FACE MASK DETECTION USING ALEXNET ARCHITECTURE

Course Project for

ARTIFICIAL INTELLIGENCE

Course Code : EC3066D

Group Number : 39

Instructor : Dr.Anup Aprem

Year : 2020/21

Govind J S B180544EC

Athul K Rajeev B180626EC

Emmanuel Thomas Deavsia B180711EC

CONTENTS

Introduction

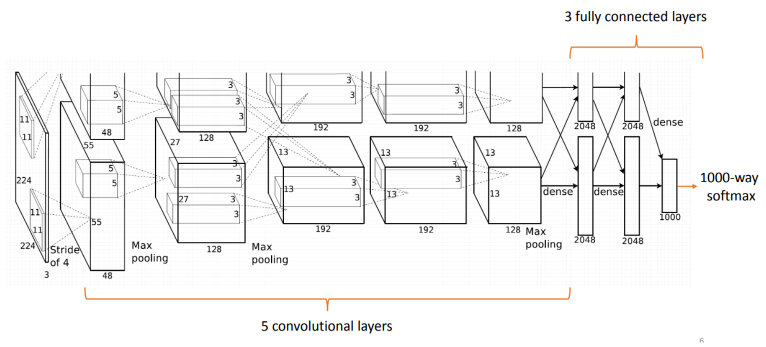
Task

Our task is to develop, train and test a CNN model in AlexNet architecture to classify masked and unmasked faces with a high accuracy. Modify the architecture to match our use case. S Determine suitable hyper parameters and other properties like optimizer and can produce good results while maintaining a feasible computational load and time required.

Architecture

We have used the AlexNet architecture for our CNN model. AlexNet architecture was proposed by **Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton** in 2012 and had won the famous ILSVRC 2012, beating the next best performer by a fair margin (15.3% VS 26.2% (second place) error rates). Various innovations used in architecture were considered revolutionary in Deep Learning at that time.

AlexNet had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision. The network had a very similar architecture as LeNet by Yann LeCun et al but was deeper, with more filters per layer, and with stacked convolutional layers. It consisted of **11×11, 5×5,3×3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum**. It attached ReLU activations after every convolutional and fully-connected layer.



The first convolutional layer filters the 224×224×3 input image with 96 kernels of size 11×11×3 with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size 5 × 5 × 48. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size 3 × 3 × 256 connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size 3 × 3 × 192 , and the fifth convolutional layer has 256 kernels of size 3 × 3 × 192. The fully-connected layers have 4096 neurons each.

AlexNet uses Rectified Linear Units (ReLU) instead of the tanh function, which was standard at the time. ReLU's advantage is in training time; a CNN using ReLU was able to reach a 25% error on the CIFAR-10 dataset six times faster than a CNN using tanh.

Dataset

Training and Testing were done on two datasets all taken from kaggle.com.

Dataset : kaggle.com/pranavsingaraju/facemask-detection-dataset-20000-images
This dataset is collection of 20000 grayscale images of masked and unmasked faces containing 2 classes, each class with a strength of 10000 images each, the dataset has a usability rating of 7.5/10 in kaggle

Dataset : kaggle.com/niharika41298/withwithout-mask
This dataset is a collection of 938 RGB images of masked and unmasked faces containing 2 classes, class 1 of masked faces has 400 images and class 2 has 538 images. The Dataset has a usability rating of 8.1 /10 in kaggle.

Dataset: github.com/cabani/MaskedFace-Net
MaskedFace-Net is a dataset of human faces with a correctly or incorrectly worn mask (133,783 images) based on the dataset Flickr-Faces-HQ (FFHQ). The wearing of the face masks appears as a solution for limiting the spread of COVID-19. In this context, efficient recognition systems are expected for checking that people's faces are masked in regulated areas. We pick 2000 images in this dataset for better evaluation of our model accuracy.

Environment

The entire model was tested on the Colab development environment provided by google. Tensorflow and keras library were used for importing the neural network and associated functions. Seaborn library is used for plotting various metrics. MatplotLib library is used for plotting various graphs .

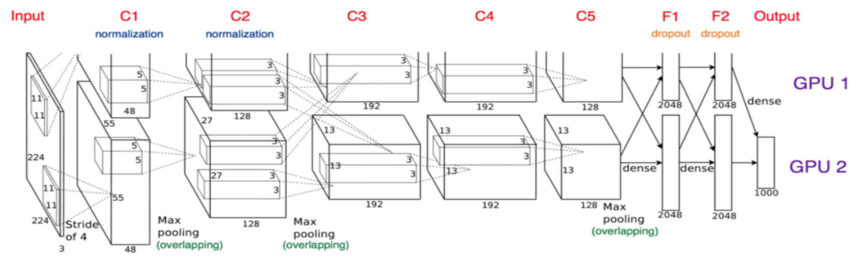
The Colab development environment provides a 12 GB RAM and 60GB Disk space.

Review of AlexNet

Overall Architecture

AlexNet consisted of **11×11, 5×5, 3×3 convolutions, max pooling, dropout,** layers with SGD with momentum as an optimizer. It attached ReLU activations after every convolutional and fully-connected layer.

