Image Classification Project

Artificial Neural Network and Deep Learning

A Convolutional Neural Network for facemask recognition

Carlo Ghiglione, Leonardo Perelli, Giacomo Vecchione

20-11-2020

Contents

1 Introduction

The aim of the project is to build a neural network model to classify pictures of people considering if:

1. all wear mask;
2. none wear mask;
3. not all wear mask.

The given dataset is made of 5614 training images, respectively 1897 for class 1, 1900 for class 2, 1817 for class 3 and 450 test images.
We developed our model starting from a very simple architecture, whose structure is a sequence of five blocks made up of one convolution, a *ReLU* activation layer and a MaxPooling (see the summay below).

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 8)       224
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 128, 128, 8)     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 16)      1168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 64, 64, 16)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2D) (None, 32, 32, 32)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2D) (None, 16, 16, 64)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2D) (None, 8, 8, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)                (None, 512)             4194816
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 1539
=================================================================
Total params: 4,294,739
```

```
Trainable params: 4,294,739
Non-trainable params: 0
```

We used a learing rate of 10^{-4}, the *Adam* optimizer and the *CategoricalCrossEntropy* loss function.

2 Hyperparameter tuning

2.1 Data Augmentation

We decided to always do data augmentation considering the much better performaces in terms of validation accuracy and loss we can see.

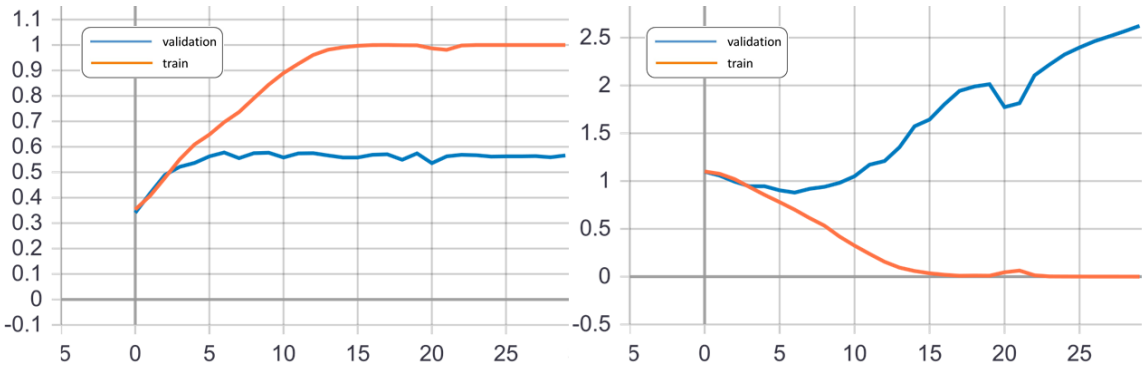

Figure 1: Accuracy (sx) and loss (dx) of the model without data augmentation

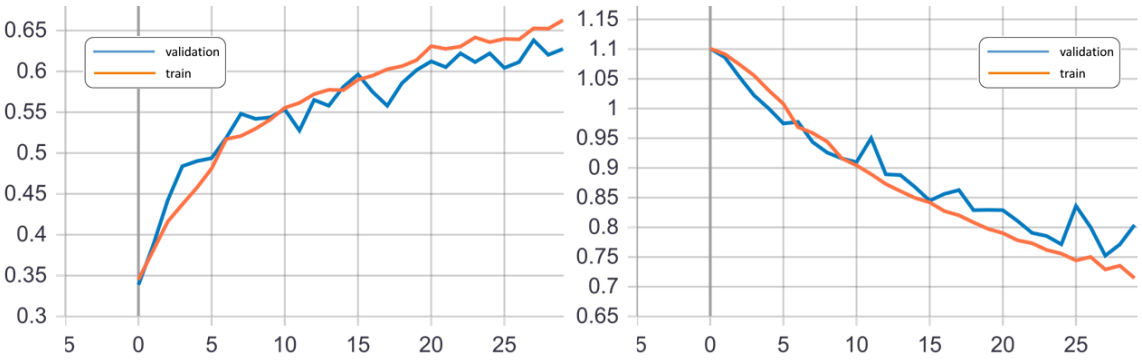

Figure 2: Accuracy (sx) and loss (dx) of the model with data augmentation

2.2 Batch size

After observing the results of the model by changing the batch size, we decided to try to keep this parameter as low as possible, compatibly to the increasing computational complexity of the further developments.

