"[ENEE633/CMSC828C] PROJECT-2"

Statistical Pattern Recognition

PROJECT REPORT

NEVIL PATEL:- 116897068

Contents

PART 1- Handwritten Digit Recognition:- 3
Project Description:- 3
Data Visualization:- 3
SVM:- 4
CNN:- 5
RESULTS:- 6
PART 2- Transfer Learning:- 7
Project Description:- 7
Program Flow:- 7
Alexnet for Transfer Learning:- 7
Simple CNN:- 8
RESULTS:- 9
LEARNING AND VALIDATION CURVES 9
REFERENCES: 11

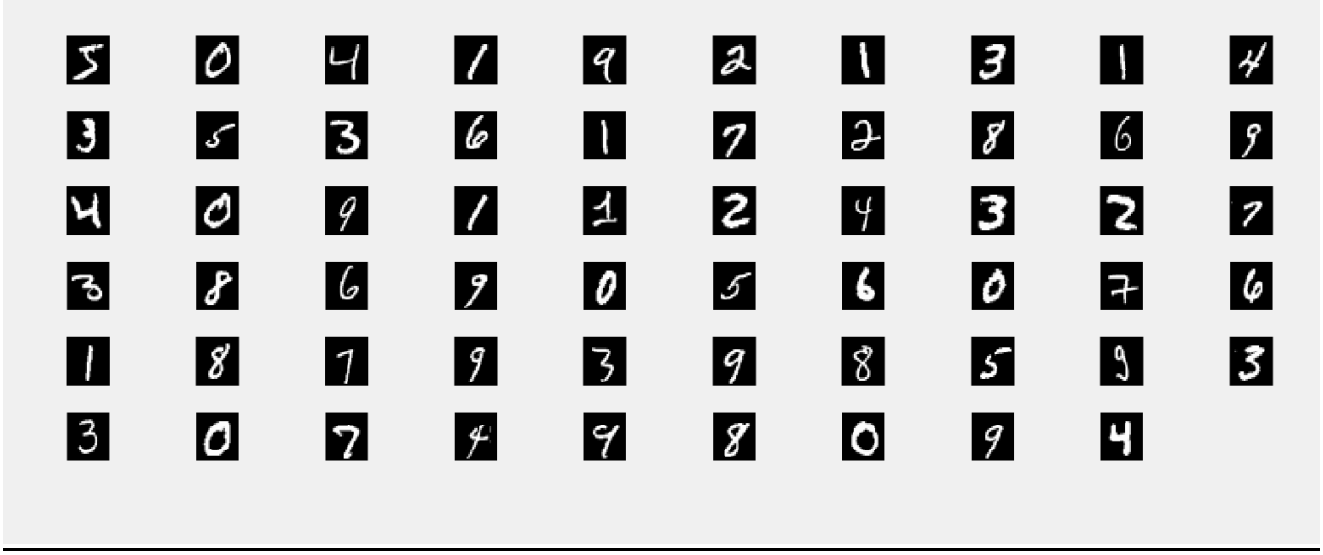
PART 1- Handwritten Digit Recognition:-

Project Description:-

SVM :- Implement linear and kernel SVM on MNIST dataset. You have to try different kernels (linear, polynomial, RBF) and compare results in your report. You can use any online toolbox for this, e.g. LIBSVM (https://www.csie.ntu.edu.tw/~cjlin/libsvm/) or any MATLAB built-in function. You can apply PCA and LDA here for dimensionality reduction.

Deep Learning :- Build a Convolutional Neural Network. Train it on MNIST training set and test it on testing set. You can design your architecture or use the architecture introduced in LeCun's paper [1]. Dataset You can download MNIST from http://yann.lecun.com/exdb/mnist/. The description of Dataset is also on the website. Basically, it has a training set with 60000 28 x 28 grayscale images of handwritten digits (10 classes) and a testing set with 10000 images.

Data Visualization:-



SVM:-

Introduction:-

Support Vector Machines are a type of supervised machine learning algorithm that provides analysis of data for classification and regression analysis. While they can be used for regression, SVM is mostly used for classification. We carry out plotting in the n-dimensional space. Value of each feature is also the value of the specific coordinate. Then, we find the ideal hyperplane that differentiates between the two classes.

