ENPM673 PROJECT 6
Perception of Autonomous Robots

By Group 2
B. Sai Praveen(DirID: spraveen)
Preyash Parikh(DirID: pparikh)
Ajinkya Parwekar(DirID: ajinkyap)

May 16, 2020

1 INTRODUCTION

In this project, the main objective is to differentiate between a dog and a cat based on the given data set.

CONVOLUTIONAL NEURAL NETWORK(CNN)

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm used to differenciate between different images. It takes an image as input, read it, assign importance to characteristics/ aspects of the image for differenciating purpose. The main advantage of CNN is that it requires much less pre-processing as compared to other classification algoritms. In premitive methods, the filters used to differentiate between images are written by humans; But CNN has the ability to learn/ adapt to these filters and characteristics with enough training.

The structure of CNN is inspired by the organization of visual cortex. i.e, the structure of CNN is analogous to that of the connectivity pattern of Neurons in the Human Brain. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area. The agenda for this field is to enable machines to view the world as humans do, perceive it in a similar manner and even use the knowledge for a multitude of tasks such as Image and Video recognition, Image Analysis and Classification, Media Recreation, Recommendation Systems, Natural Language Processing, etc.

A CNN is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

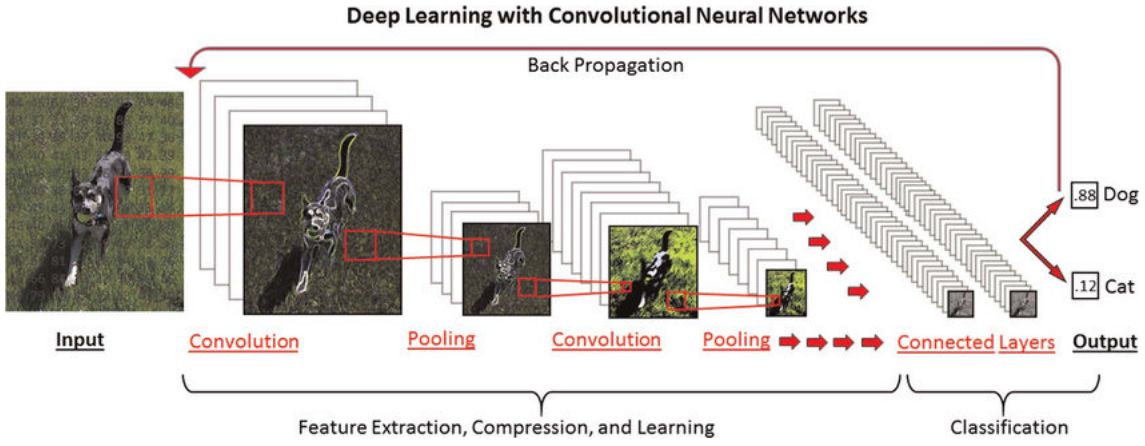

Figure 1: CNN

CONVOLUTIONAL LAYER - THE KERNEL

The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K. The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed. In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output. The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. CNN need not be limited to only one Convolutional Layer. Conventionally, the first Convolution Layer is responsible for capturing the Low-Level features in the image such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how humans would. There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying Valid Padding in case of the former, or Same Padding in the case of the latter.

POOLING LAYER

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling. The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

OPTIMIZER

The main objective is to reduce the difference between the predicted output and the input image. In order to do so, the results obtained will have to be optimitized. This will help in predicting data that was not seen before also. So in order to minimize the cost function we find the optimized value for weights. To find the optimized value we run many iterations with different weights which is known as gradient descent. There are three types of gradient descent variants such as:
1. Batch gradiant descent.
2. Stochastic Gradiant Descent.

3. Mini-batch Gradiant Descent.
In this project we have implemented Stochastic Gradiant Descent (SGD) optimizer on the images.

2 STOCHASTIC GRADIENT DESCENT

Stochastic gradient descent often referred as SGD, is an iterative method for optimizing an objective function $J(\theta)$ with suitable smoothness properties for a model by upgrading the parameters in the opposite direction of the gradient function $\Delta_\theta J(\theta)$ with respect to the parameters.

The mathematical representation of SGD function is,

$$\theta = \theta - \eta * \delta_\theta J(\theta; x^i; y^i)$$

Unlike other descent gradients, stochastic gradient descent is faster than the normal gradient descents and the computations are redundant and are updated one at a time. So, to reduce the fluctuations, the learning rate should be reduced there by giving precise information without much fluctuations.

3 ACTIVATION FUNCTION

1) In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs.

2) The activation function is a mathematical "gate" in between the input feeding the current neuron and its output going to the next layer. It can be as simple as a step function that turns the neuron output on and off or it can be a transformation that maps the input signals into output signals that are needed for the neural network to function.

