Neural computing and deep learning

Dog Breed Classification

MOD006568

SSID:2021497

Academic_Year:2020/21

Table of Contents

Introduction and Background

In this project, I intend to recognize the breed of the dogs through data obtained from Kaggle and perform a thorough data analysis. Implemented using Convoluted Neural Network (CNN) model where the model is thoroughly trained and then analysis is performed. Jupyter notebook is used for the implementation of the model.

Kaggle Database

There is a separate database for training the model and then subsequently testing the model. All these databases have their own unique identification. There is 100+ dog pictures from different races. The intention of the project is to recognize and classify breed of the dogs. To do an extensive training, there are 10K+ photographs of various breed. However, it is to be understood that it requires a lot of computational resources which is not possible for academic projects. Hence, the pretrained models are used which have been trained on ImageNet, COCO etc. For Image classification, transfer learning is generally used. Leveraging this, the current project uses NASNet pre trained network on ImageNet database based on architecture made by Convoluted Neural Networks.

Literature Review

Before starting with the project, there was an extensive literature review done to understand how such and comparable projects were undertaken in past. I have studied a few convolutional neural network architecture and their performance on this dataset.

The following were studied:

1. CNN Architecture like NASNet Large, DenseNet201, MobileNet-v2 have been studied which were reviewed by (Yufeng et al., 2020)
2. The paper by (Yufeng et al., 2020) shows how Support Vector Machine is used for enhancing accuracy in the performance.

It is important to understand that the performance of Convoluted Neural Network is understood through costs achieved through training and accuracy. The NasNet-Large model is much better and has a higher accuracy and convenience compared to other models. The dog categorization model presented was as per yolov3 that used ordinary pet dogs for sampling. (C.Wang et al., 2020). Yolov3 recognises facial characteristics of the pet dog and mark the category of the dog as per its face selection. As per the technology experiment, this procedure pinpoints the face position of the pet dog with more speed and precision. Thus, it help in detection and categorization of the pet dog based on its face.

As per (Borwarnginn et al., 2019), there was a previous attempt to recognize 100+ breeds of dog. Transfer learning has been the model leveraging image augmentation and Convoluted neural networks. Using this, an accuracy of ~90% has been achieved using different settings and simulations. From this study, I have tried using data augmentation with different transformations on my dataset which showed good results when compared.

Stanford dog repositories was used by Yizhou Chen et al. in their paper to categorize dog breeds based on two separate networks. They accurately categorize the pet dog into 40 breeds in their ultimate VGG19 model that is around 80% accurate. With the help of information technology program they display the use of convolutional neural networks. This program has a common server in centre and clients are mobile. This program also has a libraries to assess the neural network without use of internet.

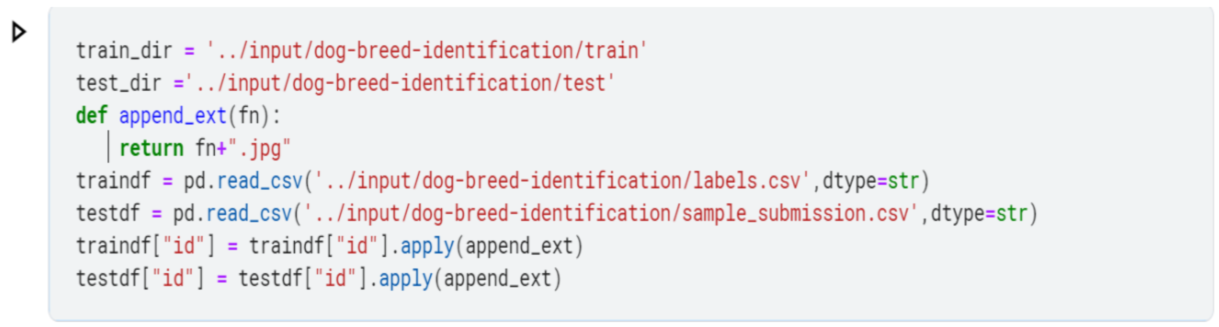
There was an experiment done with basic convolutional layers (with 6 convolutional layers) and classified dog breeds with 90% accuracy. After studying the above papers, I have trained my model with pre-trained architecture and concluded that the performance of the NasNet-Large on my model is high compared to others.

Pre-processing of the Data

Data pre-processing is done to remove all the unwanted and redundant data. The images in the current dataset are of varied sizes with a label on each image.

