IE 6318- DATA MINING AND ANALYTICS

Spring 2019

SPOT DETECTION IN X-RAY MEDICAL IMAGES

Submitted to

Dr. Shouyi Wang

Group Members

Leons Manuel Saju	1001620520
Goutham Munagala	1001565060
Shivam Goyal	1001667663

TABLE OF CONTENTS

INTRODUCTION

Abstract:

The purpose of this project is to carry out spot detection in X-ray medical images in the surgical field. The aim is to accurately classify the number of dark spots in the X-ray images. Integration of machine learning algorithms has become widespread thanks to their efficiency and adaptability. This technology has enabled many breakthroughs in computer vision, which offer new approaches to solve complex problems. For instance, deep learning algorithms based on a convolutional neural network (CNN) have great object detection capabilities which can be used in technology such as self-driving cars, satellite views, etc.

As this technology constantly progresses, we are constantly seeking to expand our knowledge of it and to use it to overcome challenges in the surgical field. As part of this long-term goal, our aim is to accurately detect spots in X-Ray medical images using machine learning algorithms. This push for innovation leads to faster and more accurate patient results, patient satisfaction and customer service.

Tools used:

- Matlab R2018b
- Python 3.7

Convolutional Neural networks (CNN), a type of Deep Neural network (DNN) is used for this image classification challenge. The workings of CNN are described in the subsequent sections.

BACKGROUND

Machine learning:

Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. It can be explained as automating and improving the learning process of computers based on their experiences without being programmed. It gives the computer that which makes it more like humans: The ability to learn. Machine learning is actively being used today, perhaps in many more places than one would expect. The process starts with feeding good quality data and then training the computers by building machine learning models using the data and different algorithms. The choice of algorithms depends on what type of data do we have and what kind of task we are trying to automate.

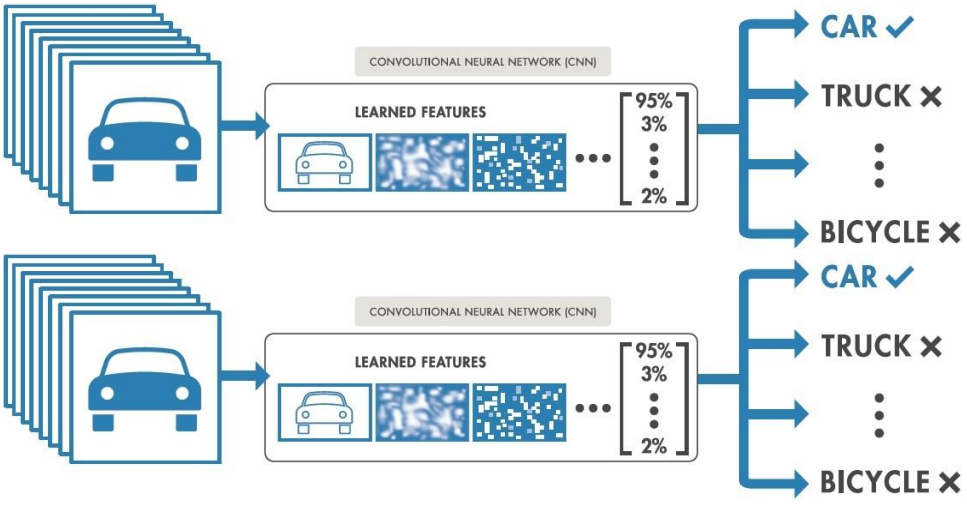
Machine learning methods:

There are mainly 3 types of machine learning algorithms.