Architecture itself is relatively simple. There are 8 trainable layers: 5 convolutional and 3 fully connected. ReLU activations are used for all layers, except for the output layer, where sigmoid activation is used. Local response normalization is used only after layers C1 and C2 (before activation). Overlapping max pooling is used after layers C1, C2 and C5. Dropout was only used after layers F1 and F2.



Distinctive Features

1. **ReLU Nonlinearity.** AlexNet uses Rectified Linear Units (ReLU) instead of the tanh function, which was standard at the time. ReLU's advantage is in training time; a CNN using ReLU was able to reach a 25% error on the CIFAR-10 dataset six times faster than a CNN using tanh.
2. **Multiple GPUs**. Back in the day, GPUs were still rolling around with 3 gigabytes of memory (nowadays those kinds of memory would be rookie numbers). This was especially bad because the training set had 1.2 million images. AlexNet allows for multi-GPU training by putting half of the model's neurons on one GPU and the other half on another GPU. Not only does this mean that a bigger model can be trained, but it also cuts down on the training time.

3. **Overlapping Pooling.** CNNs traditionally "pool" outputs of neighboring groups of neurons with no overlapping. However, when the authors introduced overlap, they saw a reduction in error by about 0.5% and found that models with overlapping pooling generally find it harder to overfit.
4. **DropoutLayers.**Use of dropout layer instead of regularisation to deal with overfitting. However, the training time is doubled with the dropout rate of 0.5.

The Overfitting Problem

Two methods were employed to reduce overfitting:

1. **Data Augmentation.** The authors used label-preserving transformation to make their data more varied. Specifically, they generated image translations and horizontal reflections, which can increase the size of the dataset.In this project we are also using the same technique to prevent overfitting

2. **Dropout**. This technique consists of "turning off" neurons with a predetermined probability (e.g. 50%). This means that every iteration uses a different sample of the model's parameters, which forces each neuron to have more robust features that can be used with other random neurons. However, dropout also increases the training time needed for the model's convergence.

The Optimizer and Learning rate heuristic

We initialized the weights in each layer from a **zero-mean Gaussian distribution** with standard deviation **0.01**. We initialized the neuron biases in the second, fourth, and fifth convolutional layers, as well as in the fully-connected hidden layers, with the constant 1. This initialization accelerates the early stages of learning by providing the ReLUs with positive inputs. We initialized the neuron biases in the remaining layers with the constant 0.

We used an **equal learning rate** for all layers, which we adjusted manually throughout training. **The heuristic which we followed was to divide the learning rate by 10 when the validation error rate stopped improving with the current learning rate**. The learning rate was initialized at 0.01 and 6 reduced three times prior to termination.

We trained the network for roughly **90 cycles** through the training set of **1.2 million images**, which took **five to six days** on two NVIDIA GTX 580 3GB GPUs.

Optimizations

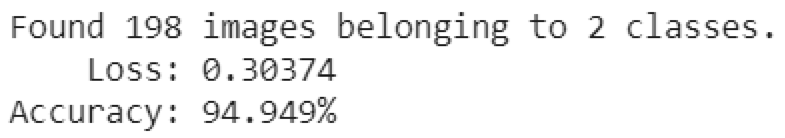
(Playing around with architecture to perfect it for our use case)

Architecture

- ***Number of filters***

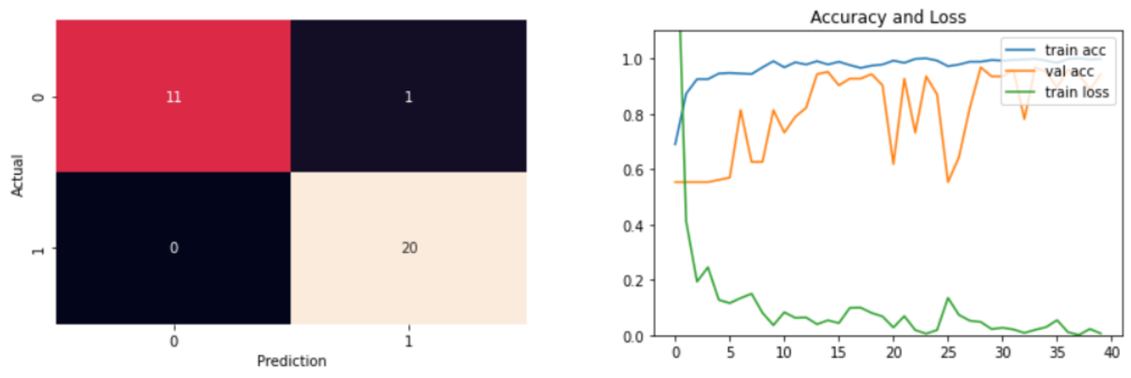
  AlexNet architecture has 5 convolution layers and 3 fully connected layers. Each convolution layer has a parameter of filter number. The number of filters is the property that determines the size of the output feature map. Increasing the number of filters can detect more intricate features.
  Original AlexNet has filter counts as 96, 256, 384, 384, 256 for the five convolution layers respectively

  Testing Loss and Accuracy for default number of filters.