1. Loading of the Dataset

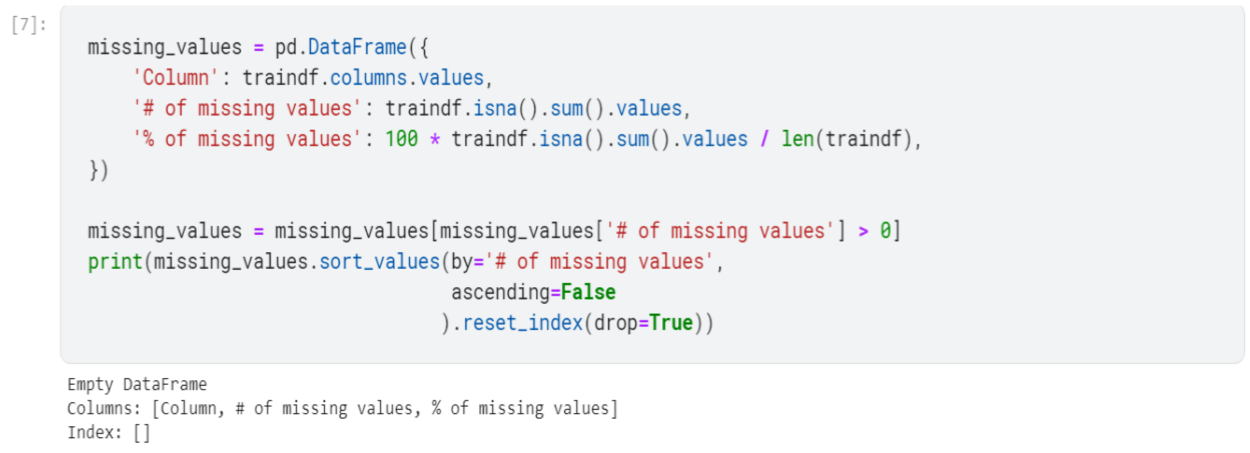
The data from the folders (Train & Test) is loaded into the file implementing the following command.



```
train_dir = '../input/dog-breed-identification/train'
test_dir ='../input/dog-breed-identification/test'
def append_ext(fn):
    return fn+".jpg"
traindf = pd.read_csv('../input/dog-breed-identification/labels.csv',dtype=str)
testdf = pd.read_csv('../input/dog-breed-identification/sample_submission.csv',dtype=str)
traindf["id"] = traindf["id"].apply(append_ext)
testdf["id"] = testdf["id"].apply(append_ext)
```

2. No Empty or Redundant Data

The following code is used to understand if the dataset has any missing values.



```
[7]:
missing_values = pd.DataFrame({
    'Column': traindf.columns.values,
    '# of missing values': traindf.isna().sum().values,
    '% of missing values': 100 * traindf.isna().sum().values / len(traindf),
})

missing_values = missing_values[missing_values['# of missing values'] > 0]
print(missing_values.sort_values(by='# of missing values',
                                 ascending=False
                                ).reset_index(drop=True))
```

```
Empty DataFrame
Columns: [Column, # of missing values, % of missing values]
Index: []
```

It is to be understood that the images are of various sizes, resolution along with various brightness and darkness. The few examples are as below:

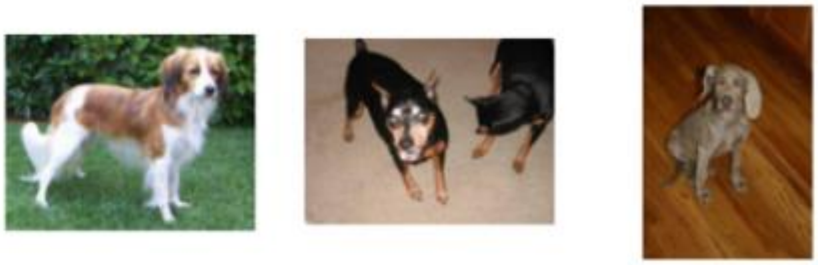


Fig. Sample images from dataset

It is important to note that the images to be processed should be of high quality and should not have noise as it may become difficult to analyze in that case.