For 2 class problem, given a training set of instance-label pairs *(**x**i, yi), i=1, …, l,* the SVM find a separating hyper-plane (weight vector ***w*** and threshold *b*) with maximum margin by solving the following optimization problem. The training vector ***xi*** can be mapped to a higher dimensional space using kernel functions such as

Linear Kernel:

The equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

$$k(x_i, x_j) = x_i * x_j$$

Polynomial kernel:

It is popular in image processing.

Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

Polynomial kernel equation

where d is the degree of the polynomial.

Gaussian radial basis function (RBF):

It is a general-purpose kernel; used when there is no prior knowledge about the data.

Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

, for:

$$\gamma > 0$$

Sometimes parametrized using:

$$\gamma = 1/2\sigma^2$$

Gaussian radial basis function (RBF)

The LIBSVM toolbox supports these kernel functions.

Multi-class SVM:-

In its most simple type SVM are applied on binary classification, dividing data points either in 1 or 0. For multiclass classification, the same principle is utilized. The multiclass problem is broken down to multiple binary classification cases, which is also called *one-vs-one*. In scikit-learn one-vs-one is not

default and needs to be selected explicitly (as can be seen further down in the code). *One-vs-rest* is set as default. It basically divides the data points in class x and rest. Consecutively a certain class is distinguished from all other classes.

The number of classifiers necessary for *one-vs-one multiclass classification* can be retrieved with the following formula (with n being the number of classes):

$$\frac{n * (n-1)}{2}$$

In the one-vs-one approach, each classifier separates points of two different classes and comprising all one-vs-one classifiers leads to a multiclass classifier.

LIBSVM toolbox:-

The *svmtrain* function and the *svmpredict* function in the LIBSVM toolbox are used for training and testing the SVM model:-

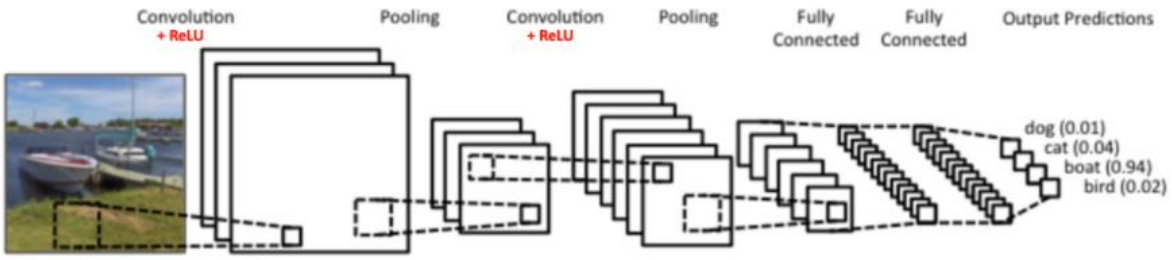
```
model{i} = svmtrain(double(y==(i-1)), X', '-s 0 -t 0 -b 1');

[~,~,p] = svmpredict(double(yt==(i-1)), XT', model{i}, '-b 1');
```

Dimensionality reduction:-

By lowering the dimension, we can speed up the training process using PCA or LDA. I implemented both the methods in this project to lower dimension from d = 784 to p = 10.

CNN:-

Convolutional Neural Networks (**ConvNets** or **CNNs**) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self-driving cars.



CNN Architecture:

The CNN architecture used in this project is as follows:

- imageInputLayer([28 28 1]
- Conv 1: filter = 32, kernel_size = 4, stride = 1;
- ReLU 1 , Normalization 1
- Max-pool 1: kernel_size = 3, stride = 3;
- Dropout layer 1
- Conv 2: filter = 16, kernel_size = 3, stride = 1;
- ReLU 2 , Normalization 2
- Dropout layer 2
- Fully connected Layer
- Softmax Layer
- Classification (Output = 10 Classes)

RESULTS:-

METHOD	SVM						CNN
	PCA (p=10)			LDA (p=10)			
	Linear	Poly	RBF	Linear	Poly	RBF	
Accuracy	72.8%	93.45%	94.4%	88.45%	88.41%	90.20%	98.66%

