University Of Maryland

College Park

Project Report

673-Perception for Autonomous Robots

Project 6

Aditya Goswami
116951968

Mahmoud Dahmani
116777896

Sukoon Sarin
116955522

May 16, 2020

Contents

List of Figures

1 Introduction

In this project our aim is to classify whether images, in the given dataset, contain either a dog or a cat. This is easy for humans, dogs, and cats. But for our computer it is difficult.

Image classification refers to a process in computer vision that can classify an image according to its visual content. For example, an image classification algorithm may be designed to tell if an image contains a human figure or not. While detecting an object is trivial for humans, robust image classification is still a challenge in computer vision applications.This has proven to be a difficult task, but major breakthroughs proved that it is attainable.

In this project we had to implement implement a Convolution Neural Network(CNN) to perform classification on the given dataset. For such classification task, generally VGG-16 architecture is deployed. Your implementation the same using tensorflow, pytorch or keras.

One relaxation of that objective is to be able to distinguish images between a smaller set of classes of artifacts conveyed in them. In that sense, Kaggle launched a competition in September, 2013 that challenged the community to perform the task of distinguishing computationally pictures with dogs from the ones with cats.

The document follows by further detailing the problem, then a revision over the classifiers used for the task, as well as the parameters used, and after that an evaluation of the results obtained and conclusions.

2 Data Set

The given training archive contains 25,000 images of dogs and cats. We were to train our network on these images to predict the labels for test1.zip (1 = dog, 0 = cat). The testing dataset contains 12500 images. We didn't necessarily need to train on all 25000 images.

The system inputs a data set of 25000 pictures, with variable dimensions, from which half correspond to pictures with dogs, and another half to pictures with cats.The dataset was provided from Kaggle and has already been segregated into train and test folder. These folders consists of the images of various cats and dogs.

So we divided our given training dataset according to our convenience. We created two folders named 'cats' and 'dogs' in 'train' folder. Divided the 25,000 labeled images accordingly into these folders. Now we have 12,500 images in each folder.

We again created another folder 'Val' (validation) in which contains two folders - 'cats' and 'dogs'. 25 percent of the each class of images were moved here from 'train' folder. Now 'train' folder has 9300 images each of cats and dogs in their respective folders. Similarly 'Val' folder has 3200 images each of cats and dogs in their respective folders.

Both dogs and cats distribute over a large set of races, which diverge a lot, not only in terms of shape, but also in color. The pictures also diverge in terms of scale, rotation and transposition, and include other artifacts (e.g. people) that further increases the complexity of the problem.

These photos are color images and are of different sizes and shapes. Therefore in order for modeling, it is necessary for these images to be reshaped.

The technique used to reshape these images are done by using the resize function to make all images of 224*224 square size.

The coloured image is transformed to tensors. The dataset contains the image of cats and dogs, the name of the image i.e. cat or dog is extracted.

The input images are normalized. The normalization is done in the following way. The image consists of three images (the three colour channels). Every image is subtracted by their mean and then it is divided by its standard deviation.

3 Architecture

The architecture that is being used for training is smaller VGG-16.

VGG-16 is a convolutional neural network architecture, it's name VGG-16 comes from the fact that it has 16 layers. It's layers consists of Convolutional layers, Max Pooling layers, Activation layers, Fully connected layers.

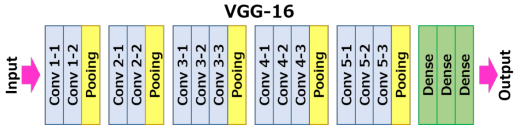


Figure 1: VGG-16 Layer Definition

Following are the characterstics of VGG-16 architecture:

- There are 13 convolutional layers, 5 Max Pooling layers and 3 Dense layers which sums up to 21 layers but only 16 weight layers.

- Conv 1 has number of filters as 64 while Conv 2 has 128 filters, Conv 3 has 256 filters while Conv 4 and Conv 5 has 512 filters.
- VGG-16 network is trained on ImageNet dataset which has over 14 million images and 1000 classes, and acheives 92.7 percent top-5 accuracy. It surpasses AlexNet network by replacing large filters of size 11 and 5 in the first and second convolution layers with small size 3x3 filters.

