

# Rpeort

*by Ankit Kumar*

---

**Submission date:** 12-Jul-2018 06:32PM (UTC+0800)

**Submission ID:** 982055366

**File name:** Report\_Ananta\_Ankit.docx (9.16M)

**Word count:** 3100

**Character count:** 16181

# CHAPTER 1

## INTRODUCTION

### 1.1 Area of Work

<sup>13</sup> Deep learning is about constructing machine learning models that learn a hierarchical representation of the data. We can describe a hierarchical model with Neural Networks where each layer of neurons represents a level in that hierarchy. There is no real restriction in how many layers we can add to a network <sup>3</sup> but going beyond two layers was practical in the past with diminishing returns. Today people started adding more layers resulting in Deep Neural Networks with much success

**Convolutional Neural** <sup>7</sup> Networks is a class of Deep neural networks that is successful method to apply for analysing visual imagery. CNN uses variation of multilayer perceptron designed to require minimal pre-processing. It uses relatively little pre-processing compared to other classification algorithms. It means that the network learns the filters that in traditional algorithms. CNN can be summarised into 4 major steps:

1. Convolutional
2. Max Pooling
3. Flattening
4. Full Connection

### CONVOLUTION

The convolution layer is applied, and convolution operation is done for the input, passing the results to the next layer. It combines and integrates two functions to generate the third function. A feature map is generated based on the feature detector which acts as the filters in CNN.

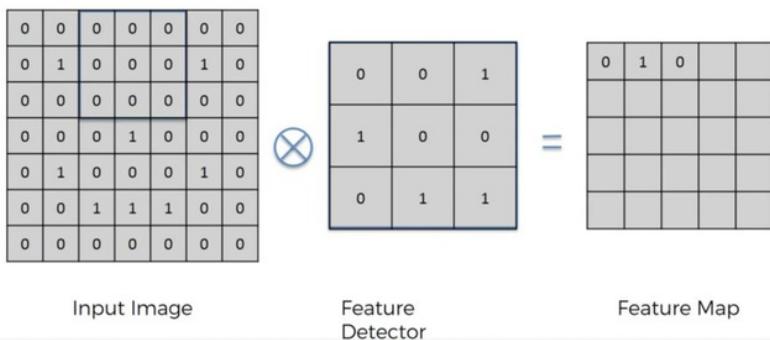


Fig 1. Convolution Step

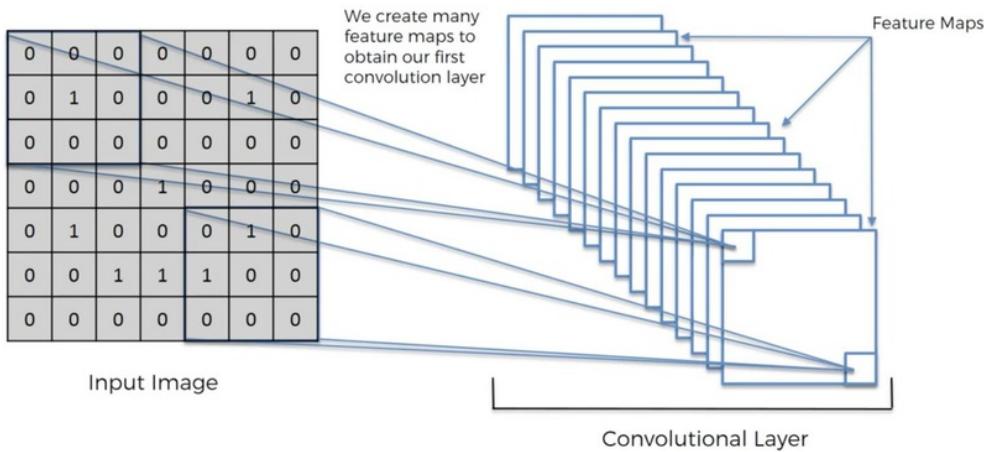


Fig 2. How Convolution works

## POOLING

Convolutional neural networks may include one or more local or global pooling layers. For example, max pooling uses the maximum value from each of the cluster of neurons formed. So, we don't lose any important information. Pooling layers are used to shrink the height and width without losing features or edges of the images.

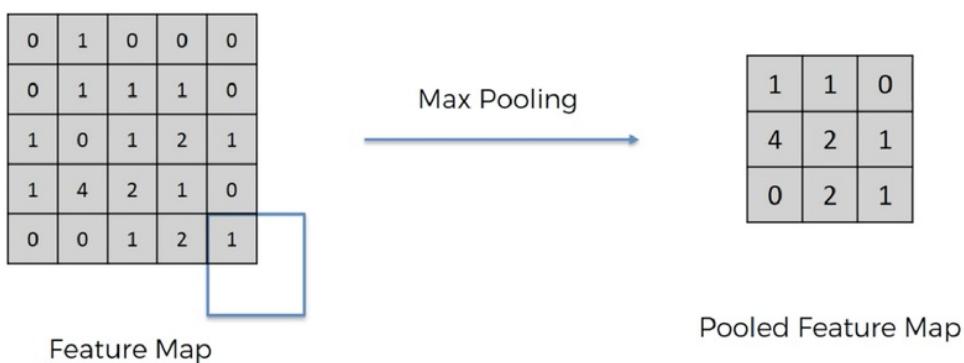


Fig 3. Pooling layer

## FLATTENING

In Flattening we convert the matrix of features into single columns and pass it through a neural network. We practically do this as we take them as input for our convolutional neural network.

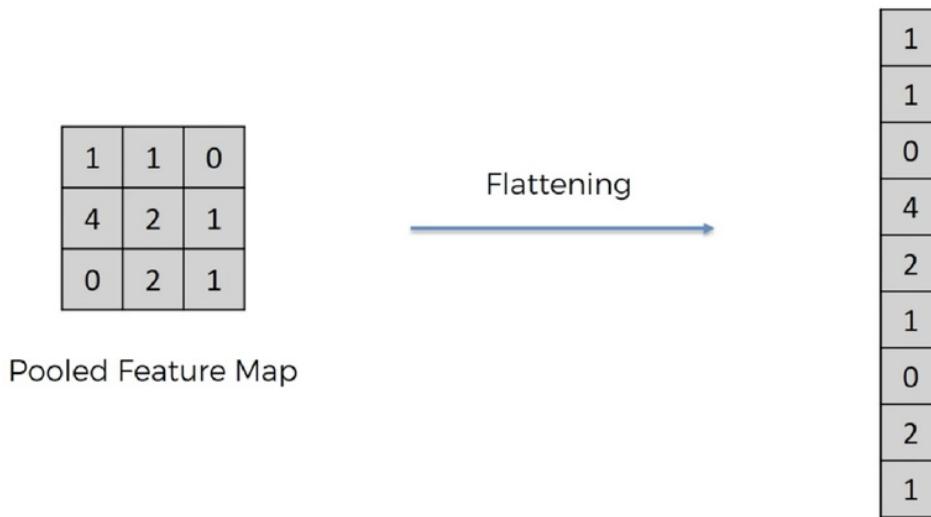


Fig 4. Flattening

## FC LAYERS

It is responsible for connecting neurons from one layer to another layer. A final dense output layer is generated on which activation function is applied for the supervised or non-supervised task.

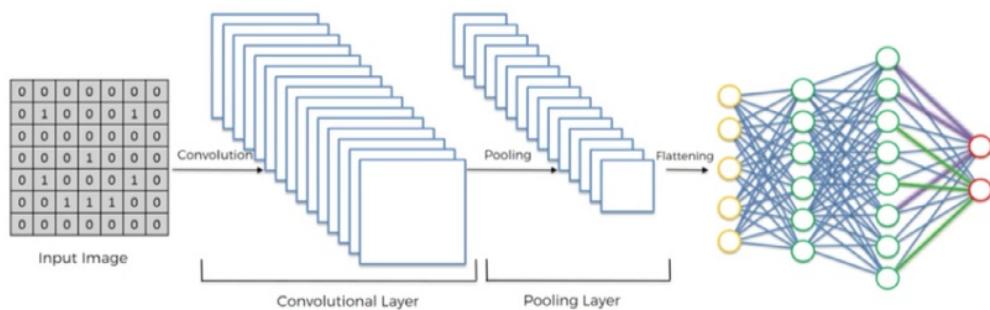


Fig 5. Fully Connected Layers

## TRANSFER LEARNING

After using a pre-trained model, our deep learning model learns faster and processes data quickly and efficiently. This method of applying pre-trained model to our dataset is called **Transfer Learning**.

8  
**Transfer Learning** is also known as inductive learning. It is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different problem which is a related problem. For example:

Knowledge gained while learning to recognise cars can also apply to trucks.