3) One such function is the Rectified Linear Activation(ReLu) function. It is a function which will give the output for the input if it is positive and if zero no output.

RELU
In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned.

A node or unit that implements this activation function is referred to as a rectified linear activation unit, or ReLU. This function is also known as piece wise linear function because for one half of the input it acts as linear function and nonlinear function for other half domain.

ADVANTAGES OF RELU

1) Computational Simplicity has it does not require to compute exponential factor.

2) Because of linear behaviour,gradients flow well on the active paths of neurons.

3) This function can be made to explore improvements and made to applied in multi layered netwroks and hence is more preferred.

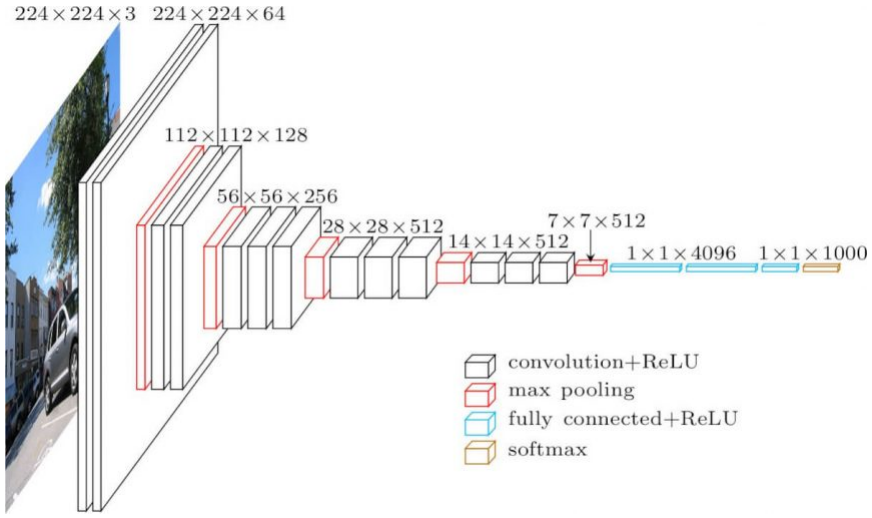

Figure 2: Relu

4 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.

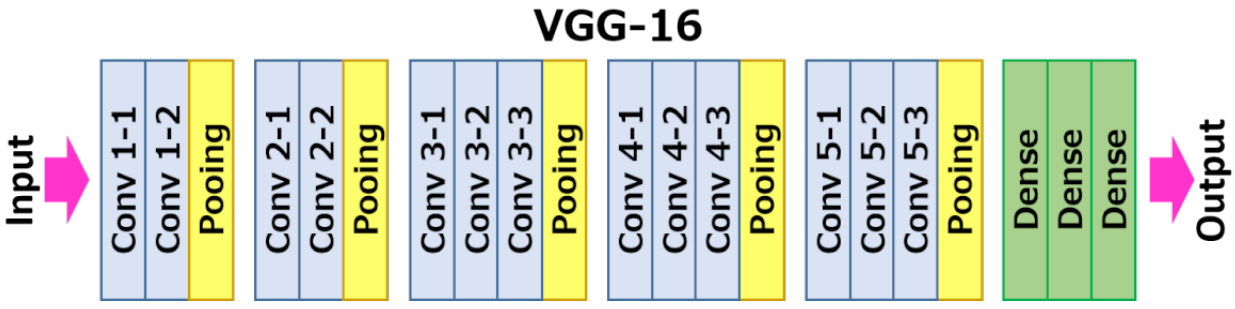

Figure 3: VGG

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.

The input to cov1 layer is of fixed size 224 x 224 RGB image. Initially, the image is passed through a stack of convolutional(conv.) layers, where the filter of 3*3 is used. Also, it utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels followed by non-linearity.

The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2. Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU) non-linearity. Various layers of VGG- 16 are :

