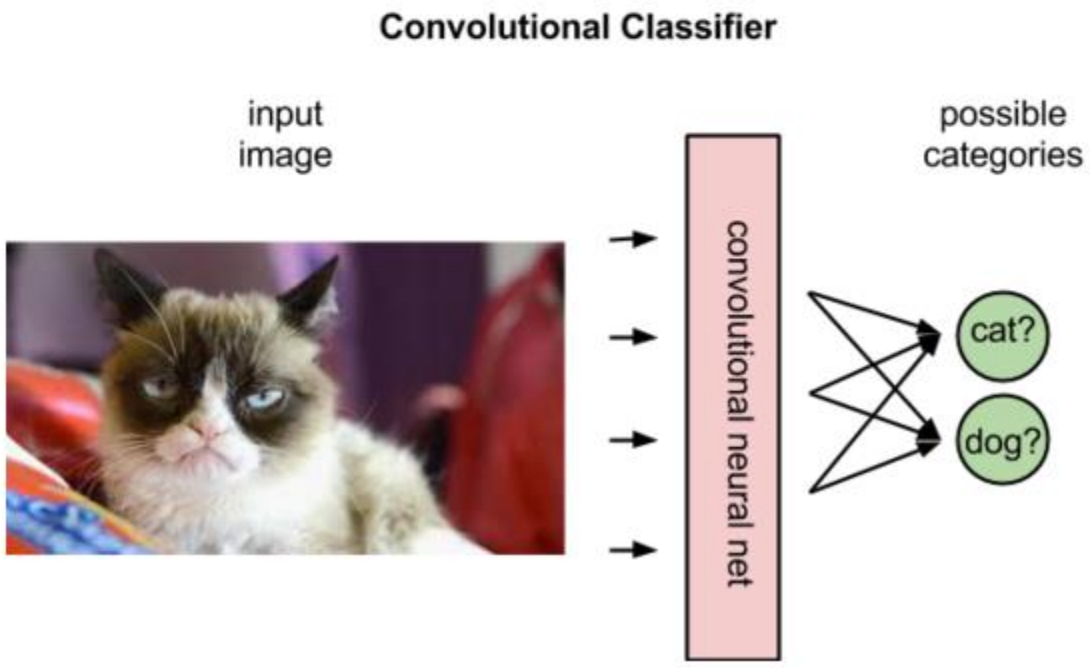
Image Classification: Fine Tuning CNN Model via Transfer Learning

1. Definition



Project Overview

Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize dogs and cats could apply when trying to recognize other animals like lion or tiger.

The problem we are tackling is having less data for training and to still classify it at a higher accuracy.

What if we don't have enough of the data to train and we want higher accuracy?

It can be achieved by Transfer learning, with which we can use already pre trained model and can adjust or fine tune according to our requirements and can train it at much faster rate which increases our accuracy with lower amount of data that can be used for training.

Dog vs Cat classification was taken into account as it is one of the binary classification problem which can be stated as 0 for cat or 1 for dog. The project started as a simple classification problem with only dog and cats in mind that we can classify the images and distinguish between them and then this issue came in my mind what if I don't have adequate amount of training data then how will I solve this problem and how will I classify them.

After researching on how to build a classifier with less data through transfer learning I came up with this project of classification with good accuracy with less amount of data and then I got the dataset of it through kaggle dataset of dogs vs cats and I saturated it according to the project.

The dataset link is provided in the ReadMe file.

Problem Statement

The problem we are talking is getting good accuracy and building classification models with less data.

In this study, we will present a few simple yet effective methods that you can use to build a powerful image classifier, using only very few training examples --just a few hundred or thousand pictures from each class you want to be able to recognize.

In our project we are taking the images of only cats and dogs so the output will be the labels of those 2 classes. To implement this project:

We will go over the following steps:

- Find the best dataset (dogs vs cats in our case) which specify our requirement
- Training a small neural network from scratch (as a benchmark) which consists of a simple stack of 3 convolution layers with a ReLU activation and followed by max-pooling layers.
- Perform data preprocessing and augmentation on the training dataset.
- Using a pre-trained network of VGG16 architecture from ImageNet.
- Calculate the accuracy of the pre-trained network with our dataset.
- Using the bottleneck features of a pre-trained network
- Fine-tuning the top layers of a pre-trained network according to our requirements.
- Compare the accuracy to get the final result.

The output of the project is to aim for getting >80% accuracy in the final model which has over 5 times the test set with respect to the training set.

The Capstone Proposal link is provided in the Readme file also below https://review.udacity.com/#!/reviews/1251377

Metrics

Metric system used in this project is classification accuracy. It is can be stated as:

$$\text{Classification Accuracy} = \frac{\text{Correct Predictions Made}}{\text{Total No. of Predictions Made}} \times 100\,\%$$

The above accuracy is calculated for all the models created.

- Benchmark
- VGG16
- Final Model

The above metric system was chosen as we have to compare multiple models and our final objective is to get the best accuracy, so to minimize the total calculation on other metric system we can chose the accuracy as a metric and calculate classification accuracy of each model and compare them in the end.

2. Analysis

Data Exploration

In our study we will use two sets of pictures, which is given in the dataset link: 1000 cats and 1000 dogs. We also use 400 additional samples from each class as validation data, to evaluate our models.

The dataset link is provided in the Readme file.

The dataset also includes more than 12500 test cases which is used for testing the accuracy.

We are constricting ourselves with only 8% of total data to train and validate our model which makes this process pretty hard.

There is not set resolution of the images, but at the time of preprocessing they are converted to 299 x 299 but as in the dataset there is no fixed resolution. But the max resolution in the dataset is 500 x 499.

Most of the images sizes are less than 25kb.

Dataset is already numerically organized so we don't need to normalize the dataset. But while processing the machine can't process all the RGB color 0-255 then we use normalization technique to rescale the value so that the machine can process.