It improves the learning processes and methods by transferring the knowledge gained by some other task. It is an optimization technique, which saves time or gives better performance than simple neural networks.

4  
So, over the years there is a trend to go more and more deeper to solve more complex tasks and to increase or improve the classification /recognition accuracy. But as we go deeper, the training of neural network becomes difficult and also the accuracy starts saturating and then degrades also. Transfer learning tries to solve these problems.

When to use TL:

- Whenever we have very huge dataset.
- Working on machines with CPU instead of GPU
- To train a model quickly for faster results

Some of the pre-trained models are as follows:

- VGG-16 and VGG-19 Model
- Inception Model
- Res-Net Model
- Alex-Net Model
- Word2vec Model

We have used ResNet-50 to train our model.

## RESNET-50

ResNet is a short form of Residual Network. As the name of this network indicates, the new terminology that this network introduces is residual learning. It is a Convolutional Neural Network of Microsoft team and it surpassed the human performance on ImageNet dataset.

Deeper Neural Networks are a little difficult to train. We are using residual network to ease our way for training a large dataset or small dataset with very deep neural network.

### 1 The issues related to deep neural networks

In the past few years, neural networks have become deeper, with simple networks going from just a few layers (e.g., Alex-Net) to over 100 layers.

The advantage of having very deep neural networks is that it can represent much more complex functions. Very deep networks can learn complex features, from edges (at the lower layers) to very complex features (at the deeper layers). A huge obstacle in training deep neural networks is vanishing gradients: very deep networks often have a gradient signal that goes to zero quickly, thus making gradient descent unbearably slow. More specifically, during gradient descent, as you backprop from the final layer back to the first layer, you are multiplying by the weight matrix on each step, and thus the gradient can decrease exponentially quickly to zero (or, in rare cases, grow exponentially quickly and "explode" to take very large values).

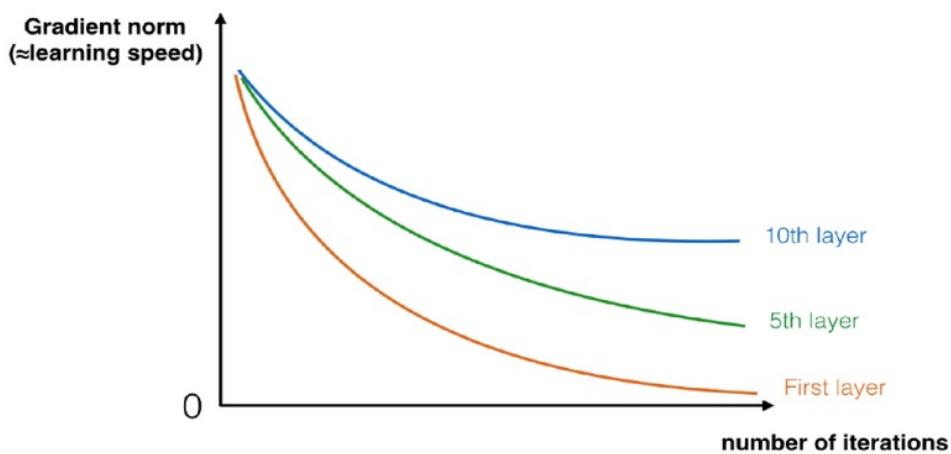


Fig 6. Problem of deep neural network

This problem can be solved by using Residual Neural Networks. In Res-Nets, a "shortcut" or a "skip connection" allows the gradient to be directly backpropagated to earlier layers:



Fig 7. Skip connections

1  
The first image shows the direct path of the neural network while the second image shows the skip connection aka shortcut. The main feature of skip connection is that it helps with vanishing gradient problem.

1  
Two main types of blocks are used in a ResNet, depending mainly on whether the input/output dimensions are same or different.

## Building Res-Net50 model

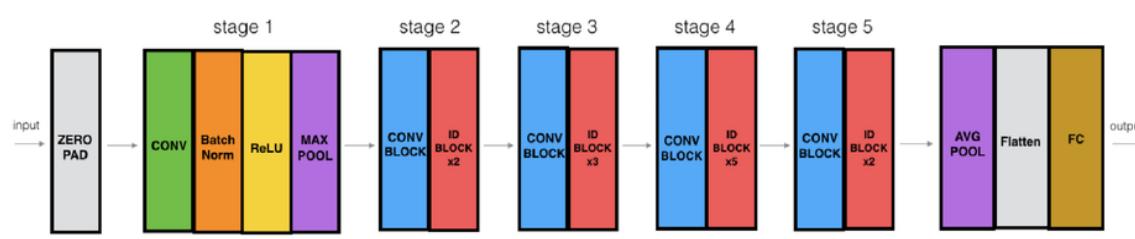
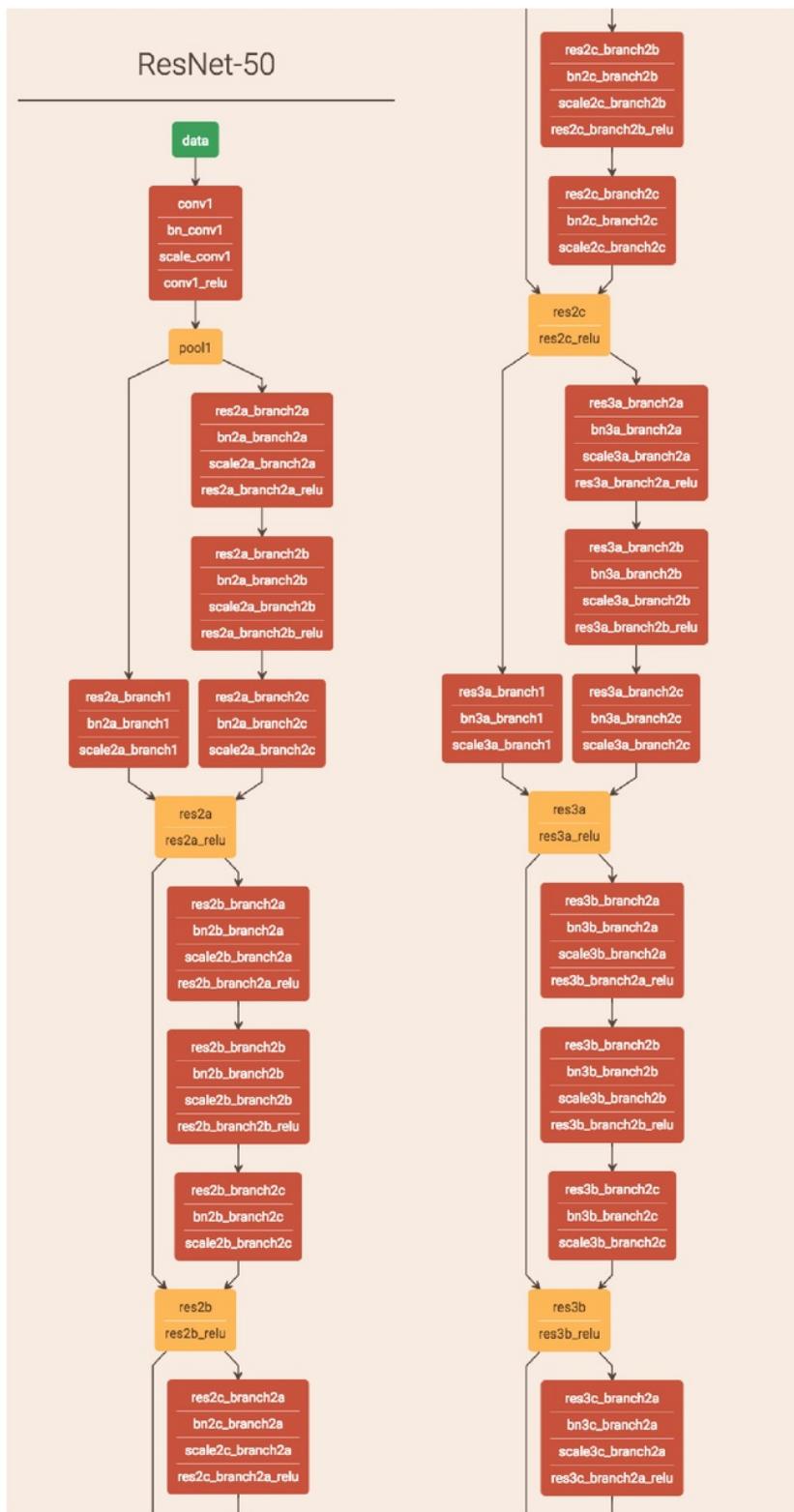
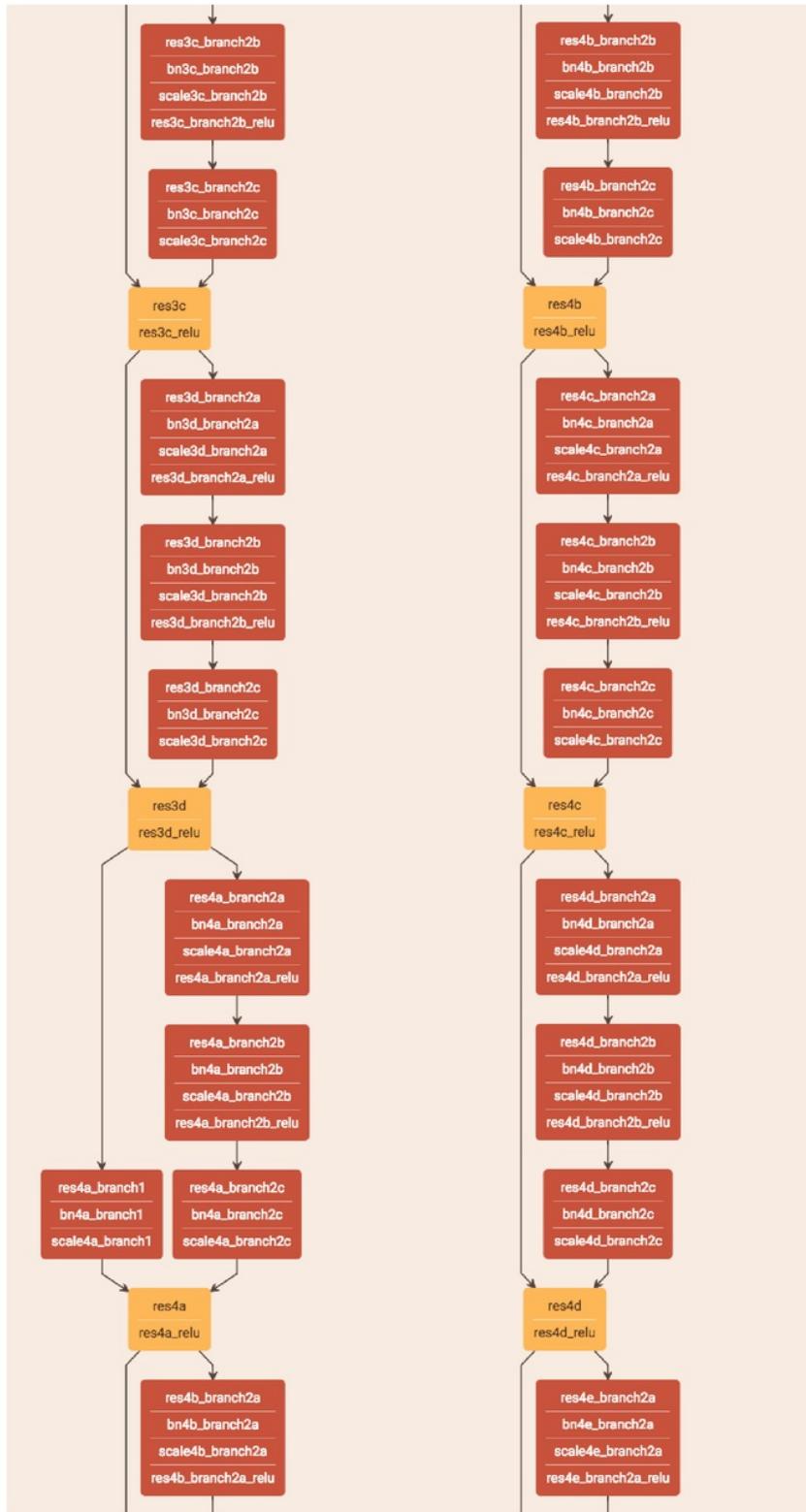


