Name: Rohith Kumar Poshala

rposhala@buffalo.edu

ImageNet Image Classification

ImageNet is a large visual database with more than 14 million human annotated images and 20k classes designed for developing computer vision algorithms. ImageNet Large Scale Visual Recognition Challenge (ILSVRC) led to developing a number of state-of-the-art algorithms.

As part of this project, 20 classes were picked from the ImageNet dataset.

Building and training Convolutional Neural Network (CNN) models, techniques for preventing overfitting and various data augmentation tools were explored on the above picked 20 classes from the ImageNet dataset and results are discussed.

Our goal is to achieve at least 50% validation/test accuracy.

Picking and loading the Specific dataset for this project:

From 20000 classes present in ImageNet dataset, we have selected below 20 classes for this project:

1. Acropolis
2. Aircraft
3. Anchor
4. Arm chair
5. cobra
6. computer keyboard
7. Cow
8. Cultivated land
9. Delivery Truck
10. aigrette
11. Jellyfish
12. Chinese lantern
13. Motorcycling
14. pay-phone
15. pygmy chimpanzee
16. rhinoceros beetle
17. snowdrift
18. ski jumping
19. giant panda
20. palm

Getting the data:

List of all classes can be found here
Linux/MacOS: https://github.com/mf1024/ImageNet-Datasets-Downloader
Windows: https://github.com/skaldek/ImageNet-Datasets-Downloader

Direct download (~150GB).

These classes are downloaded with the help of downloader.py file attached with this report.

This project has been done on **Google Colab**, so the downloaded datasets and saved into google drive (mounted to the notebook).

Codes for each class selected are found in the GitHub link provided above for respective OS used.

Below are the codes for each class picked for this project and the command to load them with the help of downloader.py

```
!python /content/downloader.py  -data_root /content/drive/My\ Drive/ImageNet \
    -use_class_list True \
    -class_list n00451635 n01747885 n02403454 n09438940 n03173929 n02510455 n027
    n02482650 n02709367 n03085013 n09260907 n00441073 n02174001 n03018712 n02676
    n02686568 n12582231 n01910747 n03902125 n02685082\
    -images_per_class 800 \2
    -multiprocessing_workers 8
```

```
Picked the following clases:
Count: 20
['motorcycling', 'cobra', 'cow', 'snowdrift', 'delivery truck', 'giant pand
a', 'armchair', 'pygmy chimpanzee', 'anchor', 'computer keyboard', 'cultivate
d land', 'ski jumping', 'rhinoceros beetle', 'Chinese lantern', 'acropolis',
'aircraft', 'palm', 'jellyfish', 'pay-phone', 'aigrette']
Multiprocessing workers: 8
  0% 0/20 [00:00<?, ?it/s]
Scraping stats:
STATS For class is_flickr:
 tried 242.0 urls with 215.0 successes
88.84297520661157% success rate for is_flickr urls
0.844525862849036 seconds spent per is_flickr successful image download
STATS For class not_flickr:
 tried 0.0 urls with 0.0 successes
STATS For class all:
 tried 242.0 urls with 215.0 successes
88.84297520661157% success rate for all urls
0.8445264949355015 seconds spent per all successful image download
```

Above classes were stored in my Google drive.

Class names were stored in a list to make retrieving the data from drive easy.

Connected to GPU, loading the data from the drive takes about 10 minutes per class for the first time. But when the connection is lost and you need to reconnect, it takes around 1 minute per class as it uses the data from cache.

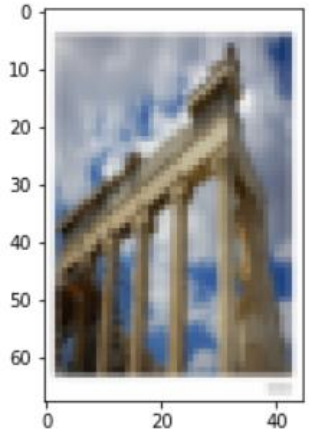
Data was loaded class by class and the images loaded were resized to 450X300X3 and later they were rescaled by 0.15 percent. And 80% of these images from each class were assigned to training data and 20% to testing/validation data.

Shape of the images are 68X45X3 now.

Sample rescaled image:

```
acropolis

<matplotlib.image.AxesImag
```



Train and test labels are one hot encoded to make them compatible to the CNN compiler.

