A Project Report

on

ANIMAL IMAGE CLASSIFIER USING DEEP LEARNING

Submitted in partial fulfillment of the

requirement for the award of the degree of

Bachelor of Technology in Computer Science

GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Dr. C Ramesh Kumar
Professor

Submitted By

Mayank Sharma - 18SCSE1010626
Amrendra pratap - 18SCSE1010576

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING / DEPARTMENT OF COMPUTERAPPLICATION
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
DECEMBER, 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled **"ANIMAL IMAGE CLASSIFIER USING DEEP LEARNING."** in partial fulfillment of the requirements for the award of the Bachelor of Technology submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of September, 2021 to December and 2021, under the supervision of Dr. C. Ramesh Kumar, Professor, Department of Computer Science and Engineering, of School of Computing Science and Engineering, Galgotias University, Greater Noida

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

Mayank Sharma,18SCSE1010626

Amrendra Pratap,18SCSE1010576

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Dr. C. Ramesh Kumar

Assistant Professor

CERTIFICATE

The Final Project Viva-Voce examination of Mayank Sharma 18SCSE1010626 has been held on _____ and his/her work is recommended for the award of Bachelor of Technology.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December, 2021

Place: Greater Noida

Abstract

Convolutional Neural Network (CNN) is an algorithm taking an image as input then assigning weights and biases to all the aspects of an image and thus differentiates one from the other. Neural networks can be trained by using batches of images, each of them having a label to identify the real nature of the image. A batch can contain few tenths to hundreds of images. For each and every image, the network prediction is compared with the corresponding existing label, and the distance between network prediction and the truth is evaluated for the whole batch. Then, the network parameters are modified to minimize the distance and thus the prediction capability of the network is increased. The training process continues for every batch similarly. The main goal of this project is to develop a system that can identify images of cats, dogs, bird, horses etc. The input image will be analyzed and then the output is predicted. The model that is implemented can be extended to a website or any mobile device as per the need. The animal dataset can be downloaded from the Kaggle website. The dataset contains a set of images of cats and dogs. Our main aim here is for the model to learn various distinctive features of cat and dog. Once the training of the model is done it will be able to differentiate images of cat and dog.

Key Words: Image Classification, Convolutional Neural Network, Kaggle, Deep Learning.

Table of Contents

List of Table

List of Figures

Acronyms

B.Tech.	Bachelor of Technology
CNN	Convolutional Neural Network
KNN	K- Nearest Neighbour
CIFAR-10	Canadian Institute Of Advanced Research
MNIST	Modified National Institute of Standards And Technology
GPU	Graphics Processing Unit
SCSE	School of Computing Science and Engineering

CHAPTER-1

Introduction

1.1 Introduction

The Image classification is one of the fundamental problems in computer vision. It forms basis for many other computer vision tasks such as object recognition, image segmentation and object detection. The task of categorizing images into one of several predefined classes is called image classification. Though the task of classifying images is easy for human beings, it is very difficult for an automated system. By using machine learning techniques, images can be classified. These machine learning algorithms falls under the category of deep learning. Deep learning is a type of neural network algorithms in which each layer is responsible for extracting one or more features of the image. A neural network is a computational model that is similar to a human brain. It is collection of nodes called as neurons. These nodes are organized into layers where each neuron in the one layer takes some input processes it and passes the output to the neuron in the next layer. Different layers may perform different kinds of transformations.Data transfers from the input layer (first layer) to the output layer (last layer) by traversing various hidden layers. One of the most popular techniques used for improving the accuracy of image classification is Convolutional Neural Networks (CNN). Neural Network Image classification can be done using both supervised classification algorithms and unsupervised classification algorithms. Supervised classification uses training data along with human intervention whereas in unsupervised classification human intervention is not required as it is fully computer operated. The supervised

classification has two phases namely training phase and classification phase. In training phase the classifier is given information about classes. This is the phase where learning of a model takes place. In classification phase it uses the information provided by the training data and classifies the image into one of the predefined classes. Various algorithms such as minimum distance algorithm, K-Nearest neighbour algorithm, Nearest Clustering algorithm, Fuzzy C - Means algorithm, Maximum likelihood algorithm and so on are used for the purpose of classification of images. Ever since Alex Krizhevsky, Geoff Hinton and Ilya Sutskevar won ImageNet in 2012, Convolutional Neural Networks (CNNs) have become the standard for image classification.

1.2 Problem Formulation

We are given a set of dog and cat images. The task is to build a model to predict the category of an animal.

