# Implementation of Computational Neural Network

## What is a Convolutional Neural Network?

The Convolutional Neural Network is a particular type of neural network inspired by biological visual systems and have been proven to be very effective for image recognition. The word ”convolution” comes from the convolution layer which consists of a dot product of a square grid of pixels (”filter”). The grid is translated across the image to form a feature map. The application of various filters (defined during training) allow for the detection of features such as edges and even complicated shapes.

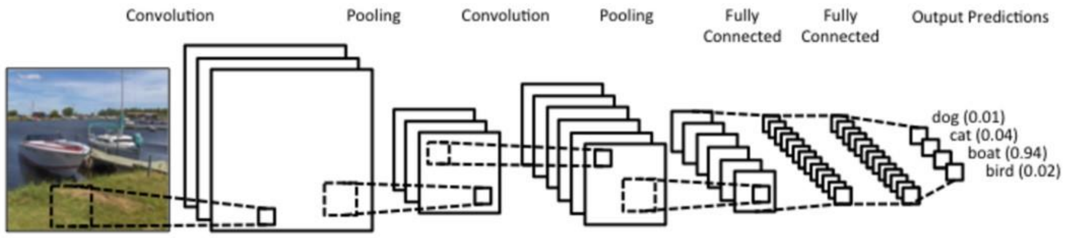


Figure 1: Representation basic flow for CNN

The aim in the project is to implement image classification on a dataset which includes images of cats and dog and classify them by training the CNN model and testing the model. The workflow I will be using for our project is as below:

- Understanding and Pre-processing Data
- Creating the Model
- Training the Model
- Testing the Model
- Improvising and repeating the process to increase its accuracy

## Understanding and Pre-processing Data

Started with understanding the dataset used for training. and classify them as ’cat’ and ’dog’ so that I can segregate the data for model to learn. I have specified information for dataset in this block such as the split ratio and image size.

Next, I focused on Data preparation where I do the following processes:

- Read image from data files
- Resize it so that features can be extracted later.

- Convert image from string type to float
- Append processed image along with the label

**Creating the Model**

The model consists of five convolution blocks with a max pool layer in each of them. There is also a fully connected layer with 256 units. This is activated using 'ReLU' activation function. ReLU stands for rectified linear unit, and is the most commonly used activation function CNNs. This function is linear for all positive values, and zero for all negative values.

Using 'ReLU' reduces the time required to train the model s it doesn't have complex mathematics associated with it.

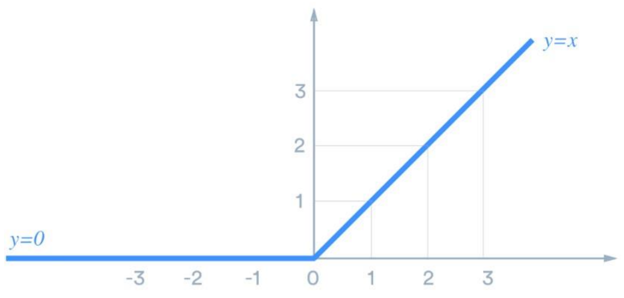


Figure 2: Graph for ReLU

I then complied with the model using binary cross entropy loss function along with ADAM optimiser.

After creating the model, I trained it based on CNN and tested the accuracy and loss percentage.

**Improvising of Computational Neural Network**

**Batch Normalization**

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

Batch Normalization helps us as it converges fast i.e. loss starts to lower faster. This can help us minimize the number of epochs required, if I stopped training after there are no more improvements.

**Dropout**

Another technique to reduce overfitting is to introduce dropout to the network. It is a form of regularization that forces the weights in the network to take only small values, which makes the distribution of weight values more regular and the network can reduce overfitting on small training examples. I then applied 0.2 dropout to few layers and 0.3 to some to regularize the model.

When you apply dropout to a layer it randomly drops out (set to zero) number of output units from the applied layer during the training process. Dropout takes a fractional number as its input value, in the form such as 0.1, 0.2, 0.4, etc. This means dropping out 10%, 20% or 40% of the output units randomly from the applied layer.

When applying 0.1 dropout to a certain layer, it randomly kills 10% of the output units in each training epoch.

I have applied 0.2 dropout to few layers and 0.3 to some to regularize the model.

**Decayed Learning Rate**

Learning rate controls the step I  make along the gradient. if the learning rate is high that means I are taking bigger steps, and hence decaying the learning rate for every epoch can help us increase the accuracy. When I  decrease the learning rate, it allows our model to descend into areas of the loss landscape that are more optimal and would have otherwise been missed entirely.