ConvNet 1: A Convolutional Neural Network has been built and trained with 8 convolutional layers with 2 max pooling layers with 5th and 8th CNN layers and even dropouts were included to reduce the overfitting.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 66, 43, 128)       3584
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 41, 64)        73792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 62, 39, 128)       73856
_________________________________________________________________
dropout (Dropout)            (None, 62, 39, 128)       0
```

conv2d_3 (Conv2D)	(None, 58, 35, 128)	409728
dropout_1 (Dropout)	(None, 58, 35, 128)	0
conv2d_4 (Conv2D)	(None, 58, 35, 256)	295168
max_pooling2d (MaxPooling2D)	(None, 19, 11, 256)	0
dropout_2 (Dropout)	(None, 19, 11, 256)	0
conv2d_5 (Conv2D)	(None, 17, 9, 128)	295040
dropout_3 (Dropout)	(None, 17, 9, 128)	0
conv2d_6 (Conv2D)	(None, 13, 5, 128)	409728
dropout_4 (Dropout)	(None, 13, 5, 128)	0
conv2d_7 (Conv2D)	(None, 13, 5, 256)	295168
max_pooling2d_1 (MaxPooling2	(None, 4, 1, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 256)	262400
dense_1 (Dense)	(None, 128)	32896
dropout_5 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 20)	2580

Total params: 2,153,940
Trainable params: 2,153,940