1.2.1 Tools And Technology Used

- Python Interpreter
- Anaconda Prompt
- Spyder

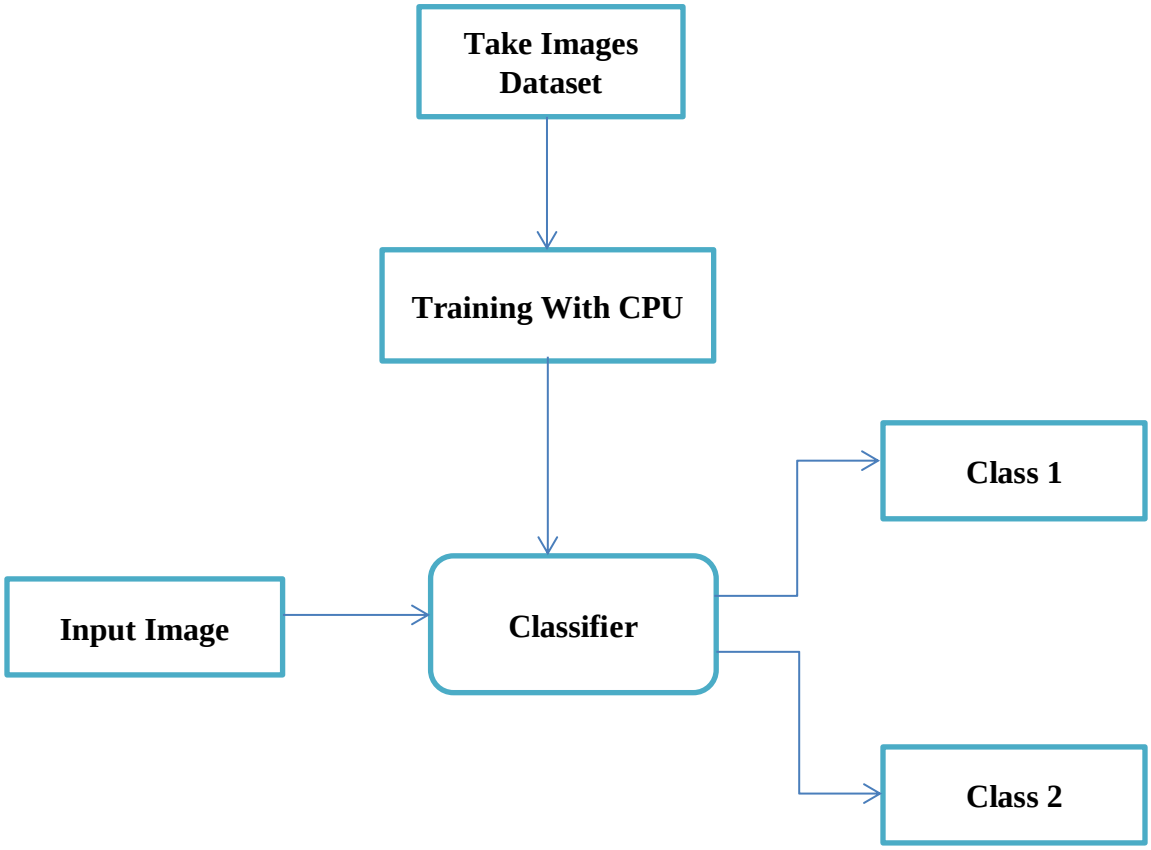
CHAPTER-2

Literature Survey

Tianmei Guo et. al.[1] explained that deep Belief Networks and Convolutional Neural Networks are commonly used models in deep learning. Among different type of models, convolutional neural networks has been demonstrated high performance on image classification. He built a simple neural network and the experiments are based on benchmarking datasets MINIST and CIFAR-10. On the basis of the convolutional neural network, different methods of learning rate set and different optimization algorithm of solving the optimal parameters of the influence on image classification are analyzed. Emine CENGIL et. al.[2] described that although several algorithms for image classification have been developed over the years, they have not been used with the discovery of Convolutional Neural Networks. Convolutional Neural Networks provide better results than existing methods in the literature due to advantages such as processing by extracting hidden features, allowing parallel processing thanks to parallel structure, and real time operation. He used The caffe library, which is often used for deep learning to train and test with images of cats and dogs taken from the kaggle dataset. 10,000 tagged data is used for training and 5,000 unlabeled data is used for testing. Owing to Convolutional Neural Networks allow parallel processing, GPU technology has been used. Travis Williams et. al.[3] explained that Convolutional Neural Networks (CNN) is a type of deep neural network that has a structure and approach that differs from other deep neural networks. Their strength and design is usage on

two-dimensional data, like images and videos. He developed a Convolutional Neural Networks (CNN) to classify handwritten digits. An algorithm is used to convert Data into wavelet domain to attain greater accuracy. Applying CNN on the raw pixels of images generates accurate results. However, the size and complexity of these images in the spatial domain causes the efficiency of the algorithm to decrease. By converting the images into the wavelet domain, they can be processed at a lower dimension, with faster processing times. Furthermore, given the varying frequencies represented in each subband, multiple CNNs performed on each subband, or a combination of them, can increase the accuracy of the classification. Sayali Jog et. al.[4] explained that remote sensing is the method used to detect and measure target characteristics using electromagnetic energy in the form of heat, light and radio waves. The process of producing thematic map from remotely sensed imagery is called image classification. Accuracy of classification depends on satellite image quality. Four steps are used for image classification, first is pre processing of image followed by selection of particular criteria feature to describe the pattern then selection of classifier and lastly accuracy assessment of the image classification. For classification, multispectral satellite images are used. Image classification can be supervised and unsupervised. There are various supervised classifiers namely minimum distance, support vector machine, maximum likelihood, and parallelepiped. The performance of these classifiers is judged on the basis of kappa coefficient and overall accuracy.

Project Design



Flow Chart of the Proposed System

Chapter – 3

Working of Project

We are building a simple convolutional neural network (CNN) of six layers that can identify and classify the images into one of the two predefined classes. In general, most of the machine learning applications requires GPU (Graphics processing unit) because of high number of computations on large amount of data. Since GPUs have almost 200 times more processors than CPU, it improves the performance of neural networks. As the number of layers in the network increases, the number of computations increases and hence the need for GPUs increases. Therefore, in order to build a neural network that can work on CPU as well a very small network is build.

There are various software libraries that are focused on machine learning. Few of them are Theano, Scikit-learn and Tensorflow. In proposed system, a convolutional neural network based image classifier is build using Tensorflow which is an open source software library focused on machine learning. [5] It is implemented using python language and ubuntu operating system. Python is used because of its simplicity and ease to learn. It provides various tools that are helpful in making machine learning applications. Apart from this, OpenCV (Open Source Computer Vision library) which is an open source C++ library for image processing and computer vision is used to read the images.

MODULE-1 (Input and Output Layers)

Convolutional neural network has input layers, output layers and hidden layers. The hidden layers are consists of convolutional layer, flattened layer and a fullyconnected layer.

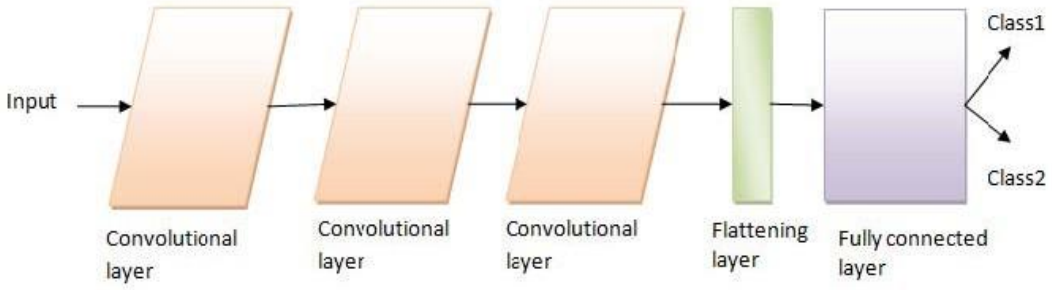
Input Layer: Input layer in CNN should contain image data. Image data is represented by three dimensional matrix as we saw earlier. You need to reshape it into a single column. Suppose you have image of dimension 28 x 28 =784, you need to convert it into 784 x 1 before feeding into input. If you have "m" training examples then dimension of input will be (784, m).

