Report

by Ankit Kumar

Submission date: 12-Jul-2018 05:00PM (UTC+0800)
Submission ID: 982048987
File name: Report_Ananta_Ankit.docx (8.81M)
Word count: 4093
Character count: 20958

CHAPTER 1
INTRODUCTION

1.1 Area of Work

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 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 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.

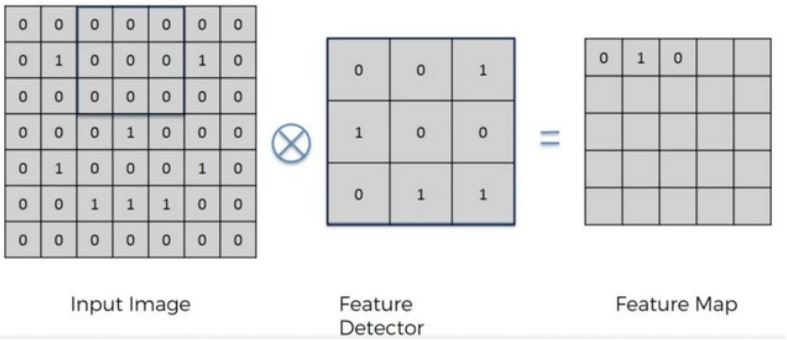


Fig 1. Convolution Step

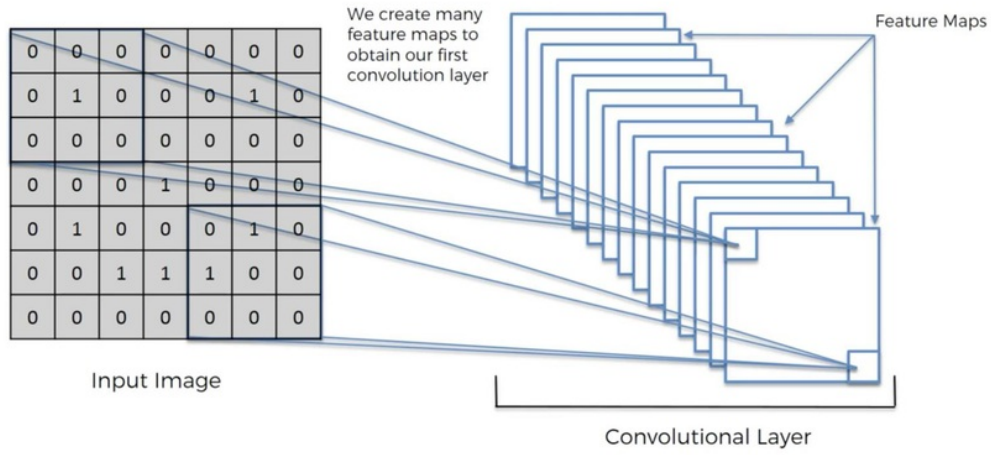


Fig 2. How Convolution works

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.

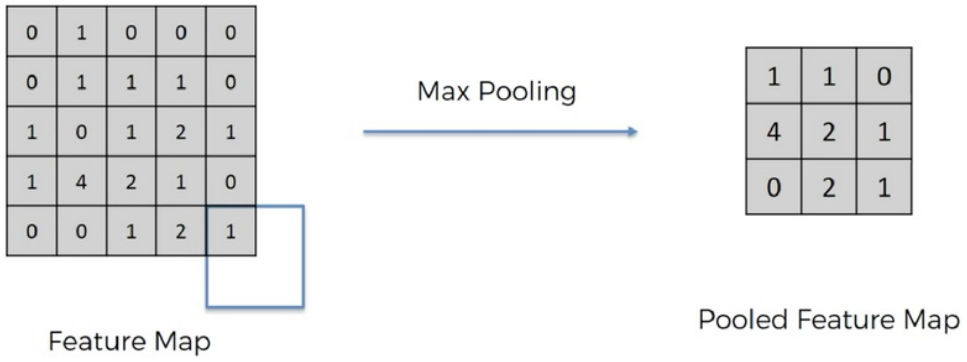


Fig 3. Pooling layer

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.

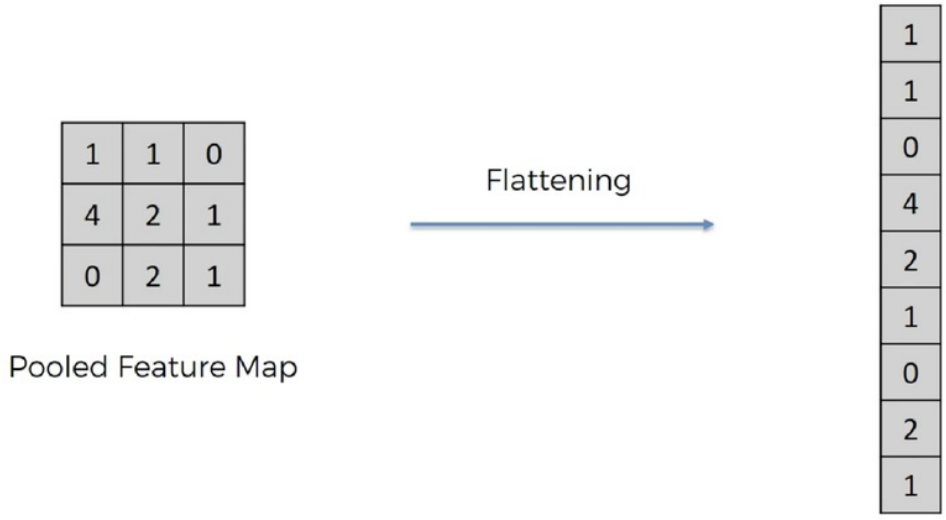


Fig 4. Flattening

FULLY CONNECTED LAYERS

Fully Connected Layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as traditional multi-layer perceptron neural network.

