POLITECNICO
MILANO 1863

Report of Homework 1 - Artificial Neural Networks and Deep Learning 2020/2021

Tommaso Fontana Daniele Comi

22/11/2020

1 Introduction

In this first homework we had to solve a classification problem, we have to classify images depicting groups of people based on the number of masked people. The classifier is meant to discriminate between images depending on the following cases illustrated in Figure 1.

No one is wearing a mask

All the people are wearing a mask

Some people are wearing a mask

Figure 1: Example of the tree classes that the model has to learn to recognize.

2 Dataset handling

The dataset available is constituted of roughly 5000 images and its classes are balanced. The dataset was then augmented by means of ImageDataGenerator class where we choose augmentation values (see Table 1) empirically based on the result of the model on the validation set.

While the RGB images were of an original dimension of about 640 x 412, we choose to handle 256 x 256 RGB images, so we can use a smaller model which we can actually train on our limited hardware, in batches of 16 (this is the biggest value that fit in our available GPU).

Table 1: Parameters used for the augmenting process

Augmentation Method	Parameter1	Parameter2
rotation_range	10 degrees	
width_shift_range	0.25	
height_shift_range	0.25	
zoom_range	0.1	
shear_range	0.2	
random_hue	0.1	
random_brightness	0.2	
random_saturation	0.8	1.2
random_contrast	0.8	1.2
horizontal_filp	True	
vertical_filp	False	
horizontal_filp	True	
rescale	1/255	
fill_mode	”constant”	

3 Model Manual Tuning

To better evaluate the performance of the model without biases we split the dataset into training set and validation set using the StratifiedShuffleSplit. The method we chose guarantees that the training and validation sets have the same balancing of classes which should result in a more realistic evaluation. We chose a single 90% - 10% splitting ratio for training and validation due to the limited computing power we have. Once the the model had been evaluated we re-trained the model using the whole dataset.

Having better hardware and time we could have used multiple holdouts, K-fold, cross-validation or Monte-Carlo Cross-validation to have a statistically significant evaluation of the performance of the model. Furthermore, having more resources we could adopt Hyper-parameters optimization strategies such as Grid-search and Bayesian Optimization which could improve the performance of the model but requires massive computational power.

4 Model Choice

The first architecture used was a Convolutionary Neural Network of 3 hidden layers using a 3x3 filter size which had subpar performance even with the usage of some regularization technique and optimization technique such as Dropout, after having reached the best result we think we could with this model we moved to another approach. The last model we experimented with can be seen in Figure 2.

We approached Transfer Learning using fine-tuning, we choose 3 Model, VGG19, ResNet50V2 and InceptionResNetV2. These models were chosen for the following reasons: VGG19 is the one we used during the exercitations, ResNet50V2 and InceptionResNetV2 have both residual connections and Batch Normalizzation which should allow to train faster deeper models and in the past we used them in other contexts and they yielded good performance.

The best architecture, of the ones we tried, is InceptionResNetV2 model with 164 layers and its Inception modules. As mentioned before, we used fine-tuning so we trained the whole network starting from the pre-learnt features of InceptionResNetV2 from the training on the ImageNet dataset. Then we added at the top a Flattening layer and some dense layers that perform the classification based on the features extracted form the CNN model.

```
i = h = Input(shape=IMAGE_SHAPE)

h = Conv2D(64, kernel_size=(6, 6), activation="relu")(h)
h = Conv2D(32, kernel_size=(3, 3), activation="relu")(h)
h = Conv2D(32, kernel_size=(3, 3), activation="relu")(h)
h = MaxPooling2D()(h)
h = Conv2D(32, kernel_size=(3, 3), activation="relu")(h)
h = MaxPooling2D()(h)
h = Conv2D(16, kernel_size=(3, 3), activation="relu")(h)
h = MaxPooling2D()(h)

h = Flatten()(h)
h = Dense(32, activation="relu")(h)
h = Dense(32, activation="relu")(h)
h = Dense(32, activation="relu")(h)
h = Dropout(0.2)(h)
h = Dense(10, activation="relu")(h)
output = Dense(
    3,
    activation="softmax",
)(h)

model = Model(i, output)
```

Figure 2: The last model we tried before transfer learning

5 Architecture of the final model

This top model is composed by 3 Dense layers of 100, 10, and 3 neurons respectively,and ReLU as activation function. We chose to apply Batch-Normalization to these layers to improve the regularization, particularly the robustness to co-variate shift. Moreover, on top of these layers we added Dropout with 0.5 and 0.2 rates to further regularize the model and prevent overfitting. These values showed to perform better.