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

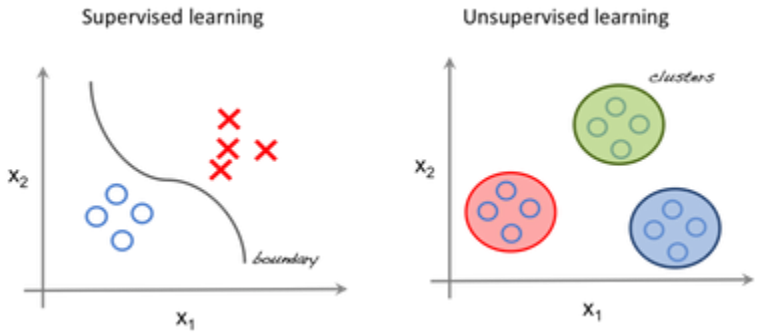
Supervised learning:

In Supervised Learning, algorithms learn from labeled data. After understanding the data, the algorithm determines which label should be given to new data based on pattern and associating the patterns to the unlabeled new data as shown in the image below.



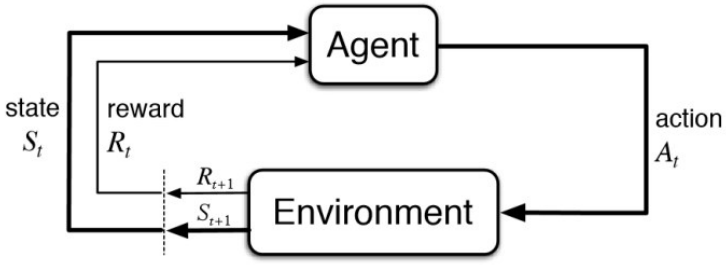
Unsupervised learning:

It is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. It is a class of Machine Learning techniques to find the patterns in data. The data given to unsupervised algorithm are not labelled, which means only the input variables are given with no corresponding output variables. The algorithms are left to themselves to discover interesting structures in the data.



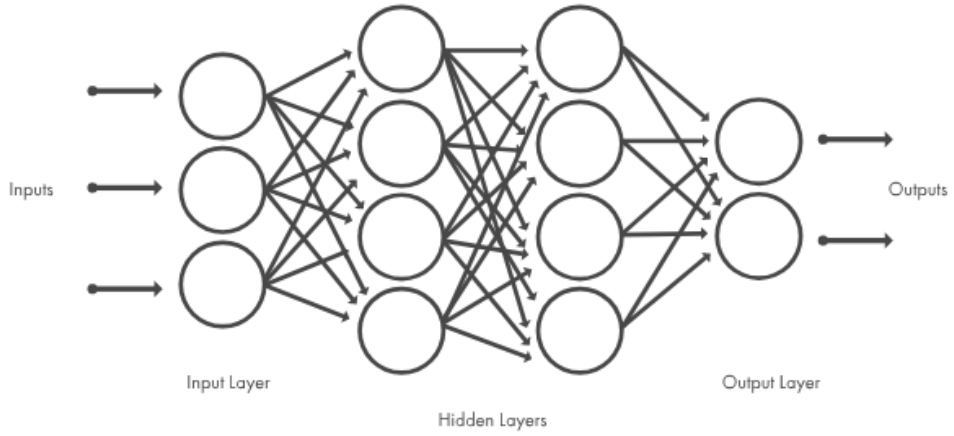
Reinforcement learning:

Reinforcement Learning is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. Reinforcement learning is quite widely used in building AI for playing computer games. AlphaGo Zero is the first computer program to defeat a world champion in the ancient Chinese game of Go. Others include ATARI games, Backgammon, etc.



Method Used:

For the purpose of this project, Deep Neural Network (DNN) is used to classify the images into their respective categories. Basic structure of DNN is shown below. We use Convolutional neural network, a type of DNN for our image classification.

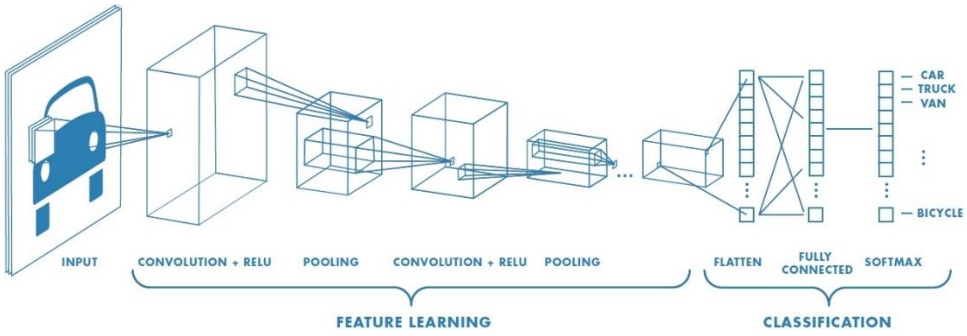


What is image classification?