Some of the images have multiple cats or dogs or both in one image. Some of them also have humans in the background.

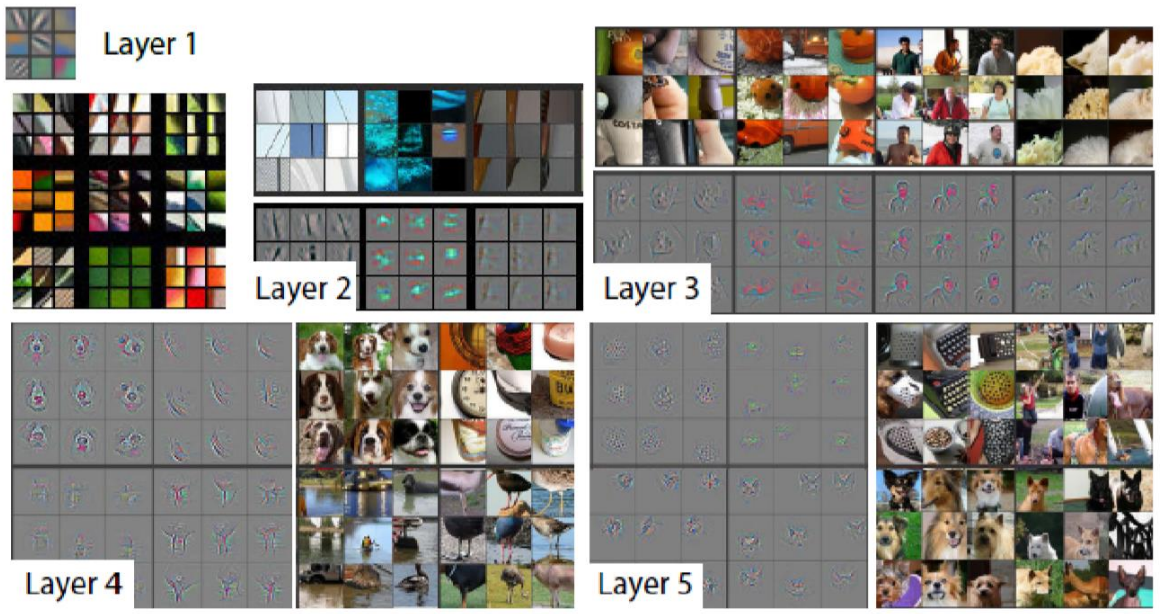
There were no abnormalities in data there are different backgrounds of the images which makes it interesting to learn.

Some of the dataset images are as follows:

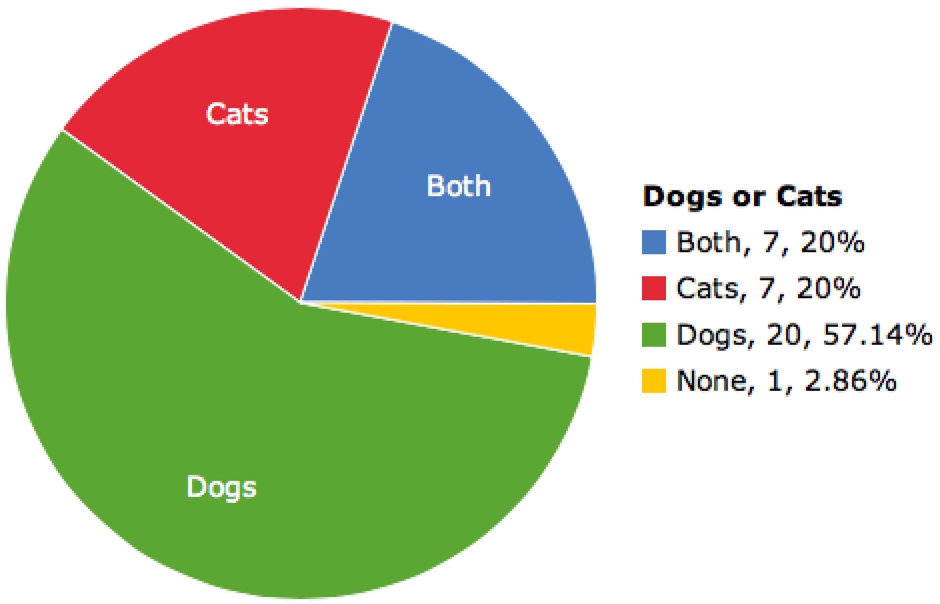
<u>Exploratory Visualization</u>

We are using a pre-trained model in our project so we can visualize the data with respect to what the model is performing on the dataset.

The regularization used in the following image is no regularization.