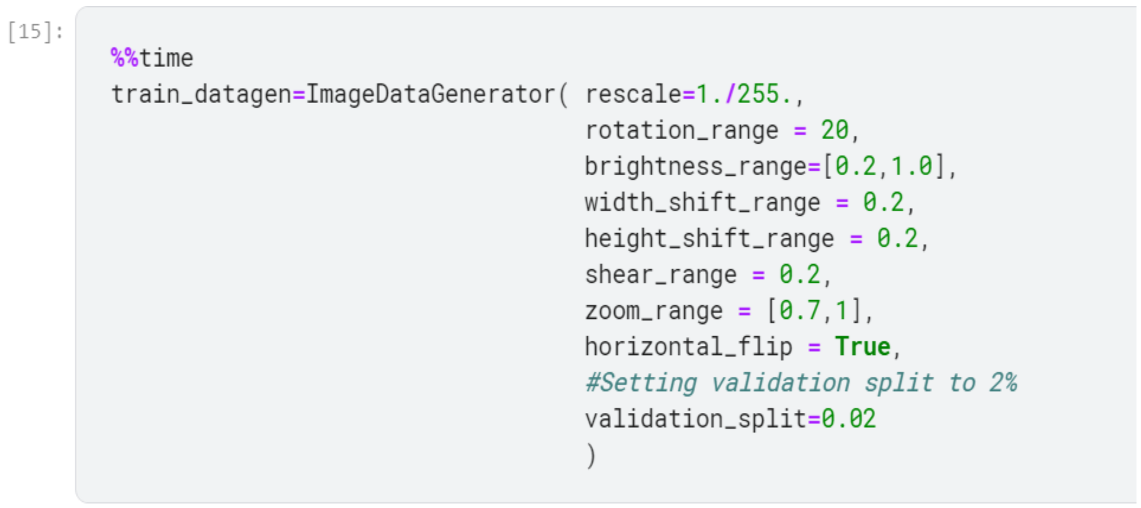
3. Excessive Overfitting can be counterproductive and hence important to reduce it

In our neural network architecture, there are around 85 million parameters. Since there are ImageNet's 1000 classes requiring each training sample, it limits the picture to be mapped to label to 10 bits. Unless there is an excruciating overfitting done, the learning won't be insignificant.

Augmentation of Data

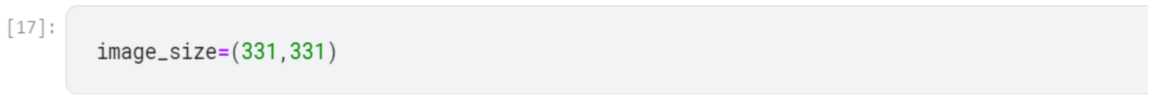
This process is done to reduce the overfitting on image data. An Image generator in Keras is implemented to generate the images. This happens in a dynamic fashion and it may introduce redundancy as a thorough database is not maintained. It has a feature of image rotation technique which rotates the image at various angles. The shift in the technique helps in positioning the images uniformly which helps in uniform data analysis. Image data generator helps in flipping of the image using horizontal image flipping. The generator also helps in setting and adjusting the brightness of the image. It also helps in setting up zoom level of each and every image. The function of validation split is used for splitting training data into two- Training set and Validation set.

After a few attempts, I decided on parameters listed below because they worked best for my dataset. Testing data only requires rescaling because the model was trained on training data.



```
[15]:
%%time
train_datagen=ImageDataGenerator( rescale=1./255.,
                                  rotation_range = 20,
                                  brightness_range=[0.2,1.0],
                                  width_shift_range = 0.2,
                                  height_shift_range = 0.2,
                                  shear_range = 0.2,
                                  zoom_range = [0.7,1],
                                  horizontal_flip = True,
                                  #Setting validation split to 2%
                                  validation_split=0.02
                                  )
```

As mentioned in the figure below, the images containing the dogs are resized to 331*331 pixels. After the pre-processing of the data, the images are converted to a single shape.



```
[17]:
image_size=(331,331)
```

Experiments

In this section, explanation about architecture, process of training, and evaluation of models I have experimented on is discussed.

Convoluted Neural Network Architecture

Prototype 1:
My first prototype comprises of primary convolutional model that has 12 layers that includes activation layer, thick layer, dropout and batch normalization layers. Through this model, I could achieve accuracy of almost 87% on train data but the log loss on the test data was lesser.

Prototype 2:
My base model has been trained with Xception, InceptionResNetV2, and NASNetLarge CNN architectures.