Image classification is the task of taking an input image and outputting a class (a cat or a dog) or a probability of classes that best describes the image.

When we see an image or just when we look at the world around us, most of the time we can immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.

We want the computer to be able to differentiate between all the images it's given and figure out the unique features that make an X-ray image with one spot different from the images with two spots or no spots.



How computer sees an image?

When a computer takes image as an input, it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 24 x 24 x 3 array of numbers in our case 24 x 24 refers to the size of the image and 3 refers to RGB values. Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point. 0 being the dark and 225 being the white.

CNNs take the image, pass it through a series of convolutional, nonlinear, pooling (down sampling), and fully connected layers, and get an output. The output can be a single class or a probability of classes that best describes the image.

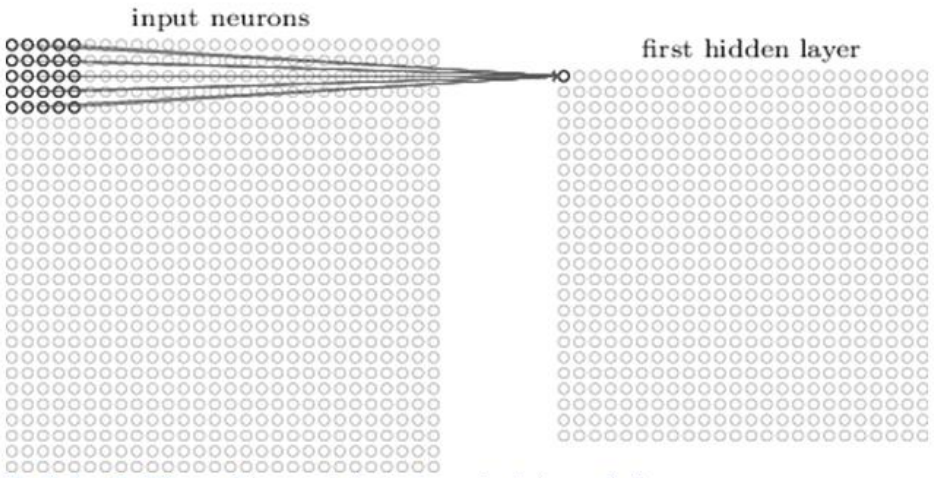
What We See

What Computers See

Convolutional layer

The input to the convolutional layer is a 24 x 24 x 3 array of pixel values. The depth of this filter is given to be the same as the depth of the input. I.e. 3. So, the dimensions of this filter are 3 x 3 x 3. As the filter is convolving around the input image; it is multiplying the values in the filter with the original pixel values of the image (element wise multiplications). These multiplications are all summed up. Every unique location on the input volume produces a number. The output of this convolution layer is an activation map. This activation map becomes a stack of filtered images. The filters in the first conv layer are designed to detect low level features such as edges and curves. we need the network to be able to recognize higher level features. we go through another conv layer; the output of the first conv layer becomes the input of the 2nd conv layer. When we were talking about the first layer, the input was just the original image. However, when we're talking about the 2nd conv layer, the input is the activation maps that result from the first layer. So, each layer of the input is basically describing the locations in the original image for where certain low-

level features appear. Now, when we apply a set of filters on top of that, the output will be activations that represent higher level features. As we go through the network and go through more conv layers, we get activation maps that represent more and more complex features.



