FINAL PROJECT: FACE MASK?

DEEP LEARNING

PROF. MAURO CASTELLI

APRIL 20TH, 2022

GROUP 10

ANTÓNIO CYMBRON, DUARTE REDINHA, GONÇALO SILVA, MARIA JOÃO MARQUES, PEDRO SEQUEIRA

NOVA IMS

1. Introduction & Task Definition

Two years ago, masks burst into our lives, becoming an indispensable object in our daily lives. Thus, for access to certain spaces, the correct use of masks has become necessary. The evaluation of the correct use of masks has been a task performed by humans, but we believe that it could be addressed by using a system based on a Deep Learning model that, when coupled with a camera, could detect if people had the mask properly placed, to be allowed to access the desired location.

Thus, in this report, we describe the development of a Deep Learning model to classify mask usage through a dataset of photos of people. The data can be retrieved from this link. There is a *"README.txt"* file in the project folder that contains information on how to save the data, alongside other relevant details.

2. Data Description

In order to train a Deep Learning model to recognize and classify the mentioned objects - or not - in images, a model should be built with the aim of having the ability to recognize certain distinct traces, such as the masks' shape, color, and tonality - that can vary -, as well as other traces, intrinsic to the individual itself, such as the nose contour and tonality, among many others. With this, we understand that it is the uniqueness of these features, that although common, that establishes the difficulty of this task. Although this is an elemental task for humans, it is commonly hard for computers to have this capability, and this is the challenge we are tackling. One should also consider the tough task for a computer to learn under other different conditions, such as the size of the objects presented, or the illumination, for instance. Besides this, there can also be situations where the object may be covered by some other object (occlusion), deformed, cluttered or even the possibility of different viewpoints or poses being used.

Regarding the data itself, it consists of four categories - Fully Covered (faces, by masks), Partially Covered (faces, by masks), Not Covered (faces, by masks), and Not (a) Face. The first category contains faces of people that are wearing face mask correctly according to The World Health Organization standards. The second category, on the other hand, contains face images where the face mask only covers the mouth but not the nose openings. Distinctively, the third category contains face images of people not wearing face mask at all. Finally, the last category, which we decided to also include in our study to make in more complete, contains images that are detected in OpenCV face detection library that do not have faces of people. We believe this last portion of images is also important to consider since we aim to make it applicable to practical cases, such as camaras that detect, at the entrance of a building, if an individual is wearing their masks correctly. For this, it is important that the model can understand where the actual faces are.

The source of this dataset is Kaggle, and its author is James Arnold Nogra. This dataset is under the license of Data files © Original Authors, and has a public visibility. It is also worth mention that the collection of the images at stake was made through OpenCV library to detect faces of people in YouTube videos or video files collected using mobile phone, and they were collected in the first place for the paper Mangmang, G. B. (2020). Face Mask Usage Detection Using Inception Network. *Journal of Advanced Research in Dynamical and Control Systems*, *12*(SP7), 1660-1667. Below is an image of each of the 4 categories previously mentioned.

Figures 1, 2, 3 and 4 - Images of classes "Fully Covered", "Partially Covered", "Not Covered" and "Not Face", respectively

3. Evaluation Measures

When building a model, it is essential to know if it is effective at predicting the intended classes. There are several techniques that can be employed to assure efficacy before deploying the model, but we choose to look at the usual metrics in Multi-Class Classification Problems such as Accuracy, Recall and F1-Score. We decided on using Scikit-Learn's Classification Report tool to look at these metrics, as we felt it gives us all the useful information to assess the models.

Since the classes in the dataset are mostly balanced, Accuracy is a good measure to look at when fine tuning the model. Precision of the Fully Covered class was also a measure we looked at closely to evaluate the models as we believe that False Positives in this class are the worse prediction mistakes the model can make and thus it needs to be kept to a minimum.

Time was also a relevant factor when considering the more suitable models. As such, since we are looking for a balance between the time it takes to train and accuracy, we decided to define a callback that specified that if in two consecutive epochs, the categorical accuracy when predicting the validation set does not improve, the training is stopped to avoid wasting computational resources and time.