Non-trainable params: 0

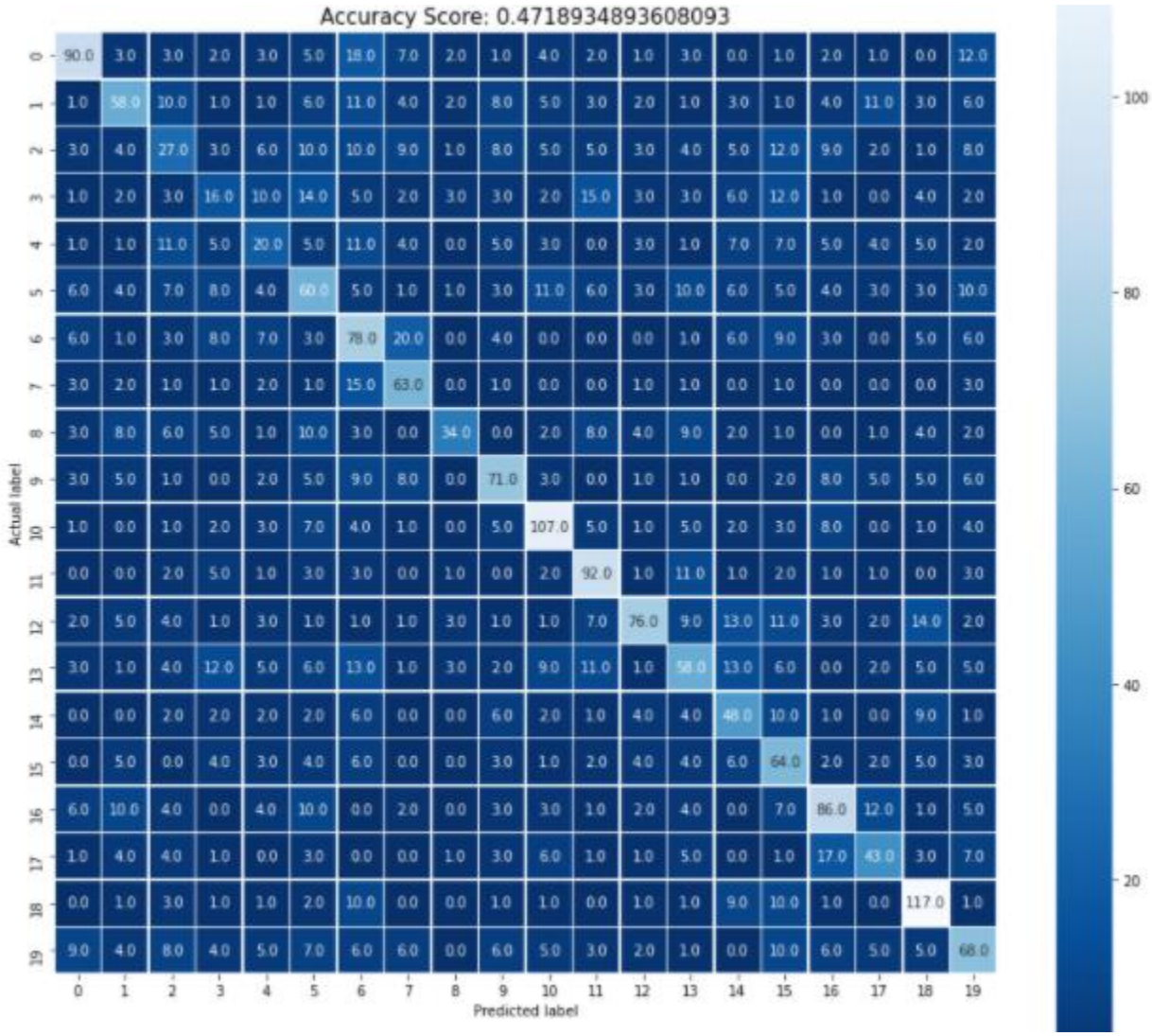
The above model was fit on the training data and validated on the testing data with batch size of 128 and 32 epochs.

Validation/Test loss obtained: `Test loss: 2.42334699630737`

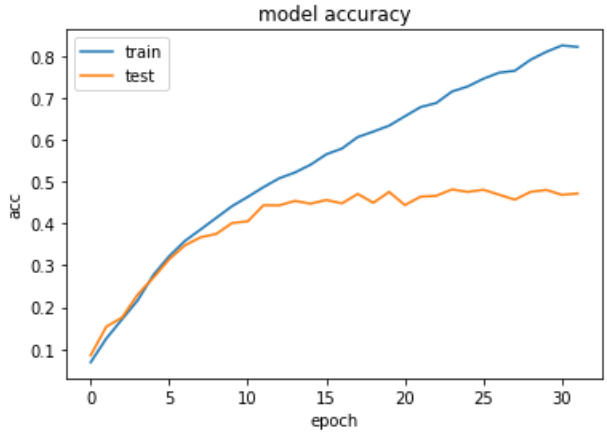
Validation/Test accuracy obtained: `Test accuracy: 0.4718934893608093`

Accuracy of 47.18% on this dataset with the ConvNet built without tuning is good. But we did not reach our goal of 50% accuracy. Confusion matrix was plotted to look into the model performance on each class. And Train Versus Validation accuracy and loss were plotted across all epochs to observe the trend and decide on which the changes which needs to be done to the CNN model.

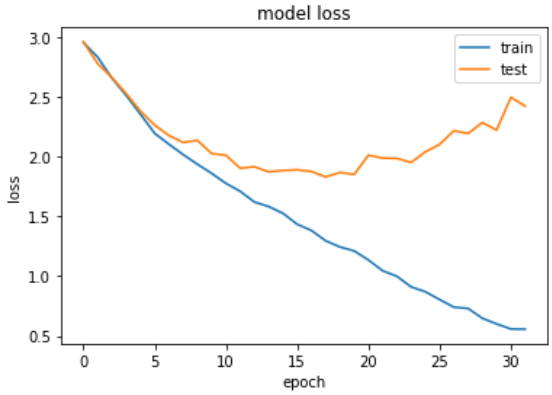
Confusion Matrix:



Train Vs Validation Accuracy:



Train Vs Validation Loss:



Observation:

We can observe that there was no significant increase in test accuracy and reduction in validation loss after 13th epoch. We can see the model overfitting the training data as training accuracy continued to increase till the last epoch. From the above verbose stats, we can observe that the model loss being oscillated. We can understand that the learning rate is not reduced enough to avoid oscillations around the minima.