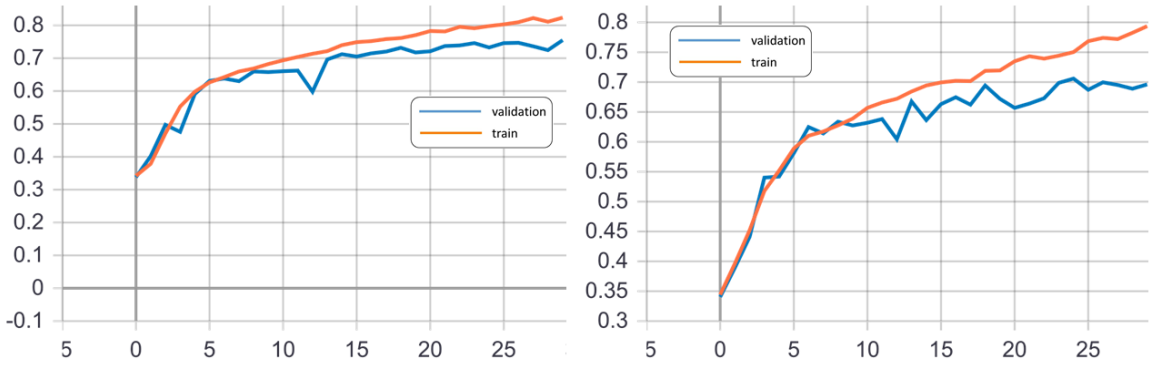

Figure 3: Accuracy with batch size 1 (sx) and 4 (dx)

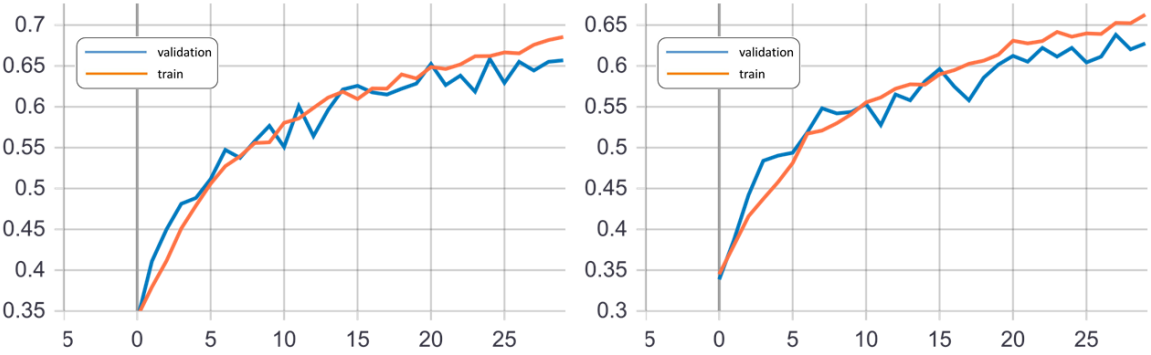

Figure 4: Accuracy with batch size 8 (sx) and 16 (dx)

2.3 Number of blocks

We tuned the number of blocks: the higher the depth is, the better the performances of the model are, obtaining the best results with seven blocks and slightly worse with eight. We suppose this is due to final convolution output images being too small.

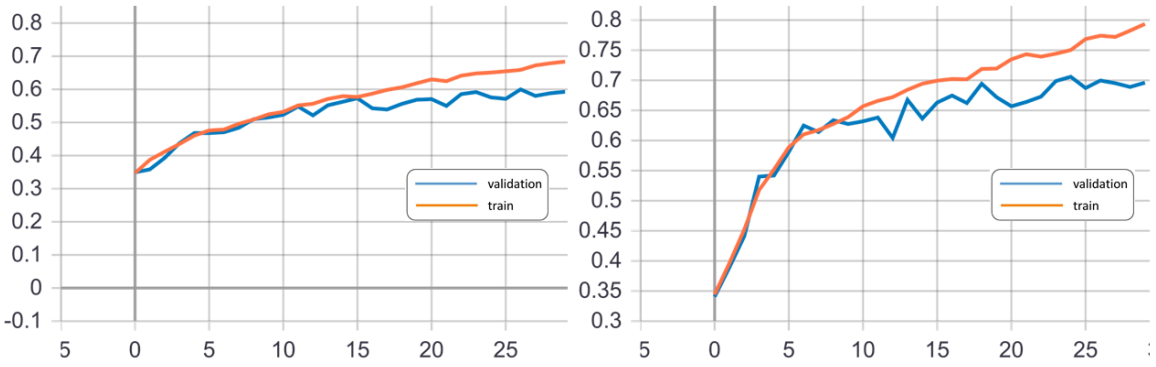

Figure 5: Accuracy of the model with 3 (sx) and 5 (dx) blocks

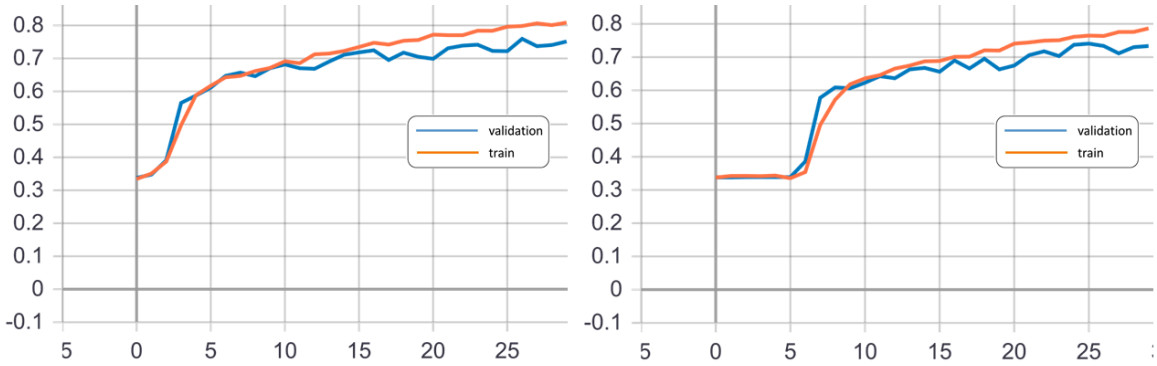

Figure 6: Accuracy of the model with 7 (sx) and 8 (dx) blocks

2.4 Number of convolutions per block

We added a convolutional layer at each block trying with four and five blocks.
We choose to further develop the first model although the other has better performances, seeing that the learning process takes many epochs to start and expecting that, increasing model complexity, it will be delayed even more.

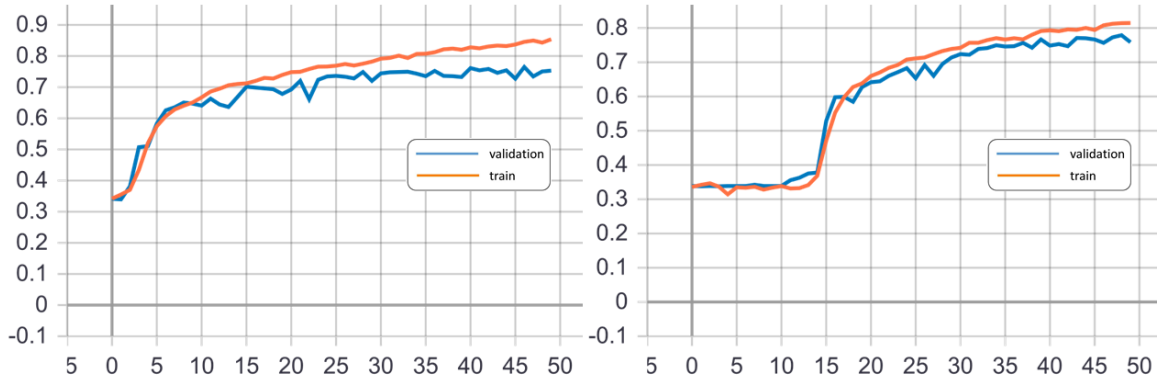

Figure 7: Accuracy of the model with 4 (sx) and 5 (dx) blocks

2.5 Kernels number

We increased the starting number of kernels up to 16 and 32, obtaining a wide complexity in the second case with approximately 26 millions of parameters.

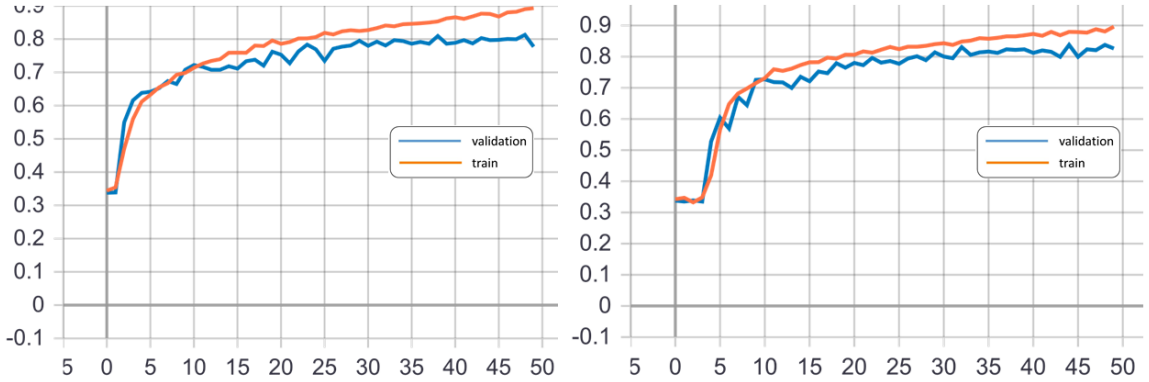

Figure 8: Accuracy of the model with 16 (sx) and 32 (dx) starting kernels

3 Towards final model

3.1 Preliminar model

Inspired by the high performing CNN architectures like VGG-16 and considering our results about depth, we increased the convolutions in the second block up to three and in last two up to four. To speed up the initial learning process, the first five epochs were trained without image augmentation.
The new model has half the parameters of the previous one but better performaces, reaching an average validation accuracy of 0.86 and an average 0.85 score on the test set (computed with the best 10 results).
Its weknesses are:

1. many epochs to get to the best result;
2. overfitting issues from the 70^{th} epoch;
3. irregular weigths distribution in some layers.

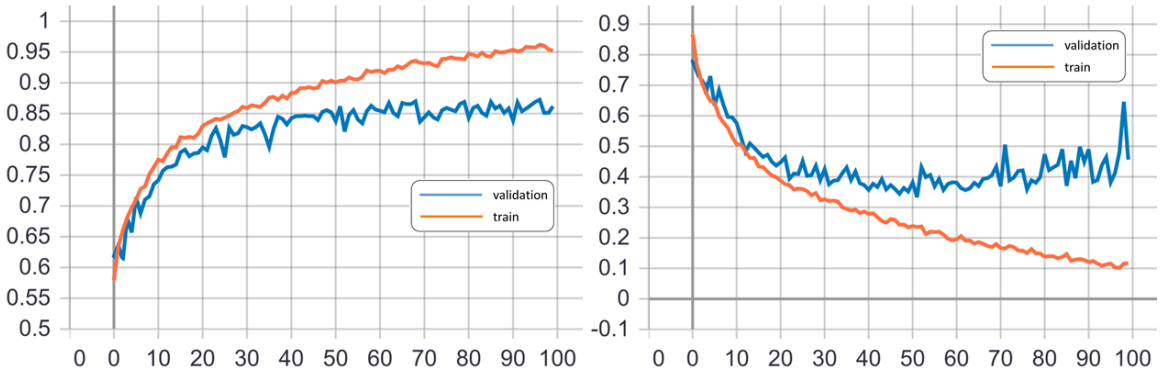

Figure 9: Accuracy (sx) and loss (dx) of the model

epoch	val_accurcay	test_score
59	0.8627	0.8533
60	0.8565	0.8511
63	0.8672	0.8644
