Final project personal report

Atharva Haldankar

DATS 6203_10: Machine Learning 2

Amir Jafari

December 10, 2022

Introduction

Group 1 is working on using CNN to predict whether a person is wearing a mask or not. In this project our main scope is to not only predict if the person is wearing a mask or not but to also compare the performance when we train using a pretrained model and when we train without a pretrained model.

For this project my team mate Osemekhian Ehilen worked on training the model with pretrained models whereas I worked on training the model without a pretrained model.

Description of my scope of work

Data and it pre-processing

We loaded our data from Kaggle and titled Face Mask Detection with 853 images belonging to 3 classes as well as their bounding boxes with the face mask's location.

After loading the data, I parsed the annotations of all the images in a single data frame. This data frame gets important information from the annotations like the name of the images, the dimensions of boxes to extract the faces and the labels of the images.

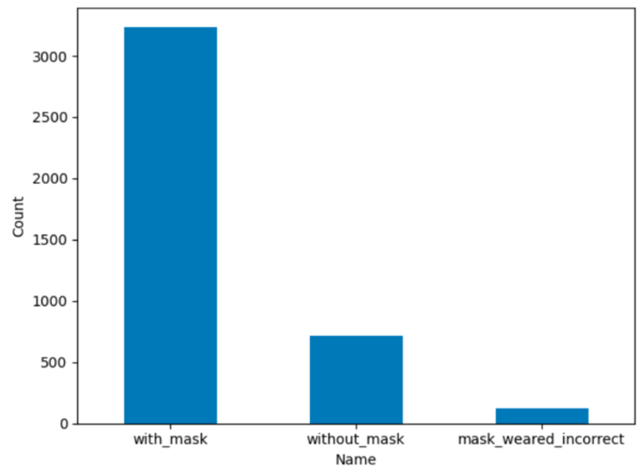
From the plot above we can see that there is significant imbalance in the data which could result in overfitting.

For this model, in spite of heavy imbalance in the data we consider all the classes of the data.

After visualizing the data distribution, with the help of dimensions provided in the annotations of the images, we extract the faces from all the images and split the data into training, validation and testing sets and save it in respective folders.

The final step before I trained the model was data augmentation. Using TensorFlow Datagenerator I flipped, rotated & shifted images to improve performance and outcomes of machine learning models by forming new and different examples to train datasets. It also helps to avoid overfitting to a certain extent by adding on to the training samples.

Before working on the model, I also visualized the counts of different classes.



Now that the data is ready for training the model, we start developing the model.

From the above bar plot we can see that after data augmentation we have significantly increased our training samples.

Architecture of the model

A hierarchical decomposition of the input is made possible by the stacking of convolutional layers. Faces, animals, houses, and other objects are extracted from very deep levels as this process proceeds. This is exactly what we observe in real life. As the depth of the network grows, features are abstracted to ever-higher levels.

Max pooling is done in part to help overfitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation.

I started developing the model layer by layer. I observed good results when I reached a 12-layer model, it gave me good accuracy of 93% but while testing the model on my own face with and without a mask, it did not work very well. So, I added 3 more layers to the model, which did result in slightly better accuracy of 94% and it also predicted my face with and without mask correctly.

Followed by the Max pooling layer, is the dropout layer. This layer plays an important role in avoiding overfitting.

Flattening layer is used to convert all the resultant 2D arrays from the pooled feature maps into a single long continuous linear vector.

And finally, a dense layer with “relu” as the activation function for the hidden layer and the output dense layer with “softmax” as the activation function is used to classify the output from the above convolution layers.

The output for each convolution layer was calculated with the following formula:

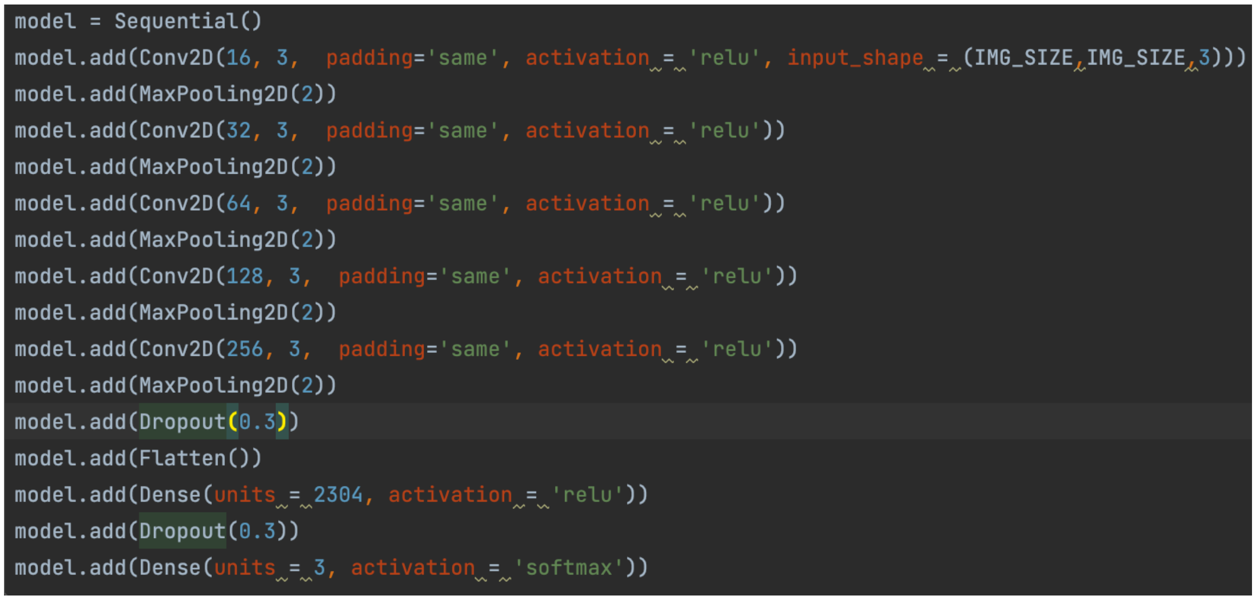
$$[(W-K+2P)/S]+1$$

Where: W is the input volume

K is the Kernel size

P is the padding

S is the stride



```
model = Sequential()
model.add(Conv2D(16, 3,  padding='same', activation = 'relu', input_shape = (IMG_SIZE,IMG_SIZE,3)))
model.add(MaxPooling2D(2))
model.add(Conv2D(32, 3,  padding='same', activation = 'relu'))
model.add(MaxPooling2D(2))
model.add(Conv2D(64, 3,  padding='same', activation = 'relu'))
model.add(MaxPooling2D(2))
model.add(Conv2D(128, 3,  padding='same', activation = 'relu'))
model.add(MaxPooling2D(2))
model.add(Conv2D(256, 3,  padding='same', activation = 'relu'))
model.add(MaxPooling2D(2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(units = 2304, activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(units = 3, activation = 'softmax'))
```

We used Adam as the optimizer for this model as we achieved the best results with this optimizer. Categorical Cross entropy was used as the loss function and the accuracy as our performance metric.

To avoid overfitting in this model we use the "ReduceLRonPlateau" scheduler.

The learning rate used for this model is 0.001 and stays constant throughout with image size as 100x100, batch size as 8 and 50 iterations.

Early stopping with a patience=8 will be used to stop the training when no significant improvements are observed or when the performance is going down.

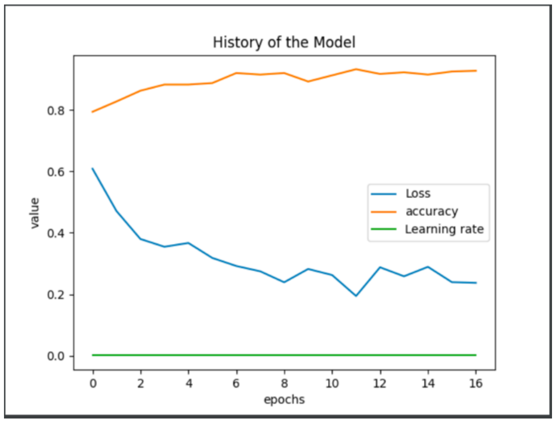
"Modelcheckpoint" callback is used to save the best model while training, this helps us save the entire model when it performs best while iterating.