Fig 8. ResNet-50 building stages





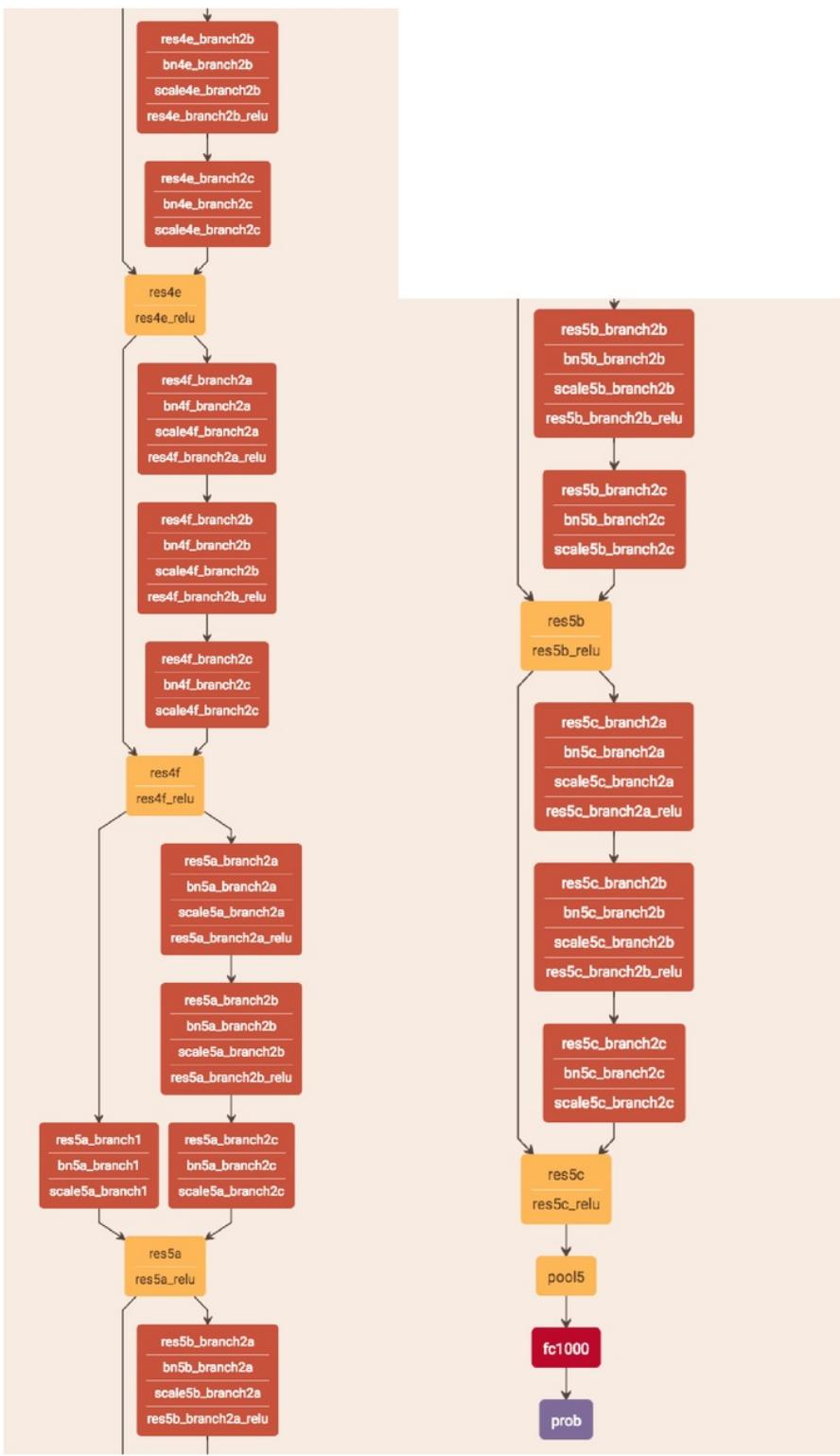


Fig 9. ResNet-50 architecture

## 1.2 Motivation

We built a CNN architecture (using keras library which uses TensorFlow as backend) to classify the breed of a dog from any supplied image from a user.

The process of classifying dog's breed from images is challenging because more than one breed looks just like other.

Our goal is that by completing this project, we can train our model for several other datasets and classifications as well. This way our model is only going to get better and better.

With good accuracy, it is going to provide better results in future. By working on this project, we acquired the basics knowledge of TensorFlow, keras frameworks. Deep Learning is one of those things where the only limit is how far it can go, is our own imaginations. Several things that were thought to be impossible by the older generation, are now in our lives.

We believe Deep learning would be one of the catalysts to change the future as we know it of now.