66	0.8672	0.8644
67	0.8654	0.8489
77	0.8665	0.8422
80	0.8690	0.8756
86	0.8690	0.8533
92	0.8681	0.8400
97	0.8725	0.8356
average	0.8664	0.8529

3.2 Final model

To address the issues, we added these features:

1. batch normatization layer at the beginning of each block between the first convolution and the activation;
2. dropout with parameter 0.5 after the flatten layer and 0.2 after the dense layer;
3. weight decay with parameter 10^{-4} in last convolution and in the dense layer.

We notice that the overfitting problems have been reduced but the time to reach a good accuracy is still long.
Furthermore, the performances have been improved, indeed the average validation accuracy and test score increased up to 0.87 (computed with the best 10 results), with a notable peak of 0.91.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1_1 (Conv2D)          (None, 254, 254, 32)      896
_________________________________________________________________
batch_normalization (BatchNo)  (None, 254, 254, 32)      128
_________________________________________________________________
re_lu (ReLU)                 (None, 254, 254, 32)      0
_________________________________________________________________
conv2d_1_2 (Conv2D)          (None, 252, 252, 32)      9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 126, 126, 32)      0
_________________________________________________________________
conv2d_2_1 (Conv2D)          (None, 124, 124, 64)      18496
_________________________________________________________________
batch_normalization (BatchNo)  (None, 124, 124, 64)      256
_________________________________________________________________
re_lu (ReLU)                 (None, 124, 124, 64)      0
_________________________________________________________________
conv2d_2_2 (Conv2D)          (None, 122, 122, 64)      36928
_________________________________________________________________
conv2d_2_3 (Conv2D)          (None, 120, 120, 64)      36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 60, 60, 64)        0
_________________________________________________________________
conv2d_3_1 (Conv2D)          (None, 58, 58, 128)       73856
_________________________________________________________________
batch_normalization (BatchNo)  (None, 58, 58, 128)       512