Output Layer: Output layer contains the label which is in the form of one-hot encoded.

MODULE-2 (Hidden Layers)

Convolutional Layer: Convolutional layer is the building block of convolutional neural network. The main task of convolutional layer is to extract features. It consists of one or more convolutional layers followed by one or more fully connected layers. This architecture is designed to support 2D structure of input such as images. This is done by using local connections and tied weights. The input to the convolutional layer is an x*x*r image where 'x' is height and width of image and 'r' is the number of channels. The convolutional layer will have filters of size y*y*r where 'y' is smaller than the size of image. A filter sized chunk from the image is selected and convolution (dot product) is calculated with the filter. This will result in a

single number as output to which a bias is added. Here convolution (dot product) is nothing but matrix multiplication of y*y*r sized chunk of image and y*y*r sized filter. Filter is slided over the whole input image to calculate the output across the image. The number of pixels through which the sliding takes place is called stride. All these outputs are concatenated to have an activation map or feature map. After each convolution, the size of the image decreases. Therefore, it is a standard practice to add zeros on the boundary of input layer such that the output is same as input layer. This is called as padding.



The Figure 1 shows the architecture of the proposed convolutional neural network.

Pooling layer: Pooling layer is used to reduce the dimensionality of the feature map in order to reduce the processing time. The sole purpose of pooling layer is to reduce the spatial size (height, width). This reduces the number of parameters; hence the number of computations is also reduced. There are three different types of pooling. They are max pooling, average pooling, min pooling. The most commonly used pooling is max pooling where we take a filter of size f*f and apply the maximum operation over the f*f sized part of image. Mostly pooling is done with a filter of size 2*2 with a stride of 2. This reduces the image size into half

Flattening Layer: The output of a convolutional layer is a multi-dimensional Tensor. A flattening layer is used in order to convert this into a single dimensional tensor. This is done using the reshape operation of tensorflow framework. It gets the output from the previous convolutional layers and flattens its structure to create a single feature vector which can be used by the fully connected layer to perform classification.

Fully connected layer: This layer performs the classification of the image based on the features extracted by the previous convolutional layers. In fully connected layer, every neuron is connected with every neuron of previous layer. A softmax function is used to convert the output of neural network into probability for each class.

After deciding the network architecture, focus should be given on parameters of the network. The best set of parameters can be found using back propagation technique. In this technique, random set of parameters are used at first. These values are changed such that for every training image we get correct output. Gradient descent is an optimizer method that is quick in finding the correct parameters. Cost is a single number that indicates the accuracy of the classifier increases as the cost decreases. Therefore, training is done till the cost remains constant. After training is done, the parameters and architecture are saved in a binary file called as model. A new image is sent as input to the same network and the probability of the new image is calculated. This is called as inference or prediction.

Instead of feeding the whole training data to the network, it is divided into batches of images called as epochs. Each epoch may contain 16 or 32 images and it takes more than 50 iterations to train the whole dataset.

Computer vision and neural networks are the hot new IT of machine learning techniques. With advances of neural networks and an ability to read images as pixel density numbers, numerous companies are relying on this technique for more data. For example, speed camera uses computer vision to take pictures of license plate of cars who are going above the speeding limit and match

the license plate number with their known database to send the ticket to. Although this is more related to Object Character Recognition than Image Classification, both uses computer vision and neural networks as a base to work.

A more realistic example of image classification would be Facebook tagging algorithm. When you upload an album with people in them and tag them in Facebook, the tag algorithm breaks down the person's picture pixel location and store it in the database. Because each picture has its own unique pixel location, it is relatively easy for the algorithm to realize who is who based on previous pictures located in the database. Of course the algorithm can make mistake from time to time, but the more you correct it, the better it will be at identifying your friends and automatically tag them for you when you upload. However, the Facebook tag algorithm is built with artificial intelligence in mind. This means that the tagging algorithm is capable of learning based on our input and make better classifications in the future.

We will not focus on the AI aspect, but rather on the simplest way to make an image classification algorithm. The only difference between our model and Facebook's will be that ours cannot learn from it's mistake unless we fix it. However, for a simple neural network project, it is sufficient.

Since it is unethical to use pictures of people, we will be using animals to create our model. My friend

The first step is to gather the data. This in my opinion, will be the most difficult and annoying aspect of the project. Remember that the data must be labeled. Thankfully, Kaggle has labeled images that we can easily download. The set we worked with can be found here: animal-10 dataset. If your dataset is not labeled, this can be be time consuming as you would have to manually create new labels for each categories of images. Another method is to create new labels and only move 100 pictures into their proper labels, and create a classifier like the one we will

and have that machine classify the images. This will lead to errors in classification, so you may want to check manually after each run, and this is where it becomes time consuming.

Now that we have our datasets stored safely in our computer or cloud, let's make sure we have a training data set, a validation data set, and a testing data set. Training data set would contain 85–90% of the total labeled data. This data would be used to train our machine about the different types of images we have. Validation data set would contain 5–10% of the total labeled data. This will test how well our machine performs against known labeled data. The testing data set would contain the rest of the data in an unlabeled format. This testing data will be used to test how well our machine can classify data it has never seen. The testing data can also just contain images from Google that you have downloaded, as long as it make sense to the topic you are classifying.