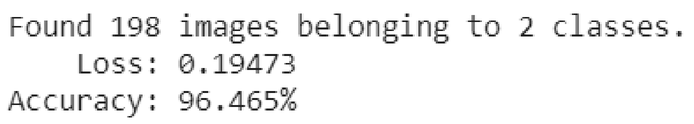
```
Found 198 images belonging to 2 classes.
    Loss: 0.30374
Accuracy: 94.949%
```

I. Doubling the number of filters



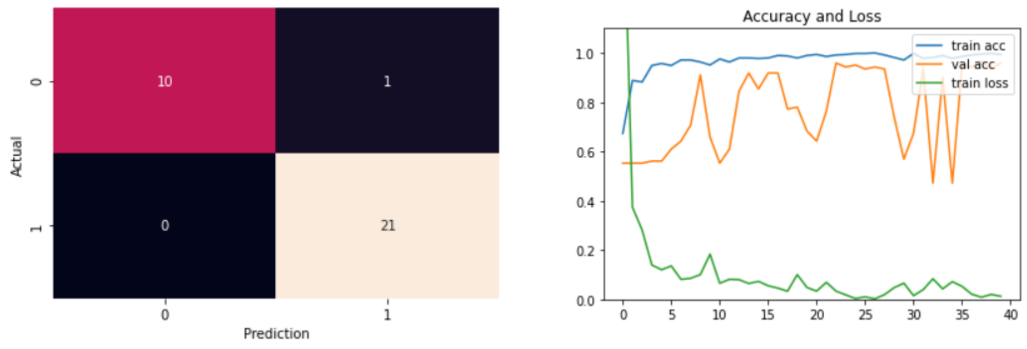
Confusion matrix and Accuracy vs Epoch plot after halving the filter count

Testing Loss and Accuracy for double the number of filters.



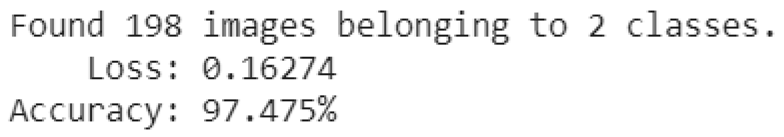
```
Found 198 images belonging to 2 classes.
     Loss: 0.19473
 Accuracy: 96.465%
```

II. Half the number of filters



Confusion matrix and Accuracy vs Epoch plot after halving the filter count

Testing Loss and Accuracy for half the number of filters.



```
Found 198 images belonging to 2 classes.
     Loss: 0.16274
 Accuracy: 97.475%
```

Better performance by the smaller filter length can be explained by the fact that the model predicts masks which occupy a large part of a portrait image of a person. So looking for a larger feature is more reasonable, hence the better performance of the latter model.

- ***Activation Function***

AlexNet was one of the first architectures to use ReLU activation functions. It was one of the distinctive features of AlexNet.

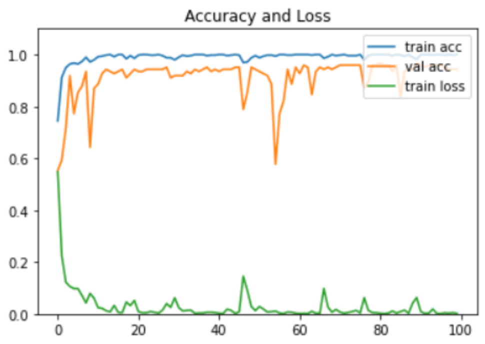
I. ReLU and tanh(x)

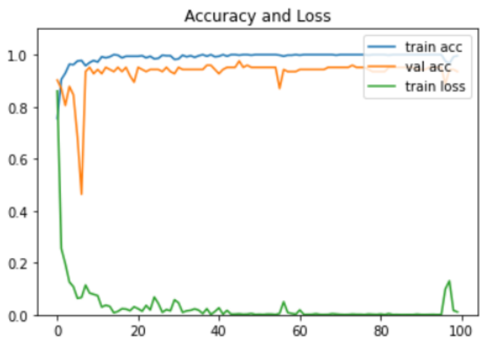
The ReLU function is a non-linear activation function that has gained popularity in the deep learning domain. ReLU stands for Rectified Linear Unit. The main

advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.
This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.
Tanh is also a very popular and widely used activation function.It is also nonlinear in nature, and bound to range (-1, 1)

$$\mathrm{ReLu}(x) = \max(0, x)$$

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$





Comparison of performance of ReLU and tanh activation functions.
Optimizer : AdaGrad at 0.01 lr

ReLU and tanh have almost similar performance if not tanh has a slight advantage but still ReLU is chosen more often as the activation for hidden layers because of its computational efficiency and simpler gradient.

II. Sigmoid vs Softmax

The output layer of AlexNet can be configured in two ways either using a softmax activation function that forces us to use categorical cross entropy as loss metric, or use sigmoid/tanh and use binary cross entropy as loss metric.

The next activation function that we are going to look at is the Sigmoid function. Sigmoid is a nonlinear function and It belongs to one of the most widely used non-linear activation functions. Sigmoid transforms the values between the range 0 and 1.

- ***Stride***

Stride is a component of convolutional neural networks, or neural networks tuned for the compression of images and video data. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal

AlexNet used max pooling of size 3 and stride 2. This means that the largest values were pooled from 3x3 regions, centers of these regions being 2 pixels apart from each other vertically and horizontally. Overlapping pooling reduced the tendency to overfit and also reduced test error rates by 0.4% and 0.3% (for top-1 and top-5 error correspondingly).