NasNetLarge outperformed the other two networks in accuracy and number of epochs. Though NasNetLarge requires lot of memory and training time for training parameters it has achieved 94% accuracy in 10 epochs. This complex network reaches higher accuracy in short time because of its Controller Recurrent network and CNNs in the architecture.

As shown in the table below, the model best worked when used with ImageNet validation dataset. The accuracy is shown below:

Experimented prototypes:
These are the experimented models, both InceptionResNetV2 and NasNetLarge are performing nearly similar, ~1% difference can be observed in training accuracy and test log score.

Models	**Training Accuracy**	**Kaggle score**
Xception	86.35%	Not- submitted
InceptionResNetV2	93.64%	0.3590
NASNetLarge	94.61%	0.2392

Fig comparison between experimented models with train and test data.

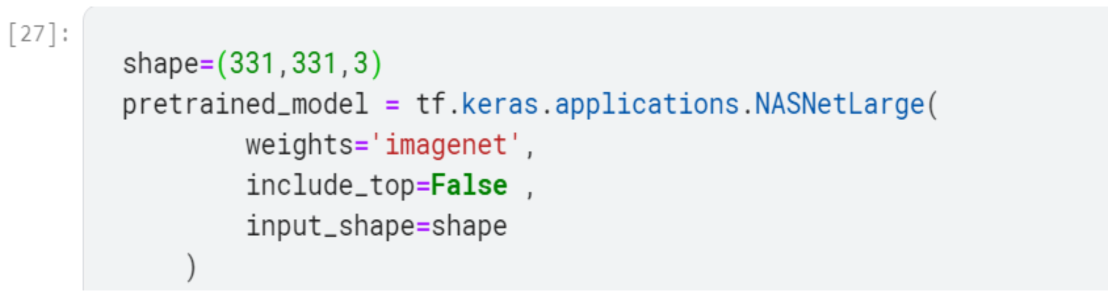
Findings:

- NASNetLarge network need more training time but log loss score is lesser as compared to Xception and InceptionResNetV2

- NASNetLarge network works more accurately (almost 92-94%) with test data
- It reaches more accuracy in less number of epochs nearly less than half of the other two models.

Workflow of the ImageNet Classes

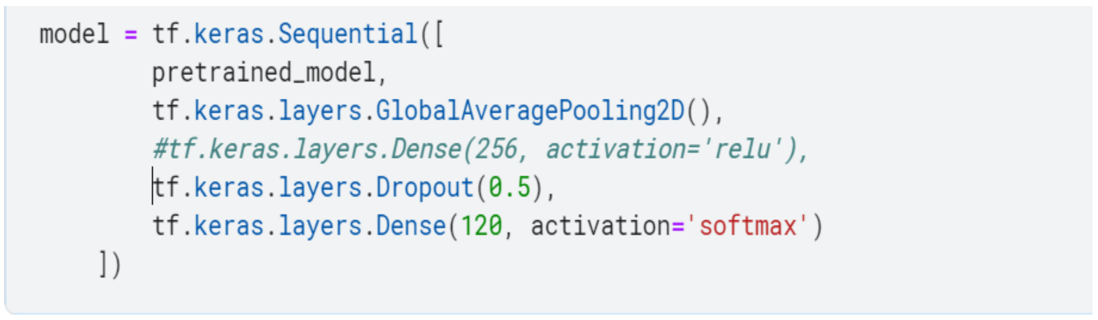
1. To start with, from imagenet database, a pre-trained model was taken. To use this model as pre-trained network as a feature extractor, this base model was preserved. Asequential model was made on top of this base model. Input shape is resized for NasNetLarge architecture as 3331 x 331.



```
[27]:
shape=(331,331,3)
pretrained_model = tf.keras.applications.NASNetLarge(
        weights='imagenet',
        include_top=False ,
        input_shape=shape
    )
```

Include_top=False states that the layer converts all the weights from trainable to non-trainable means that its value does not change during training.