Let's import all the necessary libraries first:

```
import pandas as pd
import numpy as np
import itertools
import keras
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from keras.preprocessing.image import ImageDataGenerator,
img_to_array, load_img
from keras.models import Sequential
from keras import optimizers
from keras.preprocessing import image
from keras.layers import Dropout, Flatten, Dense
from keras import applications
from keras.utils.np_utils import to_categorical
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
import math
import datetime
import time
```

Defining Dimensions and locating images:

```
#Default dimensions we found online
img_width, img_height = 224, 224

#Create a bottleneck file
top_model_weights_path = 'bottleneck_fc_model.h5'# loading up our datasets
train_data_dir = 'data/train'
validation_data_dir = 'data/validation'
test_data_dir = 'data/test'

# number of epochs to train top model
epochs = 7 #this has been changed after multiple model run
# batch size used by flow_from_directory and predict_generator
batch_size = 50
```

In this step, we are defining the dimensions of the image. Depending on your image size, you can change it but we found best that 224, 224 works best. Then we created a bottleneck file system. This will be used to convert all image pixels in to their number (numpy array) correspondent and store it in our storage system. Once we run this, it will take from half hours to several hours

depending on the numbers of classifications and how many images per classifications. Then we simply tell our program where each images are located in our storage so the machine knows where is what. Finally, we define the epoch and batch sizes for our machine. For neural networks, this is a key step. We found that this set of pairing was optimal for our machine learning models but again, depending on the number of images that needs to be adjusted.

Importing transfer learning model VGG16:

```
#Loading vgc16 model
vgg16 = applications.VGG16(include_top=False,
weights='imagenet')datagen = ImageDataGenerator(rescale=1. / 255)
#needed to create the bottleneck .npy files
```

This is importing the transfer learning aspect of the convolutional neural network. Transfer learning is handy because it comes with pre-made neural networks and other necessary components that we would otherwise have to create. There are many transfer learning model. I particularly like VGG16 as it uses only 11 convolutional layers and pretty easy to work with. However, if you are working with larger image files, it is best to use more layers, so I recommend resnet50, which contains 50 convolutional layers.

For our image classifier, we only worked with 6 classifications so using transfer learning on those images did not take too long, but remember that the more images and classifications, the longer this next step will take. But thankfully since you only need to convert the image pixels to numbers only once, you only have to do the next step for each training, validation and testing only once- unless you have deleted or corrupted the bottleneck file.

Creation of the weights and feature using VGG16:

```
#__this can take an hour and half to run so only run it once.
#once the npy files have been created, no need to run again.
Convert this cell to a code cell to run.__start =
```

```
datetime.datetime.now()

generator = datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_train_samples = len(generator.filenames)
num_classes = len(generator.class_indices)

predict_size_train = int(math.ceil(nb_train_samples / batch_size))

bottleneck_features_train = vgg16.predict_generator(generator,
predict_size_train)

np.save('bottleneck_features_train.npy', bottleneck_features_train)
end= datetime.datetime.now()
elapsed= end-start
print ('Time: ', elapsed)
```

Since we are making a simple image classifier, there is no need to change the default settings. Just follow the above steps for the training, validation, and testing directory we created above. However, you can add different features such as image rotation, transformation, reflection and distortion.

Once the files have been converted and saved to the bottleneck file, we load them and prepare them for our convolutional neural network. This is also a good way to make sure all your data have been loaded into bottleneck file. Remember to repeat this step for validation and testing set as well.

Creating a bottleneck file for the training data. (Same step for validation and testing):

```
#training data
generator_top = datagen.flow_from_directory(
   train_data_dir,
   target_size=(img_width, img_height),
   batch_size=batch_size,
   class_mode='categorical',
   shuffle=False)

nb_train_samples = len(generator_top.filenames)
num_classes = len(generator_top.class_indices)

# load the bottleneck features saved earlier
train_data = np.load('bottleneck_features_train.npy')

# get the class labels for the training data, in the original order
train_labels = generator_top.classes

# convert the training labels to categorical vectors
train_labels = to_categorical(train_labels,
num_classes=num_classes)
```

Creating our Convolutional Neural Network code:

```
#This is the best model we found. For additional models, check out
I_notebook.ipynbstart = datetime.datetime.now()
model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(100, activation=keras.layers.LeakyReLU(alpha=0.3)))
model.add(Dropout(0.5))
model.add(Dense(50, activation=keras.layers.LeakyReLU(alpha=0.3)))
model.add(Dropout(0.3))
model.add(Dense(num_classes,
activation='softmax'))model.compile(loss='categorical_crossentropy'
,
   optimizer=optimizers.RMSprop(lr=1e-4),
   metrics=['acc'])history = model.fit(train_data, train_labels,
   epochs=7,
   batch_size=batch_size,
   validation_data=(validation_data,
validation_labels))model.save_weights(top_model_weights_path)(eval_
loss, eval_accuracy) = model.evaluate(
    validation_data, validation_labels, batch_size=batch_size,
verbose=1)print("[INFO] accuracy: {:.2f}%".format(eval_accuracy *
100))
print("[INFO] Loss: {}".format(eval_loss))
end= datetime.datetime.now()
```

```
elapsed=                                                    end-start

print ('Time: ', elapsed)
```

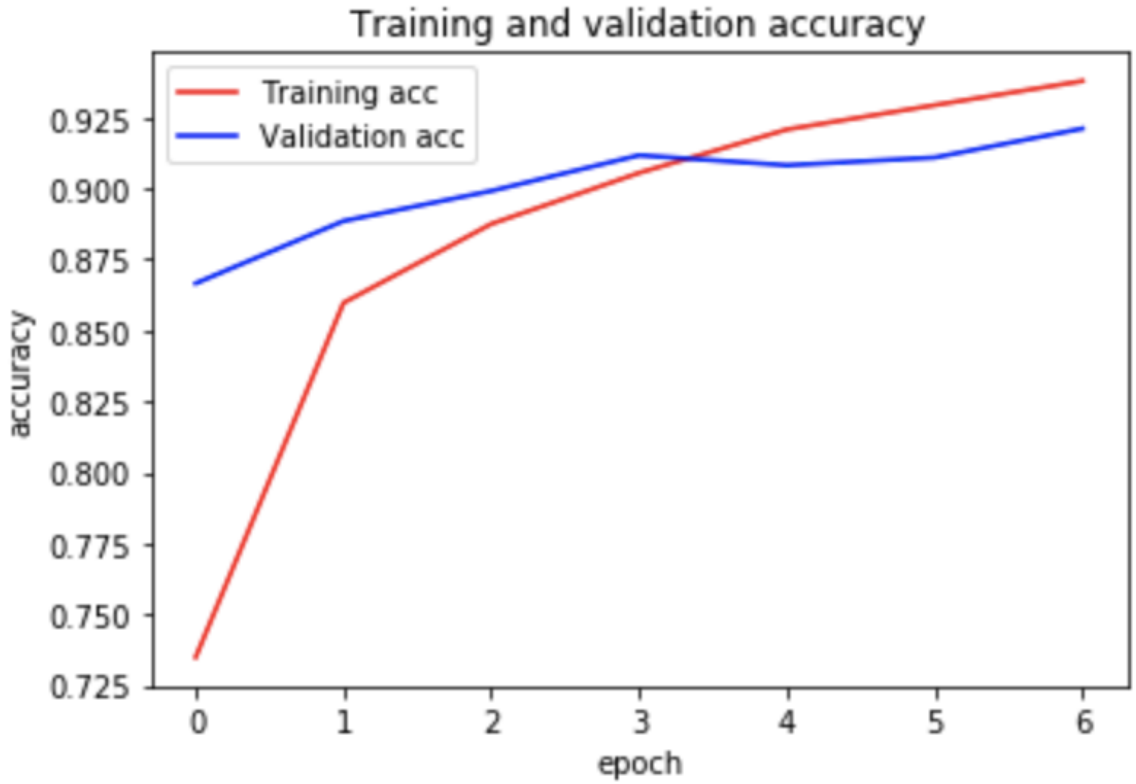
Now we create our model. First step is to initialize the model with Sequential(). After that we flatten our data and add our additional 3 (or more) hidden layers. This step is fully customizable to what you want. We made several different models with different drop out, hidden layers and activation. But since this is a labeled categorical classification, the final activation must always be softmax. It is also best for loss to be categorical crossenthropy but everything else in model.compile can be changed. Then after we have created and compiled our model, we fit our training and validation data to it with the specifications we mentioned earlier. Finally, we create an evaluation step, to check for the accuracy of our model training set versus validation set.

