MACHINE LEARNING II -

PROJECTREPORT

Face Mask Detection

Atharva Haldankar | Osemekhian Ehilen

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

OVERVIEW

One from amongst other advantageous areas of Deep Neural Networks (DNNs) is computer vision, which is the ability for a computer to see and understand and interpret accurately what is seen.

With the eruption of Coronavirus disease 2019 (COVID-19)[1] in Wuhan, China, in December 2019; the world has not remained the same. The importance of face masks[2] became mandatory to dampen the spread through the human respiratory channels.

This project will show and compare machine learning models on facemask prediction.

DESCRIPTION OF DATASET

The dataset is from kaggle, titled Face Mask Detection with 853 images belonging to 3 classes as well as their bounding boxes with the face mask's location.

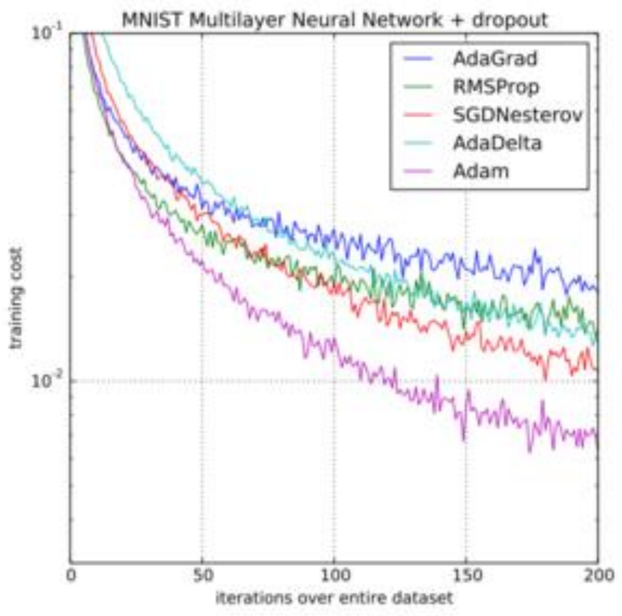
DEEP LEARNING NETWORK & TRAINING ALGORITHM

On this project task we leverage the Adam training algorithm. Adam as an optimizer is an extended version of the stochastic gradient descent (SGD) used in various deep learning applications. The Adam[3] optimizer uses estimations of the first and second moments of the gradient to adapt the learning rate for each weight of the neural network. Adam is proposed as the most efficient stochastic optimization which only requires first-order gradients where memory requirement is too little.

[1] Wikipedia- "COVID-19", https://en.wikipedia.org/wiki/COVID-19
[2] Centers for Disease Control and Prevention (5 April 2020). "What to Do if You Are Sick". U.S. Centers for Disease Control and Prevention (CDC). Archived from the original on 14 February 2020. Retrieved 24 April 2020.
[3] Adam, https://optimization.cbe.cornell.edu/index.php?title=Adam

Though Adam converges well and fast but in research it is found not converging to the optimal solution. Switching to SGD in some cases show better generalizing performance than Adam alone.

Comparison of optimizers used for the optimization training of a multilayer neural network on MNIST images. Source- Google



The Convolutional Neural Network as a deep learning network was used as it is popular in computer vision research.

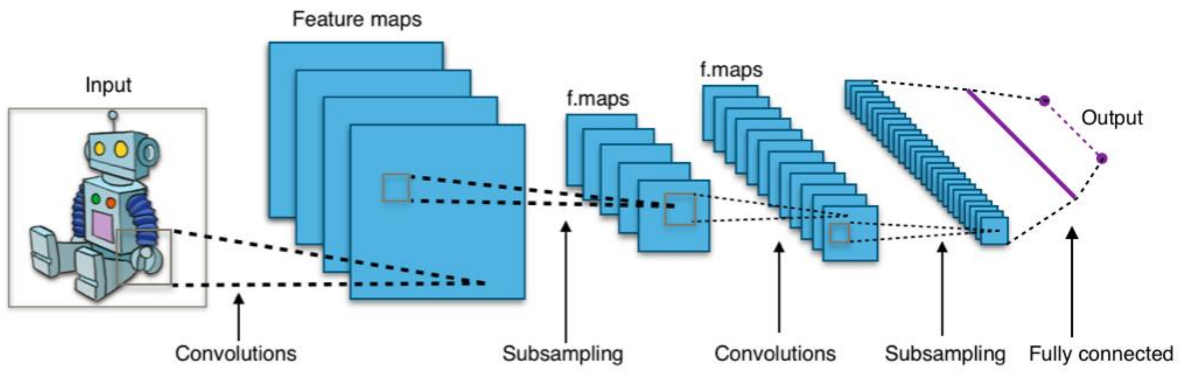
The convolutional layer is the main arm of a convolutional neural network (CNN).