With all this in mind, our strategy was to initially use the Hold-out Method to train the models and see which parameters are more promising in terms of accuracy, precision and time taken to train. Then, these more promising parametrizations are selected use the Cross-Validation method, in order to get a more realistic value of the metrics for the selected models.

4. Approach

In this section of our report, we will explain in detail our approach to this task, identifying the challenges that arose and the decisions we made during the project.

4.1. Hold-out Method

The first approach was to use the Hold-out method. We decided to save the images sequentially, assigning the first 80% of data of each image class to the train set, the following 10% to the validation set and the last 10% to the test set. Instead of doing this process manually, we used Python functions to automate it, thus creating folders for train, validation, and test sets, as well as four subfolders for each set representing each one of the four classes the dataset has. Consequently, each image will be assigned to the folder it belongs to.

The pixels of the images of the dataset contained values that ranged from 0 to 255, so we had to perform a rescaling of all the images to fit the range from 0 to 1. Knowing that we would have to test our models with different image and batch sizes, we created generators that yield batches of RGB images of different sizes and binary labels (for example, for image size *64x64px* with batch size 20, the generator yields batches of *64x64px* RGB images with shape (20,64,64,3) and binary labels with shape (20)).

We also created, from the get-go, ten functions that received a model and a directory to perform the model evaluation (one for each combination of image and batch size that we wanted to test).

After this step, we decided to try five distinct architectures, with different hyperparameters. With it, we proceeded to train them in an interactive way, changing the hyperparameters along the way, in order to find the right configuration for our goal. Being so, we tried to alter the values of the image and batch size, and some of the models' layers - as the quantity and the value in the dropout and convolutional layers.

The first 4 models we tested have a similar structure. Thus, they are composed of several convolutional layers, each one being followed by a max pooling layer; after that, we have a flatten layer; following, we resort to dense layers (the last one has 4 neurons and is the only layer of our models whose activation function is not *relu*, since we want a prediction to be made concerning the 4 classes, having therefore chosen to resort to *softmax*). The first 4 models, which generally follow the previously mentioned architecture, were tested with a batch size of 20, for 3 different image sizes (*64x64px*, *128x128px*, and *150x150px*), because we wanted to figure out if there was an ideal size. After figuring out the ideal image size for each of these 4 models, we tested each of them, for the chosen image size, for different batch sizes (32 and 64). The fifth model we trained follows the AlexNet architecture. AlexNet is the name of a convolutional neural network which has had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision. It famously won the 2012 ImageNet LSVRC-2012 competition by a large margin (15.3% VS 26.2% (second place) error rates) (Hassan, 2018). The architecture consists of eight layers: five convolutional layers and three fully connected layers (Wei, 2021). Here, the image size was *227x227px* (which is the correct image size, according to the paper where the architecture was proposed) and, knowing that larger image sizes equal longer training time, we defined that we would only train the model for a batch size of 64, which would be faster than the batch size of 20 with which we were initially training each of the first 4 models. The whole process of training the models and the various tests we performed are in the file *"Holdout_Preprocessing&Training.ipynb"*, which can be found in the folder in which this report was delivered. We chose to put this process in a separate file from the file where we compare the best models, so that the comparison becomes easier, and also because the whole pre-processing section doesn't need to be performed all the times and is quite extensive, not adding value to the conclusions we intend to draw, regarding the best model.

Thus, during these tests, we concluded that a larger image size does not result in significant performance gains, but rather results in a considerable increase in training time. Thus, for the first 3 models, the image size chosen to proceed was *64x64px*. Model 4 is an exception, where we concluded that training the model with an image size of *128x128px* was better. In the 5th model, the image size chosen had to be *227x277px*, for the reasons mentioned above. Furthermore, we also concluded that, except for the 3rd model (where the chosen batch size was 32), training the models with batches of 20 images was beneficial because the results achieved were better and the training time was not considerably longer than the other options we had for batch size. The values we decided to monitor for each of the models can be consulted later, when comparing the best models.