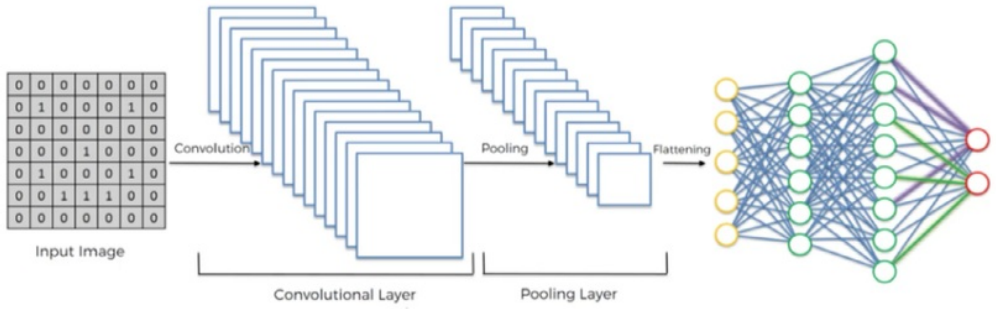


Fig 5. Fully Connected Layers

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.**

Transfer Learning is also called 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 is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned. It is related to problems such as multi-tasking and concept drift and is not exclusively an area of study for deep learning.
It is an optimization, a shortcut to saving time or getting better performance.
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 transfer learning:

- **Higher start:** The initial skill (before refining the model) on the source model is higher than it otherwise would be.
- **Higher slope:** The rate of improvement of skill during training of the source model is steeper than it otherwise would be.
- **Higher asymptote:** The converged skill of the trained model is better than it otherwise would be.

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 data.

RESNET-50

ResNet is a short name for 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 difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

The problem of very deep neural network

In recent years, neural networks have become deeper, with state-of-the-art networks going from just a few layers (e.g., Alex-Net) to over a hundred layers.

The main benefit of a very deep network is that it can represent very complex functions. It can also learn features at many different levels of abstraction, from edges (at the lower layers) to very complex features (at the deeper layers). However, using a deeper network doesn't always help. A huge barrier to training them 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).

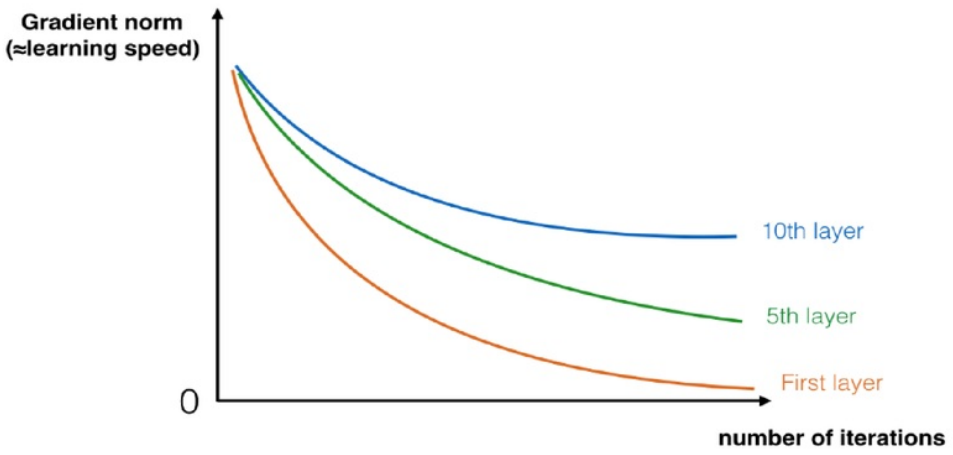


Fig 6. Problem of deep neural network

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

The image on the left shows the "main path" through the network. The image on the right adds a shortcut to the main path. By stacking these ResNet blocks on top of each other, you can form a very deep network.

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).

Two main types of blocks are used in a ResNet, depending mainly on whether the input/output dimensions are same or different. You are going to implement both of them.

Building Res-Net50 model

The details of this ResNet-50 model are:

- Zero-padding pads the input with a pad of (3,3)
- Stage 1:
  - The 2D Convolution has 64 filters of shape (7,7) and uses a stride of (2,2). Its name is "conv1".
  - BatchNorm is applied to the channels axis of the input.
  - MaxPooling uses a (3,3) window and a (2,2) stride.
- Stage 2:
  - The convolutional block uses three set of filters of size [64,64,256], "f" is 3, "s" is 1 and the block is "a".
  - The 2 identity blocks use three set of filters of size [64,64,256], "f" is 3 and the blocks are "b" and "c".
- Stage 3:
  - The convolutional block uses three set of filters of size [128,128,512], "f" is 3, "s" is 2 and the block is "a".
  - The 3 identity blocks use three set of filters of size [128,128,512], "f" is 3 and the blocks are "b", "c" and "d".
- Stage 4:

    - The convolutional block uses three set of filters of size [256, 256, 1024], "f" is 3, "s" is 2 and the block is "a".
    - The 5 identity blocks use three set of filters of size [256, 256, 1024], "f" is 3 and the blocks are "b", "c", "d", "e" and "f".
- Stage 5:
    - The convolutional block uses three set of filters of size [512, 512, 2048], "f" is 3, "s" is 2 and the block is "a".
    - The 2 identity blocks use three set of filters of size [512, 512, 2048], "f" is 3 and the blocks are "b" and "c".
- The 2D Average Pooling uses a window of shape (2,2) and its name is "avg_pool".
- The flatten doesn't have any hyperparameters or name.
- The Fully Connected (Dense) layer reduces its input to the number of classes using a softmax activation. Its name should be 'fc' + str(classes)