Example of how different breeds looks exactly the same:

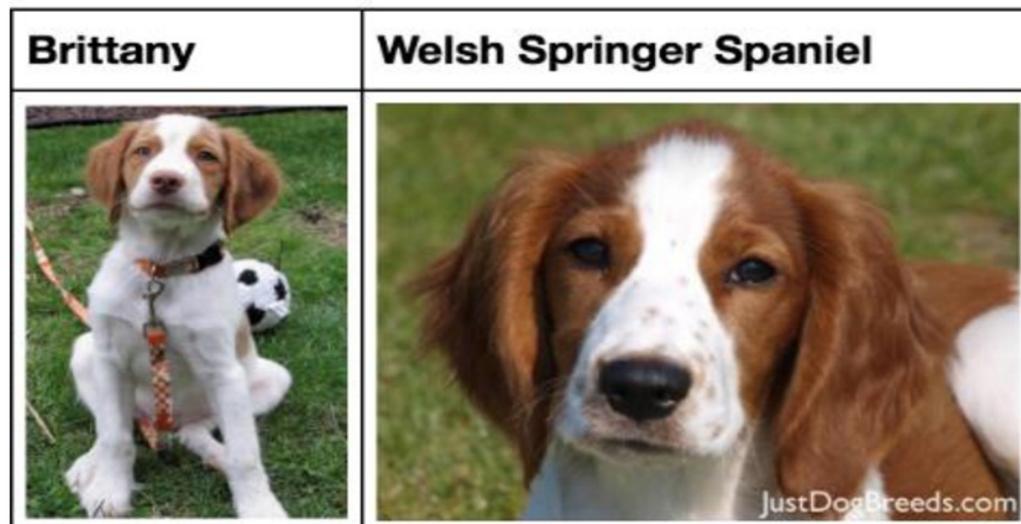


Fig 10. Similar dog breeds

### 1.3 Objective of the Work

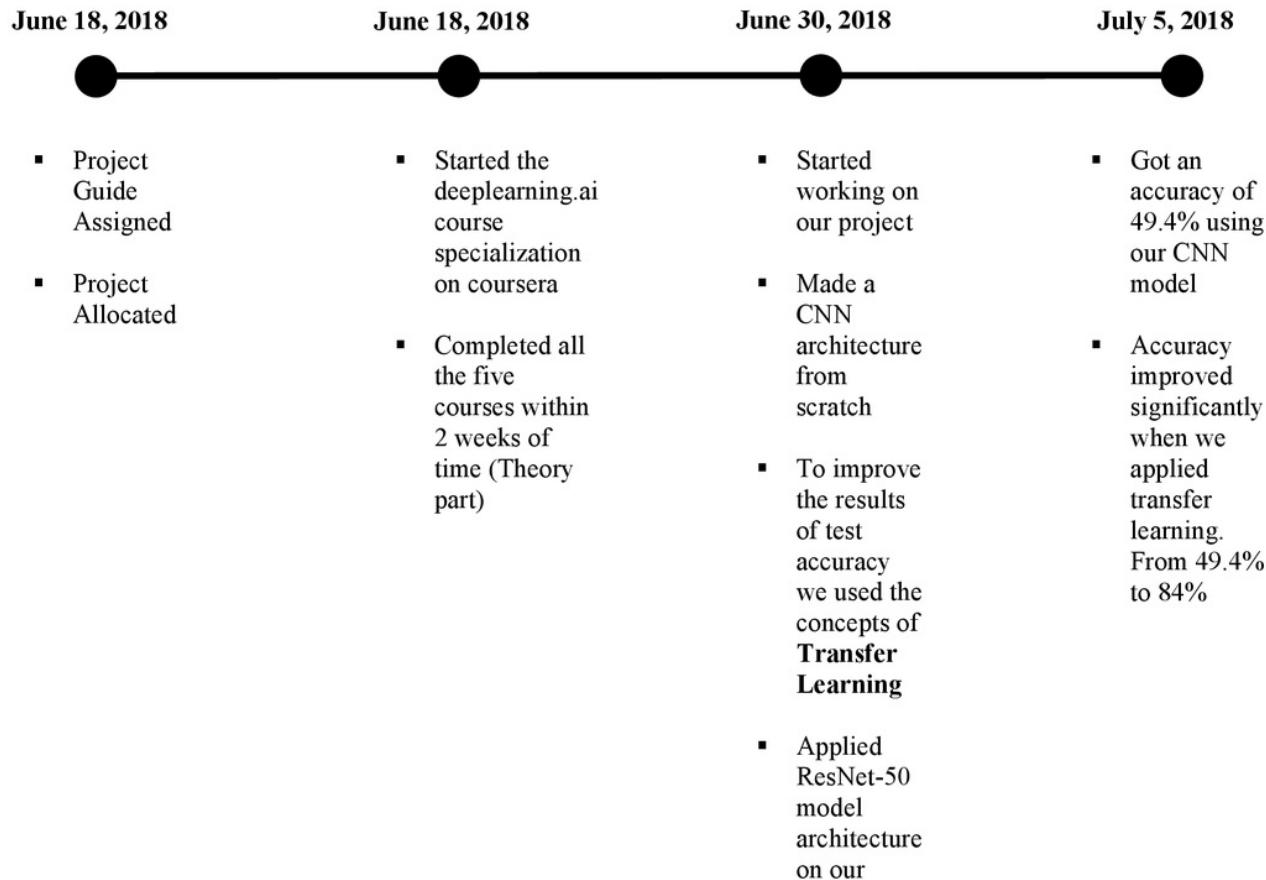
#### Objectives –

- Data Pre-processing
- Create a dog detector using ResNet-50 pre-trained model.
- Making prediction, if an image is of dog or not.
- Data Augmentation
- Build a CNN model from scratch
- Improve the efficiency using the concepts of Transfer Learning.
- Testing of our model

In the present scenario where, big industries want to replace the manual work by machine work by training that machine for a specific task. It is the science of getting computers to act without being explicitly programmed. The models build using Machine Learning and Deep Learning concepts learn to recognize the patterns in digital representations of data in a very similar way a human brain does.

- We only want things to be easier for humans by building models that can work as human brain. Deep Learning is a big change in technology as internet was. Through deep learning models, we can predict the future with their existing data and not just for historical analysis. We are actually using data for the best. The more the data is fed into our systems, the more it will learn, and the better results will be delivered.
- We wanted to analyze the functions and working of TensorFlow and use them as a go in our model. We tried building a 5-layer CNN architecture (using keras API which uses TensorFlow as backend) which will classify the breed of the dog from any supplied image from a user.  
It is an open source high performance library for numerical computation. It is used for all kinds of GPU computation.  
**Keras** is a simple, high level neural networks library written in Python that works as a wrapper to TensorFlow.
- We wanted to understand the pattern of accuracy and the results then only we can make our model better. We just want to train our model to get better and better results. The more the data we feed into our model, the better results we will obtain in future. Similar identification task can be done on different datasets also.

## 1.4 Project Work Schedule



## **1.5 Organization of the Project Report**

We worked in a team of two. So, the work was divided accordingly between us. We got a platform to apply and exchange our knowledge with each other.

It is really helpful when you are trying to learn something new. It is better to work in groups because discussions and doubts leads to better understanding of problems which should be sorted out at the right time.

The learning improves as your source of information widens up.

## Chapter 2 BACKGROUND THEORY

### 2.1 Literature Review

We read Deep Residual Learning for Image Recognition published by a team of three people. This paper is about how we can use large number of layers to build deep neural networks. One such deep neural network is called as residual network.

The main advantage of a very deep neural network is that it can represent much more complexity than simple neural network. It is also good at learning features, from edges to very complex features as we go deeper. However, using a deep neural network is not always helpful. A huge obstacle in training deep neural networks is vanishing gradients: very deep neural networks generally have a gradient signal that goes to zero very quickly, thus causing gradient descent slow.<sup>1</sup>

Using skip connections or sometimes called as shortcuts we can make residual network which can more efficiently for Image classification and recognition tasks.

### 2.2 Summarized Outcome

To get best accuracy in classification of images without losing parameters the paper uses bottleneck features for classification i.e. without losing main features or information, decreasing the size of the images to reduce the number of parameters. The parameter-free identity shortcuts are particularly important for the bottleneck architectures. We also saw that having ResNet blocks with the shortcut makes it very easy for one of the blocks to learn an identity function. This means that you can stack on additional ResNet blocks with little risk of harming training set performance. (There is also some evidence that the ease of learning an identity function--even more than skip connections helping with vanishing gradients--accounts for Res-Nets' remarkable performance).

### 2.3 Conclusion

We concluded by reading the research paper that using the pre-trained model for training our data would be beneficial and it could improve the accuracy of our model significantly.