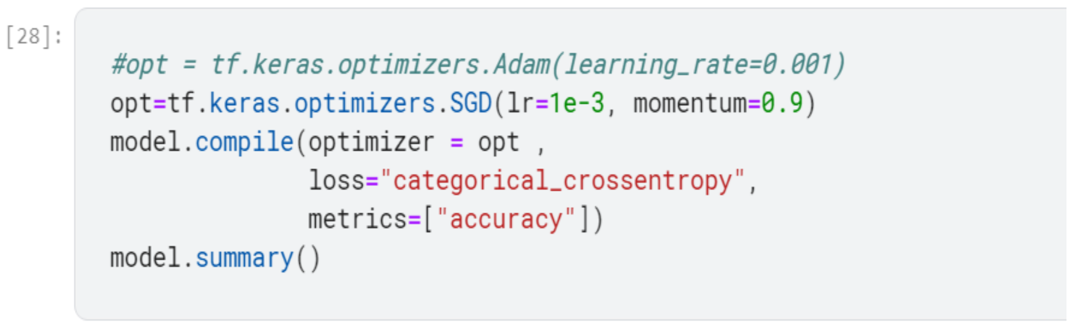
2. A sequential model on output top of layers of base model is created. After loading pre-trained model, global average pooling, a dropout, and a dense output layer are added to classify. The output layer consists of 120 dog breeds and the SoftMax activation function is used to predict the classes on basis of probabilities. This steps club all extracted features into single image.



```
model = tf.keras.Sequential([
        pretrained_model,
        tf.keras.layers.GlobalAveragePooling2D(),
        #tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(120, activation='softmax')
    ])
```

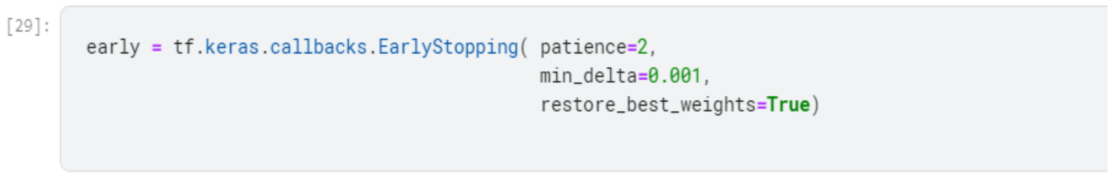
Dropout (0. 5) - The dropped-out neurons shall not participate in any propagation that may be forward or backwards. It greatly helps in reducing any overfitting.

3. The model is compiled using categorical cross entropy and accuracy is optimized using Adam optimizer. It was also compared with an alternative optimizer, Adam, RMSprop, and SGD for testing; in my final model I have used 0.001 learning rate after training my model with 0.1, 0.5, 0.01, and 0.001 learning rates.



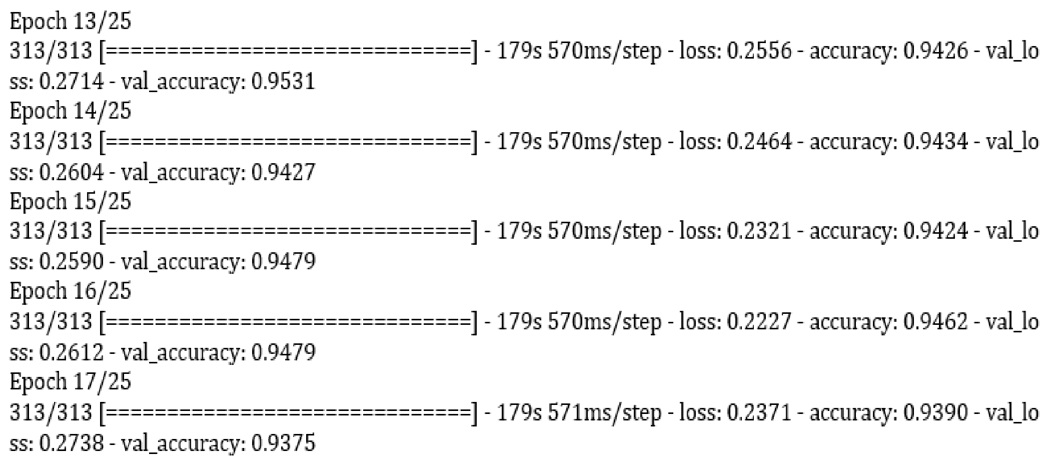
```
[28]:
#opt = tf.keras.optimizers.Adam(learning_rate=0.001)
opt=tf.keras.optimizers.SGD(lr=1e-3, momentum=0.9)
model.compile(optimizer = opt ,
              loss="categorical_crossentropy",
              metrics=["accuracy"])
model.summary()
```

I have implemented early stopping to monitor loss in my model with it triggering post 17 epochs and final loss of 0.2 with ~93% accuracy for training data and ~93% accuracy for validation data.



```
[29]:
early = tf.keras.callbacks.EarlyStopping( patience=2,
                                          min_delta=0.001,
                                          restore_best_weights=True)
```

The model is trained on train generator data from data augmentation. There are twenty-five epochs, and every epoch is of size 313.