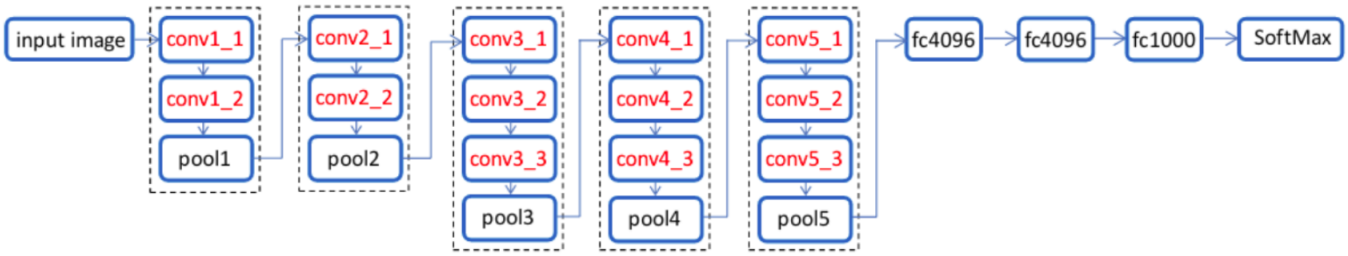

Figure 4: Convolution Flow chart

1) Convolution using 64 filters.
2) Convolution using 64 filters + Max Pooling.
3) Convolution using 128 filters.
4) Convolution using 128 filters + Max Pooling.
5) Convolution using 256 filters.
6) Convolution using 256 filters.
7) Convolution using 256 filters + Max Pooling.
8) Convolution using 512 filters.
9) Convolution using 512 filters.
10) Convolution using 512 filters + Max Pooling.
11) Convolution using 512 filters.
12) Convolution using 512 filters.
13) Convolution using 512 filters + Max Pooling.
14) Fully connected with 4096 nodes.
15) Fully connected with 4096 nodes.
16) Output layer with Softmax activation with 1000 nodes.

There are three major drawbacks with VGGNet -

1) It is painfully slow to train.
2) The network architecture weights themselves are quite large.
3) Due to its depth and number of fully-connected nodes, VGG16 is over 533MB. This makes deploying VGG a tiresome task.

5 PIPELINE

Initially after uploading the dataset, the dataset was divided into two parts based on the file names i.e. either dog or a cat.

Then we picked up a sample image from the dataset and applied convolution layer to the image. Then using keras library, we applied the pooling layer to flatten that image to 1D and extracted and saved the characteristics of the image. Then we optimized the results using SGD variant for optimization. Then we used the ImageDataGenerator function to create training dataset and validated it. The we used the keras.callbacks function and model.fit˙generator function whic yeilded the accuracy and loss data values for multiple epochs. The we plotted the data for those 7 epochs which is shown in figure 6.

Then we used the model.predict˙generator function to finally categorise the dataset into cats or dogs using the tranining dataset earlier created which is shown in figure 7.

6 TENSOR FLOW

TensorFlow is something which allows the developers to create dataflow graphs or a series of processing nodes. Each node in the graph represents a mathematical operation and each connection between nodes is a multidimensional array or tensor. Nodes and tensors in TensorFlow are are python objects and TensorFlow applications are themselves Python applications. The applications of TensorFlow can be run on any convenient targets inlcuding a local machine, CPUs or GPUs.

While creating the graph in CPU or GPU for performing operations, if nothing is specified, then the system first checks the GPU for free space. The graph can be run using the given data and then can run it multiple times for various inputs.

7 RESULTS

MODEL PARAMETERS

Model: "model˙1"
·····

Layer (type) - Output Shape - Param
==========

input˙1 (InputLayer) - (None, 224, 224, 3) - 0
·····

block1˙conv1 (Conv2D) - (None, 224, 224, 64) - 1792
·····

block1˙conv2 (Conv2D) - (None, 224, 224, 64) - 36928
·····

block1˙pool (MaxPooling2D) - (None, 112, 112, 64) - 0
·····

block2˙conv1 (Conv2D) - (None, 112, 112, 128) - 73856
·····

block2˙conv2 (Conv2D) - (None, 112, 112, 128) - 147584
·····

block2˙pool (MaxPooling2D) - (None, 56, 56, 128) - 0
·····

block3˙conv1 (Conv2D) - (None, 56, 56, 256) - 295168
·····

block3˙conv2 (Conv2D) - (None, 56, 56, 256) - 590080
·····

block3˙conv3 (Conv2D) - (None, 56, 56, 256) - 590080
·····

block3˙pool (MaxPooling2D) - (None, 28, 28, 256) - 0
·····

block4˙conv1 (Conv2D) - (None, 28, 28, 512) - 1180160

.....

block4˙conv2 (Conv2D) - (None, 28, 28, 512) - 2359808
.....

block4˙conv3 (Conv2D) - (None, 28, 28, 512) - 2359808
.....

block4˙pool (MaxPooling2D) - (None, 14, 14, 512) - 0
.....

block5˙conv1 (Conv2D) - (None, 14, 14, 512) - 2359808
.....

block5˙conv2 (Conv2D) - (None, 14, 14, 512) - 2359808
.....

block5˙conv3 (Conv2D) - (None, 14, 14, 512) - 2359808
.....

block5˙pool (MaxPooling2D) - (None, 7, 7, 512) - 0
.....

global˙max˙pooling2d˙1 (Global) - (None, 512) - 0
.....

dense˙1 (Dense) - (None, 512) - 262656
.....

dropout˙1 (Dropout) - (None, 512) - 0
.....

dense˙2 (Dense) - (None, 1) - 513
==========

Total params: 14,977,857
Trainable params: 7,342,593
Non-trainable params: 7,635,264
.....