Our experiments with different strides did not produce any difference from original accuracy. This may be due to the fact of how prominent a feature we have to extract from the image (facemask in our case). So we kept original strides values.

- ***Dropout Layers (dropout probability)***

Drop out was a state of art counter overfitting measure taken by ALexNet.
Function of a drop out can be summarised as some number of layer outputs are randomly ignored or "*dropped out*" during training. This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "*view*" of the configured layer.

Original AlexNet used a dropout layer of 0.4-0.5 probability, that is 40% - 50% of nodes will get turned off during every iteration.

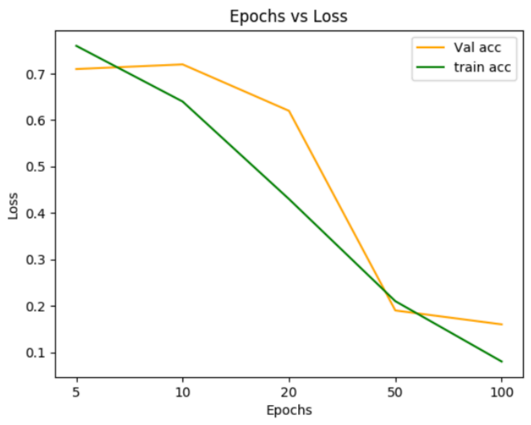
Hyper Parameters

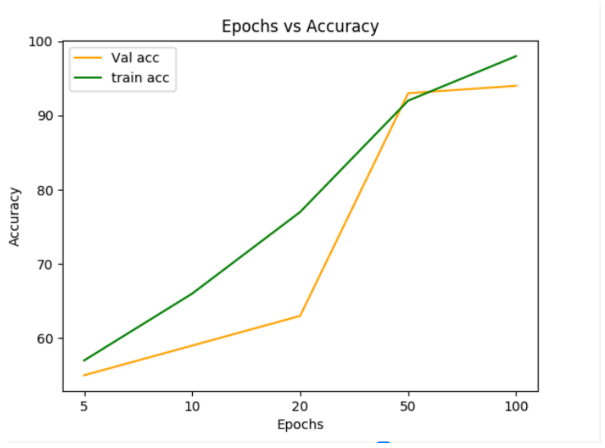
- ***Epochs***

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.

Optimizer SGD at learning rate = 0.0001

No of Epochs	Training Loss (Accuracy)	Validation Loss (Accuracy)
5	0.76 (57 %)	0.71 (55 %)
10	0.64 (66 %)	0.72 (59 %)
20	0.43 (77 %)	0.62 (63 %)
50	0.21 (92 %)	0.19(93 %)
100	0.08 (98 %)	0.16 (94 %)

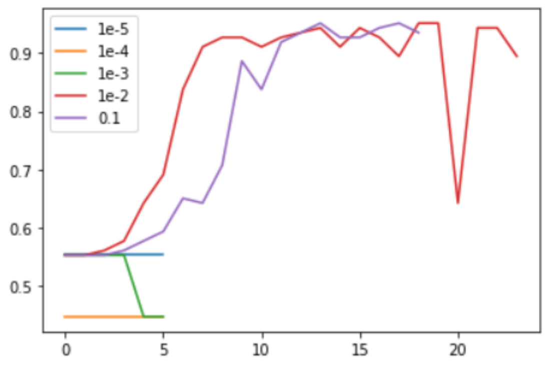




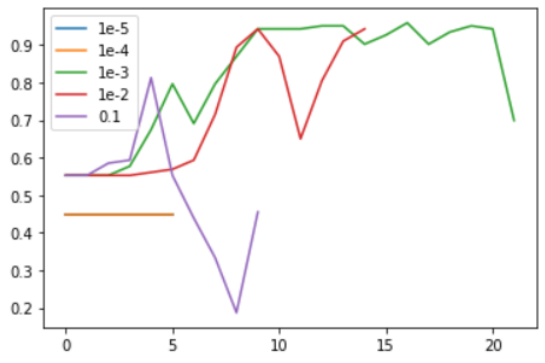
On changing to better optimizers requires only a lower number of epochs to train the model

• *Learning Rate*

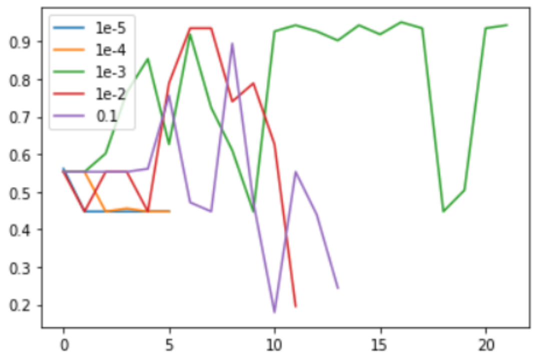
The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process. In the original version of Alexnet SGD with learning rate 0.01, momentum 0.9 and weight decay 0.0005 is used.