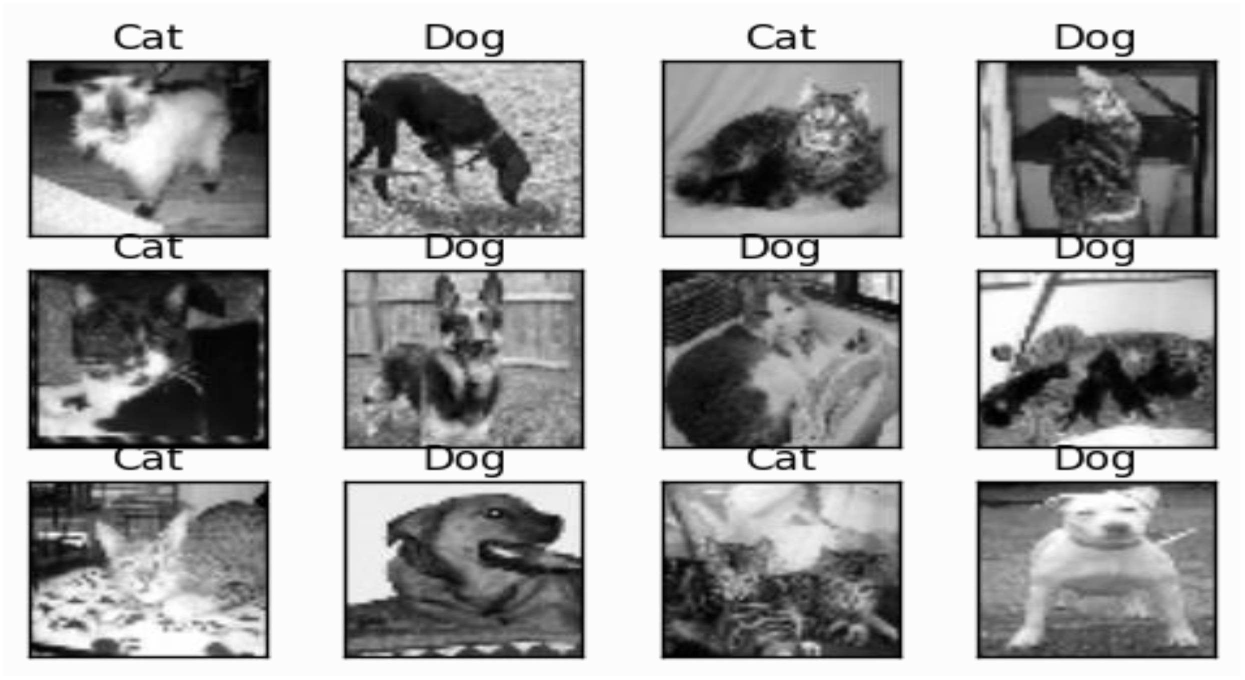


The above layers are visualized using the ImageNet library and we can see what different layers are doing and they are processing the data.



After processing the data we can observe that histogram for our test cases which tells how many dogs and cats are found in the testing data set and there are some unclassified image which is also found in kaggle dataset.

Other regularization techniques are also used such as L2 regularization and Gaussian blur techniques are used to visualize the data better.



Above image shows the final result after the processing and how to the machine is categorizing the images to their class of cat and dog.

Algorithm and Techniques

There are multiple algorithm and techniques used in the project some of them are:

Keras: It is one of the main library used in this project with which we can work with deep neural networks. This library is mainly used for the work on image and text data because of which we used this library in this project.

Dropout: It is a method used to drop the data with respect to some probability factor usually used 0.5 or 50%. This method is used to prevent overfitting of the data and also so that the same data doesn’t go through the CNN twice.

Data Augmentation: This is a method to change some part of the data so that the machine interpret as an independent data. Ex. An image of cat is rotated which will be counted as new image rather than the same image.

Binary Cross Entropy: We are using binary cross entropy loss to train our model for benchmarking.

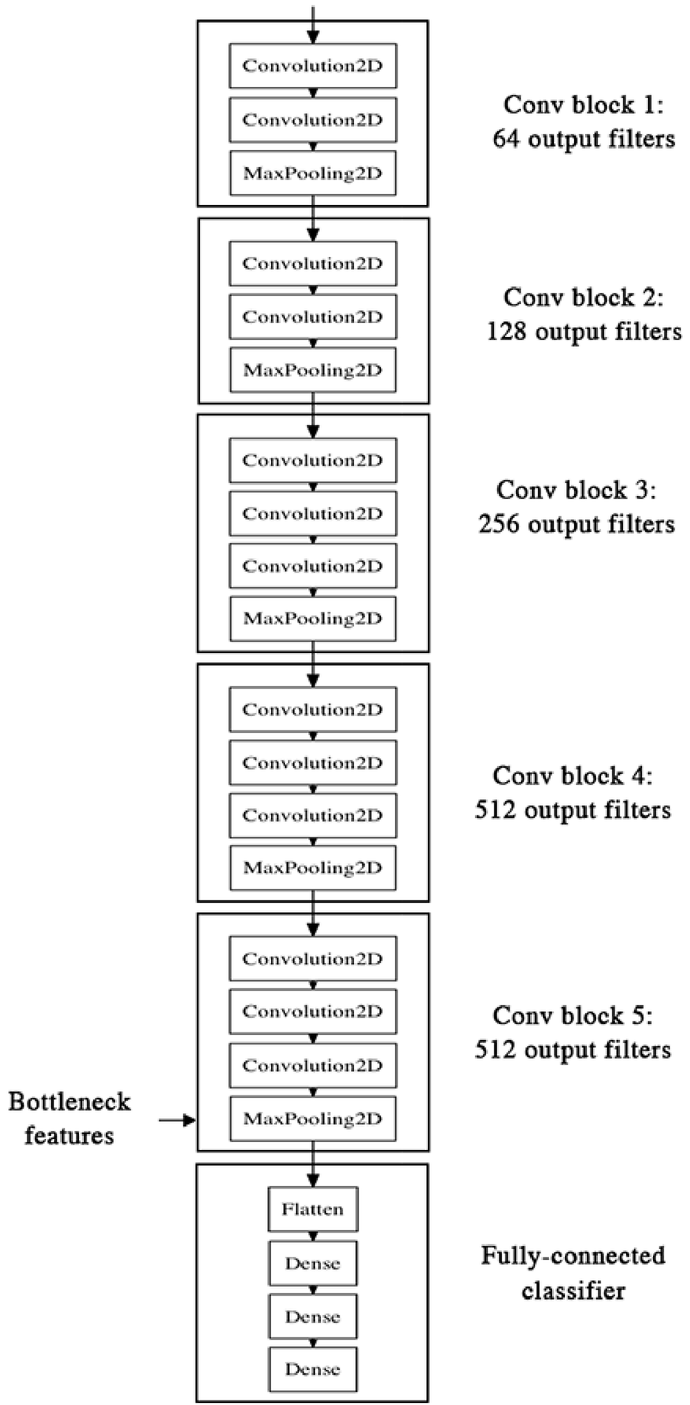
Bottleneck: A bottleneck layer is a layer that contains few nodes compared to the previous layers. It can be used to obtain a representation of the input with reduced dimensionality.