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

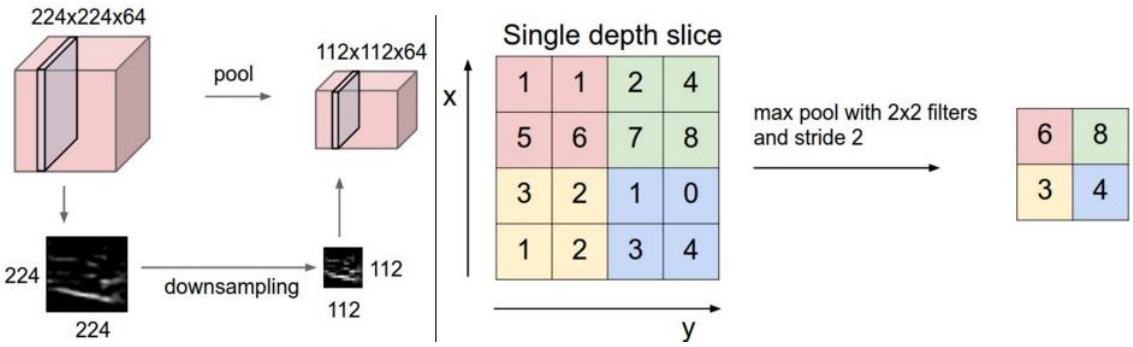
A classic CNN architecture would look like the image below.

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool ->Fully Connected

Pooling layer

In a convolution neural network model, the pooling layer, is used to reduce the spatial dimensions without reducing the depth. We choose to apply a pooling layer after ReLU layers. It is also referred to as a down sampling layer. By having less spatial dimensions, we gain computation performance. It also means less parameters and less chance to over-fit the data. The most common type of pooling is the max-pooling, which slides a window, like a normal convolution, and get the biggest value on the window as the output. Other types of pooling

are min-pooling and average-pooling. An example of the pooling layer is shown below.



Normalization layer

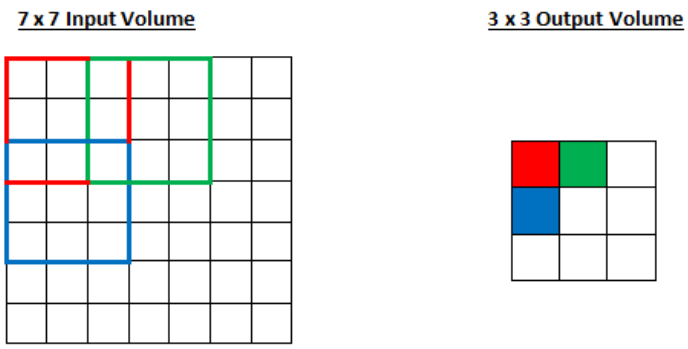
The normalization layer normalizes each input channel across a mini-batch. To speed up training of convolutional neural networks and reduce the sensitivity to network initialization, we used batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers. The layer first normalizes the activations of each channel by subtracting the mini-batch mean and dividing by the mini-batch standard deviation.

ReLU layer

ReLU is the most commonly used activation function in CNNs. ReLU is linear for all positive values, and zero for all negative values. We use ReLU layer because It's cheap to compute as there is no complicated math. The model can therefore take less time to train. Linearity means the slope converges faster. It doesn't have the vanishing gradient problem suffered by other activation functions. Since ReLU is zero for all negative inputs, it's likely for any given unit to not activate at all.

Stride and Padding

These are the two main parameters that can be changed to modify the behavior of each layer. After we choose the filter size, we also must choose the stride and the padding. The amount by which the filter shifts is the stride. Stride controls how the filter convolves around the input image. If the given stride is one, the filter convolves around the input volume by shifting one unit at a time. Stride is normally set in a way so that the output volume is an integer and not a fraction.