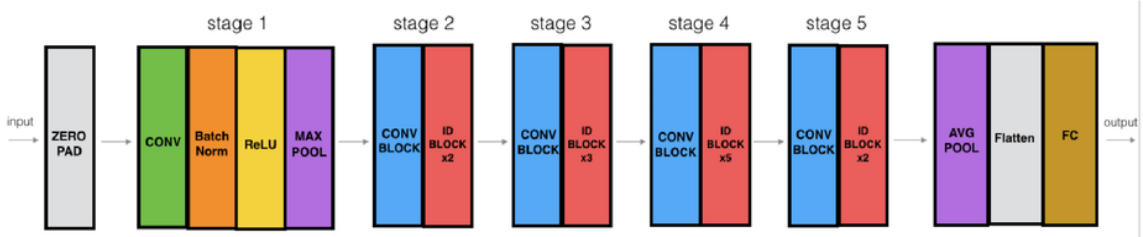
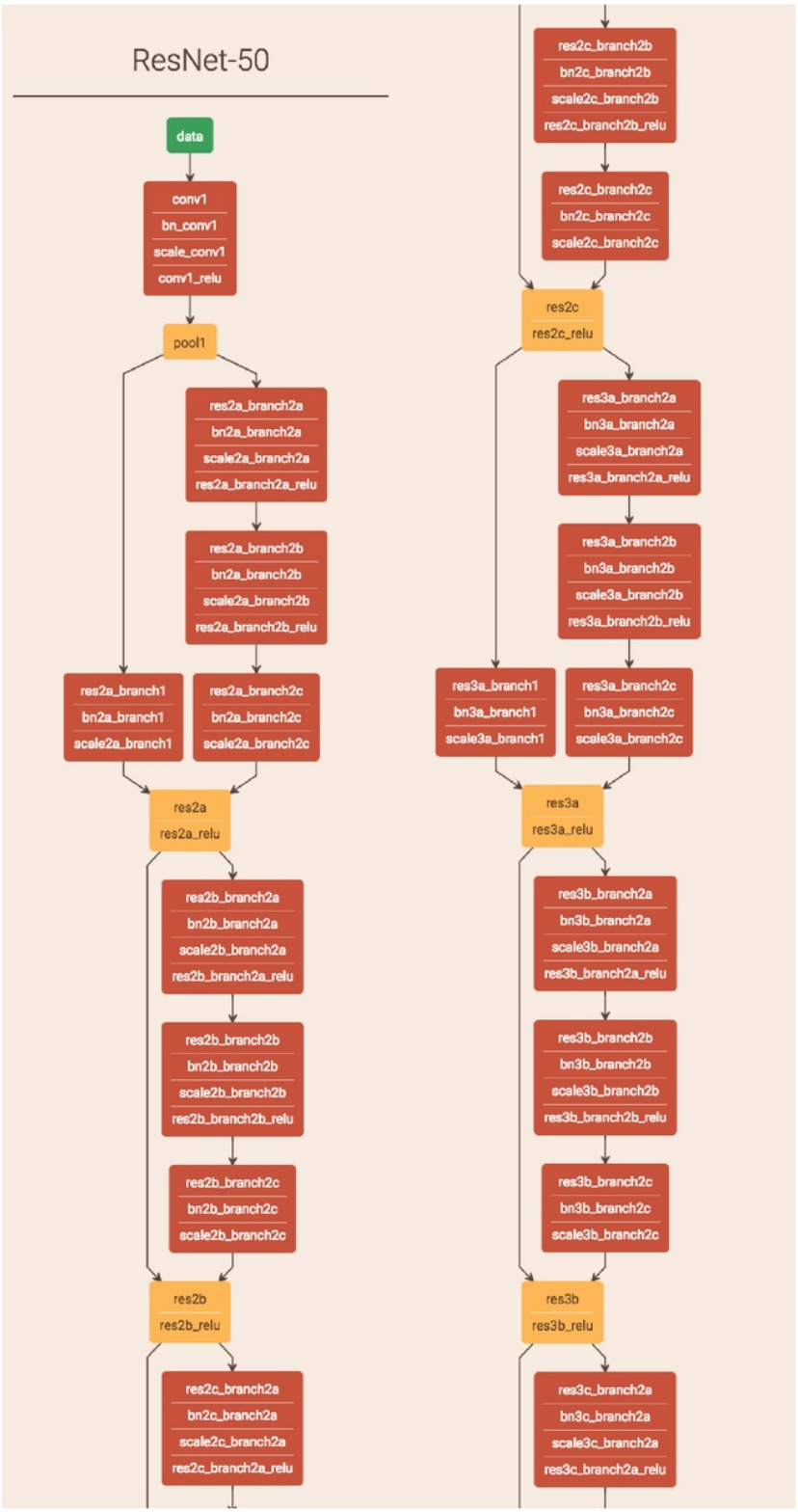
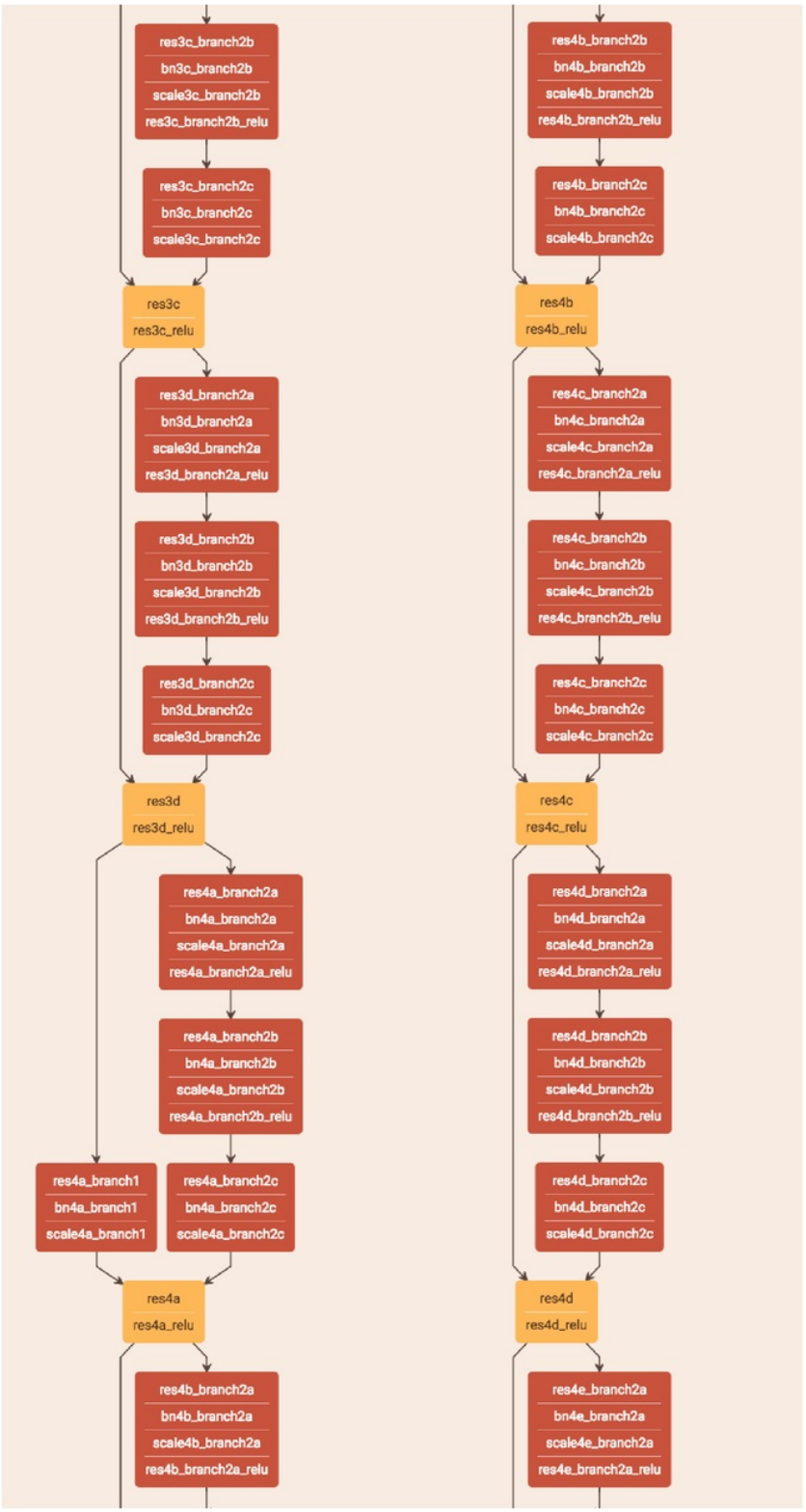