According to Wikipedia[4], the layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input

[4] Convolutional Neural Network, https://en.wikipedia.org/wiki/Convolutional_neural_network

volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the filter entries and the input, producing a 2-dimensional activation map of that filter.

As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.



By Aphex34 - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45679374

EXPERIMENTAL SETUP

We are going to leverage pre-trained models as part of transfer learning in deep neural networks. The Xception, ResNet50 and VGG16 will be utilized in solving our facemask detection problem.

The TensorFlow framework will be used in utilizing the functionality of neural network.

The data obtained from kaggle will be parsed into a pandas data frame then pre-processed using flipping, shifting, rotation and pre-trained specific pre-processing functions.

We reduced the target class to binary type i.e. a person with a face mask or without.

The loss will be captured by **cross entropy** and **accuracy** will judge the model's performance. A **batch size** of 16 will be used which will be in mini-batches.

Also, a **learning rate** of 0.0001 and 0.00001 will be used at various points of the training as there will be a reduction in the learning rate after unfreezing the pre-trained model layers.

To avoid over-fitting, the learning rate scheduler called **Performance Scheduling** will be used to adjust the learning rate during training. Early stopping with a patience value will be used to stop the training when no significant improvements are observed and when the performance is going worse; the callback in TensorFlow will help save the best model amongst the epochs.

PRE-TRAINED MODELS

Keras[5] described transfer learning as taking features learned on one problem, and leveraging them on a new, similar problem. Keras also made known that Transfer learning is usually done for tasks where your dataset has too little data to train a full-scale model from scratch.

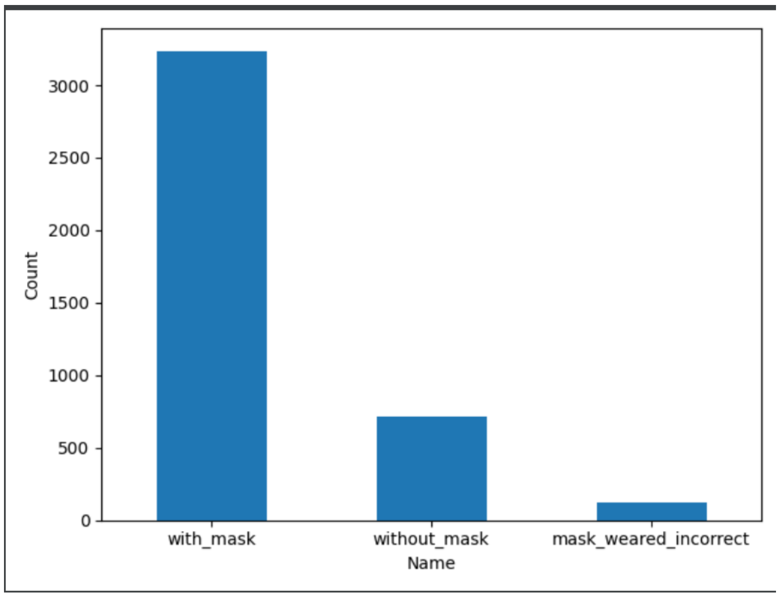
The three pre-trained models that will be leveraged are:

- Xception: this requires an input size of 299 by 299
- ResNet50: this requires an input size of 224 by 224
- VGG16: this requires an input size of 224 by 224

[5] Transfer Learning & Fine Tuning, https://keras.io/guides/transfer_learning/

WITHOUT PRETRAINED

We also tried to train the model with a custom CNN. For this we separated all the images of different categories into different folders to train the model. For training this model we have considered all the categories which were there in the data.



From the above graph we can see that there is a heavy imbalance in the samples of different categories.

This may result in overfitting and cause the predictions to be inaccurate.

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.