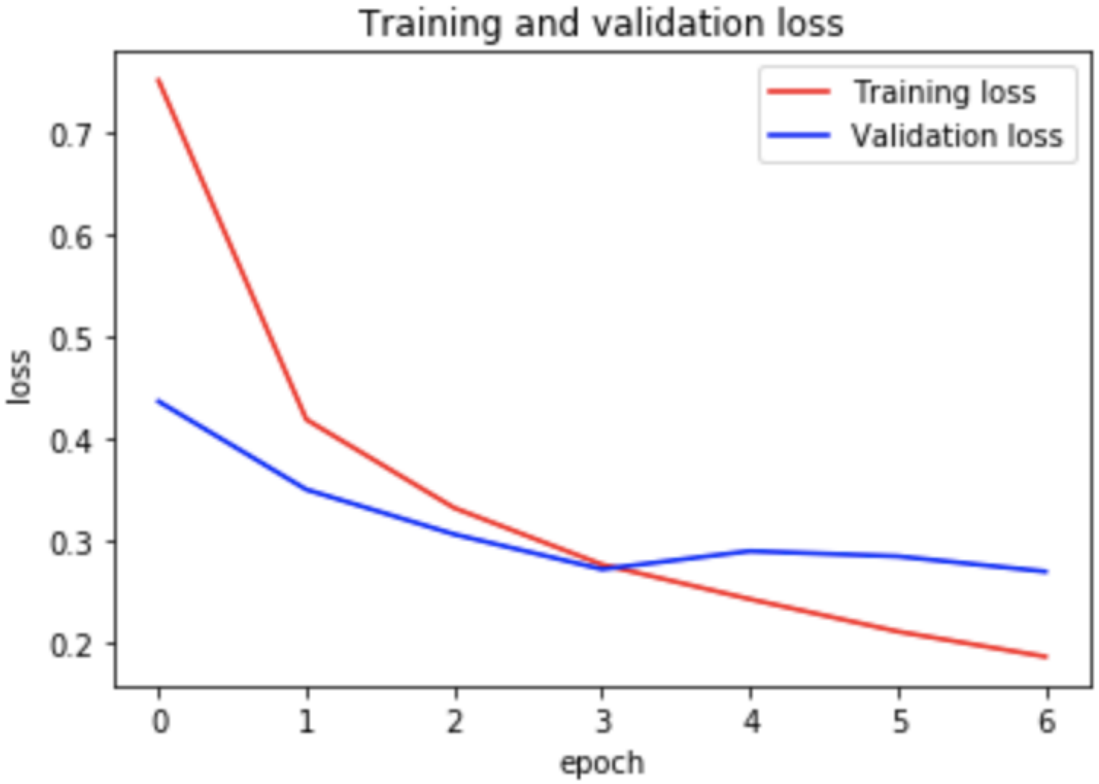
```
Train on 13412 samples, validate on 2549 samples
Epoch 1/7
13412/13412 [==============================] - 9s 638us/step - loss: 0.7518 - acc: 0.7349 - val_loss: 0.4372 - val_ac
c: 0.8666
Epoch 2/7
13412/13412 [==============================] - 6s 476us/step - loss: 0.4194 - acc: 0.8598 - val_loss: 0.3507 - val_ac
c: 0.8886
Epoch 3/7
13412/13412 [==============================] - 7s 523us/step - loss: 0.3327 - acc: 0.8876 - val_loss: 0.3068 - val_ac
c: 0.8992
Epoch 4/7
13412/13412 [==============================] - 6s 471us/step - loss: 0.2771 - acc: 0.9055 - val_loss: 0.2726 - val_ac
c: 0.9117
Epoch 5/7
13412/13412 [==============================] - 6s 444us/step - loss: 0.2433 - acc: 0.9208 - val_loss: 0.2904 - val_ac
c: 0.9082
Epoch 6/7
13412/13412 [==============================] - 6s 455us/step - loss: 0.2115 - acc: 0.9294 - val_loss: 0.2852 - val_ac
c: 0.9109
Epoch 7/7
13412/13412 [==============================] - 6s 477us/step - loss: 0.1869 - acc: 0.9379 - val_loss: 0.2703 - val_ac
c: 0.9211
2549/2549 [==============================] - 0s 96us/step
[INFO] accuracy: 92.11%
[INFO] Loss: 0.2703172541517587
Time:  0:00:47.456195
```

This is our model now training the data and then validating it. An epoch is how many times the model trains on our whole data set. Batch can be explained as taking in small amounts, train and take some more. Each epoch must finish all batch before moving to the next epoch. Training with too little epoch can lead to underfitting the data and too many will lead to overfitting the data. You also want a loss that is as low as possible. The pictures below will show the accuracy and loss of our data set