For GPU (NVIDIA GeForce - 940 MX), and 10,000 images to train, I chose the batch size of 8 and started with learning rate pf $1e^{-3}$ for epoch 10, and then reduced the learning rate to $1e^{-5}$ for epoch 5 to increase the accuracy.

| Sr. No. | No. of Images | Image size | Epoch | Training Accuracy | Validation Accuracy |
|---|---|---|---|---|---|
| 1 | 5000 | 120 | 5 | 52.79 | 69.68 |
| 3 | 10,000 | 120 | 5 | 73.00 | 60.27 |
| 4 | 10,000 | 140 | 5 | 76.53 | 48.43 |
| 5 | 10,000 | 200 | 20 | 98.00 | 88.00 |
| 6 | 15,000 | 140 | 5 | 79.60 | 43.04 |
| 7 | 15,000 | 150 | 5 | 83.66 | 35.18 |
| 8 | 15,000 | 150 | 10 | 95.07 | 81.25 |
| 9 | 18,000 | 110 | 15 | 97.59 | 90.44 |

Table 1: Training Accuracy and Validation Accuracy

I ran the model multiple time to understand how the accuracy and loss rate is affected by the changes I make. As I increase the layers of Convulation, epochs and no. of images to train I can see the accuracy increasing.

As I can see in the last rows for 10,000 and 15,000 images the model seems to be overfitting as the training accuracy is increasing linearly over time, whereas validation accuracy stalls around 81%-90% in the training process. Also, the difference in accuracy between training and validation accuracy is noticeably a sign of **overfitting**.

When there are a small number of training examples, the model sometimes learns from noises or unwanted details from training examples—to an extent that it negatively impacts the performance of the model on new examples. This phenomenon is known as overfitting. It means that the model will have a difficult time generalizing on a new dataset.

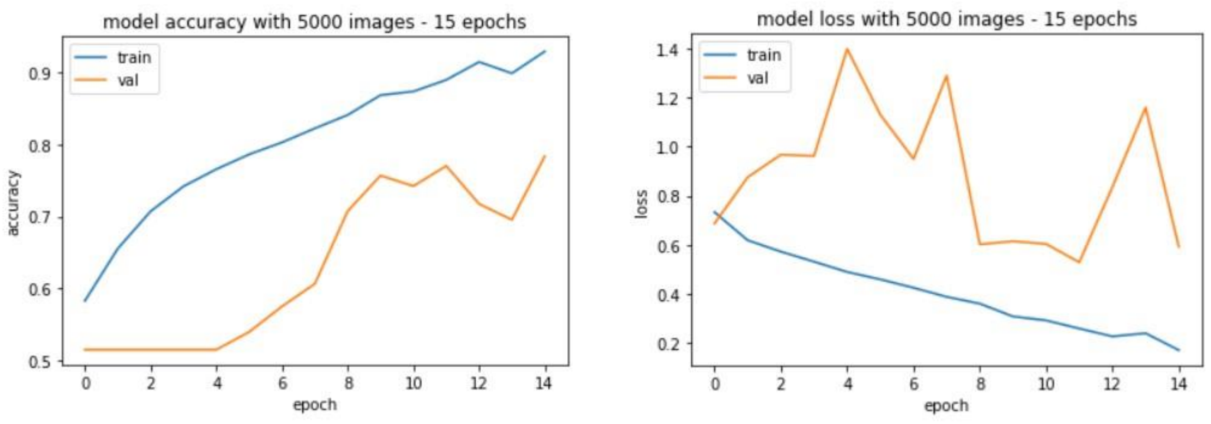
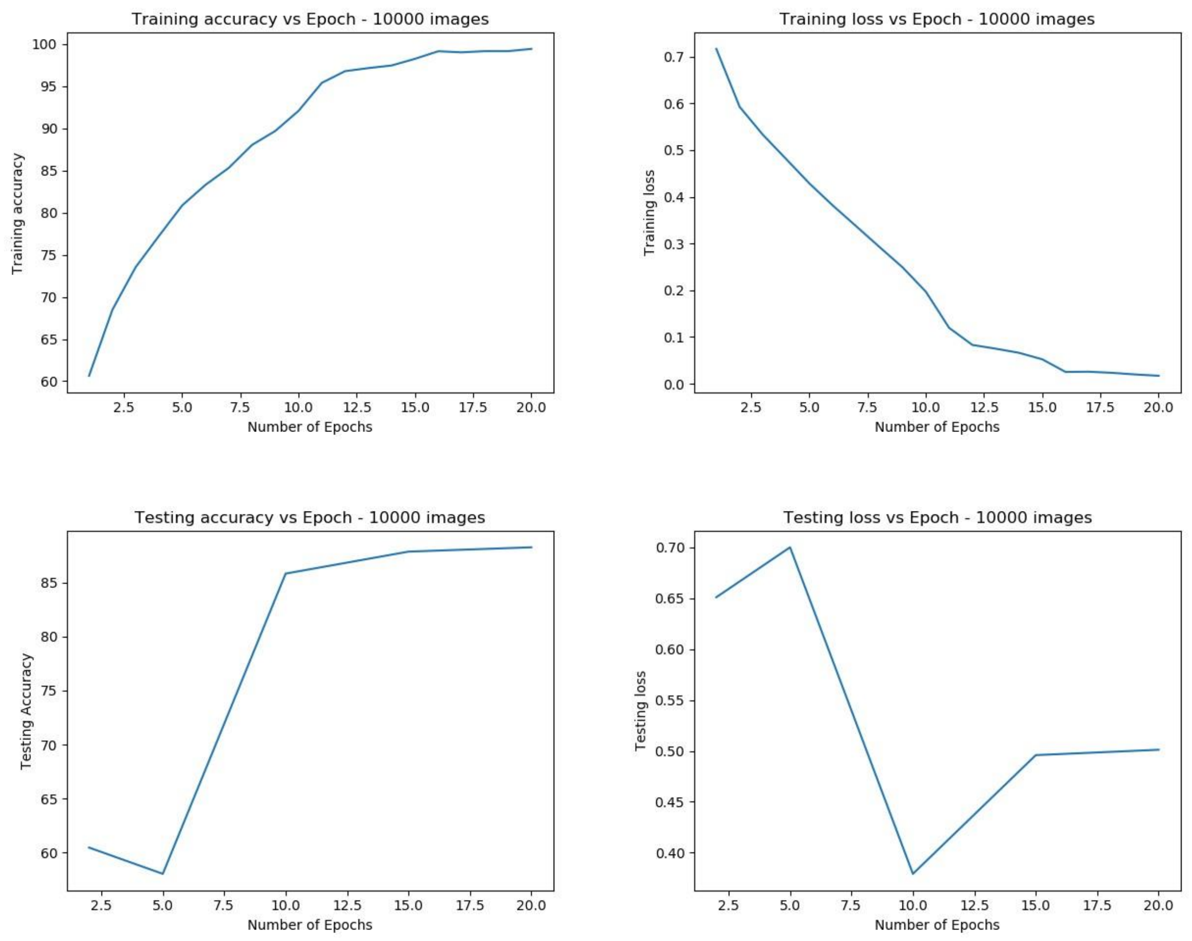