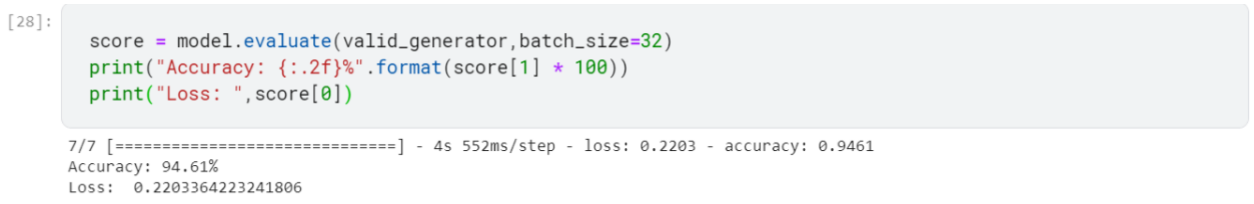
```
Epoch 13/25
313/313 [==============================] - 179s 570ms/step - loss: 0.2556 - accuracy: 0.9426 - val_lo
ss: 0.2714 - val_accuracy: 0.9531
Epoch 14/25
313/313 [==============================] - 179s 570ms/step - loss: 0.2464 - accuracy: 0.9434 - val_lo
ss: 0.2604 - val_accuracy: 0.9427
Epoch 15/25
313/313 [==============================] - 179s 570ms/step - loss: 0.2321 - accuracy: 0.9424 - val_lo
ss: 0.2590 - val_accuracy: 0.9479
Epoch 16/25
313/313 [==============================] - 179s 570ms/step - loss: 0.2227 - accuracy: 0.9462 - val_lo
ss: 0.2612 - val_accuracy: 0.9479
Epoch 17/25
313/313 [==============================] - 179s 571ms/step - loss: 0.2371 - accuracy: 0.9390 - val_lo
ss: 0.2738 - val_accuracy: 0.9375
```

Validation accuracy dropped from 13th epoch from ~95% accuracy to 93% on 17th epoch.

4. In the fine tuning, the complete model is unfreezed and then the model is further trained with a new set of data. The learning rate starts from 0.1 to 0.001. The hyperparameters are selected in the experimental stage and then the testing of model is done on the test dataset. The NasNet-Large could train the model with sample size of 32 with more accuracy with learning rate of 0.001 with adam optimizer.

Conclusion and Results

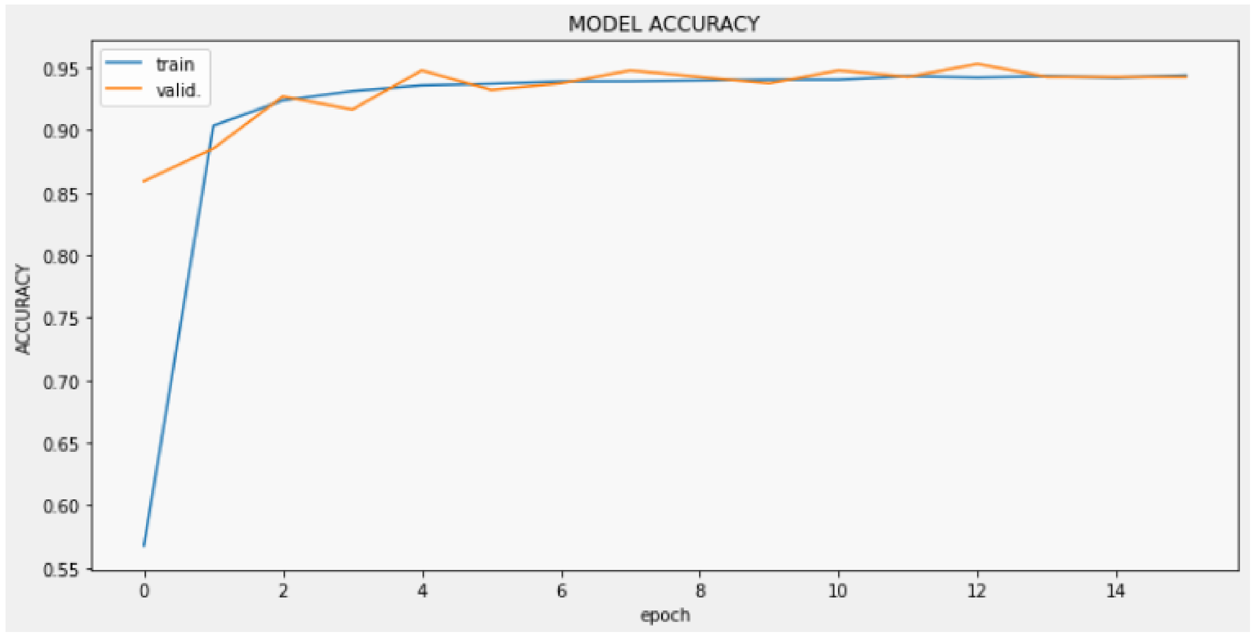
The results have been very accurate for a tiny data set and simple Convoluted Neural Network Architecture.