Fig 8. ResNet-50 building stages


ResNet-50
data
conv1
bn_conv1
scale_conv1
conv1_relu
pool1
res2a_branch2a
bn2a_branch2a
scale2a_branch2a
res2a_branch2a_relu
res2a_branch2b
bn2a_branch2b
scale2a_branch2b
res2a_branch2b_relu
res2a_branch1
bn2a_branch1
scale2a_branch1
res2a_branch2c
bn2a_branch2c
scale2a_branch2c
res2a
res2a_relu
res2b_branch2a
bn2b_branch2a
scale2b_branch2a
res2b_branch2a_relu
res2b_branch2b
bn2b_branch2b
scale2b_branch2b
res2b_branch2b_relu
res2b_branch2c
bn2b_branch2c
scale2b_branch2c
res2b
res2b_relu
res2c_branch2a
bn2c_branch2a
scale2c_branch2a
res2c_branch2a_relu
res2c_branch2b
bn2c_branch2b
scale2c_branch2b
res2c_branch2b_relu
res2c_branch2c
bn2c_branch2c
scale2c_branch2c
res2c
res2c_relu
res3a_branch2a
bn3a_branch2a
scale3a_branch2a
res3a_branch2a_relu
res3a_branch2b
bn3a_branch2b
scale3a_branch2b
res3a_branch2b_relu
res3a_branch1
bn3a_branch1
scale3a_branch1
res3a_branch2c
bn3a_branch2c
scale3a_branch2c
res3a
res3a_relu
res3b_branch2a
bn3b_branch2a
scale3b_branch2a
res3b_branch2a_relu
res3b_branch2b
bn3b_branch2b
scale3b_branch2b
res3b_branch2b_relu
res3b_branch2c
bn3b_branch2c
scale3b_branch2c
res3b
res3b_relu
res3c_branch2a
bn3c_branch2a
scale3c_branch2a
res3c_branch2a_relu


res3c_branch2b
bn3c_branch2b
scale3c_branch2b
res3c_branch2b_relu
res3c_branch2c
bn3c_branch2c
scale3c_branch2c
res3c
res3c_relu
res3d_branch2a
bn3d_branch2a
scale3d_branch2a
res3d_branch2a_relu
res3d_branch2b
bn3d_branch2b
scale3d_branch2b
res3d_branch2b_relu
res3d_branch2c
bn3d_branch2c
scale3d_branch2c
res3d
res3d_relu
res4a_branch2a
bn4a_branch2a
scale4a_branch2a
res4a_branch2a_relu
res4a_branch2b
bn4a_branch2b
scale4a_branch2b
res4a_branch2b_relu
res4a_branch1
bn4a_branch1
scale4a_branch1
res4a_branch2c
bn4a_branch2c
scale4a_branch2c
res4a
res4a_relu
res4b_branch2a
bn4b_branch2a
scale4b_branch2a
res4b_branch2a_relu
res4b_branch2b
bn4b_branch2b
scale4b_branch2b
res4b_branch2b_relu
res4b_branch2c
bn4b_branch2c
scale4b_branch2c
res4b
res4b_relu
res4c_branch2a
bn4c_branch2a
scale4c_branch2a
res4c_branch2a_relu
res4c_branch2b
bn4c_branch2b
scale4c_branch2b
res4c_branch2b_relu
res4c_branch2c
bn4c_branch2c
scale4c_branch2c
res4c
res4c_relu
res4d_branch2a
bn4d_branch2a
scale4d_branch2a
res4d_branch2a_relu
res4d_branch2b
bn4d_branch2b
scale4d_branch2b
res4d_branch2b_relu
res4d_branch2c
bn4d_branch2c
scale4d_branch2c
res4d
res4d_relu
res4e_branch2a
bn4e_branch2a
scale4e_branch2a
res4e_branch2a_relu