ConvNet 2: Regularizers were added to the previous ConvNet 1 to reduce the overfitting. And few hyper parameters like patience and factor in Reduce LR On Plateau

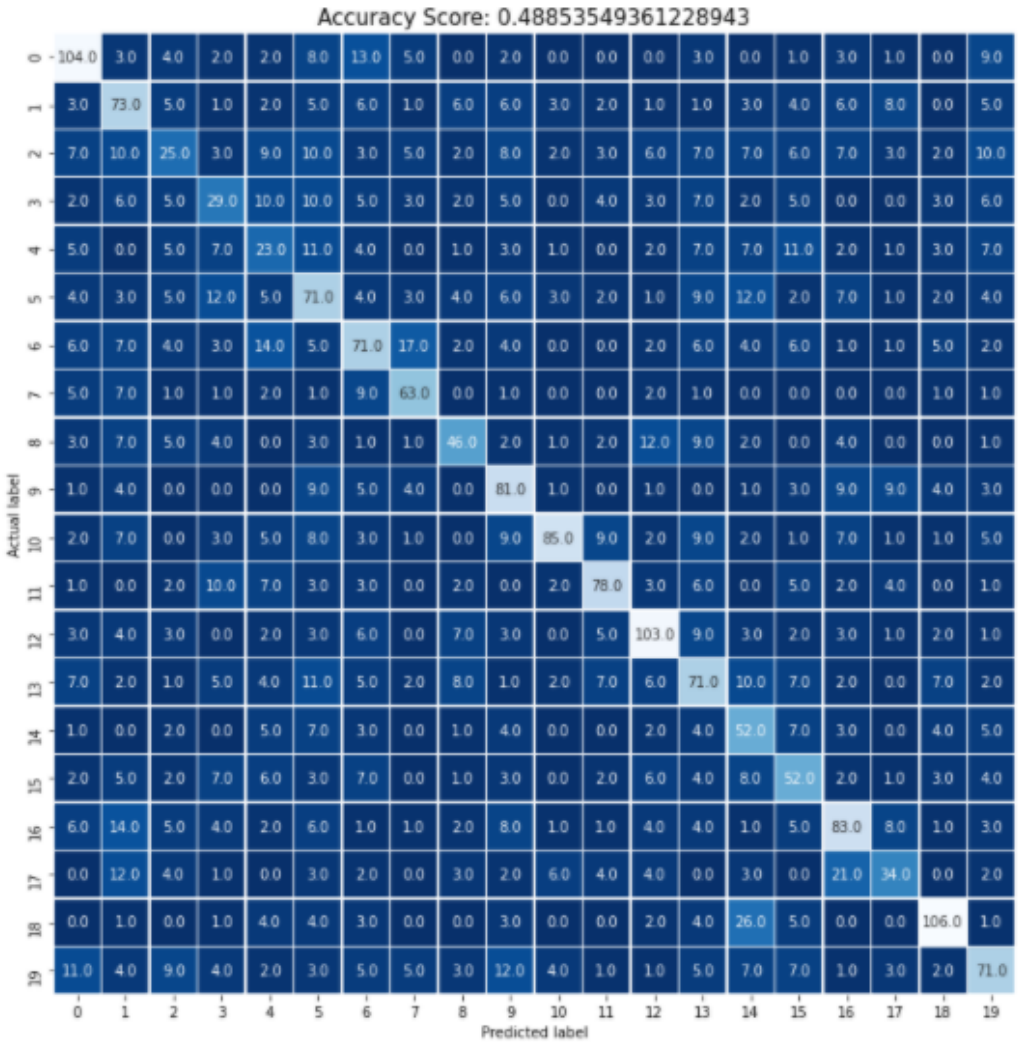
This ConvNet 2(altered ConvNet 1) model was fit on the training data and validated on the testing data with batch size of 128 and 48 epochs.

Validation/Test loss obtained: `Test loss: 1.9704837799072266`

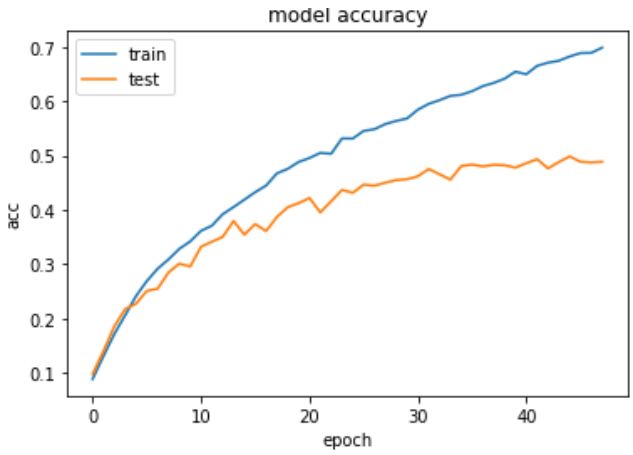
Validation/Test accuracy obtained: `Test accuracy: 0.48853549361228943`

We can see that the validation accuracy got increased to 48.85% which is a percent more than the previous ConvNet.

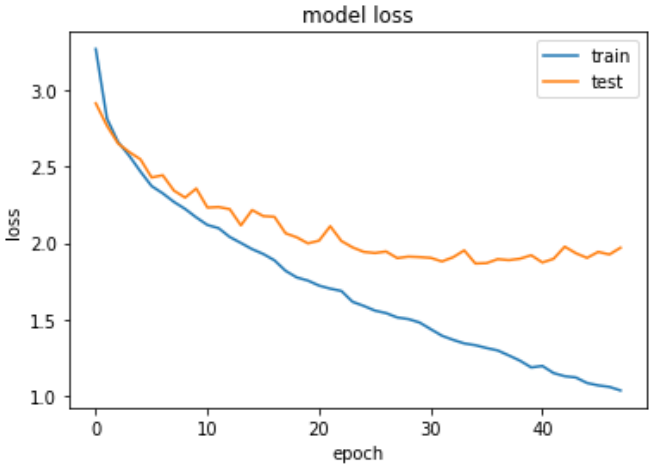
Confusion Matrix:



Train Vs Validation Accuracy:



Train Vs Validation Loss:



Observation:

By reducing patience to 2 and factor to 0.75 of ReduceLROnPlateau and adding regularizers to few layers, we can see that overfitting had been reduced and model seemed to be stabilized at the end of epochs achieving an accuracy of 48.85 % on test dataset.

We can either run for more epochs and look for better results (as the validation loss had not started to hike yet) or we can try to reduce the learning rate factor and patience more and look for better validation accuracy.