Code for visualization of the Accuracy and Loss:

```
#Graphing our training and validation
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
accuracy
epoch
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0
1
2
3
4
5
6


Training and validation loss
Training loss
Validation loss
loss
epoch
0.7
0.6
0.5
0.4
0.3
0.2
0
1
2
3
4
5
6

Even though according to this graph, it showed that epoch 3 was the best as that was the point of intersection between accuracy and loss, when we ran the model on 3 epoch, it underperformed. So this graph is not an absolute indicator of how many epoch to run on your model.

This picture below shows how well the machine we just made can predict against unseen data. Notice it says that its testing on test_data. Accuracy is the second number. However, this is not the only method of checking how well our machines performed

```
model.evaluate(test_data, test_labels)
```

```
1845/1845 [==============================] - 0s 176us/step

[0.22370996741744562, 0.926829268324989]
```

There are two great methods to see how well your machine can predict or classify. One of them is the classification metrics and the other is the confusion matrix.

```
preds = np.round(model.predict(test_data),0)
#to fit them into classification metrics and confusion metrics, some additional modificaitions are required
print('rounded test_labels', preds)
```

...

```
animals = ['butterflies', 'chickens', 'elephants', 'horses', 'spiders', 'squirells']
classification_metrics = metrics.classification_report(test_labels, preds, target_names=animals )
print(classification_metrics)
```

```
              precision    recall  f1-score   support

 butterflies       0.98      0.88      0.93       371
    chickens       0.94      0.85      0.89       203
   elephants       0.91      0.89      0.90       152
      horses       0.97      0.95      0.96       472
     spiders       0.92      0.95      0.93       403
   squirells       0.92      0.91      0.92       244

   micro avg       0.94      0.92      0.93      1845
   macro avg       0.94      0.91      0.92      1845
weighted avg       0.94      0.92      0.93      1845
 samples avg       0.92      0.92      0.92      1845
```

To use classification metrics, we had to convert our testing data into a different numpy format, numpy array, to read. That is all the first line of code is doing. The second cell block takes in the converted code and run it through the built in classification metrics to give us a neat result. Please note that unless you manually label your classes here, you will get 0–5 as the classes instead of the animals. The important factors here are precision and f1-score. The higher the

score the better your model is. Here is a great blog on medium that explains what each of those are.

Now to make a confusion matrix. There are lots on online tutorial on how to make great confusion matrix. Ours is a variation of some we found online

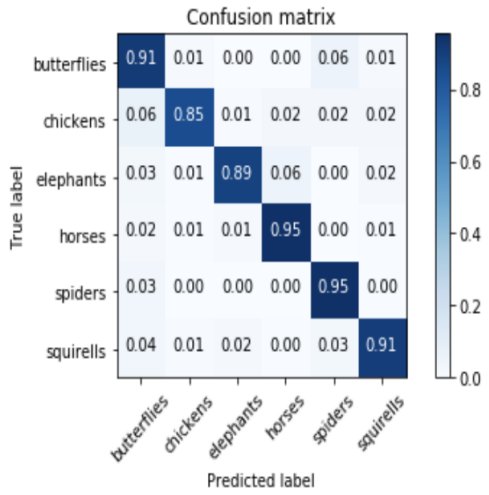
```
#Since our data is in dummy format we put the numpy array into a dataframe and call idxmax axis=1 to return the column
# label of the maximum value thus creating a categorical variable
#Basically, flipping a dummy variable back to it's categorical variablecategorical_test_labels = pd.DataFrame(test_labels).idxmax(axis=1)
categorical_preds = pd.DataFrame(preds).idxmax(axis=1)confusion_matrix= confusion_matrix(categorical_test_labels, categorical_preds)
#To get better visual of the confusion matrix:def plot_confusion_matrix(cm, classes,
   normalize=False,
   title='Confusion matrix',
   cmap=plt.cm.Blues):

#Add Normalization Option
 '''prints pretty confusion metric with normalization option '''
   if normalize:
     cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
     print("Normalized confusion matrix")
   else:
```

```
    print('Confusion matrix, without normalization')

#                                                   print(cm)

   plt.imshow(cm, interpolation='nearest', cmap=cmap)
   plt.title(title)
   plt.colorbar()
   tick_marks = np.arange(len(classes))
   plt.xticks(tick_marks, classes, rotation=45)
   plt.yticks(tick_marks, classes)

   fmt = '.2f' if normalize else 'd'
   thresh = cm.max() / 2.
   for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
      plt.text(j, i, format(cm[i, j], fmt),
horizontalalignment="center", color="white" if cm[i, j] > thresh
else "black")

   plt.tight_layout()
   plt.ylabel('True label')
   plt.xlabel('Predicted label')
```

The numpy array we created before is placed inside a dataframe. Confusion matrix works best on dataframes. The 3rd cell block with multiple iterative codes is purely for color visuals. The only important code functionality there would be the 'if normalize' line as it standardizes the data.

```
plot_confusion_matrix(confusion_matrix,
                      ['butterflies', 'chickens', 'elephants', 'horses', 'spiders', 'squirells'],
                      normalize=True)
```

```
Normalized confusion matrix
```



As we can see in our standardized data, our machine is pretty good at classifying which animal is what. Chickens were misclassified as butterflies most likely due to the many different types of pattern on butterflies. In addition, butterflies was also misclassified as spiders because of probably the same reason. Both elephants and horses are rather big animals, so their pixel distribution may have been similar.

The final phase is testing on images. The cell blocks below will accomplish that:

```
def read_image(file_path):
   print("[INFO] loading and preprocessing image…")
   image = load_img(file_path, target_size=(224, 224))
   image = img_to_array(image)
   image = np.expand_dims(image, axis=0)
   image /= 255.
   return imagedef test_single_image(path):
  animals = ['butterflies', 'chickens', 'elephants', 'horses',
'spiders', 'squirells']
```