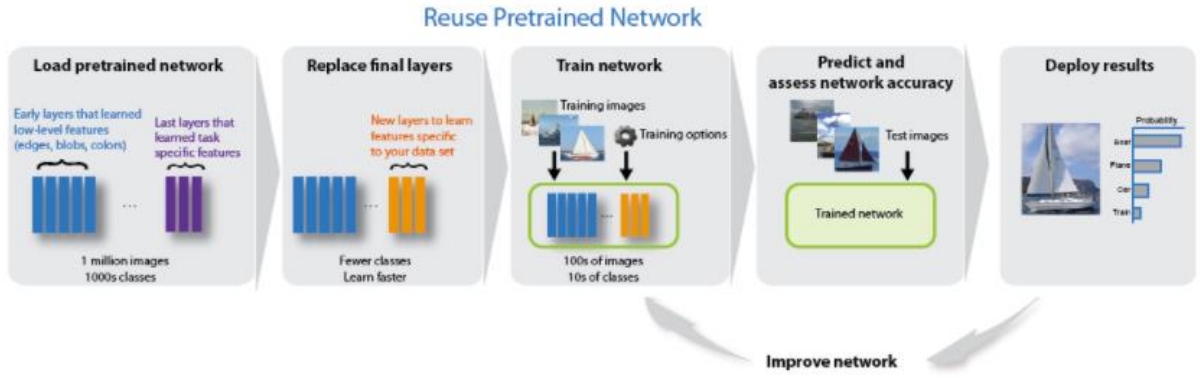
- Comparing PCA and LDA methods. LDA performs best when RBF kernel is used It's because it discriminating the high-dimension data mapped by the kernel functions. PCA has better performance comparing both.
- By looking at different kernel results we can see that RBF kernel is better then linear kernel in both the cases because RBF function maps the data into infinite dimensions and thus can have better performance.
- Comparing SVM and CNN, CNN is a winner for our case because it exploits the spatially local correlation of an image
- Hyperparameters were taken using Grid Search Method.

PART 2- Transfer Learning:-

Project Description:-

For this part of the project, you will be applying a deep learning technique commonly referred to as Transfer Learning. Specifically, Transfer Learning can be used for deep learning applications where either data or computational power are restricted. Here you are given a data set with ten classes (ten different monkey species) with only 140 images per class. The first task will be to train a simple convolutional neural network using these images (a very simple 3-5 convolutional network will suffice) and 1 test the accuracy of the model using the validation set. Because of the low number of training samples, you will see that the test accuracy of the model will be lower than expected. This is where transfer learning can help. The next task will be to download a pretrained model for image classification (for example VGG, Alexnet, InceptionNet) and use it as a feature extractor. To do this, you will remove the last fully connected layers of the pretrained model and replace it with untrained fully connected layers to classify the monkey species. During training, you will freeze the convolutional layer parameters so that they remain the same and only update the fully connected layers at the end. In this way, the convolutional layers act as generalized feature extractors that have already been pretrained on millions of other images (that werent necessarily all monkeys) while the fully connected layers are able to take these features and classify our images. You should see a substantial boost in accuracy even though we have the same amount of training samples. To further boost the performance of the network, you can unfreeze the convolutional layers and train for a few more epochs with a small step size to _ne tune the network to extract even more predictive power. Dataset You can download the dataset from here: https://www.kaggle.com/slothkong/10-monkey-species