ConvNet 3: Same previous ConvNet had been applied but with regularizer applied to additional CNN layers and further reduces learning rate factor.

This ConvNet 3(altered ConvNet 1) model was fit on the training data and validated on the testing data with batch size of 128 and 64 epochs.

Validation/Test loss obtained: `Test loss: 2.043241024017334`

Validation/Test accuracy obtained: `Test accuracy: 0.4208579957485199`

Validation accuracy is observed to be further decreased to 42% from 48.85%. Confusion Matrix, Accuracy and loss monitoring has been done but no special insight was captured from it.

Observation:

We can observe that there was not much increase in the test accuracy (48.85 %) by reducing learning rate and adding more regularization to the layers.

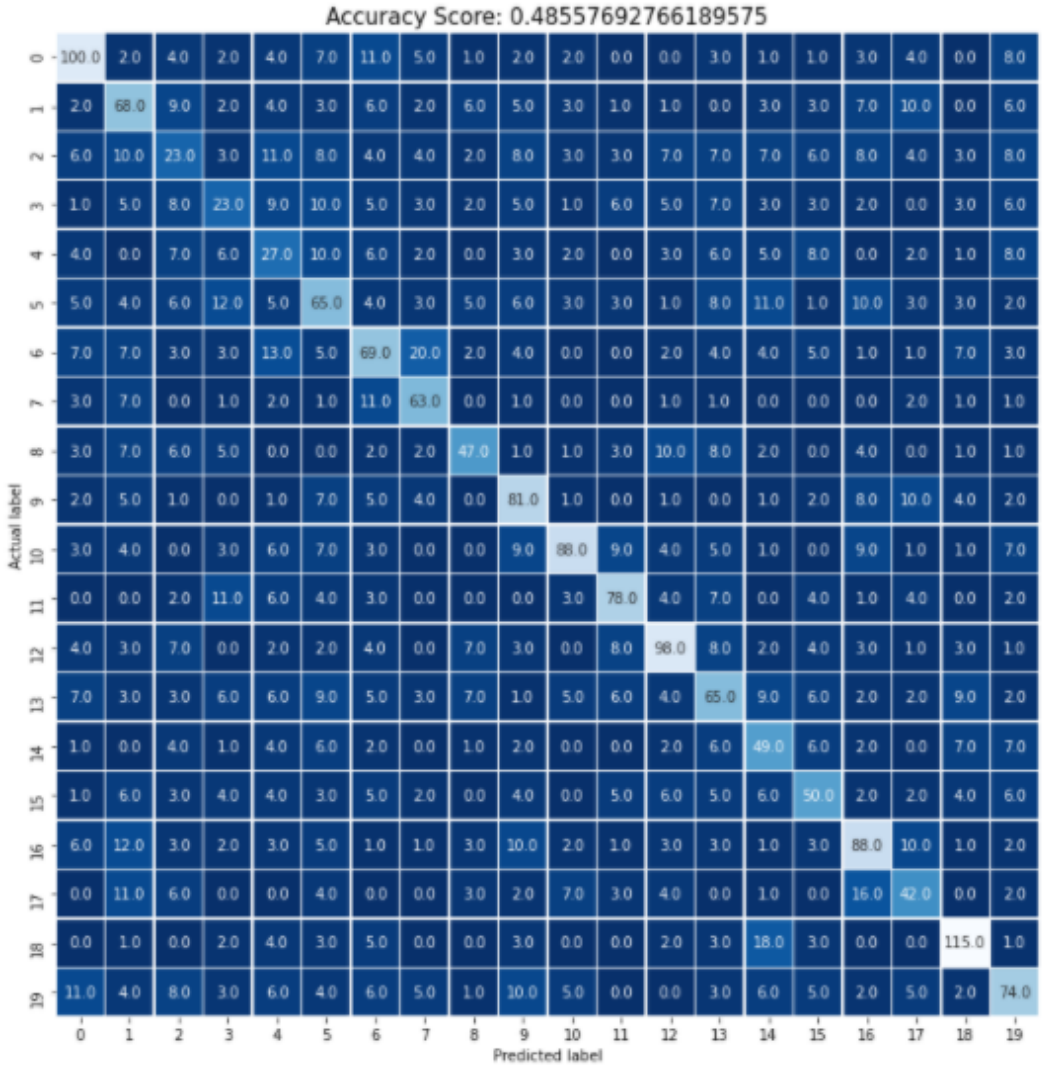
May be, we have to try a greater number of epochs on ConvNet model 2 as we were not sure about the testing accuracy stabilization.

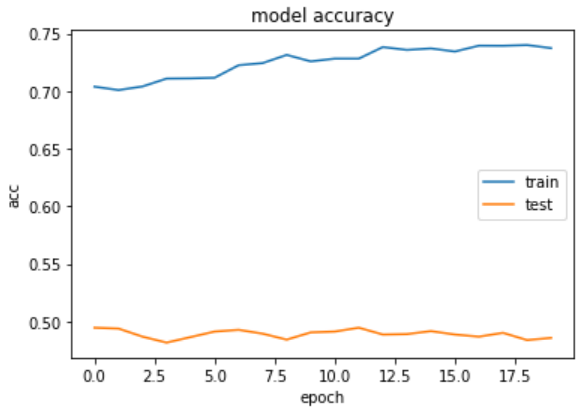
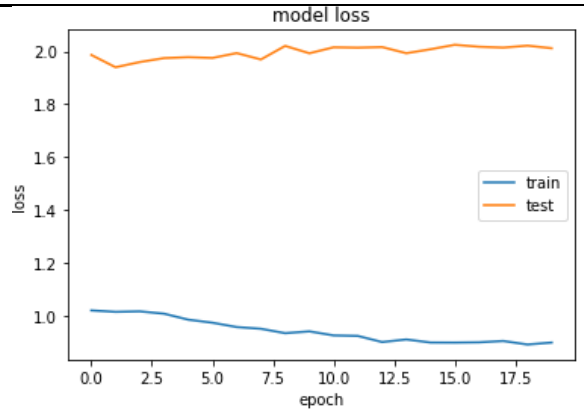
ConvNet 4: ConvNet 2 model is fit on the training data with a greater number of epochs to check the case of overfitting. (more 20 epochs i.e., 64+20 = 84 epochs)

Validation/Test loss obtained: `Test loss: 2.012007474899292`