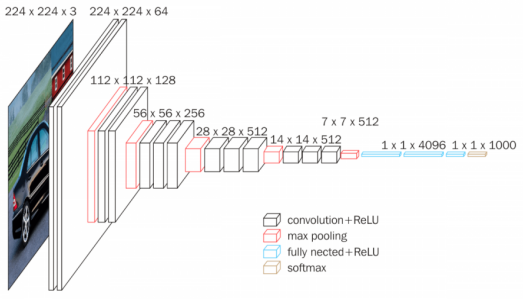


Figure 2: VGG-16 Architecture

- VGG-16 network has 138 million parameters over which the learning model is trained (selectively).

VGG16 - Structural Details													
#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total													**138,423,208**

Figure 3: VGG-16 Architecture

4 Training

The preprocessed dataset is split into training and test. Scikit learn module is used to split the preprocessed dataset into training and testing dataset. Data image generator is used to create the augmented data. Then the augmented data is sent to the architecture.

In order to obtain the best classifier for the task, all the classification algorithms were fed into a function that performed the following steps:

- Split the data in 75 percent for training and test (Val) sets and 25 percent for test.
- Here, accuracy is used to evaluate the matrices, where,

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

- The classifier that obtained the highest score is then selected.

The best model which has highest accuracy on validation set is chosen and saved.

In the training, there are mainly three major points that play key role:

1. Optimizer: Stochastic Gradient Descent with momentum with learning rate of 0.001 and momentum of 0.9.

   (a) This also the algorithm which is known as SGD with Momentum.
   (b) Here the processing of the dataset is done in mini batches.
   (c) We took the batch size of 20 because of the challenges we faced due to lack of resources (Due to this error specifically- CUDA Out of Memory)

2. Learning Rate Decay Policy: Here the learning rate and momentum adds up to the hyper-parameters.

   (a) Learning rate decay policy is required in order to achieve the goal that our model has to learn.
   (b) The policy is such that as our model starts learning and move towards the global goal, the rate has to be decreased (think it of reducing big leaps to baby steps) so that the model does not miss or 'jump over' the required goal. Meaning the error function should not saturate. If error function becomes zero than it means the saturation of error function which indicates that there is no learning happening.
   (c) As the epochs increase the learning rate decreases.
   (d) In our project, after every seven epochs the learning rate decreases by a factor of 0.1.

3. Choice of loss function: The Loss Function is one of the important components of Neural Networks. Loss is nothing but a prediction error of Neural Net. And the method to calculate the loss is called Loss Function. Here the choice of loss function depends on the last activation layer.

   (a) We have used BCE loss function here.

   (b) Binary Cross Entropy is used for the binary classification tasks.

   (c) As mentioned above the choice of loss function is associated with the choice of activation last layer which is the sigmoid activation function and the range of output is (0 – 1).

   (d) This gives probabilites.

   (e) We recognised a class with that much probabilities.

   (f) *logsoftmax* gives the probability distribution.

   Improving Accuracy of our neural network

   Now that we have a baseline neural network working, we tried to improve the accuracy and combat over-fitting by doing simple tricks.

   (a) Standardize our data input.

   (b) Decay our learning rate as you train.

   (c) Augment our data to artificially make your dataset larger.

   (d) Add Batch Normalization between layers.

   (e) Change the hyper parameters in our architecture.

5 Testing

The testing images are sorted and are preprocessed according to the same procedure used for the training dataset.

The trained model is imported and the testing images are sent to the trained model, the prediction is done whether the image is of cat or dog.

The prediction is being written in the .csv file and the classification is done.

6 Results

6.1 Experiment 1

For the sanity check, we ran our program to see the results of our model and check if we were on the right track!