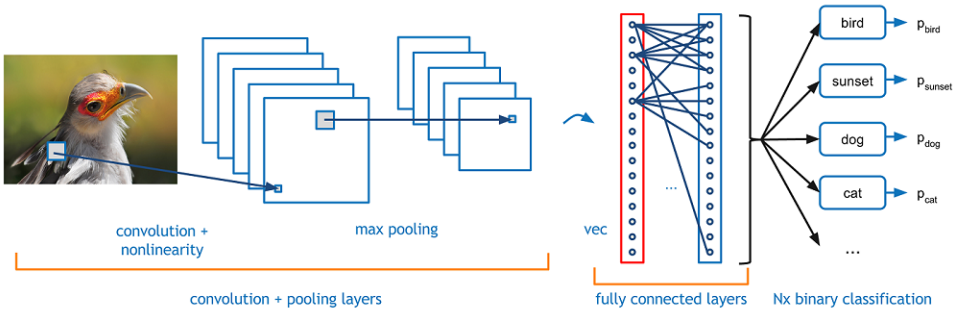


It's an additional layer that we can add to the border of an image. Zero-padding refers to the process of symmetrically adding zeroes to the input matrix. It's a commonly used modification that allows the size of the input to be adjusted to our requirement. It is mostly used in designing the CNN layers when the dimensions of the input volume need to be preserved in the output volume. Zero-padding is shown in the image below.

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

The Fully Connected Layer

The Fully Connected layer is configured exactly the way its name implies: it is fully connected with the output of the previous layer. Fully-connected layers are typically used in the last stages of the CNN to connect to the output layer and construct the desired number of outputs. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.



DATASET DESCRIPTION

The dataset for this project is small window images extracted from actual acquired data. Each dataset is composed of a .tiff images stack and a .csv file containing a description of the images stack. An images stack is a succession of 24 x 24 pixels gray scale images extracted from the actual x-ray projection using a sliding window. Several x-ray doses have been used to acquire the images resulting in images with various noise levels. Datasets are approximately composed of 48% of images without any spots, 48% of images with one spot and the last 4% are images containing 2 spots. It contains 209,933 images for training and 144,994.

The scanned object is a calibration chart composed of 61 radiopaque steel balls. The nature and geometry of this object are perfectly known, and the aim is to accurately determine the center position of the steel balls in the images. This information is then used to calibrate the acquisition device. In the future, the final goal of this operation is to bring patient 3D image reconstruction inside the medical operation rooms, based on acquisition machines such as the mobile C-Arm (2D radiology images) that are less expensive and more practical than other larger machines.

EXPERIMENTATION

Our data set is a single multi-stack tiff file in which there were 209933 images which was difficult to load on to matlab because by default matlab recognize it as a single image reading the first image. So, we used the above code to load the training data, iterate over each frame and read and save image to cell in matrix format. Then we loaded the file where we saved the images in matrix format and using for loop, we saved each image as seperate tiff files. For this project explanation we are taking a part of training set, i.e., 30000 images. Then we load every image in the folder to an image datastore. Since matlab read image files in an order like image1, image10, image100, image1000, image1001, image1002…. we used a function created by Stephen Cobeldickto to read images in the order we require. (ie, image1, image2, image3….). Since we loaded every image into imagedatastore without labels, we add labels from the training data description given. Now we have image data store with both file locations and labels. Now we split imagedatastore into training and validation data sets. For that we used splitEachLabel. We split such that 70 percent of each label into training and 30 percent to validation. Now we define the training network layers. We tried different layers. We used max pooling.

In the training options we specified the different training options to see the changes it makes in accuracy and loss. Then we trained the neural network using training data. After training, the images in validation data store are classified using our trained network and saved the labels. Comparing the predicted and known labels we find the accuracy. The following training options were changed in each iteration to see which combination is best for our dataset:

Optimizer: We trained model using three different optimizers to see the impact in our dataset. They are 1) stochastic gradient descent with momentum (SGDM) optimizer, 2) ‘adam’ and ‘rmspro’.

Learning rate: There is no best learning rate, the learning rate must be found by experimenting. If the learning rate is too low, then training takes a long time. If the learning rate is too high, then training might reach a suboptimal result or diverge. We will experiment with different learning rates and pick the one which gives the maximum accuracy and minimum loss.

The following training options were not altered:

Execution environment: Trained on GPU. Even a weak GPU is far better than fast CPU.

Epochs: An epoch is the full pass of the training algorithm over the entire training set. Since the data set is very large

Shuffle: This option is for data shuffling, there are three options: 1) 'once' — Shuffle the training and validation data once before training .2) 'never' — Do not shuffle the data. 3) 'every-epoch' — Shuffle the training data before each training epoch and shuffle the validation data before each network validation. To avoid discarding the same data every epoch, we set the 'Shuffle' value to 'every-epoch'.