Validation/Test accuracy obtained: `Test accuracy: 0.48557692766189575`

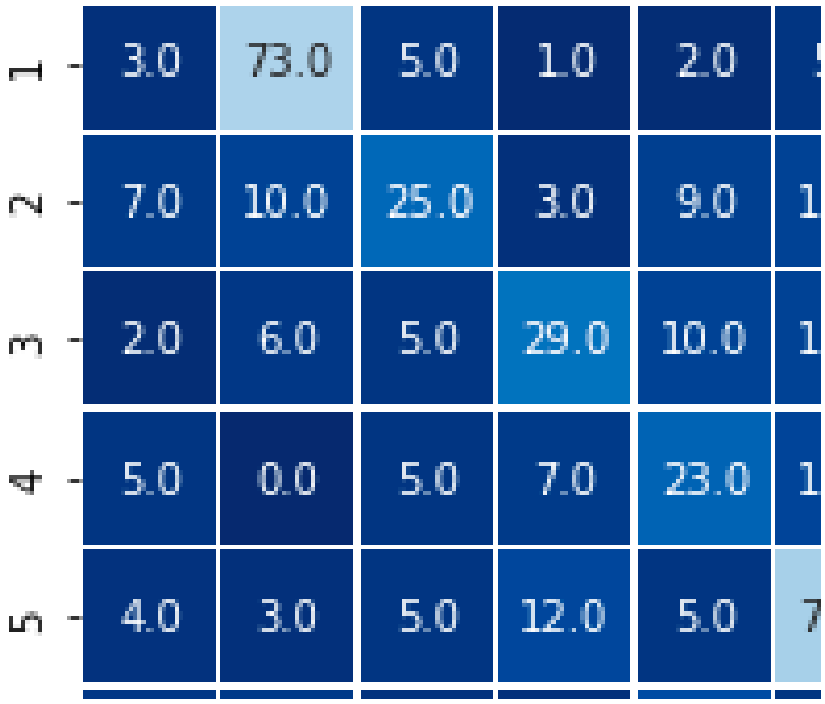
Confusion Matrix:



Training Vs Validation Accuracy (last 20 epochs):	Training Vs Validation Loss (last 20 epochs):
	

Observation:

We can see that the test accuracy did not increase much, it was still around 48.5% but training accuracy went up to 89.5%. Model is overfitting the training data when number of epochs are increased.



Looking at the Confusion Matrix, we can observe that mostly labels 2,3,4 (which are 'anchor', 'armchair', 'cobra') are wrongly classified till now for all the 3 ConvNet models.

So, in order to achieve better testing accuracy, we need to increase the size of training data.

But instead of blindly implementing data augmentation on the entire training data. We may get better results if we add more generated training data for those which are wrongly classified in most of the classes.

Extracting starting indices of the labels 2,3 and 4. As the training and testing images and labels are in a sequence, this helps us with extraction of the images.

100 training images of each of the above particular classes were extracted. Labels were generated for the extracted images.