The actual model described before can be seen in the below figure (Figure 3).

```
from tensorflow.keras.applications import InceptionResNetV2

truncated = InceptionResNetV2(
    input_shape=IMAGE_SHAPE,
    include_top=False,
    weights="imagenet"
)

i = truncated.input
h = truncated.output

h = Flatten()(h)
h = Dense(100, activation="linear",
    kernel_initializer=tf.keras.initializers.GlorotNormal()
)(h)
h = Dropout(0.5)(h)
h = BatchNormalization()(h)
h = Activation("relu")(h)
h = Dense(10, activation="linear",
    kernel_initializer=tf.keras.initializers.GlorotNormal()
)(h)
h = BatchNormalization()(h)
h = Activation("relu")(h)
h = Dropout(0.2)(h)
output = Dense(3, activation="softmax",
    kernel_initializer=tf.keras.initializers.GlorotNormal()
)(h)

model = Model(i, output)
```

Figure 3: The model generation code

The final model has 59,807,155 trainable parameters and has 167 layers.

6 Training

The training parameters are summarized in Table 2.

Table 2: Training parameters

Training argument	Parameter	Reason
loss	categorical crossentropy	It's a multi-label classification
steps_per_epochs	512	using a batch-size of 16 it should be approximately the size of the training set
validation_steps	32	using a batch-size of 16 it should be approximately the size of the validation set
Weight initialization	Xavier	To have better starting weights which should speed up the training and let us have a better final result of gradient descent
optimizer	Nadam	Dozat (the author) says that is should give better results than Adam which is the de-facto standard
learning_rate	0.001	Value suggested in the paper
beta_1	0.9	Value suggested in the paper
beta_2	0.999	Value suggested in the paper
Callback	Early Stopping	To stop the model when it reaches convergence and it act as weight decay which help preventing over-fit.
delta	0.001	A small enough value, it's $\approx \frac{1}{100}$ of the final loss of the models
patience	10	A patience big enough that the RLROP can act at least 3 times before stopping.
restoring best weights	True / False	True when we have a validation set, False otherwise to don't over-fit on the train data
Callback	RLROP	Reduce Learning Rate On Plateau, this should help the model achieve better performance in the last part of the training.
RLROP factor	0.1	This value was taken from an example.
RLROP patience	3	We set the patience so that it can trigger 3 times before the early stopping kills the process
RLROP delta	0.001	The same delta of the early stopping

For the weight initialization with Xavier Initialization we used GlorotNormal to better initialize weights W and letting backpropagation algorithm start in advantage position, given that final result of gradient descent is affected by weights initialization.

$$W \sim \mathcal{N}\left(\mu = 0,\ \sigma^2 = \frac{2}{N_{in} + N_{out}}\right)$$

We also set a checkpoint folder in order to save our model state during the training process after each epoch.
For reproducible result we set a constant seed in order to have the same pseudo-randomization on data by TensorFlow and by Numpy. The chosen seed was randomly 0xc0febabe.

7 Results

With the first architecture we were able to reach on Kaggle an accuracy on the test dataset of 84%, while with the second architecture we've reached an accuracy on the test set of 92.66% using validation and losing between validation and test accuracy only 0.31%, so it's a quite good model.
Then re-training on the whole dataset using the same model we've reached an accuaracy on the test set of 94.44%.