Thus, we used ResNet-50 model using transfer learning on our data.

## Chapter - 3

# METHODOLOGY

### 3.1 Methodology

#### Step 1 - Import Dataset

In the following code below, we are importing the dataset. We will populate our dataset using flip and some other cropping techniques.

```
In [2]: 1 from sklearn.datasets import load_files  
2 from keras.utils import np_utils  
3 import numpy as np  
4 from glob import glob  
  
Using TensorFlow backend.
```

#### Function to load train, test and validate dataset

```
In [3]: 1 def load_dataset(path):  
2     data = load_files(path)  
3     dog_files = np.array(data['filenames'])  
4     dog_targets = np_utils.to_categorical(np.array(data['target']), 133)  
5     return dog_files, dog_targets  
  
In [4]: 1 #Load train, test and validation dataset  
2  
3 train_files, train_targets = load_dataset('dogImages/train')  
4 valid_files, valid_targets = load_dataset('dogImages/valid')  
5 test_files, test_targets = load_dataset('dogImages/test')  
6  
7 #Load list of dog names  
8 dog_names = [item[20:-1] for item in sorted(glob('dogImages/train/*'))]  
9  
10 #Print different statistics about the dataset  
11 print('There are %d total dog categories.' % len(dog_names))  
12 print('There are %d total dog images.' % len(np.hstack([train_files, valid_files, test_files])))  
13  
14 print('There are %d total training images of dogs.' % len(train_files))  
15 print('There are %d total validation images of dogs.' % len(valid_files))  
16 print('There are %d total test images of dogs.' % len(test_files))
```

```
There are 133 total dog categories.  
There are 8351 total dog images.  
There are 6680 total training images of dogs.  
There are 835 total validation images of dogs.  
There are 836 total test images of dogs.
```

## Step 2 - Detect Dogs

We are using pre-trained [ResNet-50](#) model to detect the dogs in the images.

Our first line of code downloads the ResNet-50 model, along with the weights that have been trained on [ImageNet](#) a very large and popular dataset used for computer vision and image classification tasks.

```
In [5]: 1 from keras.applications.resnet50 import ResNet50
2 import h5py
3
4 #defining the model
5 ResNet50_model = ResNet50(weights='imagenet')
```

## Data Pre-processing

When using TensorFlow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape

(nb\_samples, rows, columns, channels)

where nb\_samples correspond to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.

The path\_to\_tensor function below takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is 224×224 pixels.

Next, the image is converted to an array, which is then resized to a 4D tensor. In this case, since we are working with color images, each image has three channels. Likewise, since we are processing a single image (or sample), the returned tensor will always have shape

(1, 224, 224, 3)

The paths\_to\_tensor function takes a numpy array of string-valued image paths as input and returns a 4D tensor with shape

(nb\_samples, 224, 224, 3)

Here, nb\_samples are the number of samples, or number of images, in the supplied array of image paths. It is best to think of nb\_samples as the number of 3D tensors (where each 3D tensor corresponds to a different image) in your dataset.

```
In [6]: 1 from keras.preprocessing import image
2 from tqdm import tqdm
3
4 def path_to_tensor(img_path):
5
6     #load RGB image as PIL.image.image type
7     img = image.load_img(img_path, target_size=(224, 224))
8
9     #convert PIL.image.image type to 3D tensor with shape (224, 224, 3)
10    x = image.img_to_array(img)
11
12    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
13    return np.expand_dims(x, axis = 0)
14
15 def paths_to_tensor(img_paths):
16     list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
17     return np.vstack(list_of_tensors)
```

## Making Predictions with ResNet-50

Getting the 4D tensor ready for ResNet-50, and for any other pre-trained model in Keras, requires some additional processing:

- First, the RGB image is converted to BGR by reordering the channels.
- All pre-trained models have the additional normalization step that the mean pixel (expressed in RGB as [103.939,116.779,123.68] [103.939,116.779,123.68] and calculated from all pixels in all images in ImageNet) must be subtracted from every pixel in each image.
- This is implemented in the imported function preprocess\_input.

Now that we have a way to format our image for supplying to ResNet-50, we are now ready to use the model to extract the predictions. This is accomplished with the predict method, which returns an array whose i-th entry is the model's predicted probability that the image belongs to the i-th ImageNet category. This is implemented in the ResNet50\_predict\_labels function below.

By taking the argmax of the predicted probability vector, we obtain an integer corresponding to the model's predicted object class, which we can identify with an object category through the use of this [dictionary](#).

```
In [7]: 1 from keras.applications.resnet50 import preprocess_input, decode_predictions
2
3 def ResNet50_predict_labels(img_path):
4
5     #return prediction vector for image located at img_path
6     img = preprocess_input(path_to_tensor(img_path))
7     return np.argmax(ResNet50_model.predict(img))
```

## A function for detecting dogs

In the image net data keys, the dictionary belonging to 151-268 entries contained images of dogs. Thus, we have to restrict our prediction to those points.

```
In [8]: 1 # returns "True" if a dog is detected in the image stored at img_path## return
2 def dog_detector(img_path):
3     prediction = ResNet50_predict_labels(img_path)
4     return ((prediction <= 268) & (prediction >= 151))
```

### Test the dog detector

```
In [9]: 1 dog_files_short = train_files[:100]
2 dog_count = 0
3 for img in dog_files_short:
4     isDog = dog_detector(img)
5     if isDog:
6         dog_count += 1
7     percentage = (dog_count/len(dog_files_short)) * 100
8 print('Percentage of dogs correctly classified as dogs: {}%'.format(percentage))
Percentage of dogs correctly classified as dogs: 100.0%
```

## Step 3 – A Convolutional Neural Network Classifier

### The data pre-processing work

Rescaling the data by dividing each pixel by 255.

```
In [11]: 1 from PIL import ImageFile
2 ImageFile.LOAD_TRUNCATED_IMAGES = True
3
4 # pre-process the data for Keras
5 train_tensors = paths_to_tensor(train_files).astype('float32')/255
6 valid_tensors = paths_to_tensor(valid_files).astype('float32')/255
7 test_tensors = paths_to_tensor(test_files).astype('float32')/255
100%|██████████| 6680/6680 [02:08<00:00, 51.96it/s]
100%|██████████| 835/835 [00:13<00:00, 60.10it/s]
100%|██████████| 836/836 [00:14<00:00, 59.49it/s]
```

## Data Augmentation

Creating more images from the set of images.

```
In [12]: 1 from keras.preprocessing.image import ImageDataGenerator  
2  
3 # create and configure augmented image generator  
4 datagen = ImageDataGenerator(  
5  
6     width_shift_range=0.1,    # randomly shift images horizontally (10% of total width)  
7     height_shift_range=0.1,   # randomly shift images vertically (10% of total height)  
8     horizontal_flip=True)  # randomly flip images horizontally  
9  
10 # fit augmented image generator on data  
11 datagen.fit(train_tensors)
```

## Defining the CNN architecture –

```
In [13]: 1 from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D  
2 from keras.layers import Dropout, Flatten, Dense, Activation  
3 from keras.models import Sequential  
4 from keras.layers.normalization import BatchNormalization  
5  
6 model = Sequential()  
7  
8 # Layer 1  
9 model.add(BatchNormalization(input_shape=(224, 224, 3)))  
10 model.add(Conv2D(filters=16, kernel_size=3, kernel_initializer='he_normal', activation='relu'))  
11 model.add(MaxPooling2D(pool_size=2))  
12 model.add(BatchNormalization())  
13  
14 # Layer 2  
15 model.add(Conv2D(filters=32, kernel_size=3, kernel_initializer='he_normal', activation='relu'))  
16 model.add(MaxPooling2D(pool_size=2))  
17 model.add(BatchNormalization())  
18  
19 # Layer 3  
20 model.add(Conv2D(filters=64, kernel_size=3, kernel_initializer='he_normal', activation='relu'))  
21 model.add(MaxPooling2D(pool_size=2))  
22 model.add(BatchNormalization())  
23  
24 # Layer 4  
25 model.add(Conv2D(filters=128, kernel_size=3, kernel_initializer='he_normal', activation='relu'))  
26 model.add(MaxPooling2D(pool_size=2))  
27 model.add(BatchNormalization())  
28  
29 # Layer 5  
30 model.add(Conv2D(filters=256, kernel_size=3, kernel_initializer='he_normal', activation='relu'))  
31 model.add(MaxPooling2D(pool_size=2))  
32 model.add(BatchNormalization())  
33  
34 model.add(GlobalAveragePooling2D())  
35  
36 model.add(Dense(133, activation='softmax'))  
37  
38 model.summary()
```