Following are the results:

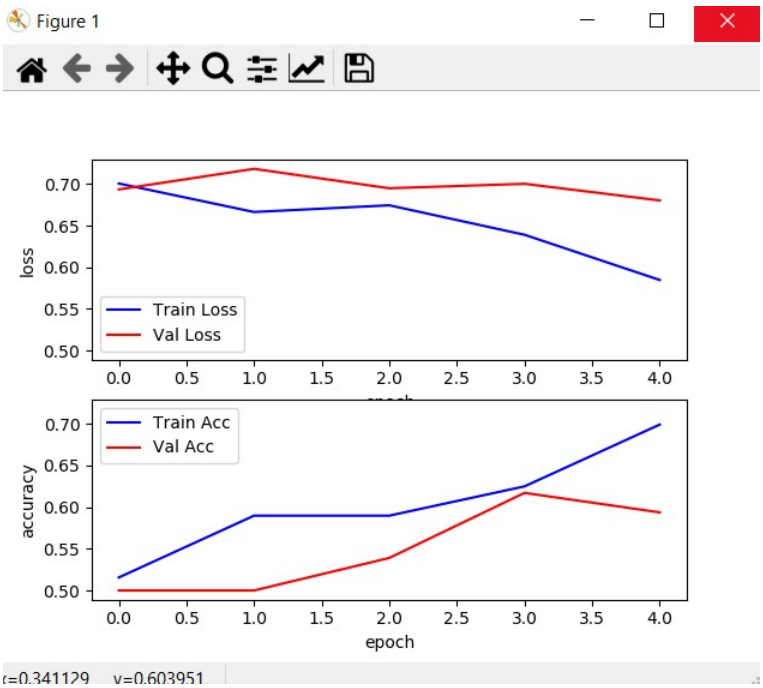


Figure 4: Accuracy and loss graphs of our first trial run!

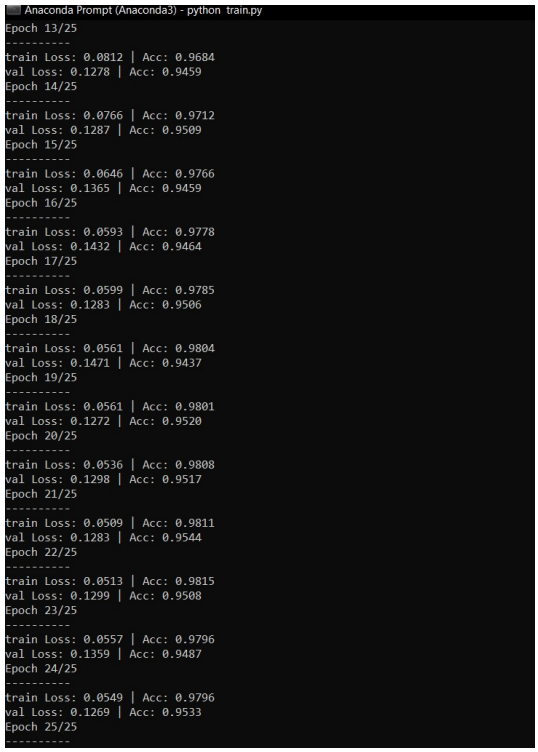
6.2 Final Run!

This is the final run of the model training and here are some minutes about some details.

1. Data could have been trained for more epochs but 20 epochs gave the desired results (also only way to train without getting CUDA Memory getting full).

2. Learning rate is 0.001. Initially learning rate of 0.01 was used but then the number of epochs had to be increased thus increasing the learning time.