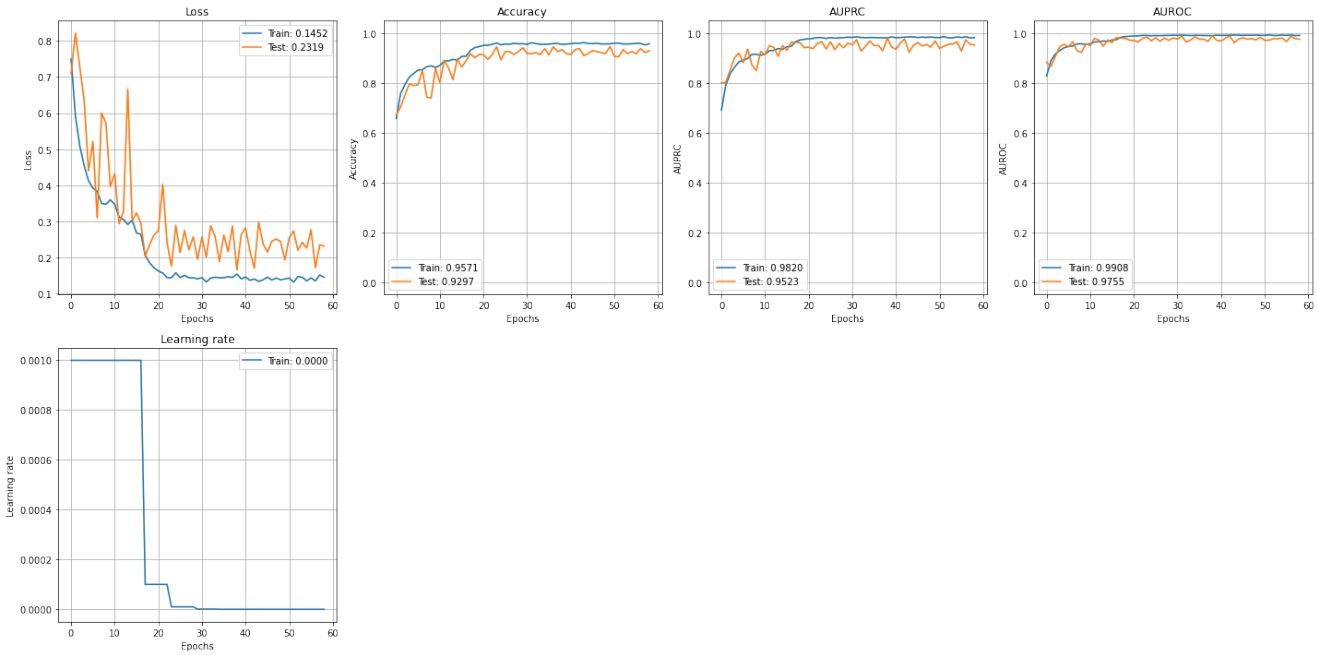

Figure 4: The training results of the run with validation set. We can see that the model does not over-fit much.

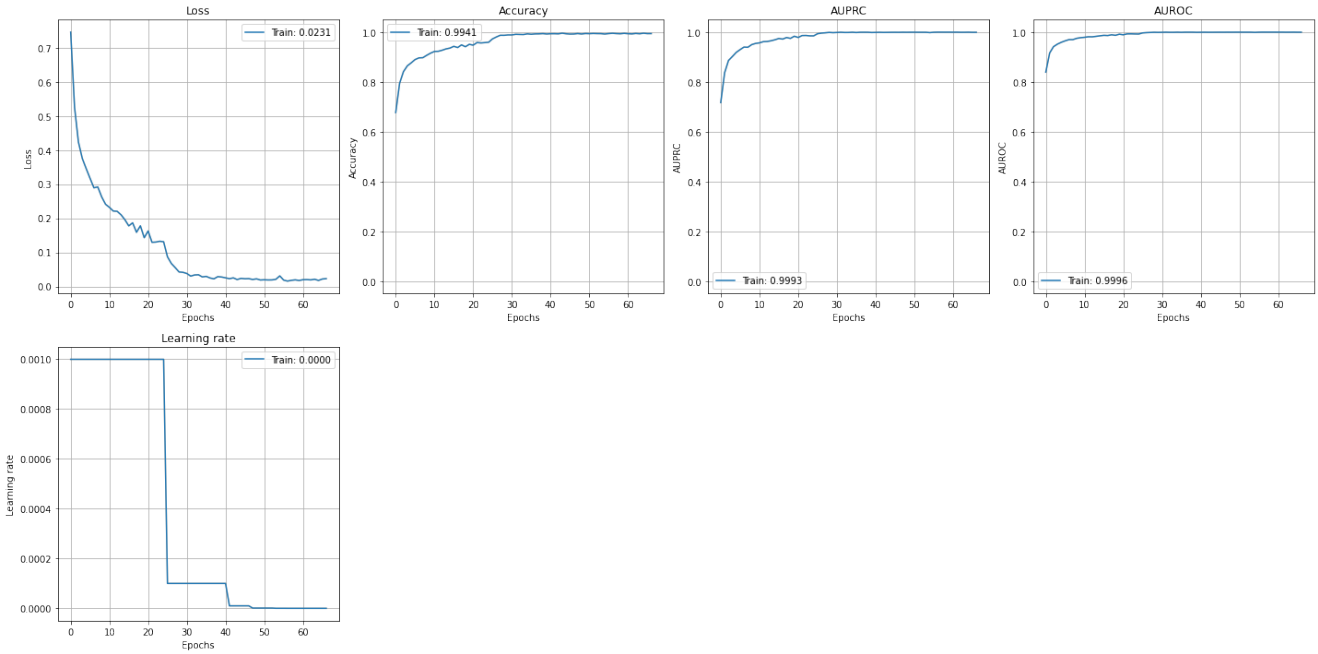

Figure 5: The training results of the whole dataset.