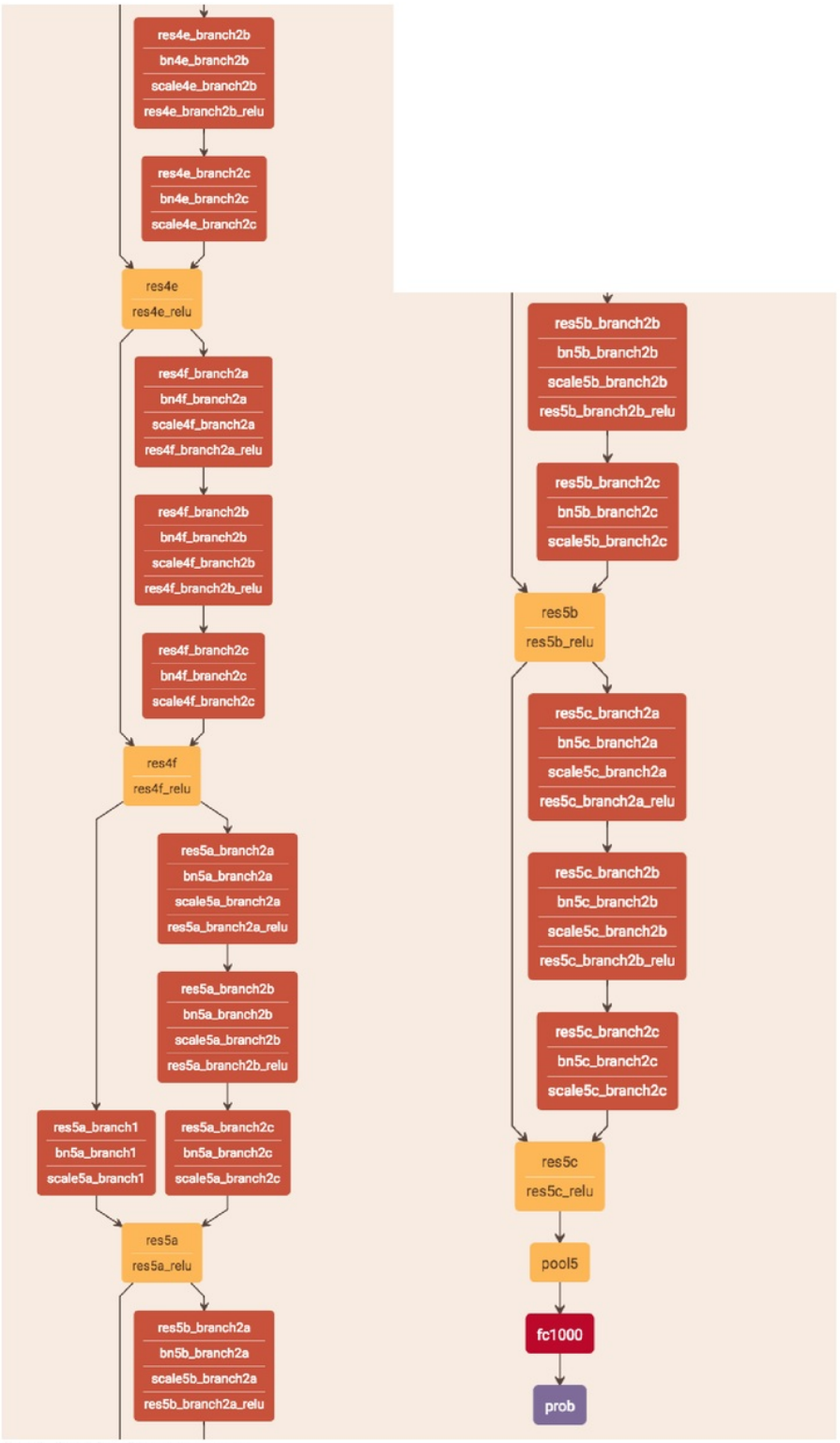


Fig 9. ResNet-50 architecture

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 task of assigning breed of dog from images is exceptionally challenging as more than one breed looks just like another.
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:

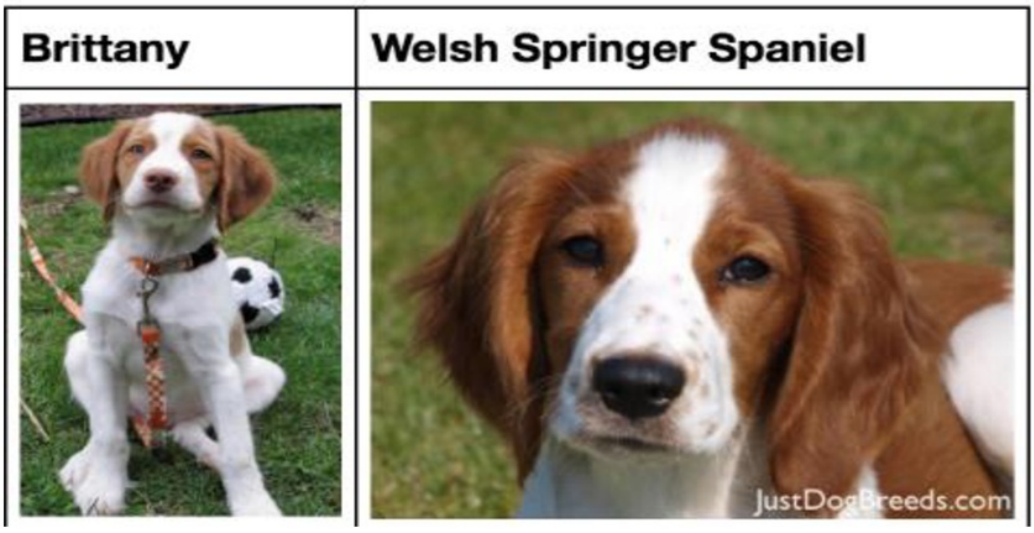


Fig 10. Similar dog breeds

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.

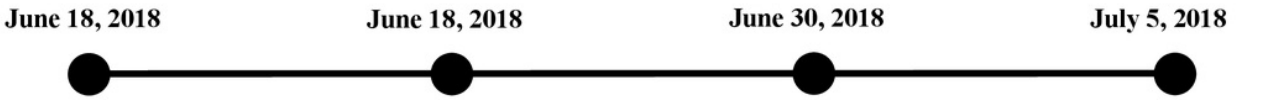
- We wanted to analyze the functions and working of TensorFlow and use them as a go in our model. We tried building a 5-layer CNN architecture (using keras API which uses TensorFlow as backend) which will classify the breed of the dog from any supplied image from a user.

  TensorFlow 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



- Project Guide Assigned
- Project Allocated

- Started the deeplearning.ai course specialization on coursera
- Completed all the five courses within 2 weeks of time (Theory part)

- Started working on our project
- Made a CNN architecture from scratch
- To improve the results of test accuracy we used the concepts of **Transfer Learning**
- Applied ResNet-50 model architecture on our

- Got an accuracy of 49.4% using our CNN model
- Accuracy improved significantly when we applied transfer learning. From 49.4% to 84%

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 benefit of a very deep network is that it can represent very complex functions. It can also learn features at many different levels of abstraction, from edges (at the lower layers) to very complex features (at the deeper layers). However, using a deeper network doesn't always help. A huge barrier to training them is vanishing gradients: very deep networks often have a gradient signal that goes to zero quickly, thus making gradient descent unbearably slow.

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 of 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]:
```
from sklearn.datasets import load_files
from keras.utils import np_utils
import numpy as np
from glob import glob
```

```
Using TensorFlow backend.
```

Function to load train, test and validate dataset

In [3]:
```
def load_dataset(path):
    data = load_files(path)
    dog_files = np.array(data['filenames'])
    dog_targets = np_utils.to_categorical(np.array(data['target']), 133)
    return dog_files, dog_targets
```

In [4]:
```
#Load train, test and validation dataset

train_files, train_targets = load_dataset('dogImages/train')
valid_files, valid_targets = load_dataset('dogImages/valid')
test_files, test_targets = load_dataset('dogImages/test')

#Load list of dog names
dog_names = [item[20:-1] for item in sorted(glob('dogImages/train/*/'))]

#Print different statistics about the dataset
print('There are %d total dog categories.' % len(dog_names))
print('There are %d total dog images.' % len(np.hstack([train_files, valid_files, test_files])))

print('There are %d total training images of dogs.' % len(train_files))
print('There are %d total validation images of dogs.' % len(valid_files))
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]:
```
from keras.applications.resnet50 import ResNet50
import h5py