```
  images                    =                    read_image(path)
  time.sleep(.5)
  bt_prediction             =             vgg16.predict(images)
  preds           =           model.predict_proba(bt_prediction)
  for  idx,  animal,  x  in  zip(range(0,6),  animals  ,  preds[0]):
   print("ID: {}, Label: {} {}%".format(idx, animal, round(x*100,2)
))
  print('Final                                          Decision:')
  time.sleep(.5)
  for               x               in               range(3):
   print('.'*(x+1))
   time.sleep(.2)
  class_predicted      =      model.predict_classes(bt_prediction)
  class_dictionary          =          generator_top.class_indices
  inv_map   =   {v:   k   for   k,   v   in   class_dictionary.items()}
  print("ID:     {},     Label:     {}".format(class_predicted[0],
inv_map[class_predicted[0]]))
  return                   load_img(path)path                    =
'data/test/yourpicturename'test_single_image(path)
```

The first def function is letting our machine know that it has to load the image, change the size and convert it to an array. *Second def function* is using transfer learning's prediction model and an iterative function to help predict the image properly. The *path* is where we define the image location and finally the *test_single_image* cell block will print out the final result, depending on the prediction from the second cell block.

Chapter-4

Results And Discussions

As a result we build a deep convolutional neural network for image classification. Despite of using only a subset of the images an accuracy of 90.10% was obtained. If the whole dataset was being used the accuracy would have been even better.

This project shows a classification process using convolutional neural networks which is a deep learning architecture. Although there are many algorithms that perform image classification, convolutional neural network is considered to be a standard image classification technique. Convolutional neural network uses GPU technology because of large number of layers which increases the number of computers. Therefore, in this project, we presented a very small convolutional neural network which can work on CPU as well. This network classifies the images into one the two predefined classes say Cat and Dog. This same network can be used for other datasets as well.

Convolutional neural network for image classification is implemented using python language in ubuntu operating system. Tensorflow which is an open source library focused on machine learning applications is used. OpenCV which is an open source library for computer vision is used to read the images from the dataset. Python is a high level, interpreted, object oriented scripting language that is easy to learn, read and maintain. It supports both functional and object oriented programming. It can be easily integrated with other programming languages. One of the reasons why python is mostly used for machine learning applications is its syntax. Since the syntax of python is mathlike, it is easy for programmers to express their ideas mathematically. It has particular tools that are very helpful in developing machine learning applications.

Frameworks, libraries and extensions like NumPy make python to accomplish the tasks easily. Languages like Java, Ruby need hard coding because of their complexity whereas python is considered as a toy language because of its simple syntax and many features. Since python can do a lot things easily, which helps for complex set of machine learning tasks it is considered as a best language for implementing machine learning tasks.

Tensorflow is an open source software library developed by google brain team for numerical computations using data flow graphs. It is a symbolic math library used for machine learning applications such as deep neural networks. The nodes of the graph represent operations whereas the edges of the graph represent multidimensional data arrays. Tensorflow provides flexibility to deploy the computation on more than one GPU or CPU.

The subset of Kaggle dog-cat dataset is used as training data. 2,000 images are used as training data and 400 images are used as testing data. The output of training phase is a classifier which can classify the given input. image as either cat or dog. If the given input image is dog then the output shows probability of dog as one or greater than probability of cat. Similarly, if input image is cat then the output shows that probability of cat as one or greater than probability of dog. Figure 4 shows the training of the dataset. Figure 5 shows the input image which is a cat and figure 6 shows the output of the classifier as “Given image is a cat”. Similarly Figure 7 shows the input image which is a dog and Figure 8 shows the output of the classifier as “Given image is a dog”.

```
Training Epoch 1 --- Training Accuracy:  50.0%, Validation Accuracy:  43.8%,  Validation Loss: 0.702
Training Epoch 2 --- Training Accuracy:  56.2%, Validation Accuracy:  53.1%,  Validation Loss: 0.687
Training Epoch 3 --- Training Accuracy:  56.2%, Validation Accuracy:  37.5%,  Validation Loss: 0.700
Training Epoch 4 --- Training Accuracy:  62.5%, Validation Accuracy:  56.2%,  Validation Loss: 0.686
Training Epoch 5 --- Training Accuracy:  62.5%, Validation Accuracy:  40.6%,  Validation Loss: 0.709
Training Epoch 6 --- Training Accuracy:  71.9%, Validation Accuracy:  43.8%,  Validation Loss: 0.718
Training Epoch 7 --- Training Accuracy:  84.4%, Validation Accuracy:  65.6%,  Validation Loss: 0.650
Training Epoch 8 --- Training Accuracy:  81.2%, Validation Accuracy:  68.8%,  Validation Loss: 0.638
Training Epoch 9 --- Training Accuracy:  81.2%, Validation Accuracy:  78.1%,  Validation Loss: 0.610
Training Epoch 10 --- Training Accuracy:  81.2%, Validation Accuracy:  62.5%,  Validation Loss: 0.634
Training Epoch 11 --- Training Accuracy:  81.2%, Validation Accuracy:  56.2%,  Validation Loss: 0.688
Training Epoch 12 --- Training Accuracy:  81.2%, Validation Accuracy:  40.6%,  Validation Loss: 0.880
Training Epoch 13 --- Training Accuracy:  84.4%, Validation Accuracy:  75.0%,  Validation Loss: 0.560
Training Epoch 14 --- Training Accuracy:  84.4%, Validation Accuracy:  59.4%,  Validation Loss: 0.596
Training Epoch 15 --- Training Accuracy:  87.5%, Validation Accuracy:  78.1%,  Validation Loss: 0.483
```

Figure 4: Training of the convolutional neural network

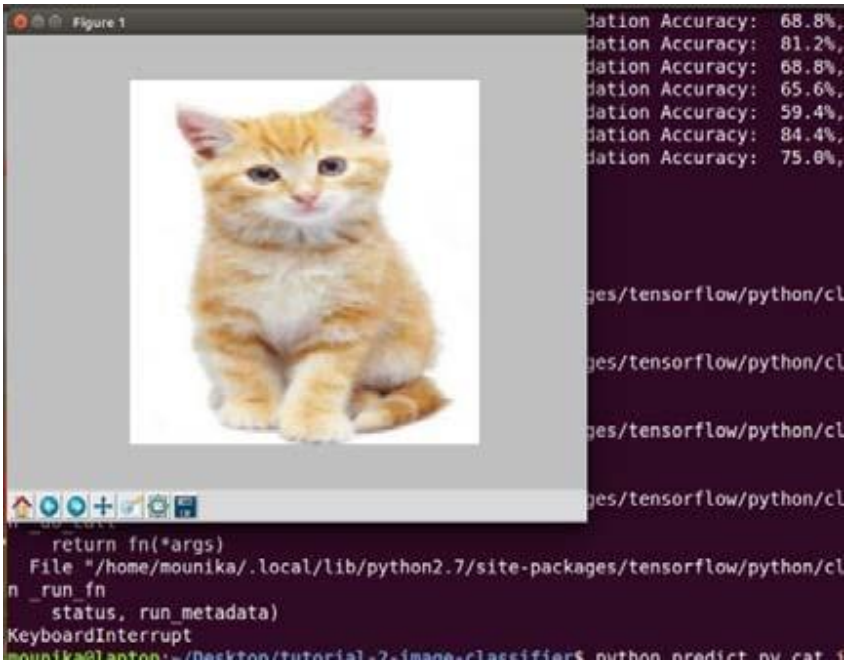


Figure 5: Input image of Cat