_________________________________________________________________
re_lu_2 (ReLU)               (None, 58, 58, 128)       0
```

```
_________________________________________________________________
conv2d_3_2 (Conv2D)           (None, 56, 56, 128)       147584
_________________________________________________________________
conv2d_3_3 (Conv2D)           (None, 54, 54, 128)       147584
_________________________________________________________________
conv2d_3_4 (Conv2D)           (None, 52, 52, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2D) (None, 26, 26, 128)      0
_________________________________________________________________
conv2d_4_1 (Conv2D)           (None, 24, 24, 256)       295168
_________________________________________________________________
batch_normalization (BatchNo) (None, 24, 24, 256)       1024
_________________________________________________________________
re_lu_3 (ReLU)                (None, 24, 24, 256)       0
_________________________________________________________________
conv2d_4_2 (Conv2D)           (None, 22, 22, 256)       590080
_________________________________________________________________
conv2d_4_3 (Conv2D)           (None, 20, 20, 256)       590080
_________________________________________________________________
conv2d_4_4 (Conv2D)           (None, 18, 18, 256)       590080
_________________________________________________________________
max_pooling2d_4 (MaxPooling2D) (None, 9, 9, 256)        0
_________________________________________________________________
flatten (Flatten)             (None, 20736)             0
_________________________________________________________________
dropout (Dropout)             (None, 20736)             0
_________________________________________________________________
dense_1 (Dense)               (None, 512)               10617344
_________________________________________________________________
dropout (Dropout)             (None, 512)               0
_________________________________________________________________
dense_2 (Dense)               (None, 3)                 1539
=================================================================
Total params: 13,305,315
Trainable params: 13,304,355
Non-trainable params: 960
_________________________________________________________________
```