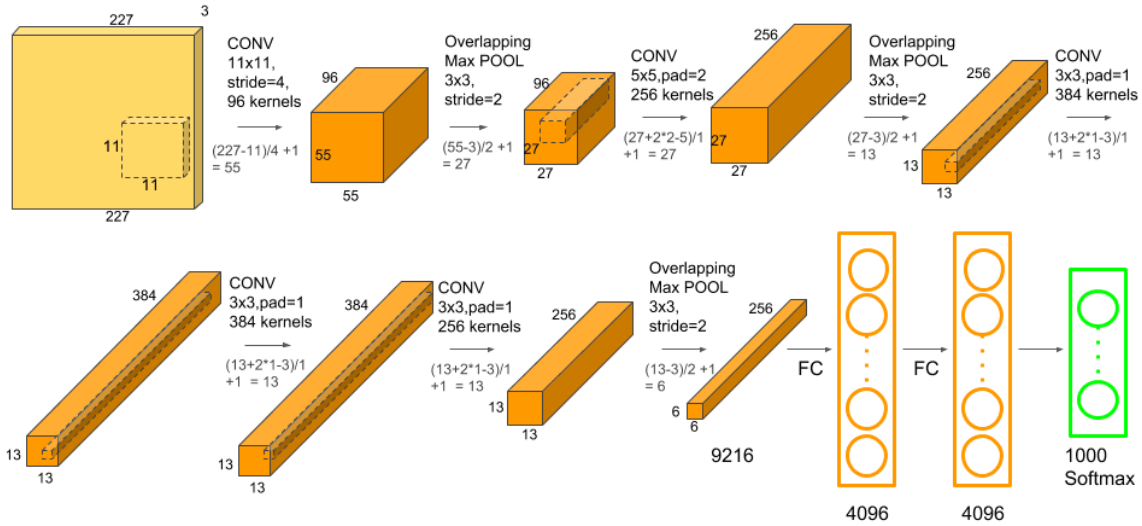
Program Flow:-



Alexnet for Transfer Learning:-

AlexNet has been trained on over a million images and can classify images into 1000 object categories (such as keyboard, coffee mug, pencil, and many animals). The network has learned rich feature representations for a wide range of images. The network takes an image as input and outputs a label for the object in the image together with the probabilities for each of the object categories.

Transfer learning is commonly used in deep learning applications. You can take a pretrained network and use it as a starting point to learn a new task. Fine-tuning a network with transfer learning is usually much faster and easier than training a network with randomly initialized weights from scratch. You can quickly transfer learned features to a new task using a smaller number of training images.



Simple CNN:-

CNN Architecture:

The simple CNN architecture used in this project 2 is as follows:

- imageInputLayer([150 150 3]
- Conv 1: filter = 6, kernel_size = 10, stride = 1;
- Normalization 1
- ReLU 1
- Max-pool 1: kernel_size = 2, stride = 2;
- Conv 2: filter = 16, kernel_size = 10,
- Normalization 2
- ReLU 2
- Max-pool 2: kernel_size = 2, stride = 2;
- Conv 3: filter = 32, kernel_size = 10,
- Normalization 3
- ReLU 3
- Max-pool 3: kernel_size = 2, stride = 2;
- Conv 4: filter = 32, kernel_size = 10,
- Normalization 4
- ReLU 4
- Fully connected Layer
- Softmax Layer
- Classification (Output = 10 Classes)

RESULTS:-

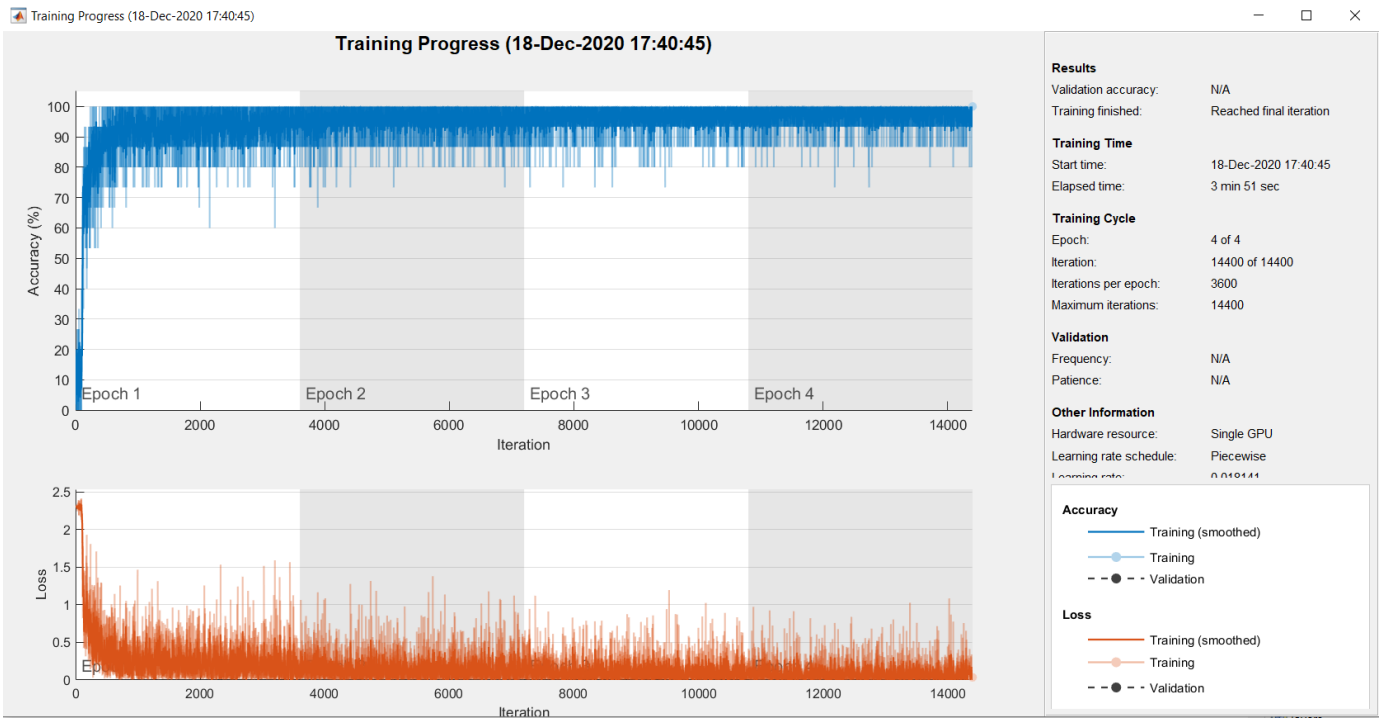
METHOD	DEEP LEARNING		
	SIMPLE CNN	TRANSFER LEARNING (Freeze Conv. Layer param)	TRANSFER LEARNING (Unfreeze Conv. Layer)
ACCURACY	45.59%	92.65%	94.12%
TIME	11s	4s	9s