Figure 3: Results for 5,000 images

The out for 18,000 images as training dataset (Figure 5), was tested on 200 test images and the link to the video output is as below.



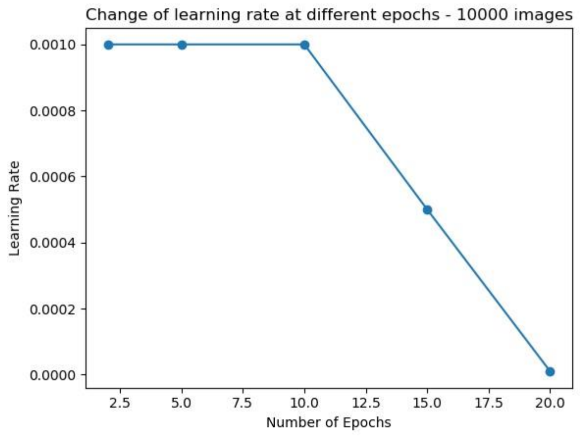


Figure 4: Results for 10,000 images

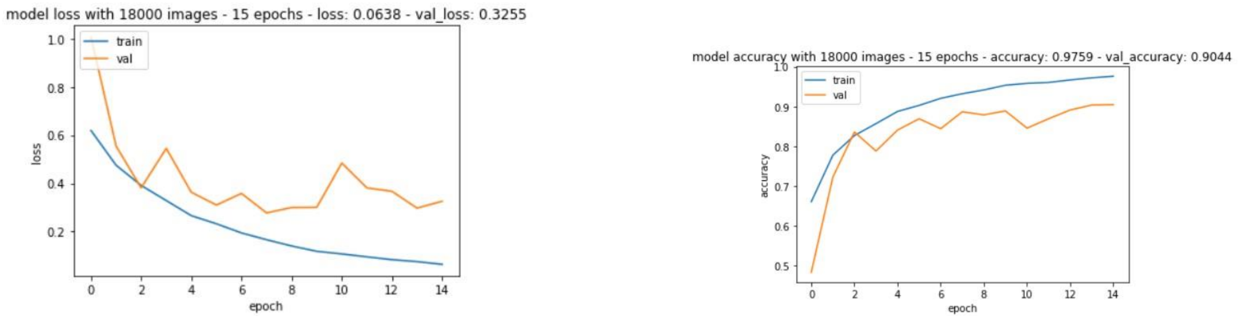


Figure 5: Results for 18,000 images