Layer (type)	Output Shape	Param #
batch_normalization_1 (Batch Normalization)	(None, 224, 224, 3)	12
conv2d_1 (Conv2D)	(None, 222, 222, 16)	448
max_pooling2d_2 (MaxPooling2D)	(None, 111, 111, 16)	0
batch_normalization_2 (Batch Normalization)	(None, 111, 111, 16)	64
conv2d_2 (Conv2D)	(None, 109, 109, 32)	4640
max_pooling2d_3 (MaxPooling2D)	(None, 54, 54, 32)	0
batch_normalization_3 (Batch Normalization)	(None, 54, 54, 32)	128
conv2d_3 (Conv2D)	(None, 52, 52, 64)	18496
max_pooling2d_4 (MaxPooling2D)	(None, 26, 26, 64)	0
batch_normalization_4 (Batch Normalization)	(None, 26, 26, 64)	256
conv2d_4 (Conv2D)	(None, 24, 24, 128)	73856
max_pooling2d_5 (MaxPooling2D)	(None, 12, 12, 128)	0
batch_normalization_5 (Batch Normalization)	(None, 12, 12, 128)	512
conv2d_5 (Conv2D)	(None, 10, 10, 256)	295168
max_pooling2d_6 (MaxPooling2D)	(None, 5, 5, 256)	0
batch_normalization_6 (Batch Normalization)	(None, 5, 5, 256)	1024
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 256)	0
dense_1 (Dense)	(None, 133)	34181
<b>Total params:</b>	<b>428,785</b>	
<b>Trainable params:</b>	<b>427,787</b>	
<b>Non-trainable params:</b>	<b>998</b>	

## Compiling the model

```
In [14]: 1 model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

## Training time

```
In [17]: 1 from keras.callbacks import ModelCheckpoint
2
3 epochs = 25
4 batch_size = 20
5
6 checkpointer = ModelCheckpoint(filepath='saved_models/weights.bestaugmented.from_scratch.hdf5',
7                                verbose=1, save_best_only=True)
8
9 # Using Image Augmentation
10 model.fit_generator(datagen.flow(train_tensors, train_targets, batch_size=batch_size),
11                      validation_data = (valid_tensors, valid_targets),
12                      steps_per_epoch = train_tensors.shape[0] // batch_size,
13                      epochs = epochs, callbacks = [checkpointer], verbose = 1)
```

```
In [69]: 1 # To save the weights
2
3 fname = "weights-Test-CNN-5-Layer-model.hdf5"
4 model.save_weights(fname, overwrite=True)
```

## Load the model with best validation accuracy

```
In [68]: 1 model.load_weights('saved_models/weights.bestaugmented.from_scratch.hdf5')
```

## Test the model and check the test accuracy

```
In [18]: 1 # get index of predicted dog breed for each image in test set
2 dog_breed_predictions = [np.argmax(model.predict(np.expand_dims(tensor, axis=0))) for tensor in test_tensors]
3
4 # report test accuracy
5 test_accuracy = 100*np.sum(np.array(dog_breed_predictions)==np.argmax(test_targets, axis=1))/len(dog_breed_predictions)
6 print('Test accuracy: %.4f%%' % test_accuracy)
```

Test accuracy: 49.4019%

After training our CNN architecture we got a test accuracy of 49.4% which is pretty bad as an image can be of dog or not. Getting 50% accuracy is bad for any deep learning architecture. Thus, we decided to use the concepts of **transfer learning** to improve our results.

VGG-16, VGG-19, Alex-Net, ResNet-50 are some of the pre-trained model available in keras module for transfer learning. We converted our dataset using bottleneck features so that we don't lose any of the information from our data and implemented **ResNet-50** architecture to our dataset.

## Obtain bottleneck features

In the code block below, I extract the bottleneck features corresponding to the train, test, and validation sets.

```
In [19]: 1 #Obtain bottleneck features from another pre-trained CNN.  
2  
3 bottleneck_features = np.load('bottleneck_features/DogResnet50Data.npz')  
4 train_ResNet50 = bottleneck_features['train']  
5 valid_ResNet50 = bottleneck_features['valid']  
6 test_ResNet50 = bottleneck_features['test']
```

## ResNet Model architecture

```
In [20]: 1 from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D  
2 from keras.layers import Dropout, Flatten, Dense, Activation  
3 from keras.models import Sequential  
4 from keras.layers.normalization import BatchNormalization  
5  
6 ResNet_model = Sequential()  
7 ResNet_model.add(GlobalAveragePooling2D(input_shape=train_ResNet50.shape[1:]))  
8 ResNet_model.add(Dense(133, activation='softmax'))  
9  
10 ResNet_model.summary()
```

Layer (type)	Output Shape	Param #
global_average_pooling2d_2 ( GlobalAveragePooling2D )	(None, 2048)	0
dense_2 ( Dense )	(None, 133)	272517
Total params:	272,517	
Trainable params:	272,517	
Non-trainable params:	0	

## Compile the model

```
In [21]: 1 from keras.optimizers import Adam, Adamax  
2  
3 ResNet_model.compile(loss = 'categorical_crossentropy', optimizer = Adamax(lr = 0.002), metrics = ['accuracy'])
```

## Training time 2.0

```
In [22]: 1 from keras.callbacks import ModelCheckpoint  
2  
3 checkpointer = ModelCheckpoint(filepath = 'saved_models/weights.best_adamax.ResNet50.hdf5',  
4                                verbose = 1, save_best_only = True)  
5  
6 epochs = 30  
7 batch_size = 64  
8  
9 ResNet_model.fit(train_ResNet50, train_targets,  
10                  validation_data = (valid_ResNet50, valid_targets),  
11                  epochs = epochs, batch_size = batch_size, callbacks = [checkpointer], verbose = 1)
```

```
In [70]: 1 # To save the weights
2
3 fname = "weights-Test-CNN-ResNet-Layer-model.hdf5"
4 ResNet_model.save_weights(fname, overwrite=True)

Load the model with best validation loss

In [23]: 1 ResNet_model.load_weights('saved_models/weights.best_adamax.ResNet50.hdf5')

Test the model accuracy

In [64]: 1 #get index of predicted dog breed for each image in test set
2 ResNet50_predictions = [np.argmax(ResNet_model.predict(np.expand_dims(feature, axis=0))) for feature in test_ResNet50]
3
4 #report test accuracy
5 test_accuracy = 100*np.sum(np.array(ResNet50_predictions)==np.argmax(test_targets, axis=1))/len(ResNet50_predictions)
6 print('Test accuracy: %.4f%%' % test_accuracy)