4.2. Cross-validation Technique

After using the hold-out method to train and evaluate our models, we decided to implement the Stratified K-Fold Cross-Validation method, which consists in splitting the available train dataset into K subsets (within each subset, the percentage of samples of each target class is roughly the same), choosing one of the subsets to be used for evaluation, and using the other K-1 subsets to train the model. This process is then repeated until all the subsets have been used to evaluate the model. Finally, the results of every iteration are then combined to get the average model evaluation. One advantage over the hold-out method is that the metrics provided by this approach generally present a more realistic value of the selected models since it helps to minimize the effect of strange or undesirable patterns present on the dataset by splitting it into subsets and then averaging the metrics.

In our case, after choosing the best 5 models with the hold-out method concerning parameters (*64x64px* for the image size, with the exception for models 4 and 5; and batch sizes of 20 for model 1, 2 and 4; 32 for model 3; and 64 for model 5), we decided to test them with this new method to observe if any changes in the models' scores would happen. However, because we realised that this method would be quite expensive in terms of time, we opted for batch sizes of 64 right from the start.

To start, we had to convert all the images in our dataset into a NumPy array, so that we could use scikit-learn train_test_split. The split consisted of saving 10% of our dataset to be used for testing purposes later on in the final models. We decided on using 5 folds (K = 5) for all the models, since a lot of our research shows us that it is a good value for K. We let each iteration run for 15 epochs, since almost all the runs done previously (on the hold-out method) stopped or started to overfit after that. One problem came up when trying to run this method on the AlexNet architecture (model 5), since it would always crash our laptops every time we ran it. Because of this, we opted to not run this model on this part of the project. This is one very big disadvantage brought by cross-validation methods, since they require much more processing power and since we are limited by the resources we have, we felt like this can be a negative point for cross-validation in general.

4.3. Overfitting control

As we are aware, overfitting is a common data science problem where a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data (test set).

We compared the results obtained by the models on the different partitions of the dataset, and the difference between the training and validation sets in accuracy was small, meaning there was little to no overfitting. The models also contained Max Pooling Layers - a function to reduce the feature dimension, number of parameters to learn and computational costs - and Dropout Layers - a method to drop out units (hidden and visible) in a neural network - to better control the overfitting.

