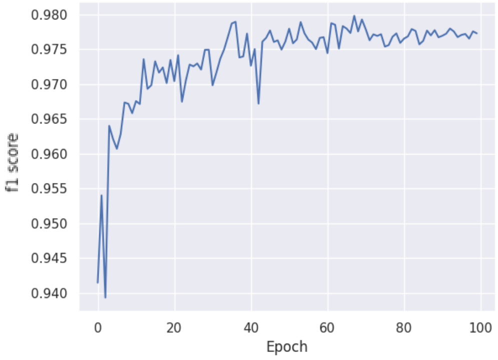
Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

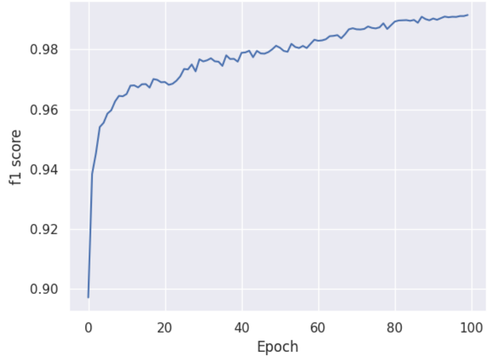
Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown table corresponds the sequence BN-ReLU-Conv.

- The only difference from the last model is the architecture itself. All the other parameters are the same for both experiments.

Graphs

Experiment 1 – “Tiramisu” (self-implement)





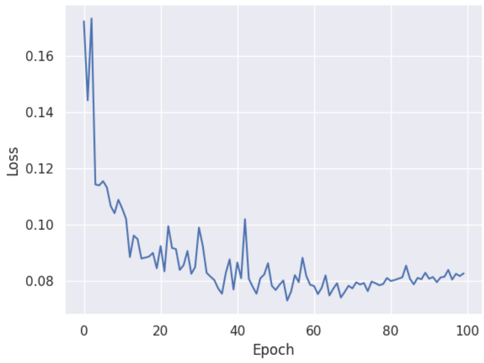


Figure 3 - Test Loss

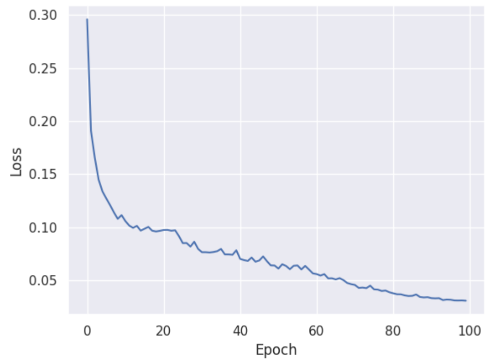


Figure 4 - Train Loss

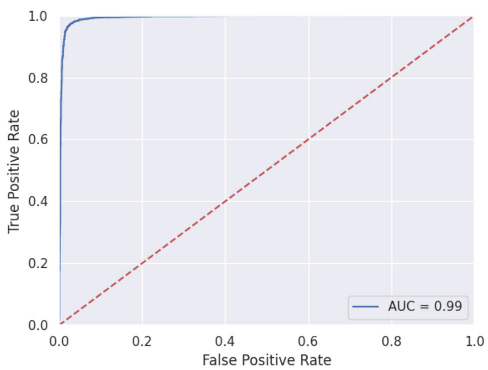


Figure 5 - Test ROC Curve

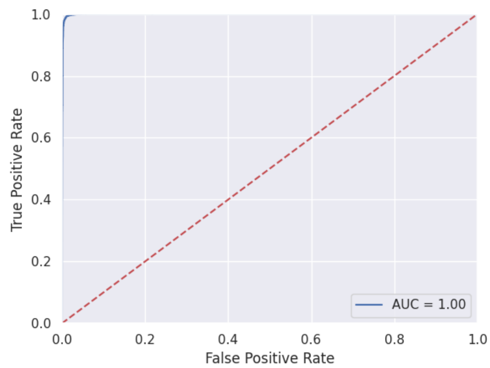


Figure 6 - Train Roc Curve

In those graphs we can see that the that the train is smoother and more monotonic then the test graphs.