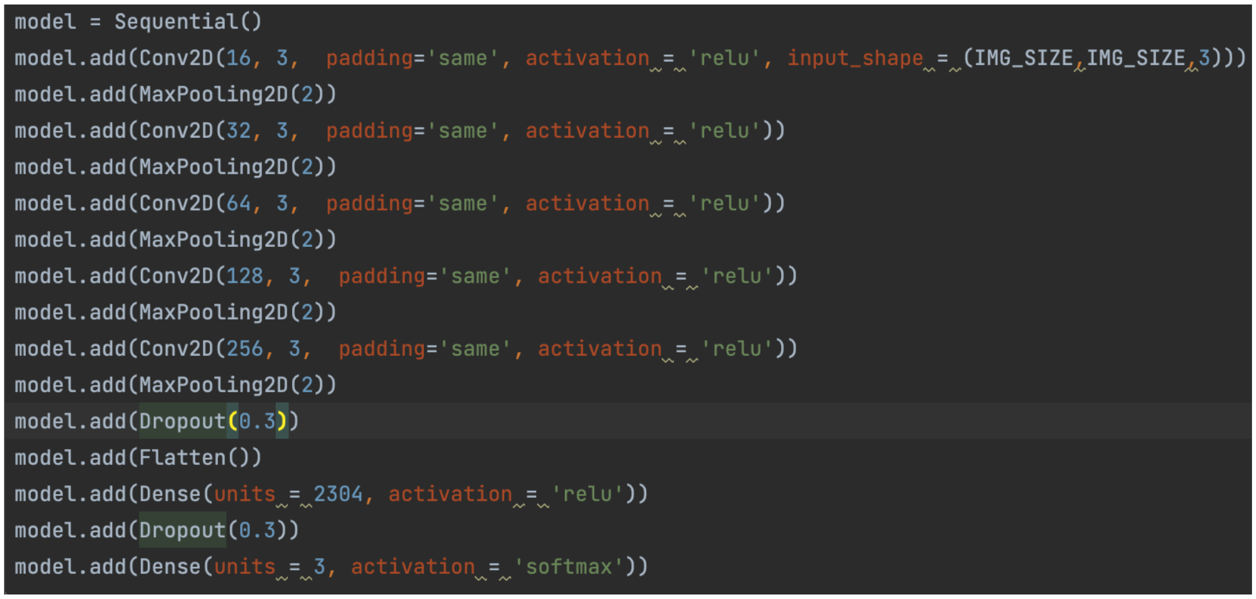
To avoid overfitting in this model we use the “ReduceLRonPlateau” performance scheduler which fine tunes the learning rate.

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.

ARCHITECTURE OF THE MODEL



```
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'))
```

A[6] 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[7] 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.

[6] https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[7] https://deepai.org/machine-learning-glossary-and-terms/max-pooling

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:

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

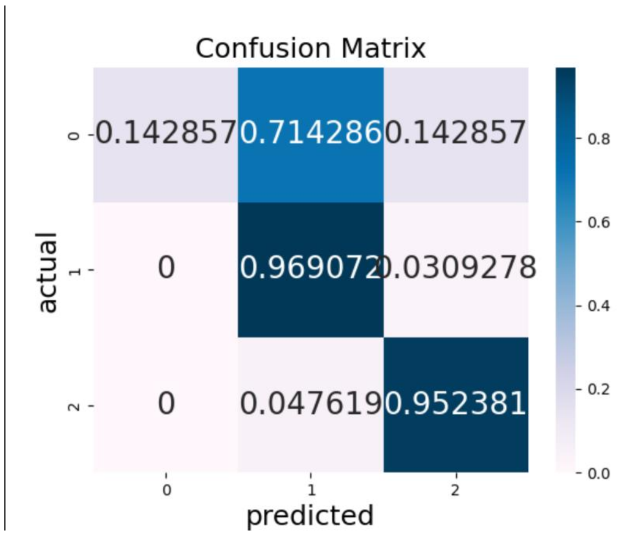
Where: W is the input volume

K is the Kernel size

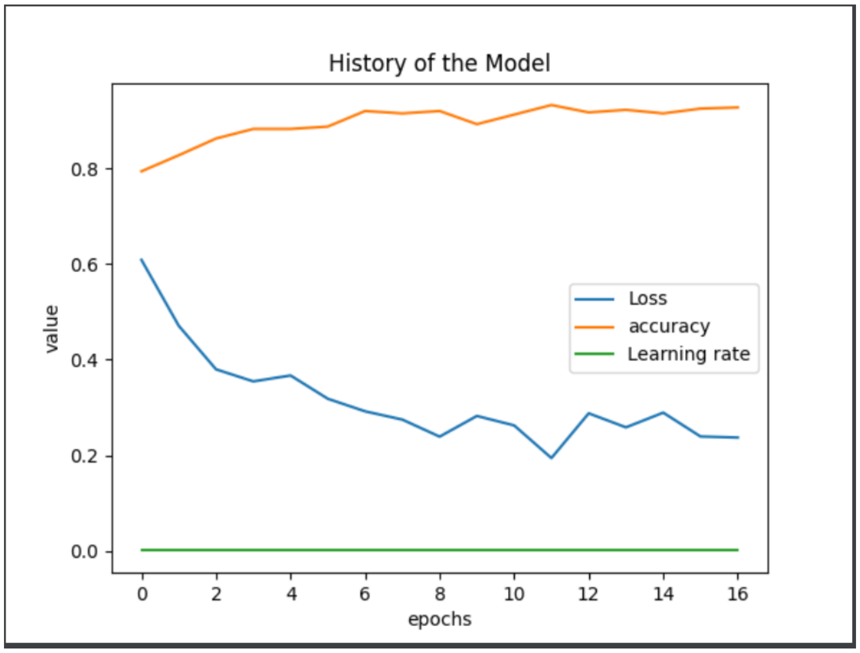
P is the padding

S is the stride

RESULTS



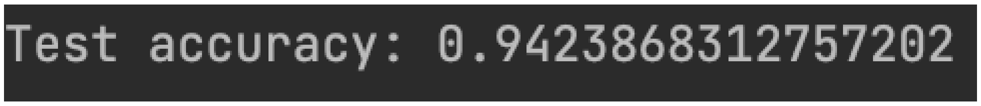
From the above confusion matrix we can see that label 1 : With_mask and 2: without_mask have the highest prediction accuracy while the label 0: mask_worn_incorrectly has the least accuracy. The bar plot above shows how imbalanced the data is and hence explains this behavior of the model.



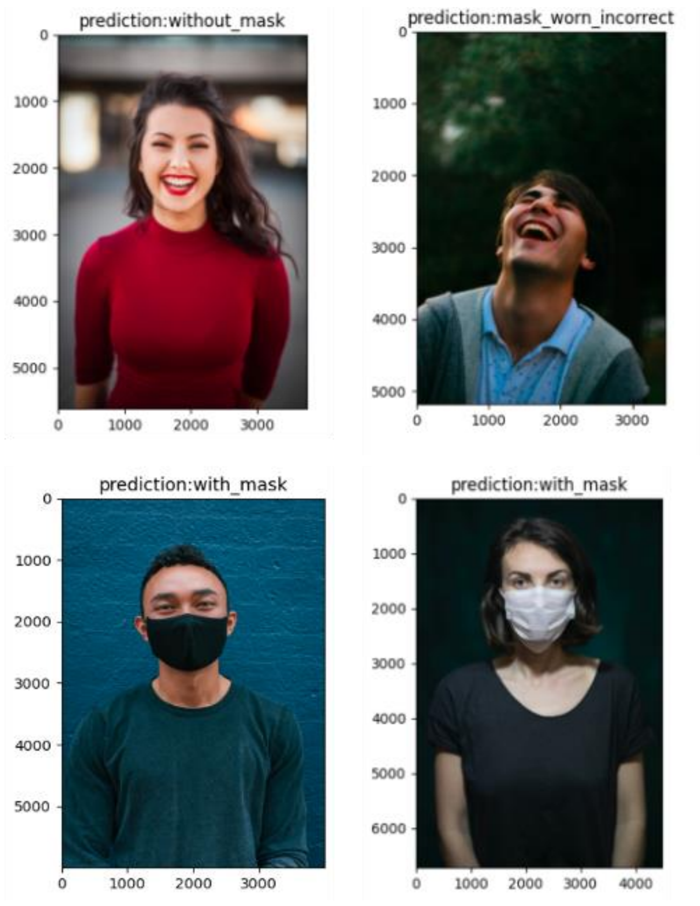
The history of the model shows healthy performance throughout. The loss reduces and the accuracy keeps improving as the number of epochs improve.