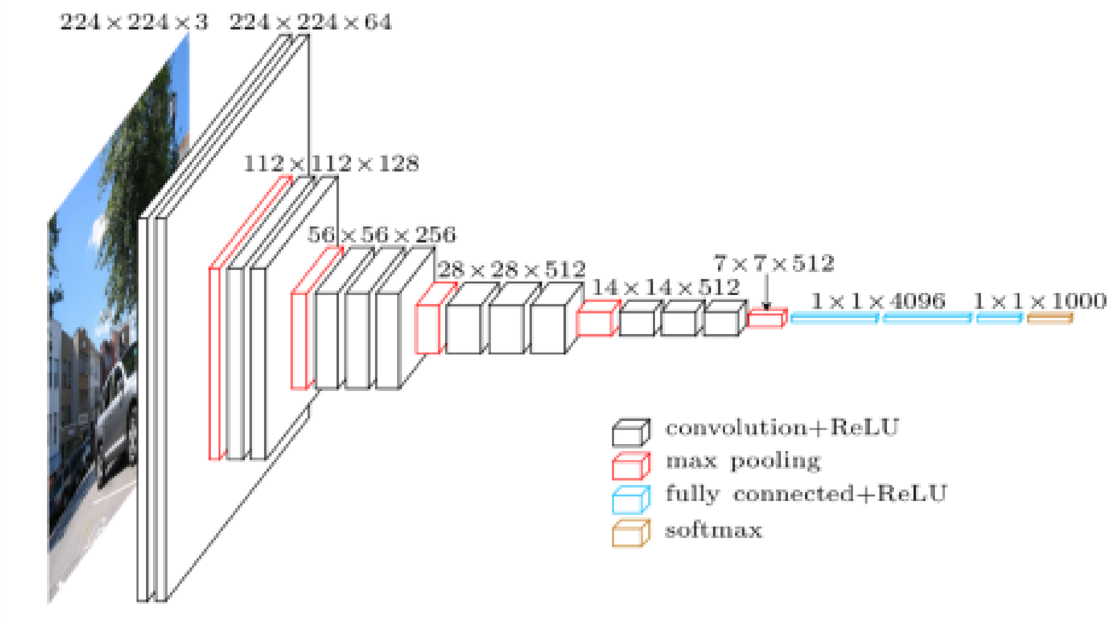
Fine Tuning: Fine tuning is just about making some fine adjustments to further improve performance. For example, during transfer learning, you can unfreeze the pre-trained model and let it adapt more to the task at hand.

Convolutional Neural Network: A **convolutional neural network** (**CNN**, or **ConvNet**) is a class of deep, feed-forward artificial neural networks, most commonly applied to analyzing visual imagery.

The CNN used in our project is based on VGG16 architecture from ImageNet.

The structure of it is as follows:





This network is characterized by its simplicity, using only *3×3* convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier (above).

Above are some of the techniques used in this project.

The techniques mentioned above are necessary to tackle the problem because it makes solving the problem easy. The techniques are chosen specifically for image processing and analyzing the image and then we are using transfer learning to transfer all the trained and processed part and use it to solve the problem.

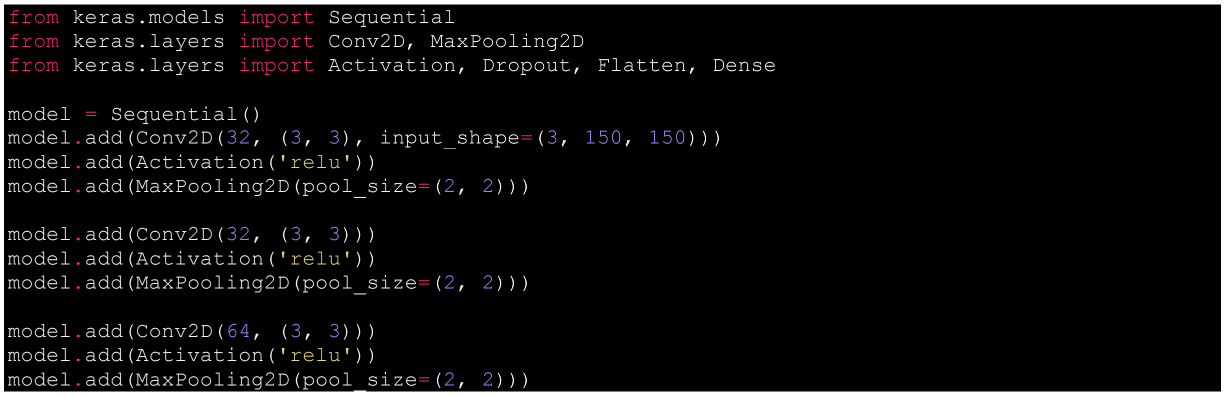
Except for data augmentation technique all the other techniques used are part of ConvNet which won't affect the datasets.

Data augmentation can create multiple copies of similar data to make machine learn more by altering some points of the data.

Benchmark

In our case we will use a very small convnet with few layers and few filters per layer, alongside data augmentation and dropout.