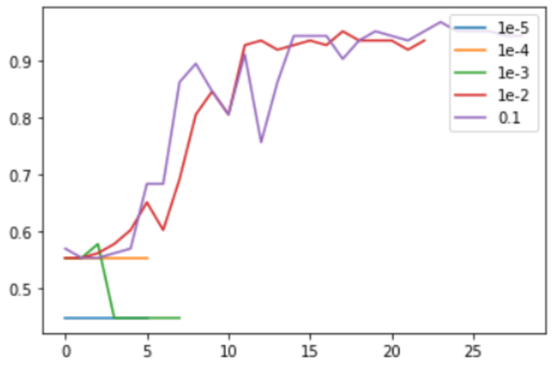
(i) SGD



(ii) Adam



(iii) RMSProp



(iv) AdaGrad

- ❏ SGD has worked well on higher learning rate while lower learning rates are very not enough for the optimizer to learn and the process is called back by the early stop.

- ❏ Other optimizers are faster,0.01 is the best choice for learning rate.

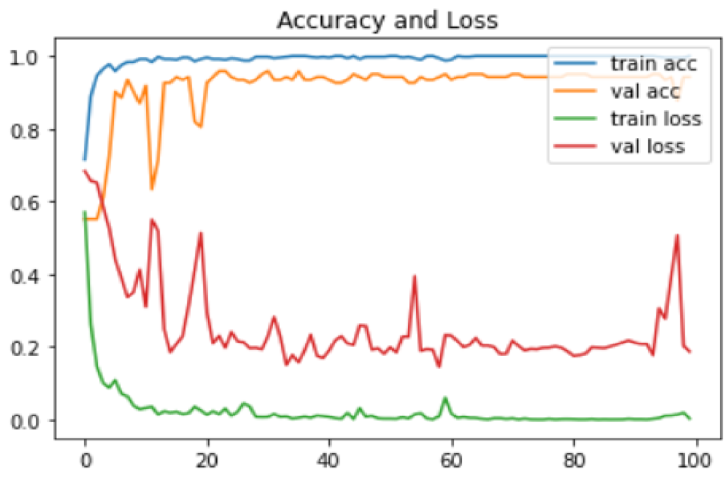
• *Optimizer*

1. *Stochastic Gradient Descent (SGD)*

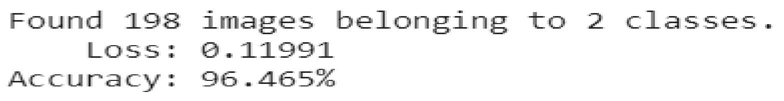
The word '***stochastic***' means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called **'batch'** which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical

Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset.

In our case SGD is the optimizer originally used in the AlexNet published in 2012. But now a new class of optimizers have come up called **'adaptive learning rate optimizers'** which are way faster and learn better than normal optimizers like SGD.



Accuracy and Loss for SGD



```
Found 198 images belonging to 2 classes.
     Loss: 0.11991
Accuracy: 96.465%
```

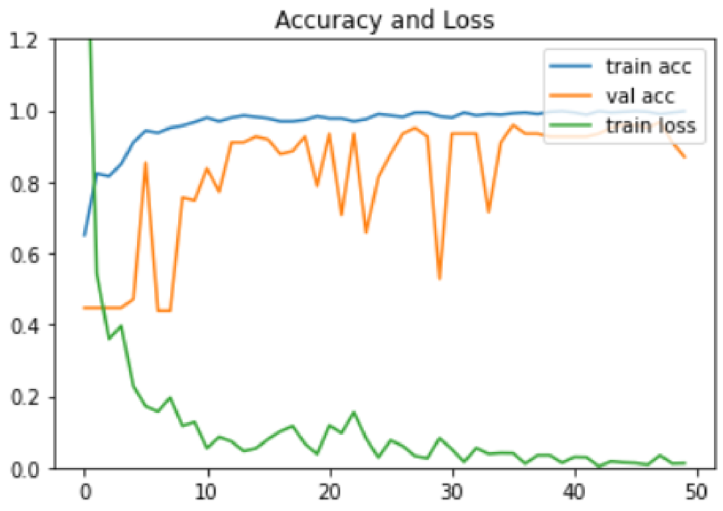
Testing Accuracy and loss for SGD

- ❑ SGD is a comparatively slower algorithm so we had to choose 100 epochs at a learning rate of 0.01. The results obtained were pretty much self explanatory. Although the final accuracy and loss are appreciable, the number of epochs and time taken to reach that is a big setback for SGD.

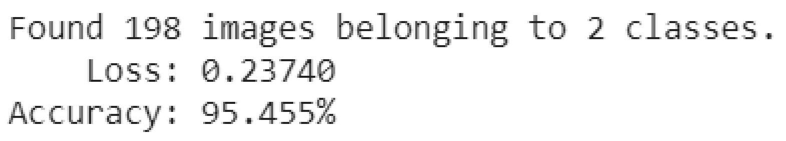
2. *Adam (Adaptive Momentum Estimation)*

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data.

Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled "Adam: A Method for Stochastic Optimization"



Accuracy and Loss for Adam



```
Found 198 images belonging to 2 classes.
     Loss: 0.23740
Accuracy: 95.455%
```

Testing Accuracy and loss for Adam

Adam is a very fast learning optimizer and a best choice among optimizers that keras provide. We tested at a learning rate of 0.01 for 50 epochs , and the results were promising. Adam combined with a suitable counter overfitting method would be a good choice for our use case.