```
mounika@laptop:~/Desktop/tutorial-2-image-classifier$ python predict.py cat.jpg
2018-03-08 12:07:16.872844: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instruct
ions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
[[0.06877851 0.9312215 ]]
Given image is a cat
mounika@laptop:~/Desktop/tutorial-2-image-classifier$
```

Figure 6: Output of classifier when input is cat

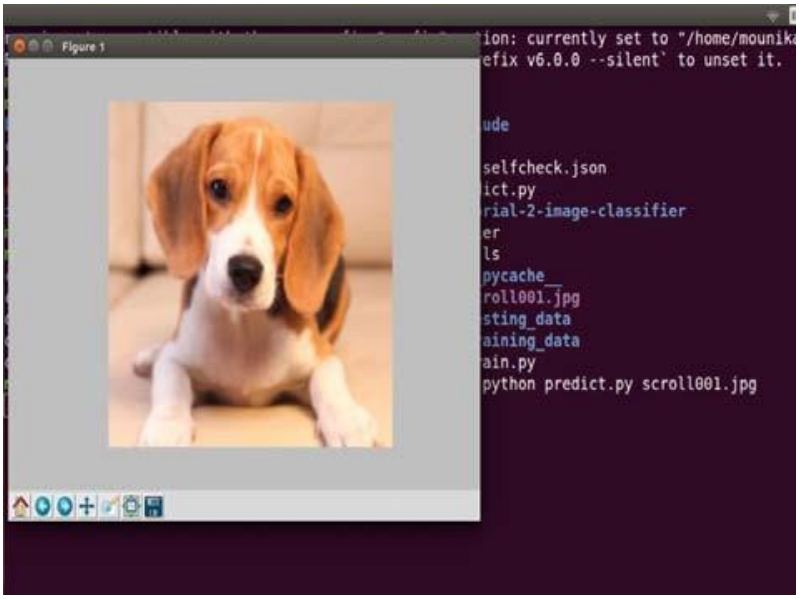


Figure 7: Input image of Dog

Figure 8: Output of the classifier when the image is dog

Chapter-5

Conclusion And Future Scope

5.1 Conclusion

In this report, I discussed how to train a CNN model to classify Dogs vs. Cats images. I trained the models from scratch at first and then using Transfer Learning, I got over 95% accuracy. Meanwhile, two ways of the model outputs visualization have been demonstrated, which can help us gain more insight into how the model works. One of the original tasks for this project is applying the CNN model to detect whether animals or pets are sick or not? In real life, it may be difficult to know if subtle changes in the animals indicate a health problem. Using the CNN model, I may be able to keep track of animals' behavior in order to prevent extreme illnesses. The discharge from eyes or nose may indicate a possible upper respiratory infection; or skin irritation or hair loss may be a sign of allergies, external parasites, or another skin condition. Because of the time limitations, I did not collect enough dataset for this kind of study. For the future works, I will keep collecting datasets for a month or a year in order to find the correlation between illnesses and animal behavior.

5.2 Future Scope

Through the deep learning model, I will be able to predict the probability of a symptom for every animal picture or video. Future work might also include robotic systems that monitor the state of the household animals, adjust food distribution depending on image readings or alert when the animals from any kind of illness. Moreover, different types of models can be employed to see, which one fits the needs the most. The project shouldn't be limited to VGG models.

Refrences

[1] Tianmei Guo, Jiwen Dong ,Henjian Li'Yunxing Gao, “Simple Convolutional Neural Network on Image Classification” , IEEE 2nd International Conference on Big Data Analytics at Beijing, China on 10-12 March 2017.

[2] Emine CENGIL , Ahmet ÇINAR , Zafer GÜLER, “A GPU-Based Convolutional Neural Network Approach for Image Classification”, International Conference on Artificial Intelligence and Data Processing Symposium at Malatya, Turkey on 16- 17 September 2017.

[3] Travis Williams, Robert Li, “Advanced Image Classification using Wavelets and Convolutional Neural Networks”, 15th International Conference on Machine Learning and Applications at Anaheim, CA, USA on 18-20 December 2016.

[4] Sayali Jog , Mrudul Dixit, “Supervised Classification of Satellite Images”, Conference on Advances on signal processing at Pune, India on 9-11 June 2016.

[5] Mart´in Abadi et.al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[6] Tianmei Guo, Jiwen Dong ,Henjian Li'Yunxing Gao, “Simple Convolutional Neural Network on Image Classification” , IEEE 2nd International Conference on Big Data Analytics at Beijing, China on 10-12 March 2017.

[7] Emine CENGIL , Ahmet ÇINAR , Zafer GÜLER, “A GPU-Based Convolutional Neural Network Approach for Image Classification”, International Conference on Artificial Intelligence and Data Processing Symposium at Malatya, Turkey on 16- 17 September

2017.

[8] Travis Williams, Robert Li, “Advanced Image Classification using Wavelets and Convolutional Neural Networks”, 15th International Conference on Machine Learning and Applications at Anaheim, CA, USA on 18-20 December 2016.

[9] Sayali Jog , Mrudul Dixit, “Supervised Classification of Satellite Images”, Conference on Advances on signal processing at Pune, India on 9-11 June 2016.

[10] Mart´in Abadi et.al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.