The code snippet below is our first model, a simple stack of 3 convolution layers with a ReLU activation and followed by max-pooling layers.



```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(3, 150, 150)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

This approach gets us to a validation accuracy of 0.79-0.81 after 50 epochs.

This benchmark result is used to check whether our CNN after tuning will give us the result better than our benchmark or it fails. If the result is lower we make other changes to make the fine tuning better and compare again.

3. Methodology

Data pre-processing and Data Augmentation

According to the dataset we are taking image width and height in our case it is 299 x 299

Batch size taken is 32

Learning rate is 1e-4

Transformation ratio is set to 0.05

Momentum is set to 0.9

No. of epochs is set to 50

In our study we will use two sets of pictures, which is given in the dataset link: 1000 cats and 1000 dogs. We also use 400 additional samples from each class as validation data, to evaluate our models.

That is very few examples to learn from, for a classification problem that is far from simple. So this is a challenging machine learning problem.

In order to make the most of our few training examples, we will "augment" them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better.

An example of this is:

```
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

All the abnormalities and other process have been mentioned in the data exploration section and yes data preprocessing is required as our data set size is low so to make the machine learn we can preprocess the data so that machine doesn't see the similar picture twice.

Implementation

At first we had to search for the best language which can be used while implementing the project. And as I am fluent with python and there is a good support of machine learning libraries in python. The language chosen was python.

After finalizing the project and what we have to implement next comes the concerned libraries which have to be used, after researching on the this topic we found out about keras and how it can be used in neural network specially for images and text recognition.
The only external library used in this project is keras all other libraries are internal.

After above the project implementation starts as following steps:

1. **The dataset was chosen**: Dataset was taken from the kaggle dataset of cats and dogs and we took out 1000 cats and 1000 dogs for both training and validation set and keep 12500 data for testing. Finding the dataset was not that hard as kaggle have most of the dataset available in it.
2. **Benchmark Model was created**: For benchmark model we used a simple stack of 3 CNN layers with a ReLU activation following with max-pooling layers to get a classification accuracy which can be compared with the models. This benchmark model was thought after learning about imagenet and how it works. The benchmark model chosen should be similar to the pre-trained model as we have to compare the result of our benchmark model and the pre-trained model so that we can see how it improved from the benchmark model. I didn't use any other projects benchmark as I wanted to use the same dataset to know accurately how good my final model will be and there will be no redundancy in the dataset.
3. **Searching of Pre-Trained model**: We had to search which model will be most suitable and which have our data pre-trained within itself and VGG16 from ImageNet was

chosen as it already has cats and dogs classification pre-trained to it. It is not the best pre-trained model we can get ResNet and other pre-trained model but as our model specifically already contains all the cats and dogs classification (of different species) this model was chosen.

4. **Calculating Accuracy**: The accuracy of the pre-trained model was calculated and compared to the benchmark result which is used to make changes and give us idea on how to increase the accuracy more.
   The Classification Accuracy was taken into account for all the models used, the pre-trained model accuracy was checked against benchmark and the changes were taken into account.
5. **Fine Tuning the model**: Fine tuning the model was done and the last of the CNN layer was altered while freezing every other layers and then with slow learning rate training was done on the tuned model.
   Fine-tuning consist in starting from a trained network, then re-training it on a new dataset using very small weight updates. In our case, this can be done in 3 steps:

- instantiate the convolutional base of VGG16 and load its weights
- add our previously defined fully-connected model on top, and load its weights
- freeze the layers of the VGG16 model up to the last convolutional block

6. **Final Accuracy calculation**: After training the final model testing was done and the final result was obtained which is compared with all the other models results and checked if we reach our desired accuracy. If this doesn't happens then we go back to step 5. And fine tune the model again and train and test it again.

Complications faced in coding while implementation happened while doing bottlenecking of the pre-trained network. As ImageNet contains thousands of classes and we only need several cats and dogs classes for our project, we have to ignore all the other classes or remove them.
To work through this problem instead of removing them it will be better to generalized it into a broader range then we can fine tune it.

Our strategy will be as follow: we will only instantiate the convolutional part of the model, everything up to the fully-connected layers. We will then run this model on our training and validation data once, recording the output (the "bottleneck features" from the VGG16 model: the last activation maps before the fully-connected layers) in two numpy arrays. Then we will train a small fully-connected model on top of the stored features.

The reason why we are storing the features offline rather than adding our fully-connected model directly on top of a frozen convolutional base and running the whole thing, is computational efficiency.

Fine Tuning the Top Layers Pre-Trained Network (Transfer Learning)