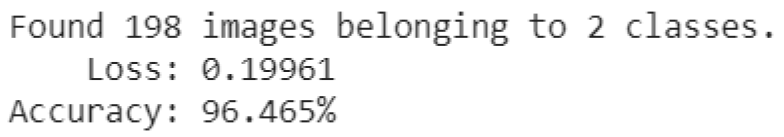
3. *RMSprop*

The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated.

The following equations show how the gradients are calculated for the RMSprop and gradient descent with momentum.

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon} \qquad v_{dw} = \beta \cdot v_{dw} + (1-\beta) \cdot dw^2$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon} \qquad v_{db} = \beta \cdot v_{dw} + (1-\beta) \cdot db^2$$

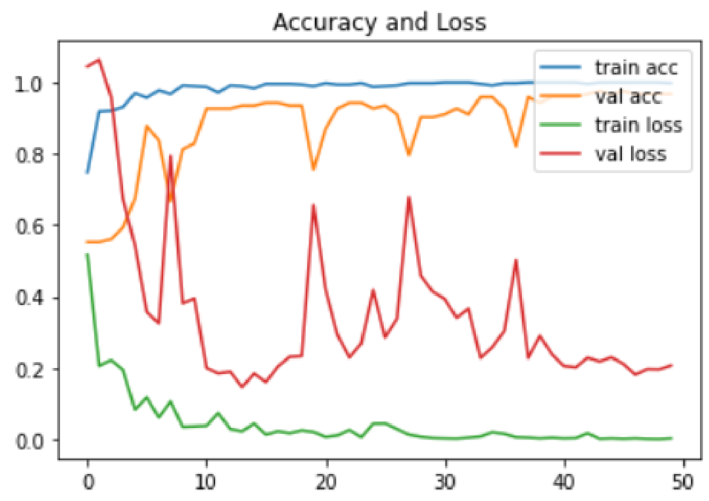


```
Found 198 images belonging to 2 classes.
     Loss: 0.19961
Accuracy: 96.465%
```

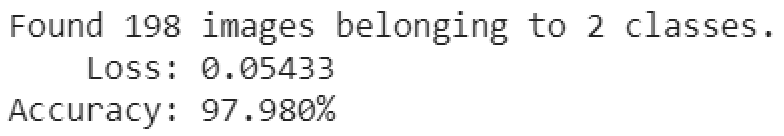
Testing Accuracy and loss for RMSprop

4. *AdaGrad*

Adagrad is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data. Adagrad greatly improved the robustness of SGD.



Accuracy and Loss for AdaGrad



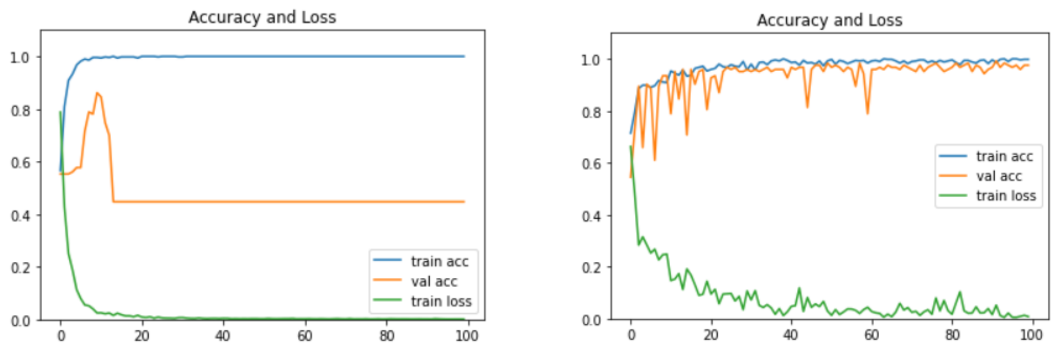
```
Found 198 images belonging to 2 classes.
     Loss: 0.05433
Accuracy: 97.980%
```

Testing Accuracy and loss for AdaGrad

● *Batch Size*

The **batch size** is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated. The number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset.

AlexNet is originally trained on a batch size of 128 samples/batch, considering the large dataset it was trained on, the batch size of 128 was a reasonable choice. As batch size decreases the computational load as well as training time increases. In our case we have to choose a apt batch that can train the model fast without compromising the performance of the model



Comparison of performance between Batch size of 256 samples/batch and 8 samples/batch

From the graph we observe the better performance of smaller batch size, but this costs heavy computational load. A batch size of 16 would be appropriate for the model.

But when dataset size increases small batch size won't be enough to get a better sample of the dataset from a single batch. Thus then we would have to increase the batch size.

Regularization effect of batchsize

Smaller batch size is observed to provide an advantage against overfitting. Thus selecting a smaller batch size would reduce the chance of overfitting.

- ***Early stopping***

  Early stopping was also implemented to avoid overfitting.Early stopping was implemented using a keras callback feature which interrupts the training when a mentioned criteria is fulfilled.

Final Model

From inferences obtained from several experiments we came to a reasonable conclusion on the parameters and final architecture perfect for our use case.