Experiment 2 – DenseNet (GitHub)

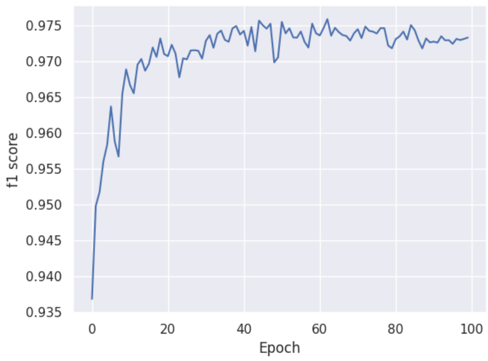


Figure 7 - Test F1

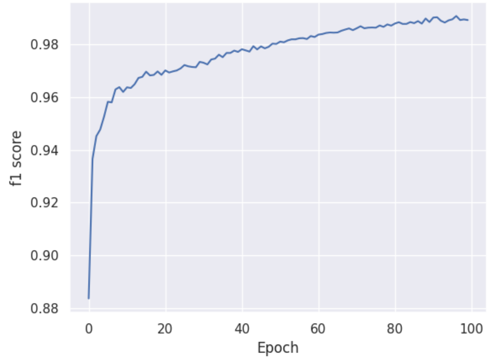


Figure 8 - Train F1

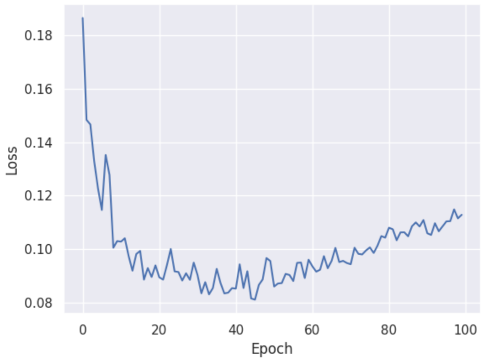


Figure 9 - Test Loss

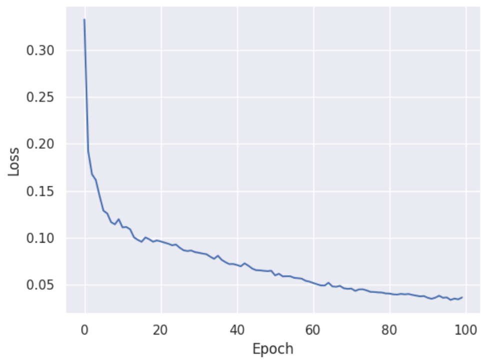


Figure 10 - Train Loss

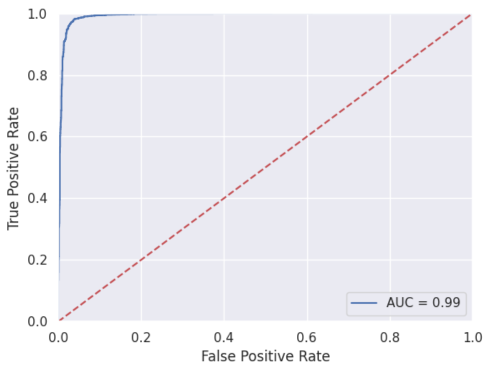


Figure 11 - Test ROC Curve

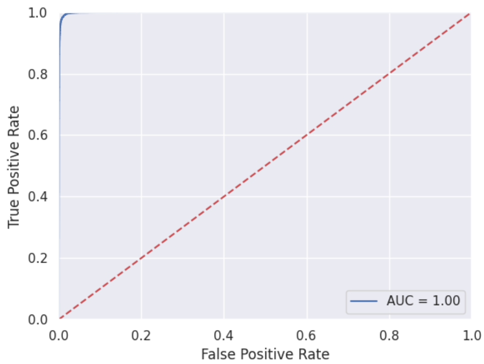


Figure 12 - Train ROC Curve

In those graphs we can see a bit of over fiiting – in figure 9 and 10 we can see that the loss of the trainig decreases while the loss of the test increases by the end of the training.

Results

Experiment1 “Tiramisu” based model F1 result – 0.977

Experiment2 DenseNet F1 result – 0.974

The “Tiramisu” (self implemented) model outperformed the DenseNet architectue (GitHub) by a small gap of less than 1%.

A bit about the process of choosing the model during the training:

We initially divided the ‘train’ images to validation and train (15% and 85%). During the epochs we saved the best model in terms of F1 score result on the validation in order to prevent overfit scenerio.

Bibliography

1. Simon Jegou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Workshop on Computer Vision in Vehicle Technology CVPRW*, 2017. https://arxiv.org/pdf/1611.09326.pdf
2. G. Huang, Z. Liu, K. Q. Weinberger, and L. Maaten. Densely connected convolutional networks. In *CVPR*, 2017. https://arxiv.org/pdf/1608.06993.pdf