In order to find which optimizer and learning rate is best for our model, we trained using layers and options that were used for MNIST example in MATLAB, changing the optimizer and learning rate and obtained the following results.

EXPERIMENTATION RESULTS

After training using the different learning rate, optimizer combinations, this is what we got:

	adam optimizer		sgdm optimizer		rmspro optimizer	
Learning rate	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
.01	97.46	.0923	97.69	.0856	96.58	.0815
.001	98.39	.0651	96.86	.1226	98.14	.1056
.0001	95.24	.1472	88.88	.2740	95.38	.1324

From the summary of experiments ran in the above table, we can see that for all the optimizers, learning rate of .001 is the best. Among the optimizers adam optimizer is the showing more accuracy and less loss. Also, from the graph we can see that experiments ran using rampro and sgdm optimizers shown some peaks and drops. So, from the nine experiments we ran, the best learning rate observed is 0.001 and optimizer is adam.

Now we have come with a set of training options that would be the best for our model to produce good results. Next, we will experiment on how many convolutional layers our model should have.

No. of convolutional layers	Accuracy	Loss
1	86.78	.3326
2	95.39	.1292
3	98.39	.0651
4	98	.0478
5	98.89	.0403

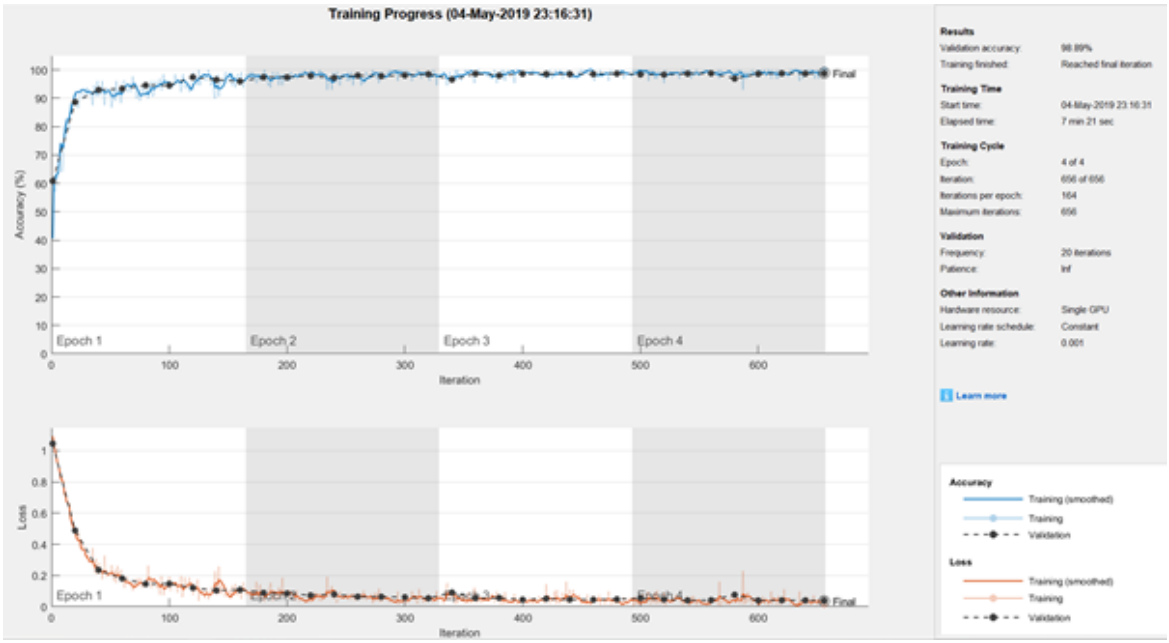
From the above result we can see that three or more convolutional layers give good accuracy. But the validation loss is decreased in the 5^{th} experiment. Also, during training with 4 and 5 layers many of the minibatch validation accuracy reached 100% accuracy.