Refinement

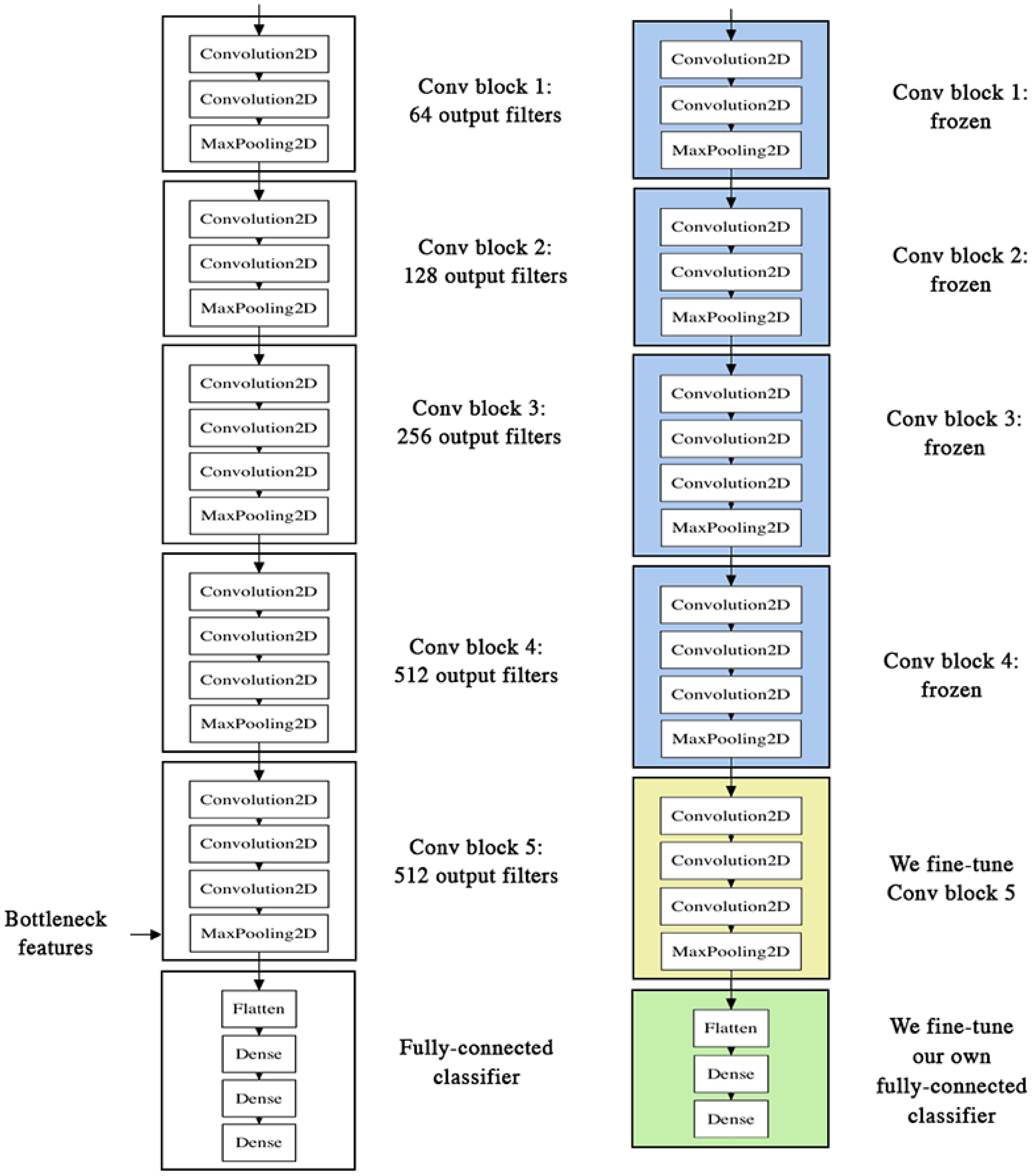
To further improve our previous result, we can try to "fine-tune" the last convolutional block of the VGG16 model alongside the top-level classifier. Fine-tuning consist in starting from a trained network, then re-training it on a new dataset using very small weight updates. In our case, this can be done in 3 steps:

- instantiate the convolutional base of VGG16 and load its weights
- add our previously defined fully-connected model on top, and load its weights
- freeze the layers of the VGG16 model up to the last convolutional block

Note that:

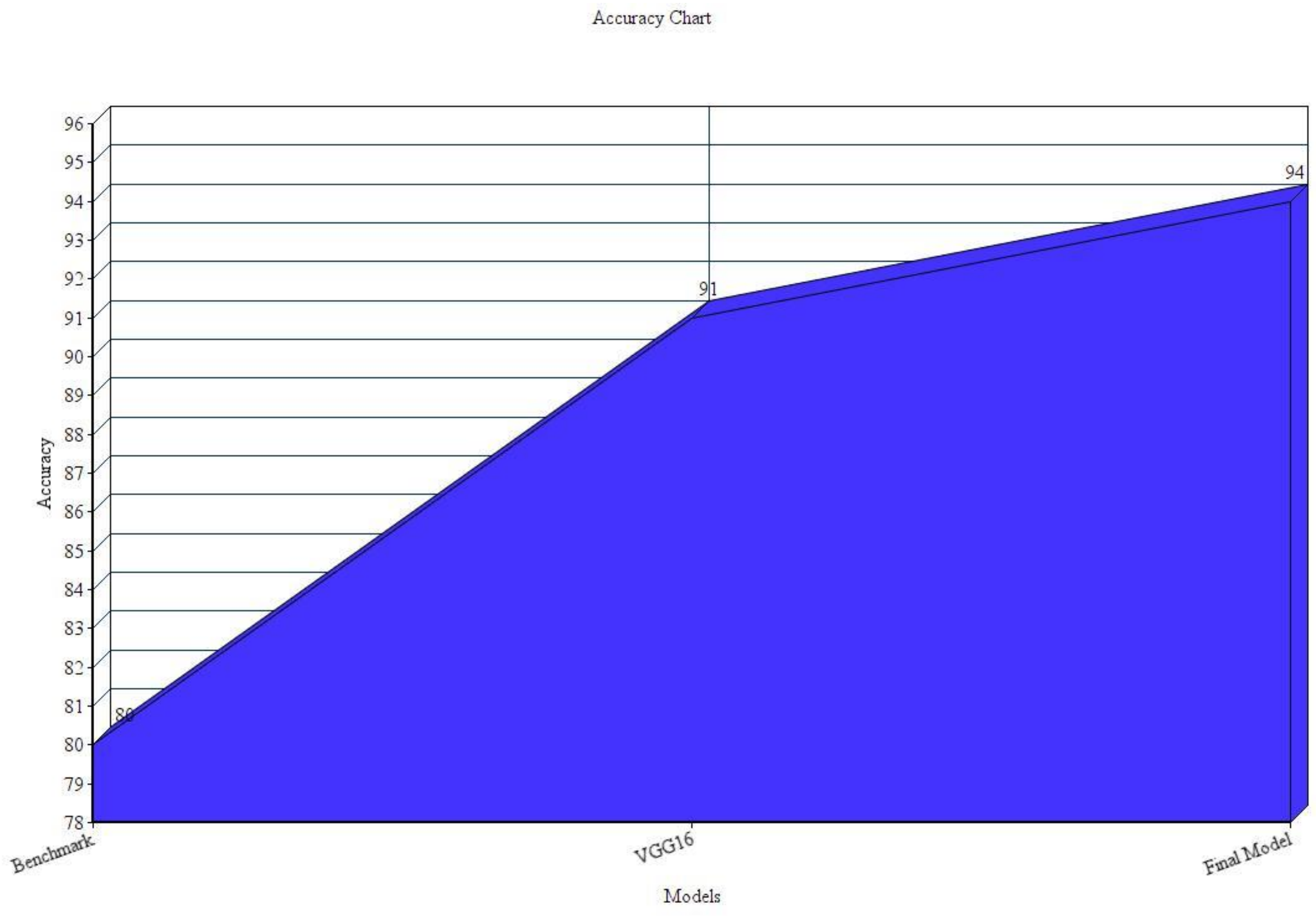
- In order to perform fine-tuning, all layers should start with properly trained weights: for instance you should not slap a randomly initialized fully-connected network on top of a pre-trained convolutional base. This is because the large gradient updates triggered by the randomly initialized weights would wreck the learned weights in the convolutional base. In our case this is why we first train the top-level classifier, and only then start fine-tuning convolutional weights alongside it.
- We choose to only fine-tune the last convolutional block rather than the entire network in order to prevent overfitting, since the entire network would have a very large entropic capacity and thus a strong tendency to overfit. The features learned by low-level convolutional blocks are more general.
- Fine-tuning should be done with a very slow learning rate, and typically with the SGD optimizer rather than an adaptive learning rate.