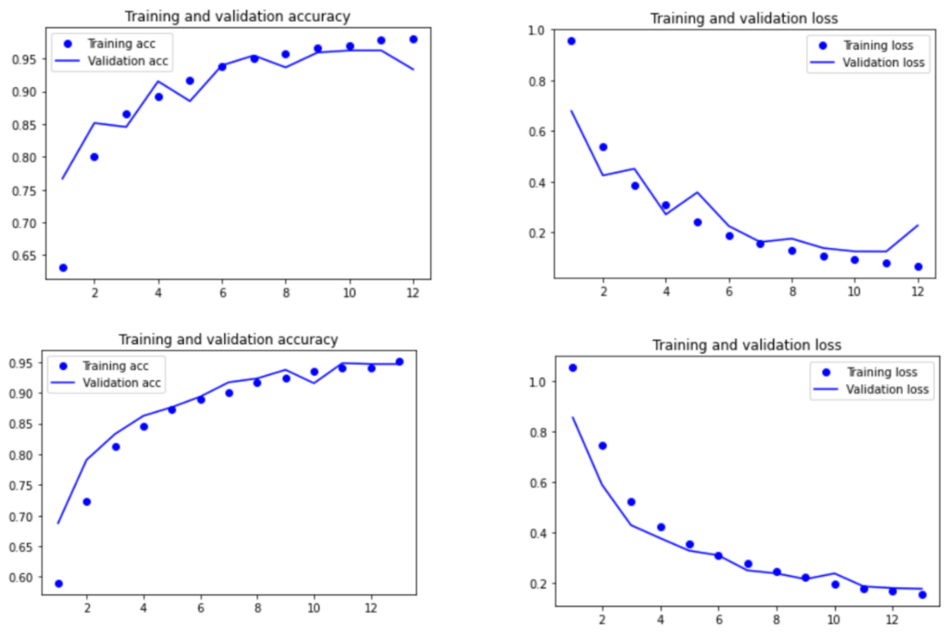
4.4. Results & Error Analysis

Thus, the table below shows the architecture of each of the 5 best models, the batch size that returned the best results in the holdout method, the selected image size, the results obtained in the holdout method, the training time in the holdout method, and the average results, obtained through the cross-validation technique previously discussed. In the table below, in the row "Architecture", the numbers without letters refer to the number of neurons in the convolutional layers (each one being followed by a max pooling layer); "f" refers to a flatten layer; "d(x)" refers to a dropout layer, with value x; and “D(y)” refers to a dense layer, with value y. The values mentioned in the lines "Accuracy (hold-out method)", "Fully covered class Precision (hold-out method)" and "Average accuracy (cross-validation method)", can be found in the file *“GROUP10_notebook.ipynb”*. On the other hand, the training time value can be found in the file *“Holdout_Preprocessing&Training.ipynb”.*

Table 1 - Results obtained for the 5 models chosen

	MODEL 1	MODEL 2	MODEL 3	MODEL 4	MODEL 5
Architecture	32, 64, 128, 128, f, D(512), D(4)	32, 64, 128, 128, f, d(0.5), D(512), D(4)	32, 64, 128, 128, f, d(0.2), D(512), D(4)	16, 32, d(0.2), 64, d(0.2), 128, f, d(0.2), D(512), D(4)	96, 256, 384, 384, 256, f, D(4096), d(0.5), D(4096), d(0.5), D(4)
Batch size	20	20	32	20	64
Image size	64x64px	64x64px	64x64px	128x128px	227x227px
Accuracy (hold-out method)	0.97	0.93	0.95	0.95	0.91
Fully Covered class Precision (hold-out method)	0.98	0.94	0.96	0.94	0.90
Training time, in minutes (hold-out method)	3 minutes and 43 seconds	2 minutes and 6 seconds	3 minutes and 7 seconds	7 minutes and 43 seconds	61 minutes and 21 sec
Average accuracy (cross-validation method)	0.95	0.94	0.93	0.95	NA

When evaluating accuracy, precision, and training time, we can see that the models with the best results are model 1 and model 3. In terms of architecture, the difference between model 1 and model 3 is that model 3 has a dropout layer of 0.2 whereas model 1 has no dropout layer. Both have small image sizes (64x64px) and a batch size of 20 and 32, respectively. In terms of training time both are reasonably quick to train. Model 1 has better overall accuracy and precision but is slightly slower to train than model 3. As such, we believe that both models are equally suitable and the best we have found. Looking at the graphs bellow, that show the training and validation accuracy and loss, of models 1 and 3, both seem very positive. However, we can notice that the 3rd model seems to have less overfitting, in the whole training process. If we only look at the 3rd last epoch of the training processes (which is the one where we save the model's parametrization, due the callback mentioned above), we would choose the 1st model, since it seems that it doesn't have overfitting problems, while also giving better results, in terms of the evaluation metrics.



Figures 5, 6, 7 and 8 - Training and validation accuracy and loss graphs, for models 1 (on the top) and 3 (on the bottom)