Test accuracy: 84.0909%
```

After training for 30 epochs we got an accuracy of 84.09% which is pretty good for test accuracy.

Train accuracy was equal to 99.8% and validation loss = 0.54

We wrote some functions to predict the breed of the dog using our model by providing an image of a dog or any other thing to our **predict\_breed** function.

## Training Algorithm –

We used **object localization** technique which is implemented by convolutions in CNN architecture. In this algorithm, we apply several points on the feature of say face and the localize them. Further when we form a fully connected layer these points together make a feature set.

For example, in our dog breed classifier the feature set which define a breed as different from other breed from just seeing the face is the location of eyes, nose, ears, mouth etc.

## Optimization Algorithm –

We used *Adam Optimization Algorithm* which is a replacement of classic stochastic gradient descent algorithm. Its performance increases significantly with increase in layers of deep learning architecture.

Some of the features are:

- Straightforward to implement
- Computationally efficient
- Appropriate for non-stationary objects

## Batch Normalization –

As the parameters keep changing in the hidden layers of a deep neural network it is a complicated task in itself to learn all the parameters throughout the training process. The training process slows down as we have to carefully chose the parameters and apply lower learning rates to avoid non-linearity. We refer to this phenomenon as internal covariate shift and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases, eliminates the need for Dropout.

Batch Normalization achieves the same accuracy with 14 times fewer training steps and beats the original model by a significant margin.

## Activation Function –

15

15e used ReLU (Rectified Linear Unit) activation function within our hidden layers. ReLU doesn't suffer from vanishing gradient problem. It doesn't update the weights when the output of a neuron is negative as the output of ReLU for a negative value is zero. To overcome this problem, we generally use Leaky ReLU.

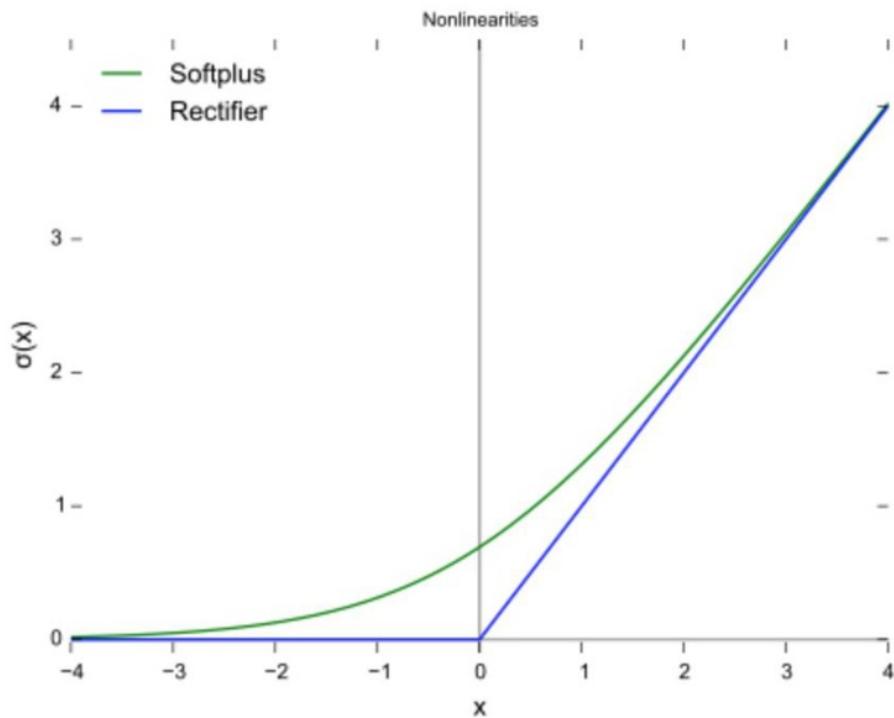


Fig 11. ReLU activation function

Further, at the output layer we used **Softmax** activation function. Since, softmax is responsible for providing different number of values between 0 and 1. It gives the best probability as an output. Thus, using **softmax** activation function for multi class classification problem is best way to go.

## Predict the dog breed using this model

```
In [29]: 1 def extract_Resnet50(tensor):
2     from keras.applications.resnet50 import ResNet50, preprocess_input
3     return ResNet50(weights='imagenet', include_top=False).predict(preprocess_input(tensor))

In [51]: 1 #To show the images
2 import matplotlib.pyplot as plt
3 import matplotlib.image as mpimg
4
5 #This function takes a path as input image and return the breed of the dog.
6 def ResNet50_predict_breed(img_path):
7     # extract bottleneck features
8     bottleneck_feature = extract_Resnet50(path_to_tensor(img_path))
9
10    # obtain predicted vector
11    predicted_vector = ResNet_model.predict(bottleneck_feature)
12
13    # return dog breed that is predicted by the model
14    breed = dog_names[np.argmax(predicted_vector)]
15    image = mpimg.imread(img_path)
16    imgplot = plt.imshow(image)
17
18    if dog_detector(img_path) == True:
19        return print("The breed of dog is a {}".format(breed))

In [52]: 1 def dog_detector(img_path):
2     prediction = ResNet50_predict_labels(img_path)
3     return ((prediction <= 268) & (prediction >= 151))

In [53]: 1 def predict_breed(img_path):
2     isDog = dog_detector(img_path)
3     if isDog:
4         print("Detected a dog.\n")
5         breed = ResNet50_predict_breed(img_path)
6         return breed
7     else:
8         print('No dog picture found.')
```

## 3.2 Tools Used

14

- We used **Python** language to implement our code. Modules are defined in Python code, which is compact, easier to debug, and allows for ease of extensibility.  
9
- **Keras**, a high-level neural networks API (programming framework), written in Python and capable of running on top of several lower-level frameworks including TensorFlow and CNTK.  
6
- **TensorFlow** is a Python library <sup>6</sup> that allows users to express arbitrary computation as <sup>6</sup> graph of data flows. We chose to use it because of its design and ease of use. It is primarily used for Deep Learning in practice and research.  
4
- We matched our dataset with the data on **ImageNet**. ImageNet is an image dataset organized according to the WorldNet hierarchy in which each node of hierarchy is depicted by hundreds and thousands of images.
- We used **Jupyter Notebook** to write code and documentation.

## Chapter 4

# RESULT ANALYSIS

### 4.1 Result Analysis

$$\text{Accuracy} = \frac{\text{Number of correct prediction}}{\text{Number of total predictions}}$$

We obtained a test accuracy of 49.4% on training our dataset with the model we create using CNN architecture.

*Test accuracy = 49.4%*

*Number of epochs = 25*

*Batch size = 20 images*

*Where 1 epoch means one complete cycle in the batches of 20 images for all the images.*

*Training accuracy = 84%*

*Validation loss = 22.8%*

*The lower the loss will be the better our model will be. Validation loss refers to the loss we are getting on the validation data.*

On improving the architecture i.e. by using Res-Net50 architecture for training the dataset and creating a model we got an accuracy of 84.09%

*Test accuracy = 84.09%*

*Number of epochs = 30*

*Batch size = 64 images*

*Training accuracy = 99.8%*

*Validation loss = 5.4%*

The following graph can illustrate the details:

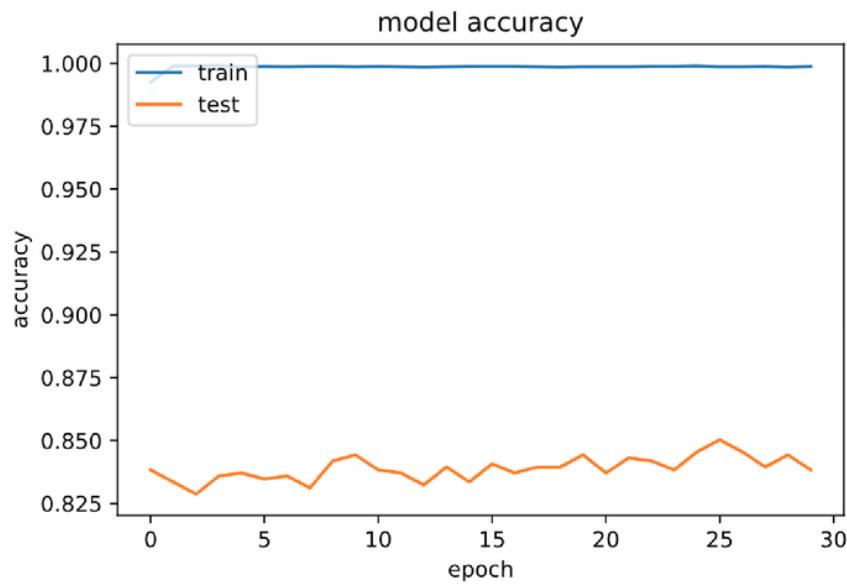


Fig 12. Model accuracy

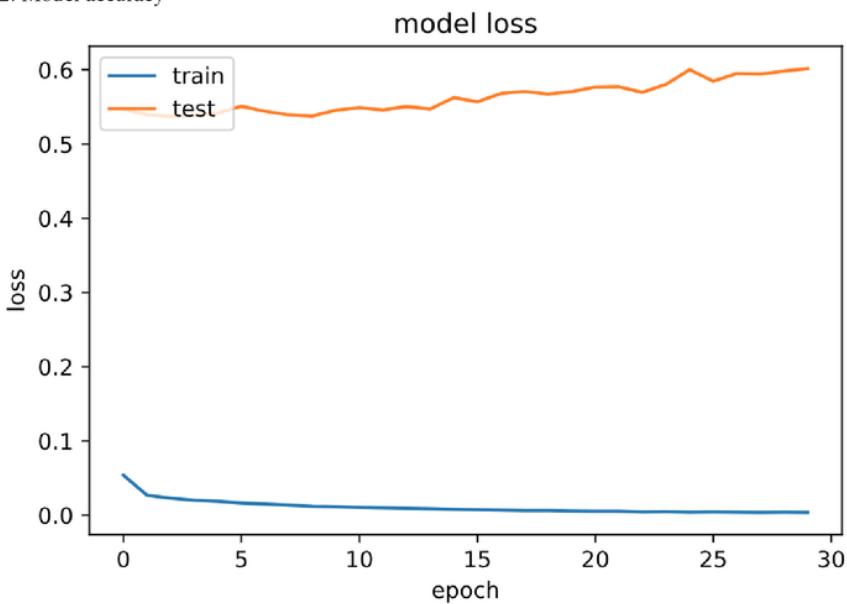


Fig 13. Model loss graph

## **4.2 Significance of the result obtained**

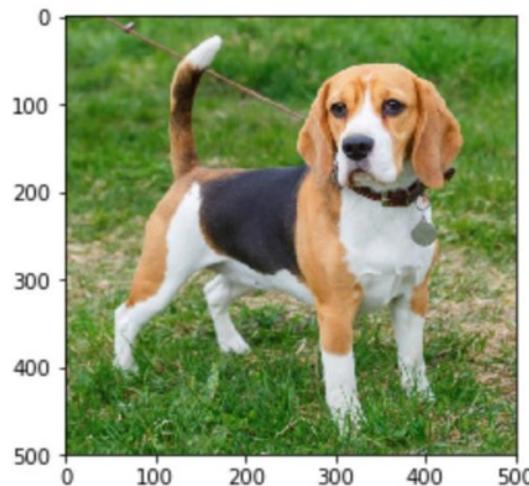
We got pretty good results on improving our model using the concepts of transfer learning. We got a leap from 49% to 84% which led to less error in the testing of the images.

On testing 10 images of dogs and non-dogs, we got one error and rest were detected as follows:

```
In [54]: 1 predict_breed('testImages/beagle.jpg')
```

Detected a dog.

The breed of dog is a Beagle



```
In [55]: 1 predict_breed('testImages/afghan_hound.jpeg')
```

Detected a dog.

The breed of dog is a Afghan\_hound

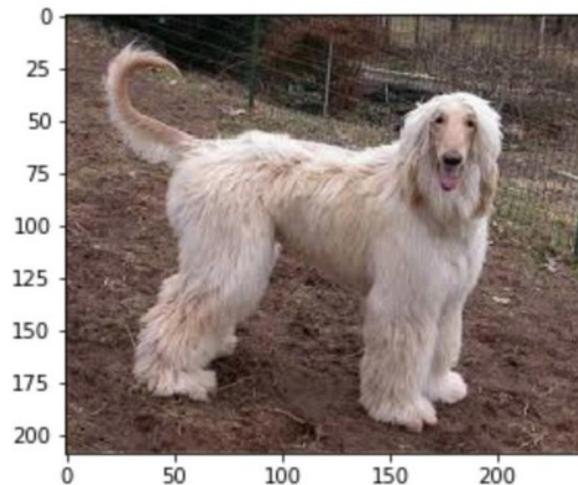
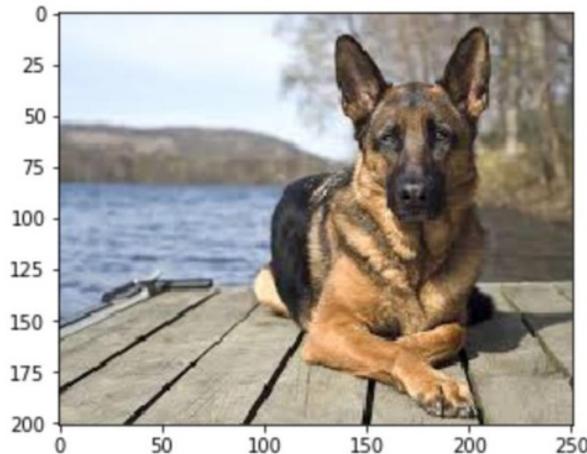


Fig 14. Test Images

```
In [56]: 1 predict_breed('testImages/german_shepherd.jpeg')
```

Detected a dog.

The breed of dog is a Belgian\_malinois



Here we can see the first error made by our model..

---

```
In [57]: 1 predict_breed('testImages/Labrador_retriever.jpg')
```

Detected a dog.

The breed of dog is a Labrador\_retriever



Fig 15. Test Images

```
In [58]: 1 predict_breed('testImages/American_water_spaniel.jpg')
```

Detected a dog.

The breed of dog is a American\_water\_spaniel



```
In [59]: 1 predict_breed('testImages/german_shepherd2.jpeg')
```

Detected a dog.

The breed of dog is a German\_shepherd\_dog

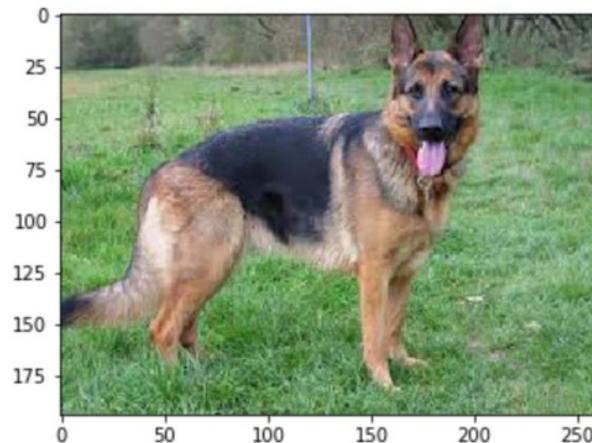


Fig 16. Test Images

### **4.3 Any deviation from the expected results**

Out of 10 images we tested, we found one error. As we have mentioned earlier our test accuracy is 84.09% which is not at par as we compare other test accuracies using Neural Networks. Nowadays people achieve accuracy up to 99% and then only the test results are up most accurate.

### **4.4 Conclusion**

Our CNN model is able to detect a dog's breed. It returns the picture provided in the image path for a specific dog breed.

Using our CNN architecture which was made from scratch we got a test accuracy of 49.4% which was way too low. As it is always a 50-50 chance if a dog belongs to a specific breed or not.

Thus, we applied transfer learning concepts and applied a pre-trained model to our dataset. We used ResNet-50 model which was trained on 1.4 million images on Image-Net dataset. In the dataset of Image-Net, the images from index 151 to 268 belonged to dogs. We scraped the indexes and trained our data using the ResNet-50 model.

This increased our test accuracy significantly from 49% to 84% which is pretty good for a classifier.

## Chapter 5

### FUTURE SCOPE OF WORK

#### 5.1 Future Scope of Work

We can detect a dog using various models now. We can apply different transfer learning models like ResNet-101 and ResNet-152.

We can also apply Alex-Net model and try on our dataset. This will improve the efficiency of our model. This machine learning model can be used in development of dog breed identifier application for mobile phones.

We can use TensorFlow API which is helpful in creating mobile applications for IOS and Android. The concept used in this model can be used to build other machine learning models for classification.



# Report

## ORIGINALITY REPORT



### PRIMARY SOURCES

- |   |   |    |
|---|---|----|
| 1 | <a href="http://sandipanweb.wordpress.com">sandipanweb.wordpress.com</a>  | 4% |
| 2 | <a href="http://proceedings.mlr.press">proceedings.mlr.press</a>  | 3% |
| 3 | <a href="http://drrajivdesaimd.com">drrajivdesaimd.com</a>  | 2% |
| 4 | Neha Sharma, Vibhor Jain, Anju Mishra. "An Analysis Of Convolutional Neural Networks For Image Classification", Procedia Computer Science, 2018 | 2% |
| 5 | Submitted to National Research University Higher School of Economics  | 2% |
| 6 | <a href="http://www.safaribooksonline.com">www.safaribooksonline.com</a>  | 1% |
| 7 | Submitted to Universiti Teknikal Malaysia Melaka  | 1% |
- 
- 1 sandipanweb.wordpress.com  
Internet Source
- 2 proceedings.mlr.press  
Internet Source
- 3 drrajivdesaimd.com  
Internet Source
- 4 Neha Sharma, Vibhor Jain, Anju Mishra. "An Analysis Of Convolutional Neural Networks For Image Classification", Procedia Computer Science, 2018  
Publication
- 5 Submitted to National Research University Higher School of Economics  
Student Paper
- 6 www.safaribooksonline.com  
Internet Source
- 7 Submitted to Universiti Teknikal Malaysia Melaka  
Student Paper

8	<a href="http://www.cpij.or.jp">www.cpij.or.jp</a> Internet Source	1 %
9	<a href="http://arun-aiml.blogspot.de">arun-aiml.blogspot.de</a> Internet Source	1 %
10	<a href="http://en.wikipedia.org">en.wikipedia.org</a> Internet Source	1 %
11	Submitted to Nanyang Technological University, Singapore Student Paper	1 %
12	Submitted to University of South Australia Student Paper	1 %
13	Submitted to Griffith University Student Paper	<1 %
14	<a href="http://github.com">github.com</a> Internet Source	<1 %
15	"Speech and Computer", Springer Nature, 2016 Publication	<1 %
16	Submitted to University of Sydney Student Paper	<1 %
17	Submitted to BITS, Pilani-Dubai Student Paper	<1 %
18	Submitted to University of Macau Student Paper	<1 %

---

Exclude quotes      On                  Exclude matches      < 10 words

Exclude bibliography      On