#defining the model
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]:

```
from keras.preprocessing import image
from tqdm import tqdm

def path_to_tensor(img_path):

    #load RGB image as PIL.image.image type
    img = image.load_img(img_path, target_size=(224, 224))

    #convert PIL.image.image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)

    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    return np.expand_dims(x, axis = 0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    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]:

```
from keras.applications.resnet50 import preprocess_input, decode_predictions

def ResNet50_predict_labels(img_path):

    #return prediction vector for image located at img_path
    img = preprocess_input(path_to_tensor(img_path))
    return np.argmax(ResNet50_model.predict(img))
```

Write a dog detector

While looking at the dictionary, you will notice that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from 'Chihuahua' to 'Mexican hairless'. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained ResNet-50 model, we need only check if the ResNet50_predict_labels function above returns a value between 151 and 268 (inclusive).

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

Test the dog detector

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

```
Percentage of dogs correctly classified as dogs: 100.0%
```

Step 3 - Create a CNN to classify dog breeds

Pre-process the data

We rescale the data by dividing every pixel in every image by 255.

In [11]:
```
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

# pre-process the data for Keras
train_tensors = paths_to_tensor(train_files).astype('float32')/255
valid_tensors = paths_to_tensor(valid_files).astype('float32')/255
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]:

```
from keras.preprocessing.image import ImageDataGenerator

# create and configure augmented image generator
datagen = ImageDataGenerator(

    width_shift_range=0.1,   # randomly shift images horizontally (10% of total width)
    height_shift_range=0.1,  # randomly shift images vertically (10% of total height)
    horizontal_flip=True)    # randomly flip images horizontally

# fit augmented image generator on data
datagen.fit(train_tensors)
```

Defining the CNN architecture –

In [13]:

```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense, Activation
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization

model = Sequential()

# Layer 1
model.add(BatchNormalization(input_shape=(224, 224, 3)))
model.add(Conv2D(filters=16, kernel_size=3, kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

# Layer 2
model.add(Conv2D(filters=32, kernel_size=3, kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

# Layer 3
model.add(Conv2D(filters=64, kernel_size=3, kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

# Layer 4
model.add(Conv2D(filters=128, kernel_size=3, kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

# Layer 5
model.add(Conv2D(filters=256, kernel_size=3, kernel_initializer='he_normal', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(BatchNormalization())

model.add(GlobalAveragePooling2D())

model.add(Dense(133, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_1 (Batch (None, 224, 224, 3)       12
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 222, 222, 16)      448
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 16)      0
_________________________________________________________________
batch_normalization_2 (Batch (None, 111, 111, 16)      64
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 109, 109, 32)      4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 54, 54, 32)        128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 52, 52, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 26, 26, 64)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 26, 26, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 128)       73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 256)       295168
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 5, 5, 256)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 5, 5, 256)         1024
_________________________________________________________________
global_average_pooling2d_1 ( (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 133)               34181
=================================================================
Total params: 428,785
Trainable params: 427,787
Non-trainable params: 998
```

Compiling the model

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

Training time

In [17]:
```
from keras.callbacks import ModelCheckpoint

epochs = 25
batch_size = 20

checkpointer = ModelCheckpoint(filepath='saved_models/weights.bestaugmented.from_scratch.hdf5',
                               verbose=1, save_best_only=True)

# Using Image Augmentation
model.fit_generator(datagen.flow(train_tensors, train_targets, batch_size=batch_size),
                    validation_data = (valid_tensors, valid_targets),
                    steps_per_epoch = train_tensors.shape[0] // batch_size,
                    epochs = epochs, callbacks = [checkpointer], verbose = 1)
```

In [69]:
```
# To save the weights

fname = "weights-Test-CNN-5-Layer-model.hdf5"
model.save_weights(fname, overwrite=True)
```

Load the model with best validation accuracy

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

Test the model and check the test accuracy

In [18]:
```
# get index of predicted dog breed for each image in test set
dog_breed_predictions = [np.argmax(model.predict(np.expand_dims(tensor, axis=0))) for tensor in test_tensors]

# report test accuracy
test_accuracy = 100*np.sum(np.array(dog_breed_predictions)==np.argmax(test_targets, axis=1))/len(dog_breed_predictions)
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]:
```
#Obtain bottleneck features from another pre-trained CNN.

bottleneck_features = np.load('bottleneck_features/DogResnet50Data.npz')
train_ResNet50 = bottleneck_features['train']
valid_ResNet50 = bottleneck_features['valid']
test_ResNet50 = bottleneck_features['test']
```

ResNet Model architecture

In [20]:
```
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense, Activation
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization

ResNet_model = Sequential()
ResNet_model.add(GlobalAveragePooling2D(input_shape=train_ResNet50.shape[1:]))
ResNet_model.add(Dense(133, activation='softmax'))

ResNet_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_2 ( (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]:
```
from keras.optimizers import Adam, Adamax

ResNet_model.compile(loss = 'categorical_crossentropy', optimizer = Adamax(lr = 0.002), metrics = ['accuracy'])
```

Training time 2.0

In [22]:
```
from keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint(filepath = 'saved_models/weights.best_adamax.ResNet50.hdf5',
                               verbose = 1, save_best_only = True)

epochs = 30
batch_size = 64

ResNet_model.fit(train_ResNet50, train_targets,
          validation_data = (valid_ResNet50, valid_targets),
          epochs = epochs, batch_size = batch_size, callbacks = [checkpointer], verbose = 1)
```

In [70]:
```
# To save the weights

fname = "weights-Test-CNN-ResNet-Layer-model.hdf5"
ResNet_model.save_weights(fname, overwrite=True)
```