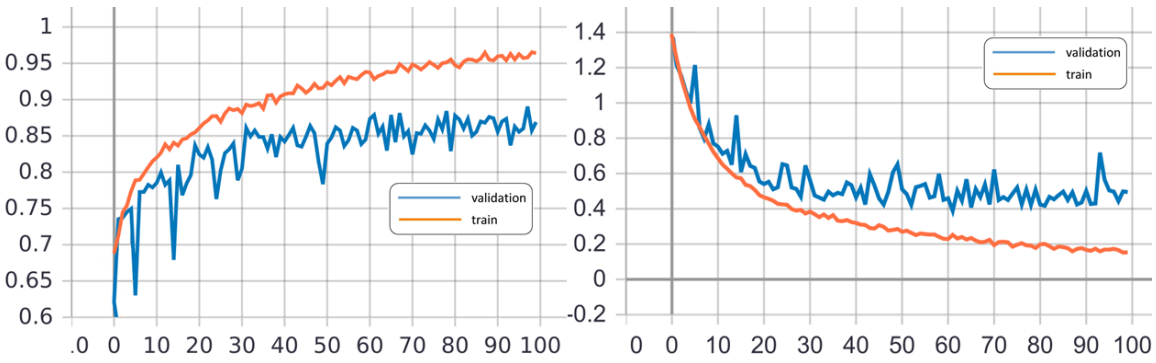

Figure 10: Accuracy (sx) and loss (dx) of the model

epoch	val_accurcay	test_score
61	0.8734	0.8644
66	0.8788	0.8644
68	0.8815	0.9111
74	0.8645	0.8622
77	0.8725	0.8777
82	0.8725	0.8755
84	0.8734	0.8777
87	0.8699	0.8644
90	0.8743	0.8844
92	0.8699	0.8644
average	0.8731	0.8764

4 Conclusions

Considering we obtained a best test score of 0.91 with a model built from scratch and trained on a local GPU without transfer learning, we are very satisfied.
To set a benchmark, we fine-tuned the last three convolutional layers of a VGG-16 architecture using the same classifier structure of our model. Considering it reached a test score of 0.85, we state that our model has relatively high performances.
As further development, if we had more computational capacity to speed up the training process, we would implement the following improvements:

1. tuning different Fully Connected architectures;
2. tuning the dropout parameters and the learning rate;
3. increasing the number of kernels;
4. increasing the input size.

We are enthusiastic about what we learned with this project because we had the opportunity to get our hands on the practical implementation of a Convolutional Neural Network to solve a real-life problem.