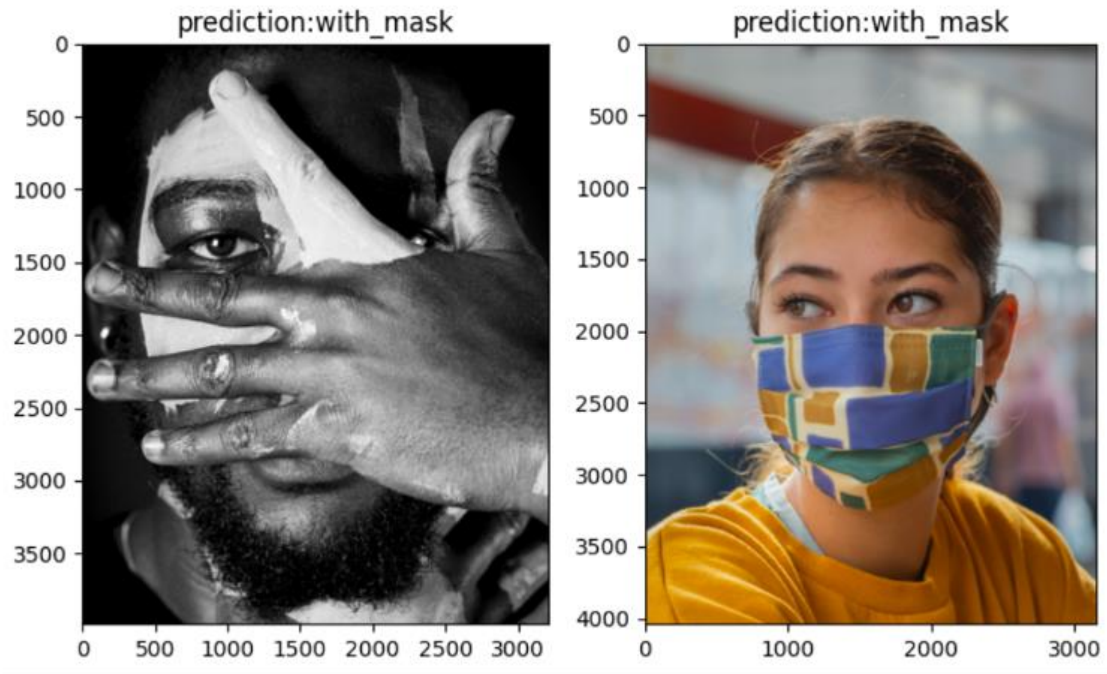
We can also see that even though we ran the model for 50 epochs, the best model was found at the 16th epoch and hence it saved that model. This was done with the help of the modelcheckpoint callback.

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



```
Test accuracy: 0.9423868312757202
```


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



[8]

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

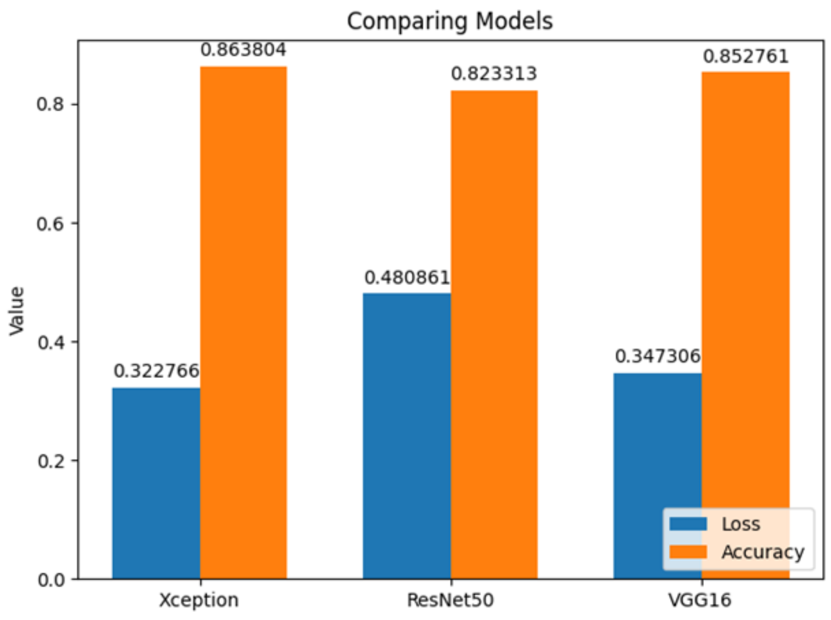
From the above results, it can be seen that pictures which focus on the faces tend to work better.