Before Fine Tuning After Fine Tuning



4. Result

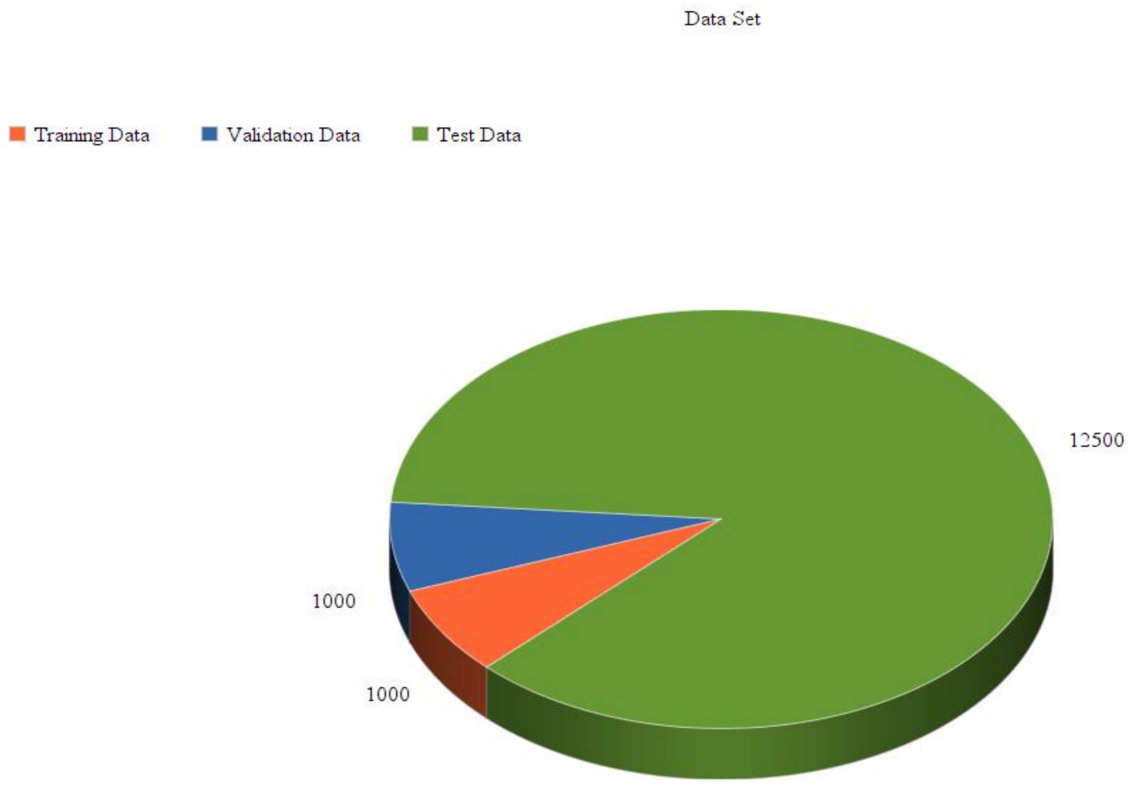
Model Evaluation and Validation



After instantiating the VGG base and loading its weights, we add our previously trained fully-connected classifier on top as mentioned above in the fine tuning.
After freezing the layers and adding our own we will start training the whole thing at a slow learning rate.
This approach gets us to a validation accuracy of 0.94 after 50 epochs.

The final model is robust enough to get 94% accuracy which is tested for over 12500 test data while training is done on only 2000 training data.
The final model easily cross over our expected accuracy which is over 60%.
The model easily distinguishes between dogs and cats and can be trusted.