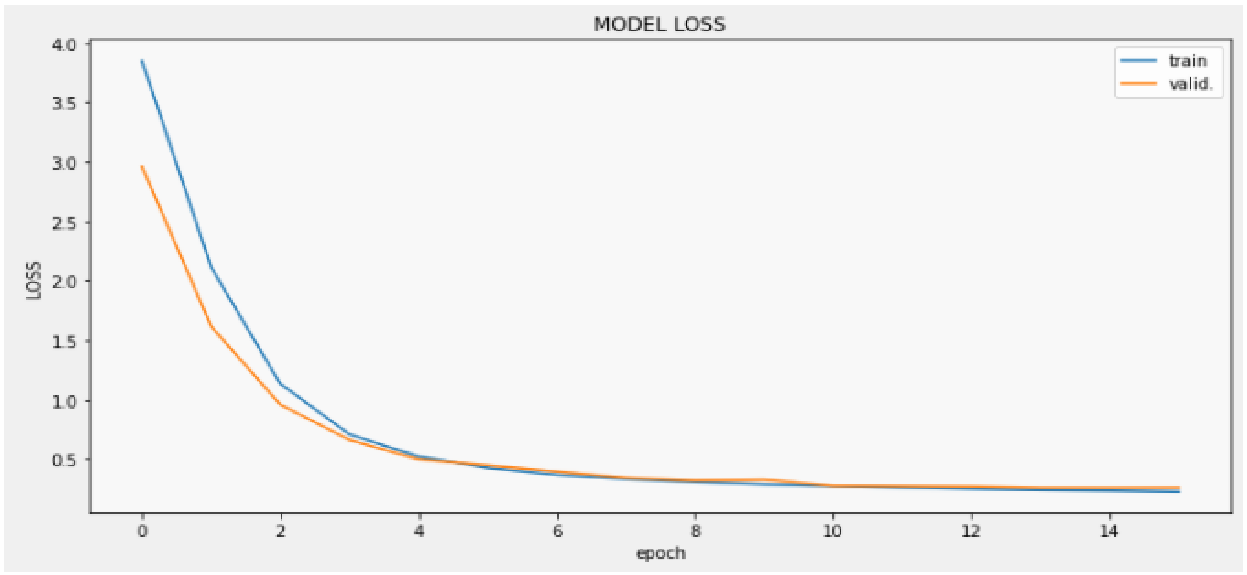
```
[28]:
score = model.evaluate(valid_generator,batch_size=32)
print("Accuracy: {:.2f}%".format(score[1] * 100))
print("Loss: ",score[0])
```

```
7/7 [==============================] - 4s 552ms/step - loss: 0.2203 - accuracy: 0.9461
Accuracy: 94.61%
Loss:  0.2203364223241806
```

With each passing epoch, the accuracy of data analyzed in the training and test becomes better. After couple of epochs, the analysis has received approximately 90% accuracy. The NASNet Large architecture is known to achieve extremely high accuracy of ~94%.



It is observed that validation data is performing well along with training data. It is implied from the learning graphs that since there is no huge gap between two curves; model is trained well and not overfitted.