Load the model with best validation loss

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

Test the model accuracy

In [64]:
```
#get index of predicted dog breed for each image in test set
ResNet50_predictions = [np.argmax(ResNet_model.predict(np.expand_dims(feature, axis=0))) for feature in test_ResNet50]

#report test accuracy
test_accuracy = 100*np.sum(np.array(ResNet50_predictions)==np.argmax(test_targets, axis=1))/len(ResNet50_predictions)
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

3
TensorFlow: learning_rate = 0.001, beta1 = 0.9, beta2 = 0.999, epsilon = 1e-08.
Keras: learning_rate = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e-08, decay = 0.0

Batch Normalization –

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization and makes it notoriously hard to train models with saturating nonlinearities. 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. Using an ensemble of batch-normalized networks.

Activation Function –

We 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 neutron is negative as the output of ReLU for a negative value is zero. To overcome this problem, we generally use Leaky ReLU.

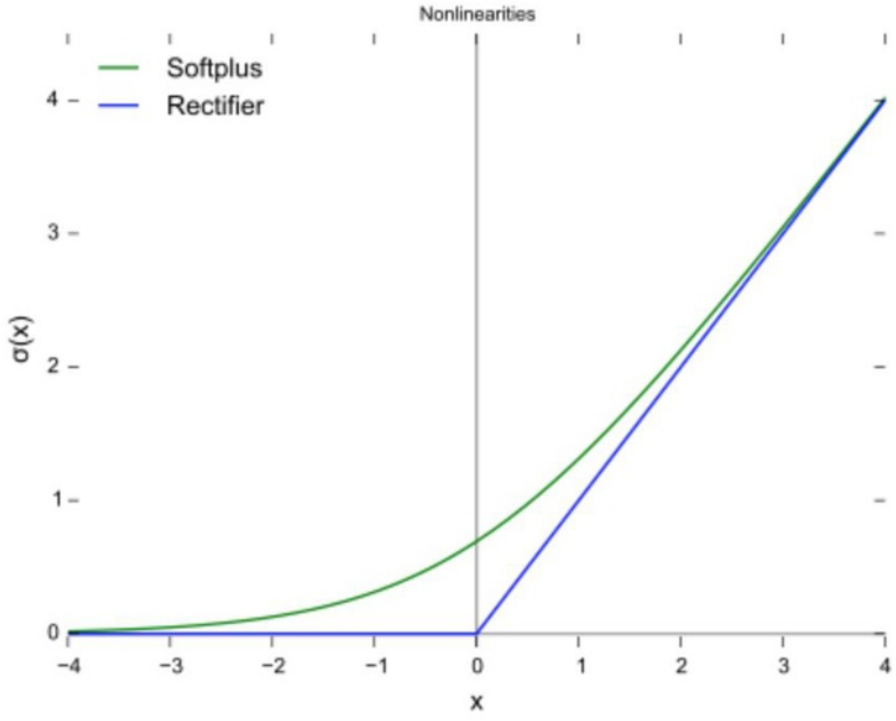


Fig 11. ReLU activation function

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

In [51]:
```
#To show the images
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

#This function takes a path as input image and return the breed of the dog.
def ResNet50_predict_breed(img_path):
    # extract bottleneck features
    bottleneck_feature = extract_Resnet50(path_to_tensor(img_path))

    # obtain predicted vector
    predicted_vector = ResNet_model.predict(bottleneck_feature)

    # return dog breed that is predicted by the model
    breed = dog_names[np.argmax(predicted_vector)]
    image = mpimg.imread(img_path)
    imgplot = plt.imshow(image)

    if dog_detector(img_path) == True:
        return print("The breed of dog is a {}".format(breed))
```

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

In [53]:
```
def predict_breed(img_path):
    isDog = dog_detector(img_path)
    if isDog:
        print("Detected a dog.\n")
        breed = ResNet50_predict_breed(img_path)
        return breed
    else:
        print('No dog picture found.')
```

3.2 Tools Used

- 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.
- **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.
- **TensorFlow** is a Python library that allows users to express arbitrary computation as a 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.
- 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

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

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:

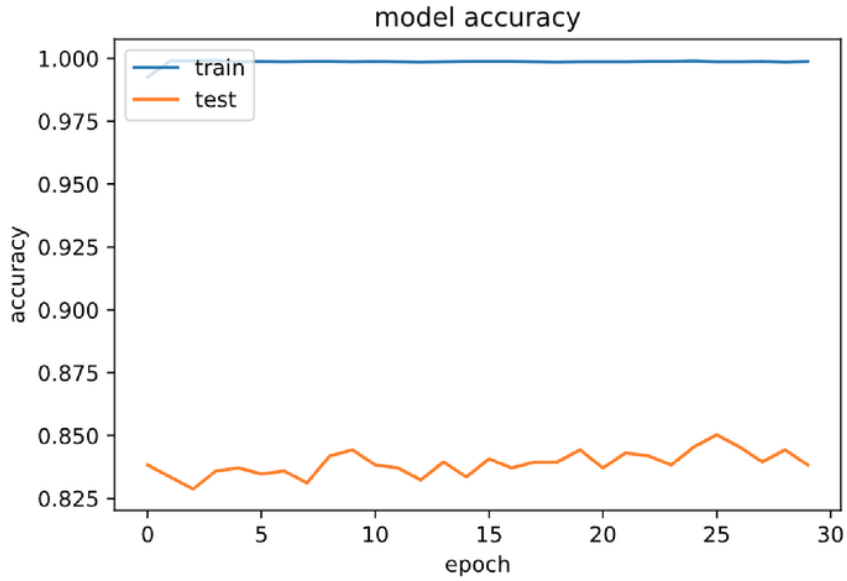


Fig 12. Model accuracy

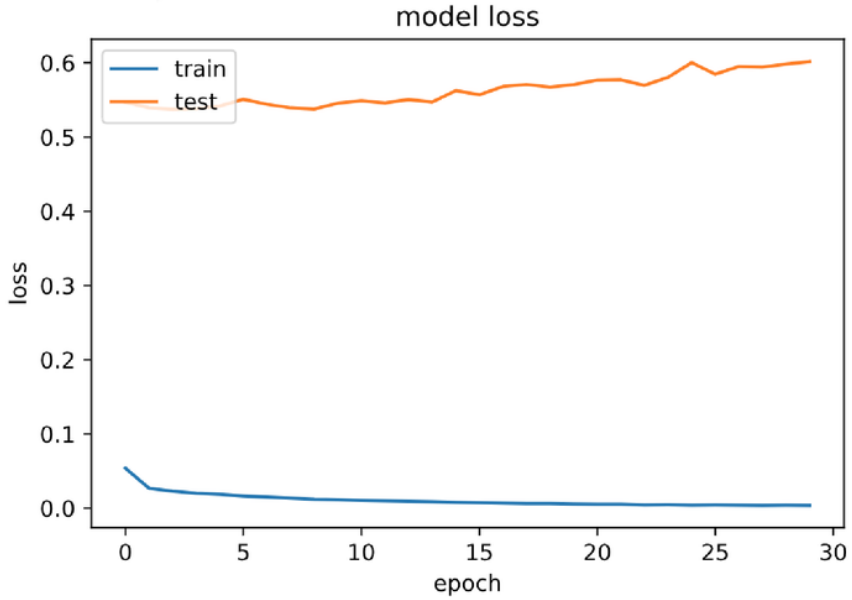


Fig 13. Model loss graph

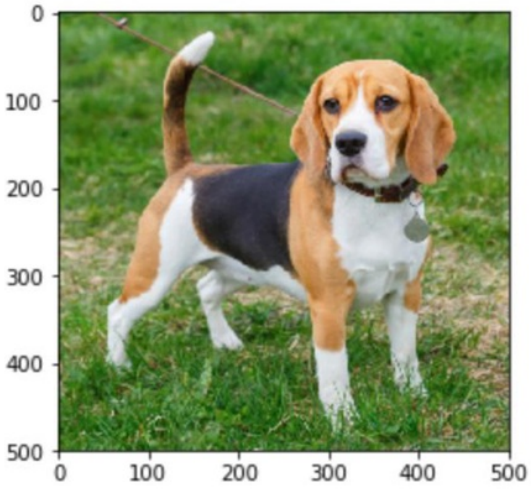
4.2 Significance of the result obtained

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
```