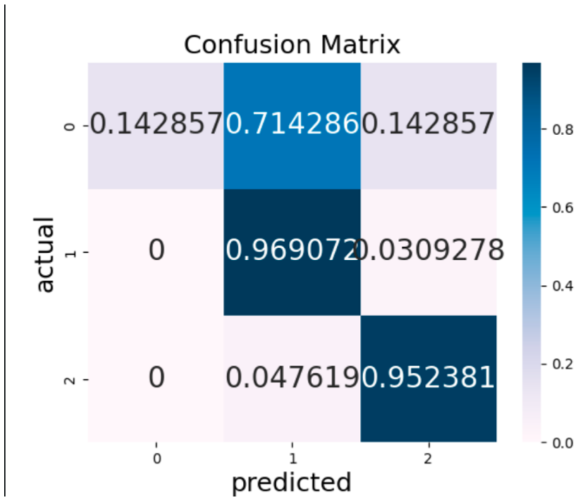
Results

The history of the model shows healthy performance throughout. The loss reduces and the accuracy keeps improving as the number of epochs progress.

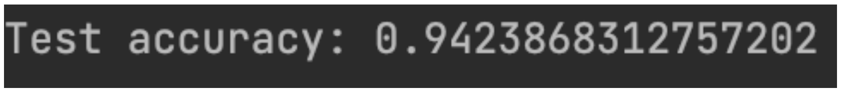
We can also see that even though we ran the model for 50 epochs, the best model was found at the 16th epoch and hence it saved that model. This was done with the help of the "modelcheckpoint" callback.

In spite of the test accuracy being so high, the model does not perform very well with real images, because of heavy imbalance in the data.



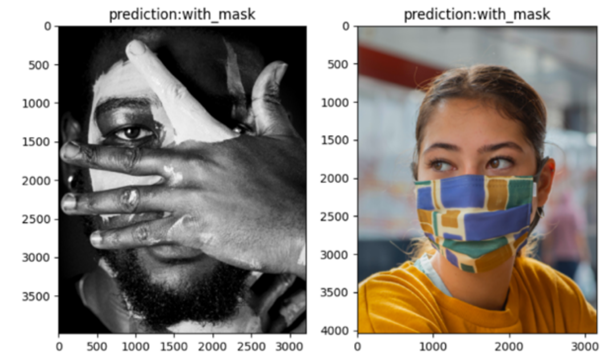


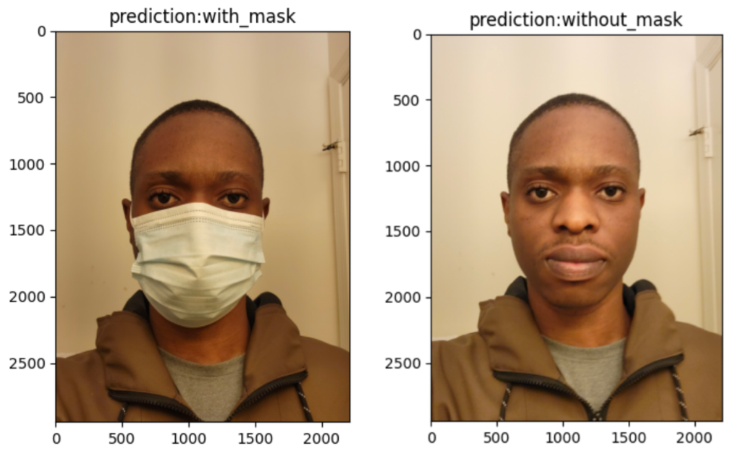
From the above confusion matrix, we can see that label 1: "with_mask" and 2: "without_mask" have the highest prediction accuracy while the label 0: "mask_worn_incorrectly" has the least accuracy. The bar plot above shows how imbalanced the data is and hence explains this behavior of the model.