ACCURACY

Epoch 1/16
7/5625 [.....] - ETA: 1:33:33 - loss: 0.8336 - accuracy: 0.6071
/usr/local/lib/python3.6/dist-packages/keras/callbacks/callbacks.py:95: RuntimeWarning: Method (on˙train˙batch˙end) is slow compared to the batch update (0.210629). Check your callbacks.

5625/5625 [==========] - 354s 63ms/step - loss: 0.2289 - accuracy: 0.8988 - val˙loss: 0.157 - val˙accuracy: 0.9328

Epoch 2/16
5625/5625 [==========] - 347s 62ms/step - loss: 0.1287 - accuracy: 0.9484 - val˙loss: 0.1566 - val˙accuracy: 0.9688

Epoch 3/16
5625/5625 [==========] - 344s 61ms/step - loss: 0.1092 - accuracy: 0.9568 - val˙loss: 6.5549e-04 - val˙accuracy: 0.9664

Epoch 4/16
5625/5625 [==============================] - 343s 61ms/step - loss: 0.0904 - accuracy: 0.9631 - val_loss: 0.1137 - val_accuracy: 0.9572

Epoch 5/16
5625/5625 [==============================] - 343s 61ms/step - loss: 0.0850 - accuracy: 0.9674 - val_loss: 4.0925e-04 - val_accuracy: 0.9668

Epoch 6/16
5625/5625 [==============================] - 342s 61ms/step - loss: 0.0774 - accuracy: 0.9685 - val_loss: 0.2138 - val_accuracy: 0.9640

Epoch 7/16
5625/5625 [==============================] - 346s 62ms/step - loss: 0.0716 - accuracy: 0.9713 - val_loss: 0.0091 - val_accuracy: 0.9716

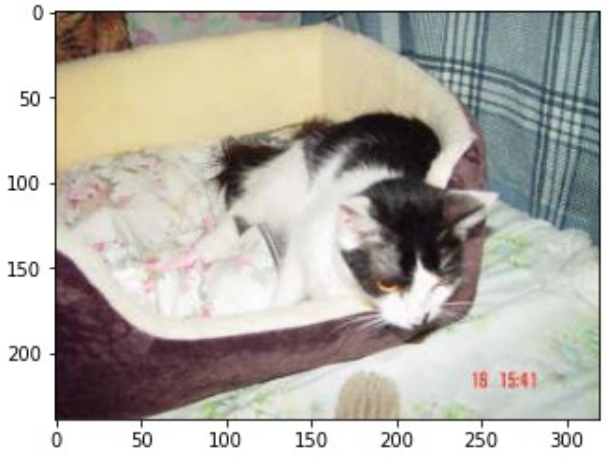

Figure 5: Cat

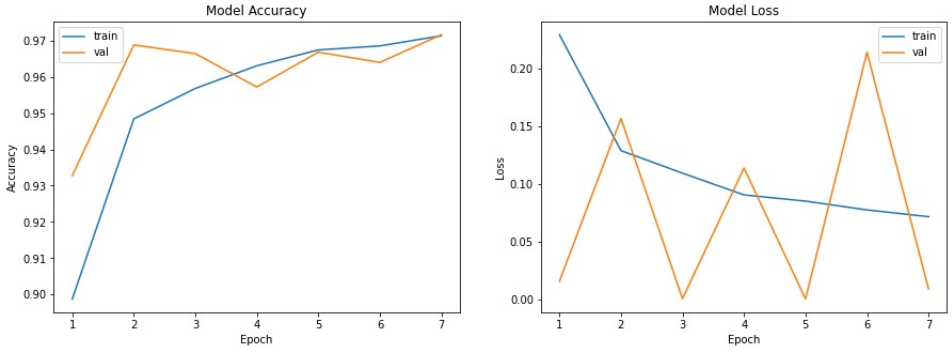

Figure 6: Data for 7 epochs

Figure 7: Cats and Dogs

8 CHALLENGES FACED

1) It takes about 15 minutes to compute accuracy. So, making changes in the program was quite time consuming.
2) Initially we tried to run the program on Laptop GPU by installing CUDA. But we faced difficulty in connecting CUDA with the jupyter notebook.
3) Since we have chosen to read images by uploading a zip file on the google colab, every time we refreshes the runtime we have to upload the zip file again.

9 REFERENCES

1) https://medium.com/datadriveninvestor/cnn-architecture-series-vgg-16-with-implementation-part-i-bca79e7db
2) https://neurohive.io/en/popular-networks/vgg16/.
3) https://en.wikipedia.org/wiki/Activation_function.
4) https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.
5) https://www.researchgate.net/figure/Deep-learning-schematic-with-a-CNN-designed-to-classify-an-input-imag
fig2_323950097.