Following are the results:



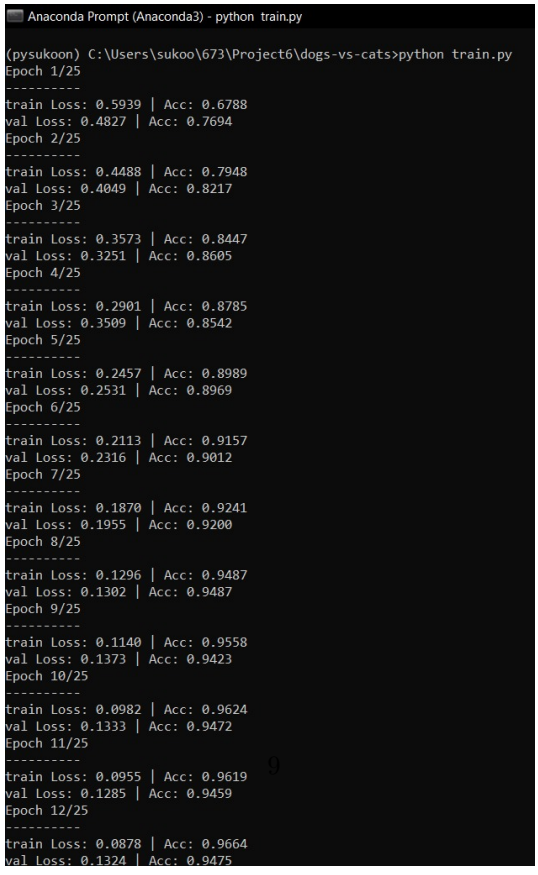


Figure 5: Output during training

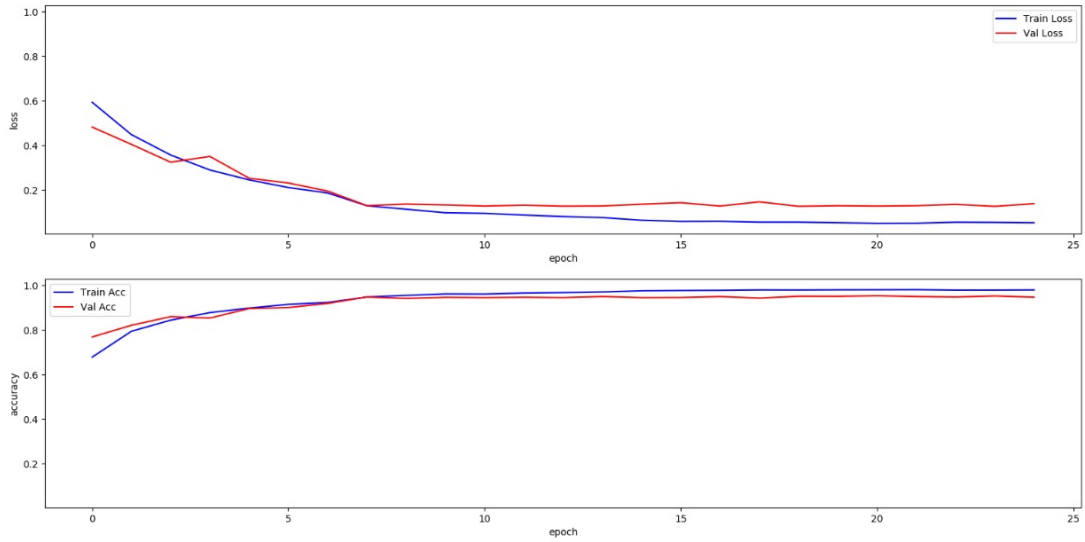


Figure 6: Final Accuracy and Loss Graphs

Zero written on the image denotes that the image is a of class zero i.e. It's a cat!