8 Conclusions

We've come to the conclusion that such Deep Learning and Machine Learning problems are much affacted by little but important changes, there is much more than, for example, the number of layers in a model to be considered. We've seen the importance of having to regularize a model in presence of little data in order to get the best of it, helping us also with data augmentation, transfer learning and various other techniques.
As last note, there are really a lot of potential hyperparameters but it's not easy given our hardware resources available to search for the optimum in the space of these hyperparameters.

9 Possible improvements

We want the network to focus on the foreground of the image so we might want to add a channel to the image (so we make a tensor (256, 256, 4)) where we compute the Difference of Gaussians filter like in Image 6 which should highlight the "foreground".

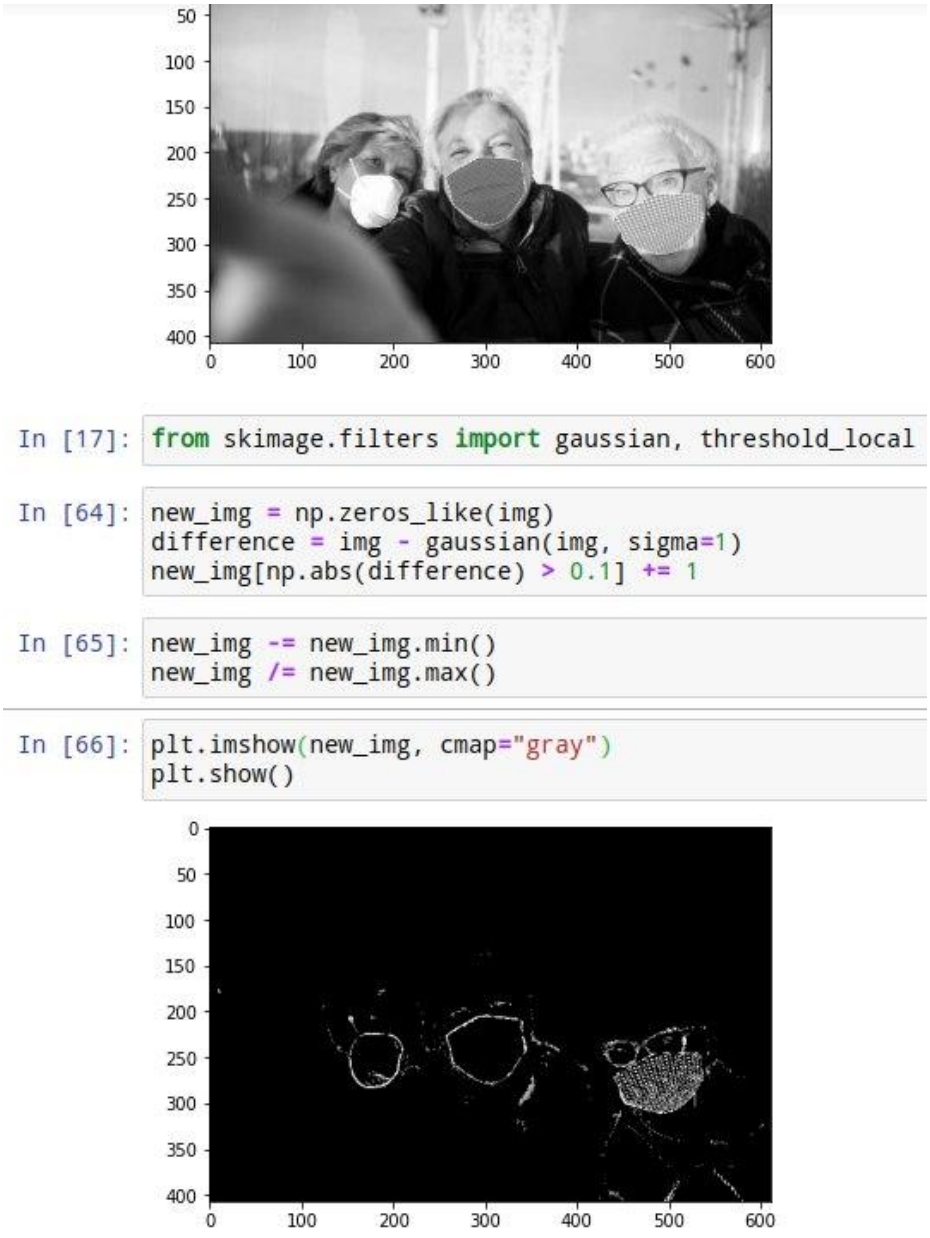

Figure 6: Example of the Difference of Gaussian filter