- Filter count **was reduced to half *** since it produced better computational time and better performance

- Activation function was **maintained as ReLU** it self since it provided a much better computational advantage over tanh

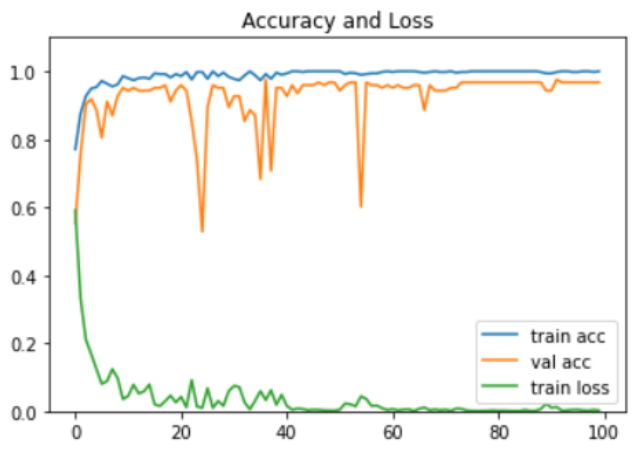
- Optimizer **was opted to be AdaGrad** as it provided slight accuracy over Adam

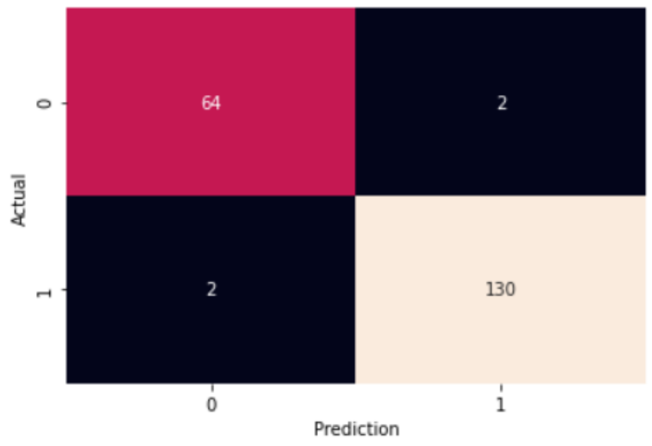
- Different learning rates were experimented and the best outcome was observed for a learning **rate of 0.01.**

- Batch size was selected as 32 since best computation time and accuracy

  ***** this change was avoided while testing in dataset with third class of improper masks

The final model was trained with the Adagrad optimizer for 100 epoch; an early stopping callback was also used in case overfitting happened.
The growth of model accuracy is during training and validation is given below.
After the training and test phase the model was tested on the test dataset it yielded a 97% accuracy and loss of 0.04773. The confusion matrix of model performance is included below.





Growth of training accuracy and loss of the final model and confusion matrix of prediction of the final model on the test dataset

```
Found 198 images belonging to 2 classes.
     Loss: 0.04773
Accuracy: 97.980%
```

Testing accuracy of model

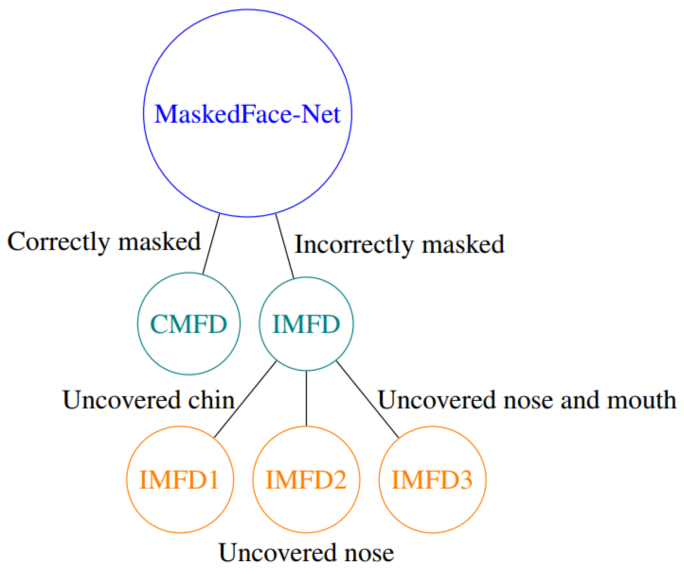
Model predicting on a sample of data from test dataset

Application on a Bigger Dataset

The MaskedFaces-Net

MaskedFaces-Net is a large database with 133,783 images of masked faces belonging to different classes. The dataset was made after the outbreak of global pandemic to help data scientists get a better picture from the mask detection Deep Learning models.

View of the MaskedFace-Net data Tree



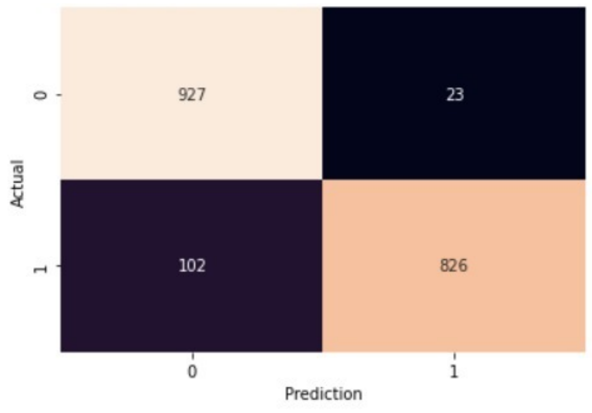
Changes required to test on a larger database

- Filter count was doubled from the final model since the new dataset has a new class of improperly worn masks which requires fetching more intricate features from the image.