Justification

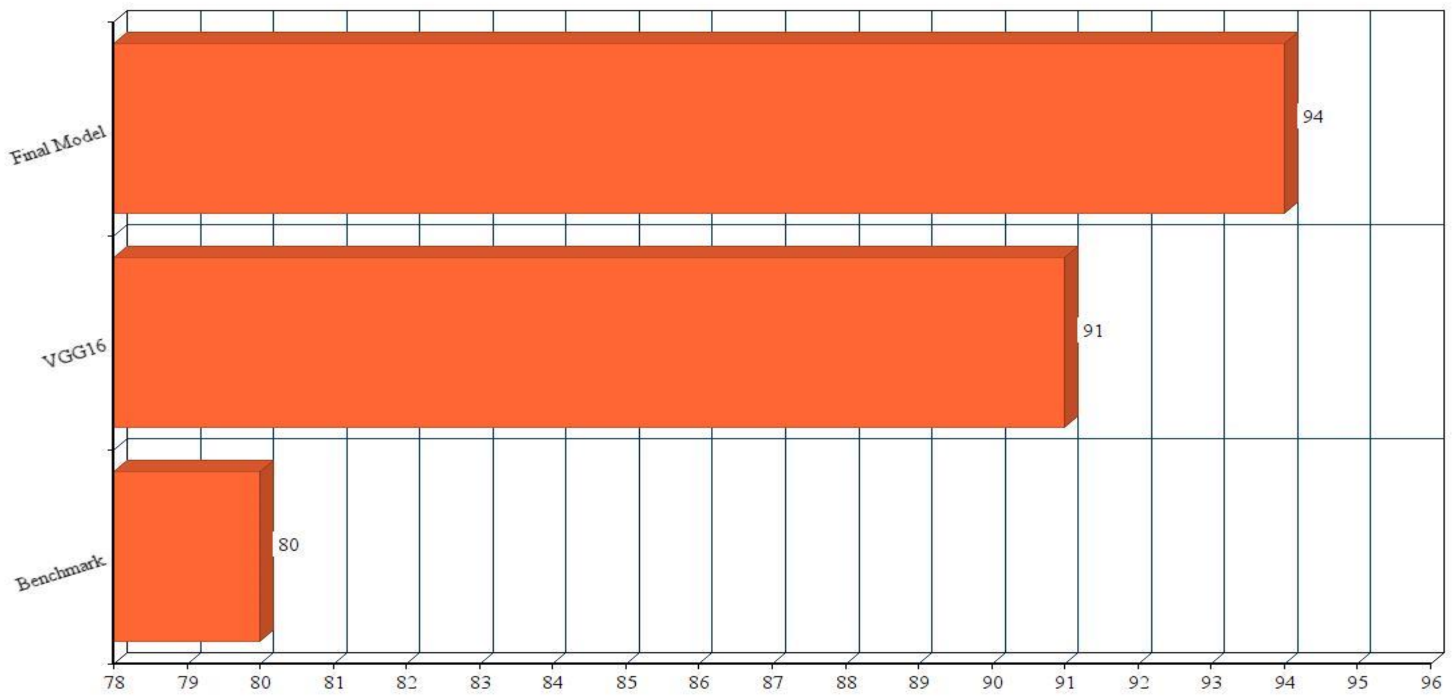


As mentioned above our benchmark model gets an accuracy of around 0.79-0.81 which is approx. 80% and our final result shows the accuracy increase to 0.94 which is 94%. We see the accuracy increase of 14% which is very significant than our benchmark model which isn't that accurate. We can see the difference above the size of data for training, validation and training set. We have achieved what our problem statement was and created a classifier which classify with high accuracy and has low training set.

Yes, the final model is capable enough to solve the problem of making a good classifier with lower size of dataset through transfer learning.

5. Conclusion

Free Form Visualization



After instantiating the VGG base and loading its weights, we add our previously trained fully-connected classifier on top as mentioned above in the fine tuning.
After freezing the layers and adding our own we will start training the whole thing at a slow learning rate.
This approach gets us to a validation accuracy of 0.94 after 50 epochs.
When we compare the result we got to our other data and accuracy of previous models and benchmark we can see the difference in the accuracy we get.

The above chart shows the difference between the accuracy and we can see how much we have improved with some changes in the CNN.

Reflection

The project have thoroughly summarized and everything of process have been mentioned with all the steps taken to reach the end result.

The most interesting part of the project was how transfer learning works and how it makes the overall work less. It makes the process faster while giving higher classification accuracy. It makes computing fast and can be improved easily by just altering the pre-trained model.

The most difficult aspect of the project was to find the pre-trained model and to fine-tune the model. As to fine tune you have to understand the working of the CNN layers of the model and how they are working. But after learning through the documentation it was possible.

The first problem faced was finding the benchmark model. At first I thought I can easily chose other project as a benchmark (cat vs dog project) but I didn't want any redundancy as my dataset may be different from them. To overcome this I first searched on which pre-trained model I will chose then I worked through it and I finally considered the benchmark model which is similar to the pre-trained model and after training on it received 80% accuracy.

The second problem I faced was while bottlenecking the pre-trained model as there are multiple classes which was of no need so it may cause the overall processing time to extend, therefore to remove it I instead of removing the classes generalize it to cats and dogs. This problem took most of my time.

At the time of fine-tuning I didn't face much of a problem as I saw an article on keras which helped me understand the process better and helped me on what to do and how to do it.

Yes, the final model and solution fit my expectation for the problems, and yes it can be used to solve these type of problems and can be used to train large neural network which uses multiple type of input to form a result can use multiple pre trained models which makes it more accurate and reliable.

Improvement

This approach can also be further improved by using the following techniques:

- More aggressive data augmentation
- More aggressive dropout
- Fine Tuning one more convolutional block
- Weight Decay

Fine tuning multiple blocks helps us to generalize better as we can make the CNN more according to our requirements.
As above we used VGG16 which can classify all the animals but we are using it only for cats and dogs which we made changes to in the last layer of the CNN but we can also categorize it

further by making changes in the layer before it so that we can focus only on those two categories not on other animals.

6. References

1. https://machinelearningmastery.com/transfer-learning-for-deep-learning/
2. https://my.pcloud.com/publink/show?code=VZvzMlZyPYO1hSn92LXdzmGNr9y1j7qKDzX
3. https://www.wikipedia.org/
4. https://becominghuman.ai/transfer-learning-retraining-inception-v3-for-custom-image-classification-2820f653c557