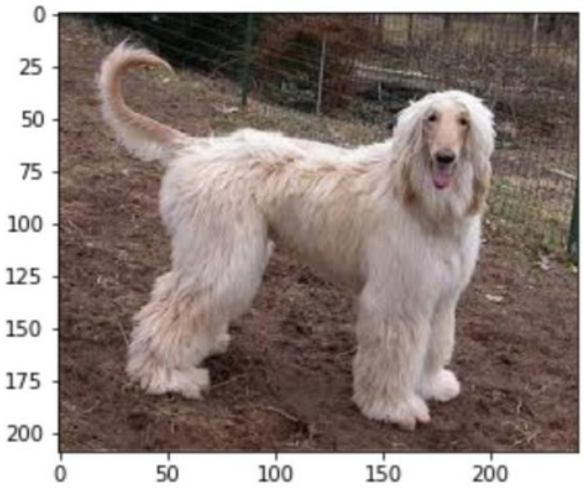
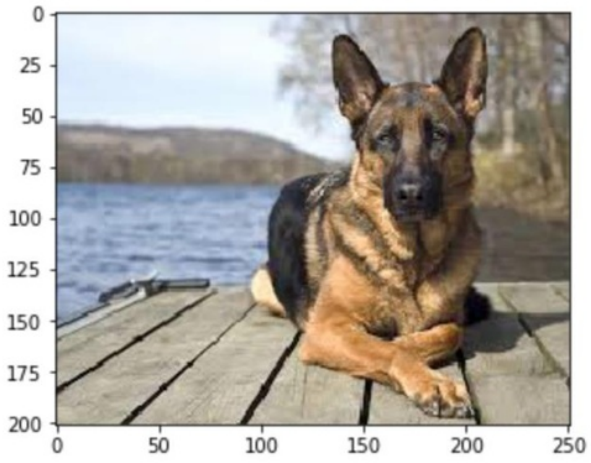


Fig 14. Test Images

```
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
```

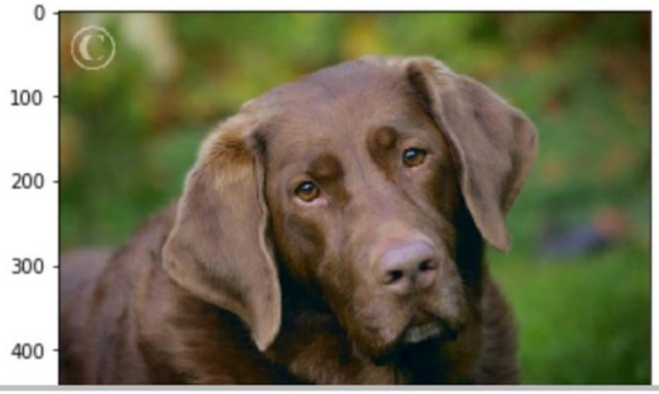


Fig 15. Test Images

```
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
```

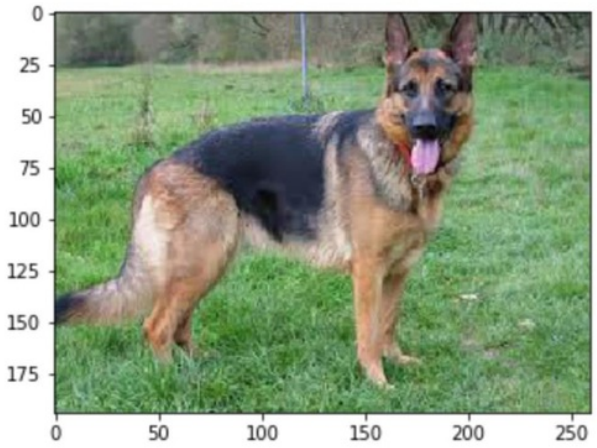


Fig 16. Test Images

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

46%	40%	11%	15%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	codeinpython.com Internet Source	13%
2	sandipanweb.wordpress.com Internet Source	12%
3	machinelearningmastery.com Internet Source	3%
4	videolectures.net Internet Source	3%
5	Submitted to National Research University Higher School of Economics Student Paper	3%
6	drrajivdesaimd.com Internet Source	1%
7	Neha Sharma, Vibhor Jain, Anju Mishra. "An Analysis Of Convolutional Neural Networks For Image Classification", Procedia Computer Science, 2018 Publication	1%
	www.safaribooksonline.com	

8	Internet Source	1%
9	docplayer.net Internet Source	1%
10	en.wikipedia.org Internet Source	1%
11	Submitted to Universiti Teknikal Malaysia Melaka Student Paper	1%
12	Submitted to University of Guelph Student Paper	1%
13	Submitted to International Islamic University Malaysia Student Paper	1%
14	recommend-papers.org Internet Source	1%
15	arun-aiml.blogspot.de Internet Source	1%
16	Submitted to Nanyang Technological University, Singapore Student Paper	<1%
17	Submitted to University of South Australia Student Paper	<1%
18	Submitted to Griffth University Student Paper	<1%

19	github.com Internet Source	<1%
20	Submitted to University of Sydney Student Paper	<1%
21	Submitted to BITS, Pilani-Dubai Student Paper	<1%
22	Submitted to University of Macau Student Paper	<1%
23	Submitted to Pace University Student Paper	<1%

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		