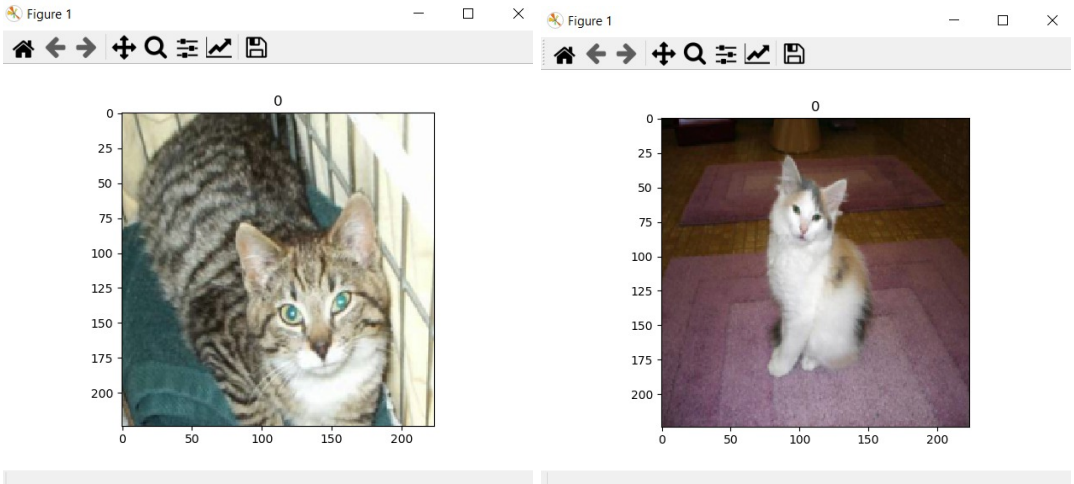


Figure 7: Final Accuracy and Loss Graphs

One written on the image denotes that the image is a of class one i.e. It's a dog!

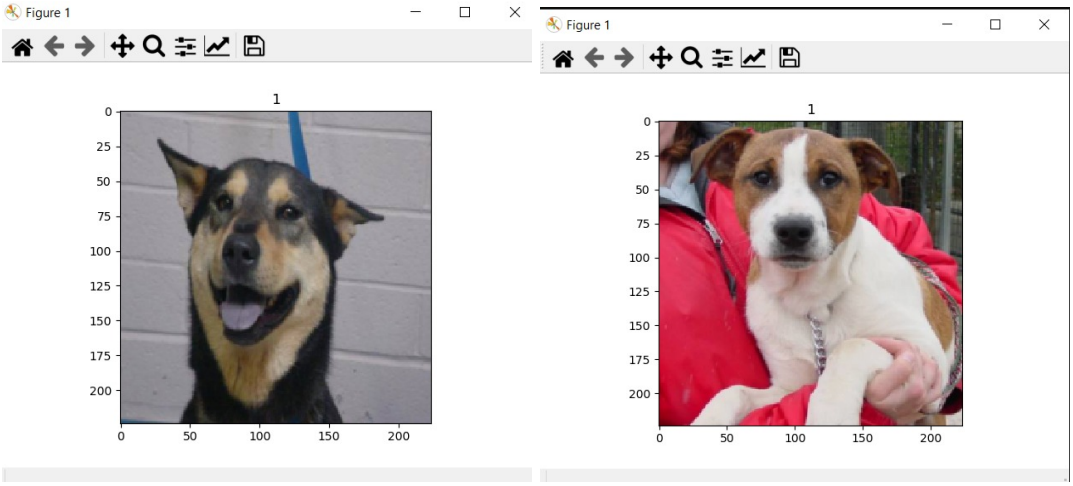


Figure 8: Final Accuracy and Loss Graphs

This is the list of prediction of first 26 images in the test data set!

	A	B	C	D	E	F	G
1	SID	Label (1: Dog, 0: Cat)					
2	1	1					
3	2	1					
4	3	1					
5	4	1					
6	5	0					
7	6	0					
8	7	0					
9	8	0					
10	9	0					
11	10	0					
12	11	0					
13	12	1					
14	13	0					
15	14	0					
16	15	0					
17	16	0					
18	17	1					
19	18	1					
20	19	0					
21	20	0					
22	21	1					
23	22	0					
24	23	1					
25	24	1					
26	25	1					
27	26	1					

submission

Figure 9: Final Accuracy and Loss Graphs

References

[1] https://www.youtube.com/watch?v=WvoLTXIjBYUlist=PLQVvvaa0QuDfhTox0AjmQ6tvTgMBZBEXN index=3

[2] https://www.youtube.com/watch?v=HMcx-zY8JSglist=PL9Hr9sNUjfsmEu1ZniY0XpHSzl5uihcXZ index=4

[3] https://www.pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/