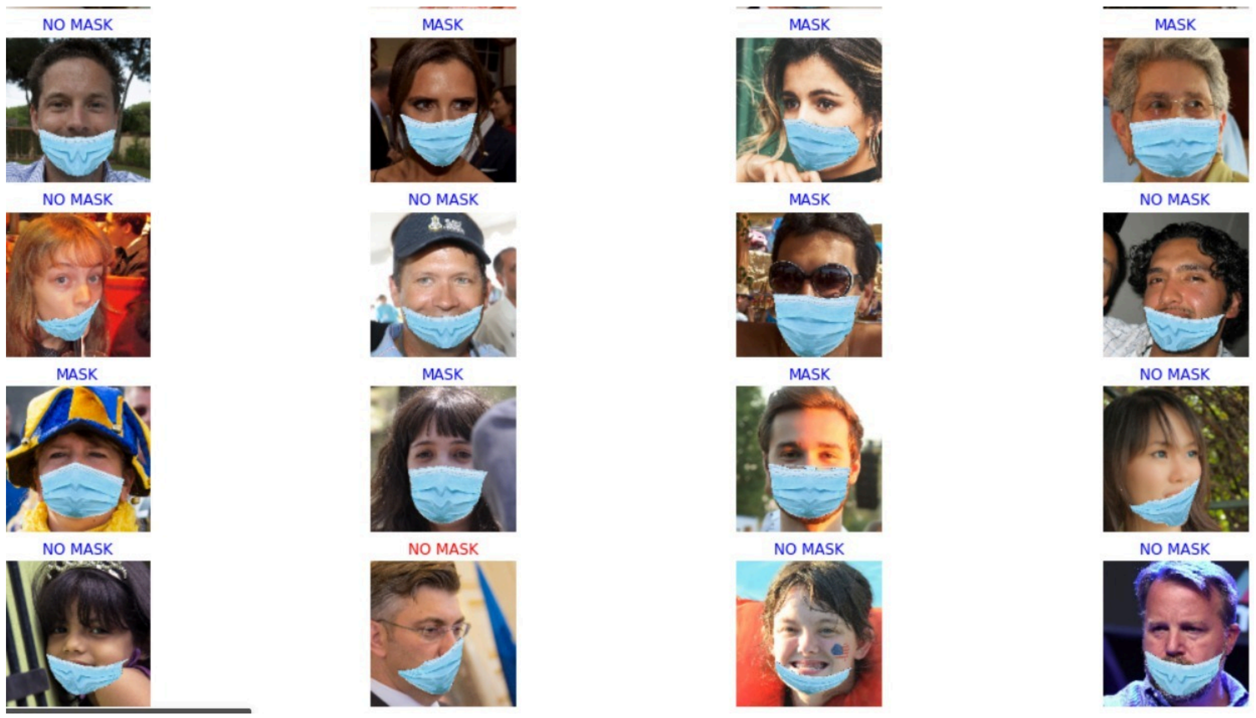
- The batch size was increased to 128 to get a better sample of dataset in each batch.

Results on testing in MaskedFaces-Net

The model produced an accuracy of 93% after testing in the new dataset the model was able to classify new classes of improperly worn masks as no mask worn. Out of the total 1848 images tested the model predicted 1723 images correctly and the remaining 125 wrongly. The result is appreciable because of the fact the model was trained on a smaller dataset and still is able to get an accuracy of 93% in a new dataset with new classes.



Confusion matrix of model testing in a sample of data taken from MaskedFaces Net



A sample of predictions made by the model notice how model classifies the class improperly masked faces as no mask

Conclusion

Our objective was to develop a CNN based Deep Learning model for mask detection with keras and tensorflow keeping the AlexNet architecture as a base architecture. After the project we were able to fulfill the objective to develop a powerful model for classifying the masked, improperly masked and not masked faces.

The model was tested in samples of data from 3 different datasets and all yielded good results with accuracy above 90% each case.

After testing our samples we made slight modifications in architecture like changing number of filters, activation function between other layers(ReLu and Tanh), optimizers and by varying the number of epochs, learning rate, batch size. All these modifications are implemented successfully .

Finally, we plotted corresponding graphs for each modification and tabulated the changes occurring in the output.

References

1. *Alex Krizhevsky,Ilya Sutskever and Geoffrey E. Hinton, 2012: “**ImageNet Classification with Deep Convolutional Neural Networks**”*

*2. Md Zahangir Alom , Tarek M. Taha , Chris Yakopcic , Stefan Westberg , Paheding Sidike, Mst Shamima Nasrin , Brian C Van Essen, Abdul A S. Awwal, and Vijayan K. Asari ,2018: “**The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches**”.*

*3. Soad Samir , Eid Emary, Khaled El-Sayed and Hoda Onsi, 2020 (Article): “**Optimization of a Pre-Trained AlexNet Model for Detecting and Localizing Image Forgeries**”*

*4. “**Artificial Intelligence: A Modern Approach**” Textbook by Peter Norvig and Stuart J. Russell*