5. Conclusion

To conclude our work, we understood that our best model was the model number 1, whose optimal image size is *64x64px*, having been trained with batches of 20 images and which has the following architecture: 32, 64, 128, 128, f, D(512), D(4). This dataset was previously used to perform the same task of training a model for this purpose, by 2 Kaggle users. The first one, Gabriel Sousa, achieved an accuracy of 0.97 for the model, and a precision for the Fully Covered class of 0.91, with the following architecture: 20, 30, d(0.25), 64, d(0.25), f, D(512), d(0.5), D(128), d(0.5), D(4). The second one, Tetsuya Sasaki, achieved an accuracy of 0.97 for the model, and a precision for the Fully Covered class of 0.98, but the model was based on a pre-trained model (InceptionV3), and had the following architecture: InceptionV3 + d(0.2),f, D(128),D(128),D(4). The model we have chosen showed excellent results, having an accuracy of 0.97 for the whole model, and a precision for the Fully Covered class of 0.98. The results obtained by us turned out to be better than the first solution mentioned, but worse than the second one. However, this is due to the fact that the second Kaggle user used only a pre-trained model, followed by dense layers, instead of using convolutional layers, like us and the first Kaggle user.

Besides the named goals for the chosen model, we believe this study can go further and be used to detect more distinct positions that a mask can be used, but still badly placed. We believe this aspect can be a flaw in the current model, given that the majority of images that concerned this category were similar. Regarding some limitations, we believe it is worth mentioning the lack of computer power to test and improve other types of networks. For instance, we couldn't test the AlexNet model for the Cross-Validation Process because we didn't have the needed computing power to do so. We believe better solutions could be left to test due to this. Finally, to go further in our research, to innovate and to "stamp" our approach, we decided to test the model with our own images, produced by each of the group members, with one photograph for each category. With model 1, the results obtained were worse than with the 3rd model. This can be explained by what was mentioned above, when looking at the graphs. Also, another reason for not obtaining better results, is the fact that all the 6621 images in the dataset were very similar between each other, and very different than the ones uploaded by us.

6. References

Agrawal, S. (2022, January 6). How to split data into three sets (train, validation, and test) And why? Towards Data Science. Retrieved April 15, 2022, from https://towardsdatascience.com/how-to-split-data-into-three-sets-train-validation-and-test-and-why-e50d22d3e54c

Alake, R. (2021, December 15). Implementing AlexNet CNN Architecture Using TensorFlow 2.0+ and Keras. Towards Data Science. Retrieved April 16, 2022, from https://towardsdatascience.com/implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98

Allibhai, E. (2021, October 17). Hold-out vs. Cross-validation in Machine Learning - Eijaz Allibhai. Medium. Retrieved April 15, 2022, from https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f

Amazon Web Services. (n.d.). Cross-Validation - Amazon Machine Learning. Retrieved April 18, 2022, from https://docs.aws.amazon.com/machine-learning/latest/dg/cross-validation.html

Brownlee, J. (2020, August 27). Dropout Regularization in Deep Learning Models With Keras. Machine Learning Mastery. Retrieved April 16, 2022, from https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

Nogra, J. A. (2022, February 26). Face Mask Usage. Kaggle. Retrieved April 13, 2022, from https://www.kaggle.com/datasets/jamesnogra/face-mask-usage

Sasaki, T. (2022, April 9). Mask Classification Model by InceptionV3. Kaggle. Retrieved April 15, 2022, from https://www.kaggle.com/code/sasakitetsuya/mask-classification-model-by-inceptionv3

Sousa, G. (2022, April 5). topicos-ii-classificacao-de-imagem. Kaggle. Retrieved April 15, 2022, from https://www.kaggle.com/code/gaspii/topicos-ii-classificacao-de-imagem

When to use "categorical_accuracy vs sparse_categorical_accuracy" in Keras. (2021, December 13). Knowledge Transfer. Retrieved April 15, 2022, from https://androidkt.com/when-use-categorical_accuracy-sparse_categorical_accuracy-in-keras/

Wikipedia contributors. (2022, January 28). AlexNet. Wikipedia. Retrieved April 16, 2022, from https://en.wikipedia.org/wiki/AlexNet

Hassan, M. U. (2018, November 1). AlexNet – ImageNet Classification with Deep Convolutional Neural Networks. Neurohive. Retrieved April 19, 2022, from https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

Wei, J. (2021, December 10). AlexNet: The Architecture that Challenged CNNs - Towards Data Science. Medium. Retrieved April 19, 2022, from https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951