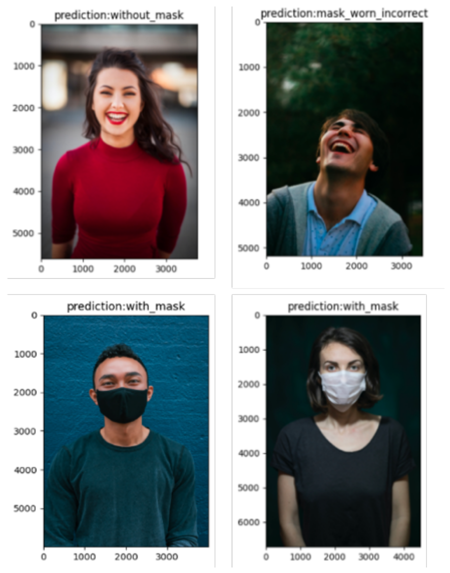


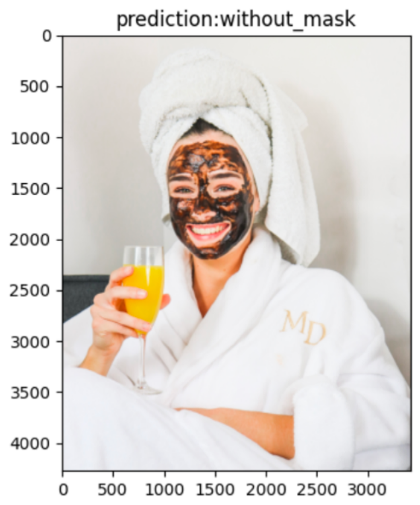
```
Test accuracy: 0.9423868312757202
```

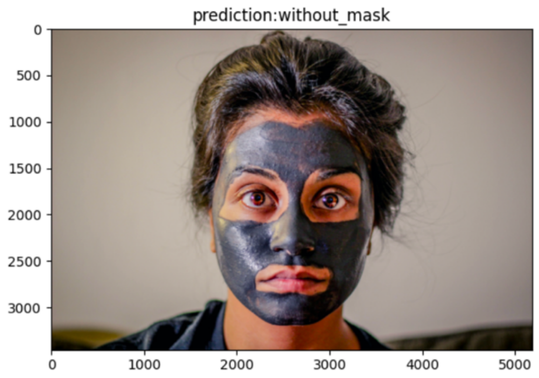
In spite of the test accuracy being so high, the model does not perform very well with real images, because of heavy imbalance in the data.


prediction:with_mask
prediction:with_mask


prediction:with_mask
prediction:without_mask


prediction:without_mask
prediction:mask_worn_incorrect
prediction:with_mask
prediction:with_mask





The title of these images is the prediction of the image given to the model.

From the above results, it can be seen that pictures which have a better focus on the faces work better.

We tested this model with the same images we tested the pre-trained model with.

This model even though has a higher accuracy, does not predict very accurately with certain images. This is due to the imbalance in the data.

Summary and Conclusion

From the above results we can conclude that we're able to achieve such high accuracy is because majority of the data is with "with_mask" label, but when used with real images, the model tends to get a confused and gives us the wrong result, which indicates that it's overfitting.

Even though it works well with images which especially focus on the face of the person, it is not ready for real world application as it can easily get confused when the image gets a little complex or tricky. This behavior can be seen while predicting the boy laughing and the man who is covering his face with his hand in the above figures.

For this model to work better, we need more training data for the non-dominant classes i.e., “without_mask” and “mask_worn_incorrectly”.

This model could also work better with a face detection model, as the predictions are accurate when the images are focused on the face properly.

The other alternative could be, to use FasterRCNN to train the model which has the ability to create bounding boxes around the targets.

Calculation of % code copied from internet

Lines copied = 113

Lines modified = 22

Lines added = 147

% copied = $\frac{113-22}{113+147} * 100$ = 35%

Outcome

From this project I have learned the following:

1. Load custom data and parse the annotations to a dataframe.
2. Use ImageDataGenerator for data augmentation.
3. Construct a CNN model from scratch and calculate the outputs of every layer.
4. Use and importance of early stopping, scheduler and model checkpoint callback.
5. Understanding of different loss functions, activation functions and optimizers.
6. Importance of dropout layer and pooling layers.
7. Save the model and use it for future prediction.

Reference:

1. https://medium.com/@cmukesh8688/activation-functions-sigmoid-tanh-relu-leaky-relu-softmax-50d3778dcea5

2. https://deepai.org/machine-learning-glossary-and-terms/max-pooling

3. https://www.kaggle.com/code/daniel601/pytorch-fasterrcnn/notebook

4. https://optimization.cbe.cornell.edu/index.php?title=Adam

5. Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: neural networks for machine learning, 4(2):26–31, 2012.

6. Géron, Aurélien (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. Sebastopol, CA: O'Reilly Media. ISBN 978-1-492-03264-9., pp. 448