- Hyperparameters were taken using Grid Search Method.
- Here we can see that training a simple convolutional neural network using these images (simple 3-5 convolutional network) we found out that the Test accuracy of the model is very low.
- By using Transfer Learning, we can see that the Accuracy is twice compared to simple CNN.
- Comparing Time cost we get the minimum time when Transfer learning is implemented while freezing of Conv. Layer parameters.
- We got best predictive power in Transfer learning when unfreezing Conv. Layer.
- Hyperparameters were taken using Grid Search Method.

LEARNING AND VALIDATION CURVES

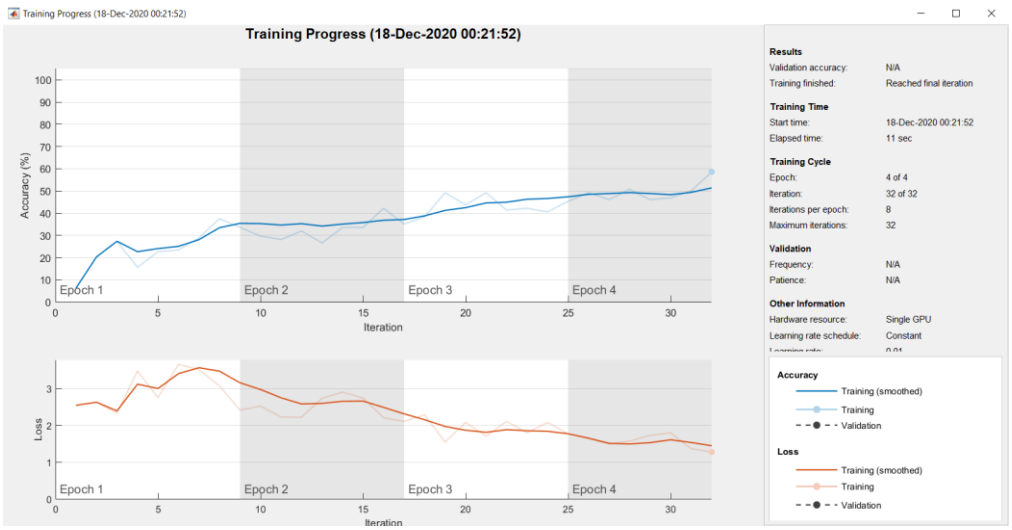
PART -1

CNN-MNIST



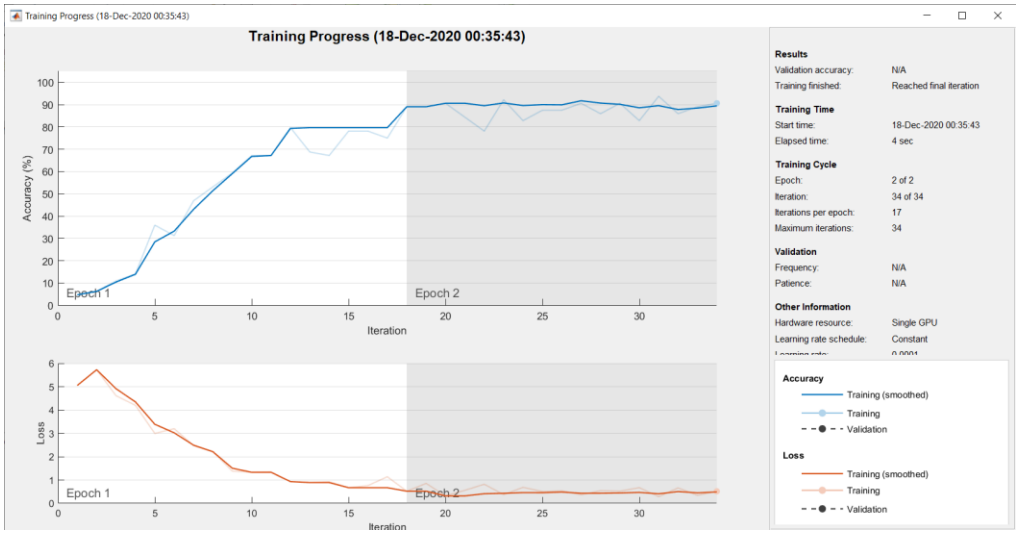
PART -2

SIMPLE CNN-MONKEY SPECIES



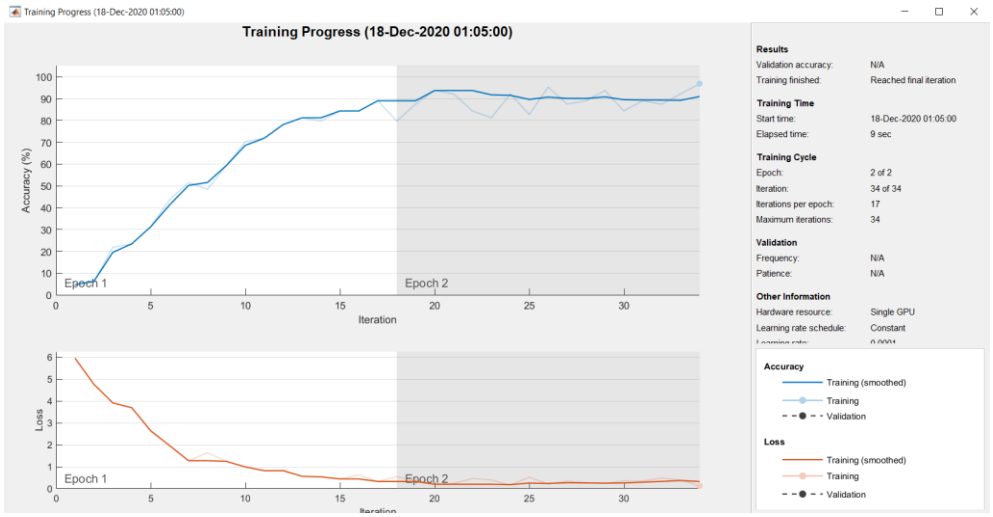

```
Training on single GPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:01 |       10.94% |       2.6693 |          0.0100 |
|       4 |          32 |       00:00:11 |       50.78% |       1.4503 |          0.0100 |
|=========================================================================================|
```

TRANSFER LEARNING CNN-MONKEY SPECIES-(FREEZE)




```
Training on single GPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:00 |        4.69% |       5.0596 |      1.0000e-04 |
|       2 |          34 |       00:00:04 |       90.63% |       0.5186 |      1.0000e-04 |
|=========================================================================================|
```

TRANSFER LEARNING CNN-MONKEY SPECIES-(UNFREEZE)




```
Training on single GPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:01 |        4.69% |       5.9642 |      1.0000e-04 |
|       2 |          34 |       00:00:09 |       96.88% |       0.1268 |      1.0000e-04 |
|=========================================================================================|
```

REFERENCES:

https://data-flair.training/blogs/svm-kernel-functions/#:~:text=SVM%20Kernel%20Functions&text=Different%20SVM%20algorithms%20use%20different,images%2C%20as%20well%20as%20vectors.

https://www.mathworks.com/help/deeplearning/ug/transfer-learning-using-alexnet.html