RESULTS FOR PRETRAINED MODEL

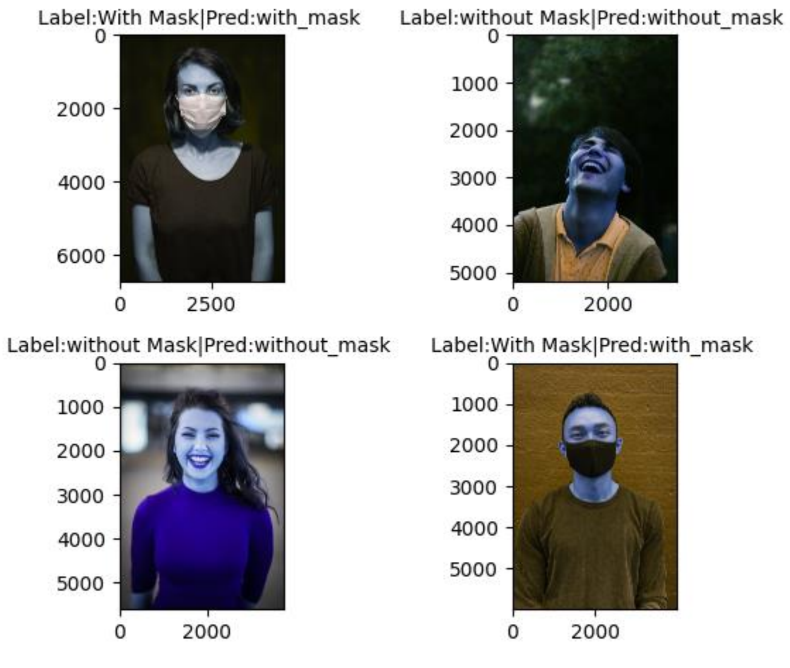
We have made splits on the dataset into training, validating and test set.

After training using Xception, ResNet50 and VGG16 pre-trained models, we got this:

[8] https://unsplash.com/s/photos/facemask

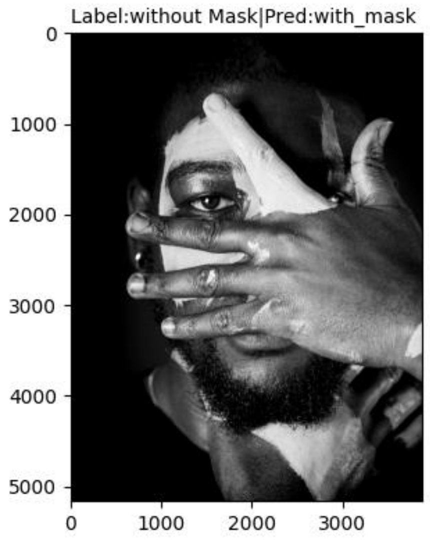


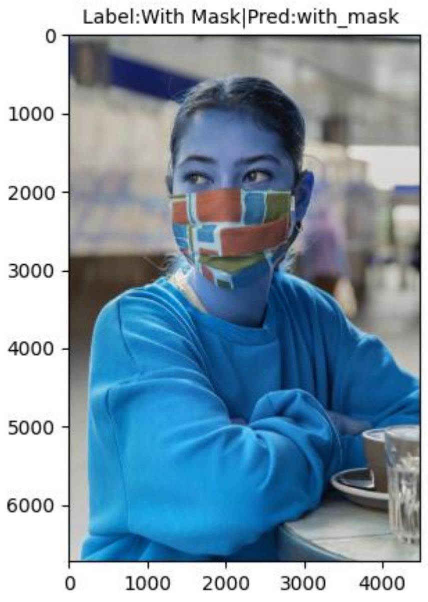
With Xception having the best score we tested some images from unsplash.com to see its performance. Below is the result of the test.


Label:With Mask|Pred:with_mask
Label:without Mask|Pred:without_mask
Label:without Mask|Pred:without_mask
Label:With Mask|Pred:with_mask

Predicting Facemask with Xception Model

We also gave a tricky image to the model and it performed strangely.





The model seems to think that the person’s hand is the facemask on the left image.

SUMMARY, RECOMMENDATION AND CONCLUSION

We discovered that both models are sensitive to the face alone to predict correctly.

Still the custom made CNN model performed better that our best pre-trained model.

We have learnt [using Tensorflow] to:

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 early stopping, scheduler and model checkpoint callbacks.
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.

REFERENCES

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

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

https://unsplash.com/s/photos/facemask

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.

https://medium.com/@cmukesh8688/activation-functions-sigmoid-tanh-relu-leaky-

relu-softmax-50d3778dcea5

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

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