After trying various training options, the best results the we obtained is validation accuracy 98.89% and loss 0.0403. This was for the following configuration.

```
layers = [
    imageInputLayer([24 24 3])
    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(2,16,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(3,'Stride',2)
    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding','same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,128,'Padding','same')
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(3)
    softmaxLayer
    classificationLayer];
```

```
options = trainingOptions('adam', ...
    'ExecutionEnvironment','multi-gpu', ...
    'InitialLearnRate',0.001, ...
    'MaxEpochs',4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',20, ...
    'Verbose',true, ...
    'Plots','training-progress');
```

The training graph and tabular result is shown below.



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	40.63%	60.82%	1.0921	1.0455	0.0010
1	20	00:00:17	91.41%	88.62%	0.5066	0.4887	0.0010
1	40	00:00:28	90.63%	92.93%	0.2787	0.2351	0.0010
1	50	00:00:30	92.19%		0.2075		0.0010
1	60	00:00:39	94.53%	93.33%	0.1551	0.1813	0.0010
1	80	00:00:50	93.75%	94.57%	0.1572	0.1460	0.0010
1	100	00:01:00	92.97%	94.52%	0.1934	0.1468	0.0010
1	120	00:01:11	99.22%	97.41%	0.0762	0.1213	0.0010
1	140	00:01:22	96.09%	96.50%	0.1316	0.1038	0.0010
1	150	00:01:23	96.88%		0.0804		0.0010
1	160	00:01:33	95.31%	95.94%	0.1120	0.1081	0.0010
2	180	00:01:44	99.22%	97.38%	0.0367	0.0885	0.0010
2	200	00:01:54	97.66%	97.37%	0.0696	0.0846	0.0010
2	220	00:02:09	100.00%	97.88%	0.0262	0.0724	0.0010
2	240	00:02:23	96.09%	97.19%	0.1032	0.0791	0.0010
2	250	00:02:26	98.44%		0.0603		0.0010
2	260	00:02:38	97.66%	98.01%	0.0824	0.0640	0.0010
2	280	00:02:52	98.44%	97.80%	0.0554	0.0634	0.0010
2	300	00:03:07	96.88%	98.07%	0.0632	0.0604	0.0010
2	320	00:03:20	97.66%	98.46%	0.0765	0.0545	0.0010
3	340	00:03:35	96.88%	96.57%	0.0828	0.0913	0.0010
3	350	00:03:37	97.66%		0.0660		0.0010
3	360	00:03:49	97.66%	98.61%	0.0620	0.0549	0.0010
3	380	00:04:03	98.44%	98.00%	0.0270	0.0564	0.0010
3	400	00:04:17	99.22%	98.59%	0.0331	0.0449	0.0010
3	420	00:04:31	96.88%	98.52%	0.1452	0.0516	0.0010
3	440	00:04:45	96.88%	98.51%	0.1095	0.0462	0.0010
3	450	00:04:47	100.00%		0.0164		0.0010
3	460	00:05:00	96.88%	98.51%	0.0696	0.0451	0.0010
3	480	00:05:14	100.00%	98.66%	0.0113	0.0477	0.0010
4	500	00:05:28	99.22%	98.38%	0.0224	0.0551	0.0010
4	520	00:05:42	95.31%	98.29%	0.0930	0.0468	0.0010
4	540	00:05:56	97.66%	98.64%	0.0470	0.0407	0.0010
4	550	00:05:59	97.66%		0.0484		0.0010
4	560	00:06:10	99.22%	98.74%	0.0417	0.0420	0.0010
4	580	00:06:25	99.22%	96.91%	0.0232	0.0765	0.0010
4	600	00:06:39	99.22%	98.57%	0.0260	0.0403	0.0010
4	620	00:06:53	99.22%	98.73%	0.0534	0.0413	0.0010
4	640	00:07:07	100.00%	98.66%	0.0191	0.0409	0.0010
4	650	00:07:09	100.00%		0.0100		0.0010
4	656	00:07:21	100.00%	98.68%	0.0113	0.0403	0.0010

```
accuracy =

    0.9889
```

CONCLUSION

We tried the classification in python and matlab with different options in CNN as mentioned in the experimentation section. We had a highest accuracy of 98.89%.