For Data Augmentation, we have used Image Data Generator for rotation, random cropping(zoom), horizontal flip and vertical flip. This is data augmentation function is then fitted to the recently formed training data. Each of the above training image is passing to the function with batch size 1 to get the generated images which can be included into training data. The same one hot encoding and reshaping which was earlier applied to the original formed training is applied to the augmented data.

Finally, the augmented data is combined with the original data and a new final training data and test/validation data.

ConvNet 5: Same ConvNet 1 model is fit to the latest training data (with augmented data)

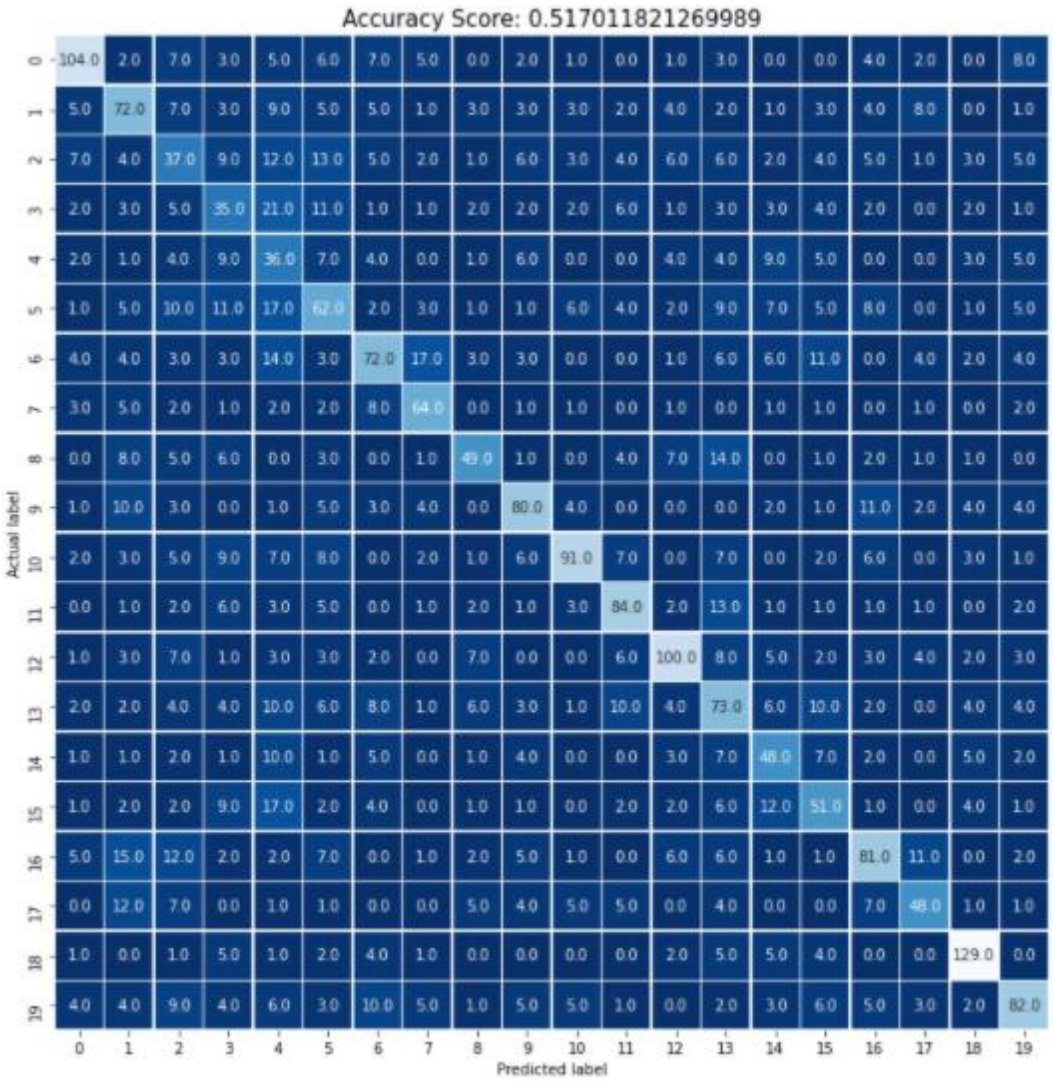
This ConvNet 5 model compiled on batch size of 128 and 48 epochs.

Validation/Test loss obtained: `Test loss: 2.1945579051971436`

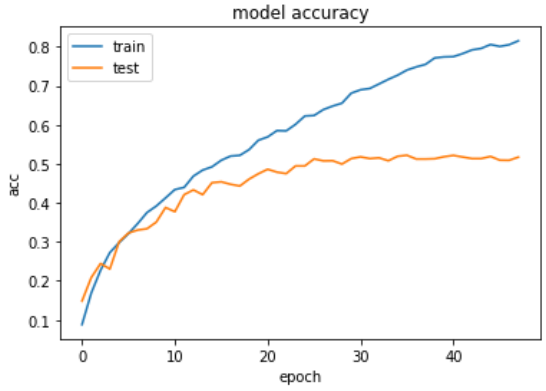
Validation/Test accuracy obtained: `Test accuracy: 0.517011821269989`

Our main goal was reached by achieving 51.7% with the help of specific data augmentation.

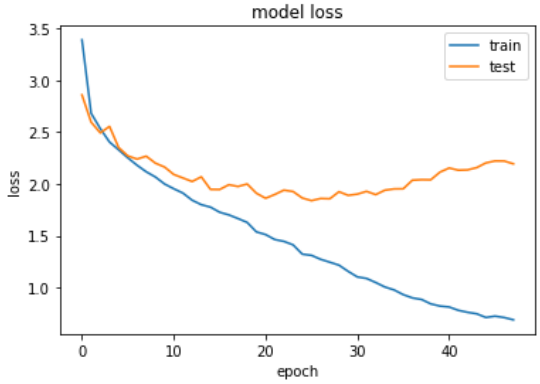
Confusion Matrix:



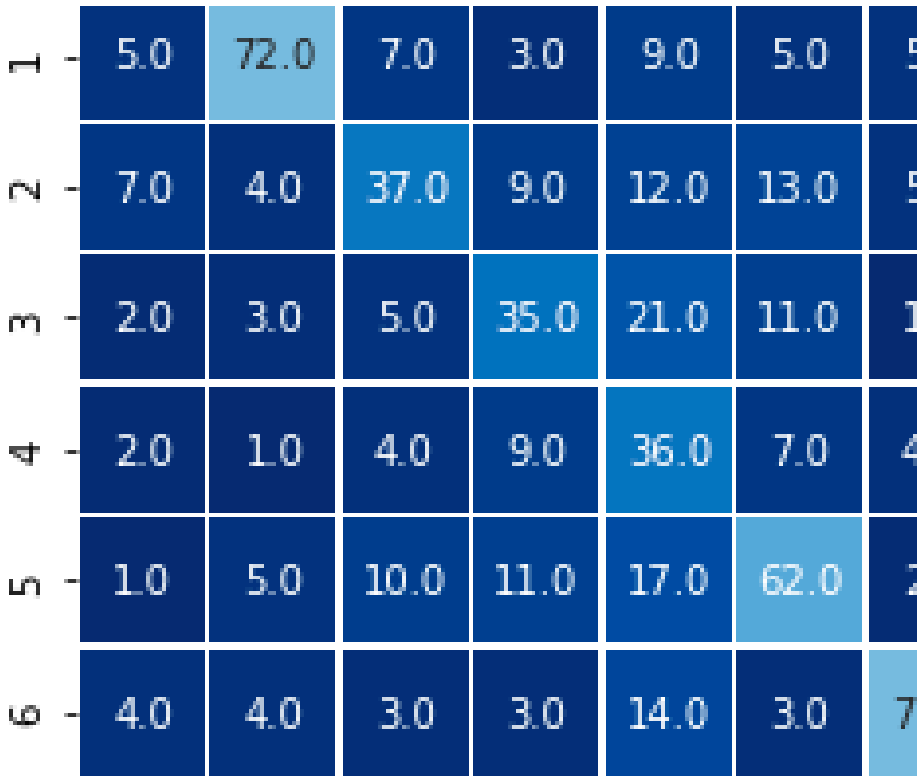
Training Vs Validation Accuracy:



Training Vs Validation:



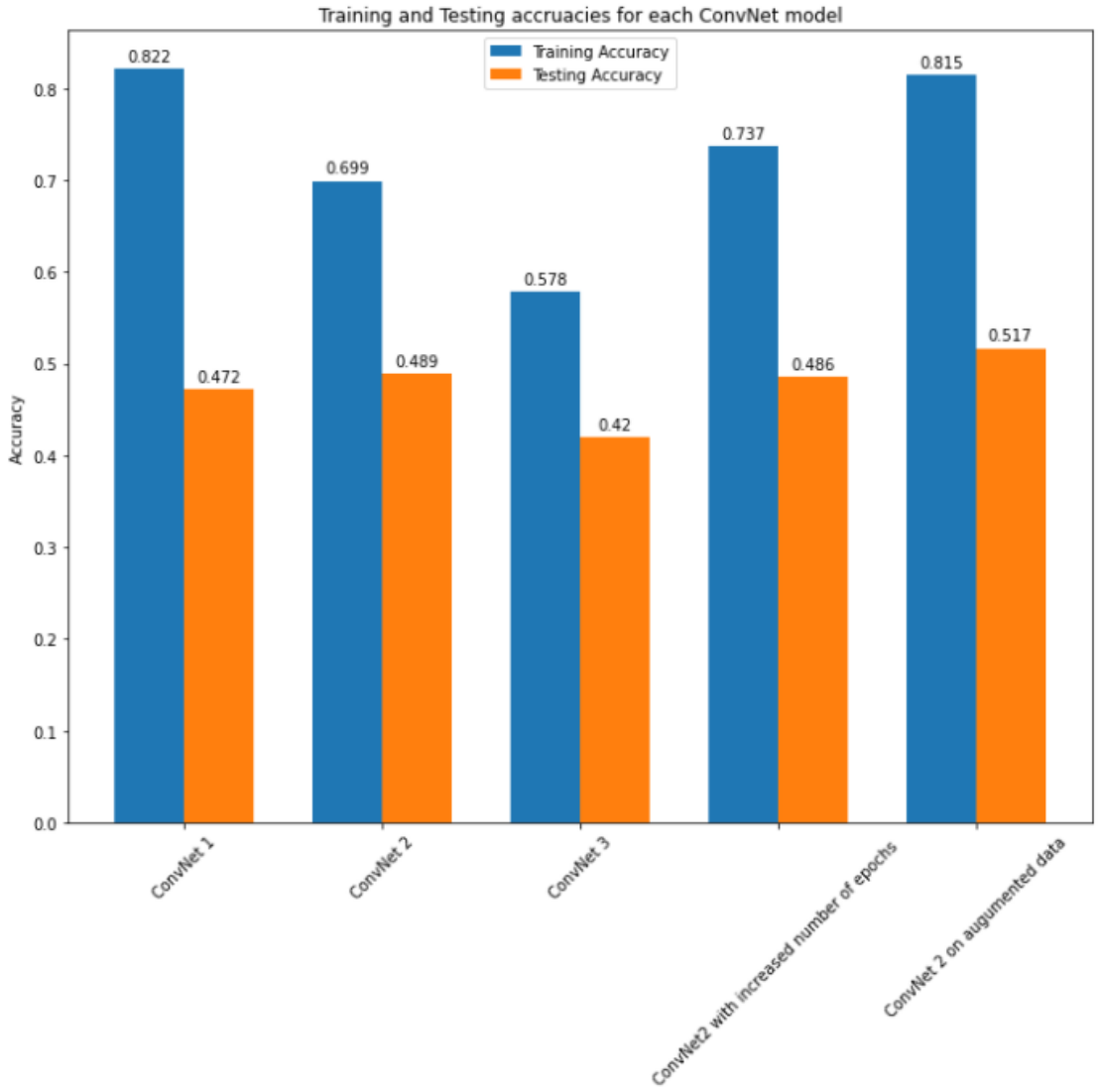
Observation: We can observe an increase in test accuracy. Augmenting a particular section of data which was misclassified had yielded a better generalization of the model. Test/Validation accuracy: 51.7%



We can see that labels 2,3 and 4 are less misclassified this time. So, a little more addition of augmented data can be beneficial.

A little more data of these classes was augmented and the same model was fit to that training data but it was found that model performance on Validation/test data was very much similar to this case (This model was also included in the notebook attached). May be picking more labels which are correlated to the most mis-classified labels for data augmentation would help us increase test accuracy.

Above ConvNet models were compared and below are the comparisons and the deductions:



Observation:

From the comparison, we can observe that model can be restricted to not overfit further but that doesn't always necessarily increase test accuracy of the ConvNet. Model may not increase the validation/test accuracy because may be train and test dataset are from different distributions or the model need more training data to perform better on testing data.

But sensible augmentation of the available training data can help us increase the model performance on the validation data.