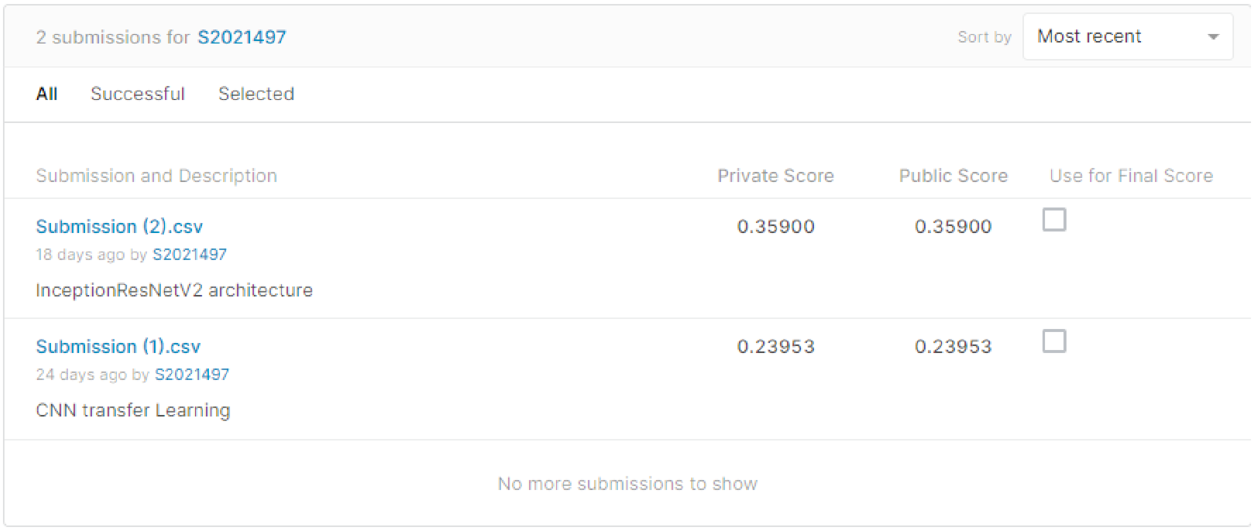
Loss for two curves is nearly same from 4th epoch which need to be observed. All these finding show that this Complex NasNetLarge network is accurate in classification when there is less data. for Large datasets, this network may not perform well because it takes nearly twice learning parameters, lot of memory, and lot of training time which is not feasible.

Submission to Kaggle

Following are two submissions I have submitted in kaggle

1) NASNet Large Architecture (377 out of 1280 teams participated and achieved 0.23953)
2) Inception ResNet-v2 deep architecture has achieved 0.3590 log score.

There has been an achievement of 0.23953 with NASNet Large Network.



2 submissions for S2021497 Sort by Most recent

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
Submission (2).csv 18 days ago by S2021497 InceptionResNetV2 architecture	0.35900	0.35900	☐
Submission (1).csv 24 days ago by S2021497 CNN transfer Learning	0.23953	0.23953	☐

No more submissions to show

References

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, et al. 2017. *Mobilenets: Efficient conolutional neural networks for mobile vision applications*, *CoRR*, vol. abs/1704.04861.

Borwarnginn, P., Kusakunniran, W., Karnjanapreechakorn, S. *et al.* 2019. *Knowing Your Dog Breed: Identifying a Dog Breed with Deep Learning. Int. J. Autom. Comput*. [online] Available at: <https://doi.org/10.1007/s11633-020-1261-0>

C. Wang, J. Wang, Q. Du and X. Yang, 2020. *Dog Breed Classification Based on Deep Learning*, 13th International Symposium on Computational Intelligence and Design (ISCID), 2020, pp. 209-212, doi: 10.1109/ISCID51228.2020.00053.

Yizhou Chen, Xiaotong Chen, Xuanzhen Xu. 2018. *Dog Breed Classification via Convolutional Neural Network*, [online]. Available at:<http://noiselab.ucsd.edu/ECE228_2018/Reports/Report17.pdf>

Yufeng Zheng, Hongyu Wang, Yingguang Hao, 2020. *CNN study of convolutional neural networks in classification and feature extraction applications,* [e-journal] Proc. SPIE 11395, Big Data II: Learning, Analytics, and Applications, 113950K, 21 April 2020 Available at: <https://doi.org/10.1117/12.2560372>