Challenges:

The dataset in the form of ***.tiff** format. Extracting the images from this file format proved to be a challenge as Matlab was not able to go beyond the index of 65536 initially. Considerable amount of time and methods were tried to overcome this. Other than experimentation, other part that took time is the experimentation with different optimization algorithms, choosing the number of layers and layers options. A number of iterations are done to ensure results consistency.

Future scope:

The scope of the project limits us to fining the number of spots on the x-ray images. More work can be done on localizing the spots and finding their dimensions to have a clear picture of what's happening with the patient and realize how serious the circumstances are. This helps in a more timely and accurate treatment of the specific condition.

REFERENCES

1. Dr. Wang's in class content and lecture slides.
2. Matlab R2018b documentation.
3. Beginner's Guide to Understanding Convolutional Neural Networks. https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
4. Geeks for geeks ML repository. https://www.geeksforgeeks.org/machine-learning/
5. Pooling Layer. https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/pooling_layer.html
6. An intuitive guide to Convolutional Neural Networks. https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050
7. https://www.mathworks.com/matlabcentral/fileexchange/47434-natural-order-filename-sort
8. Python Library guide documentation

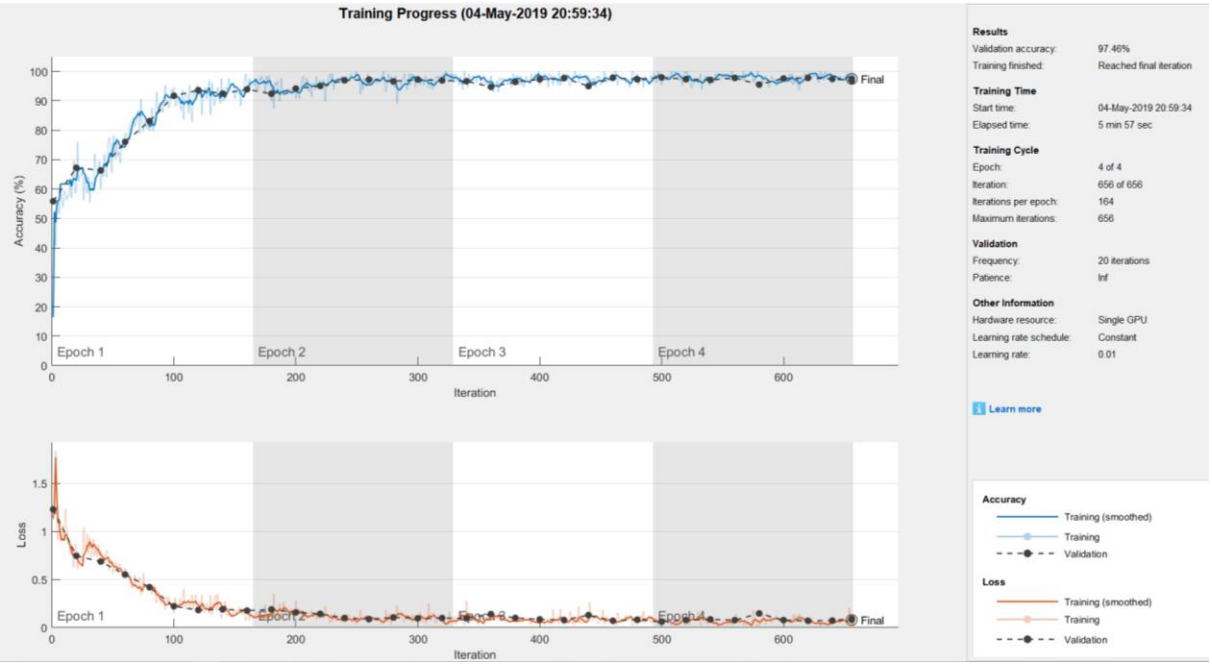
APPENDICES

Appendix A

Graphs and Results:

The following are the results of experiments mentioned in the experimentation summary.

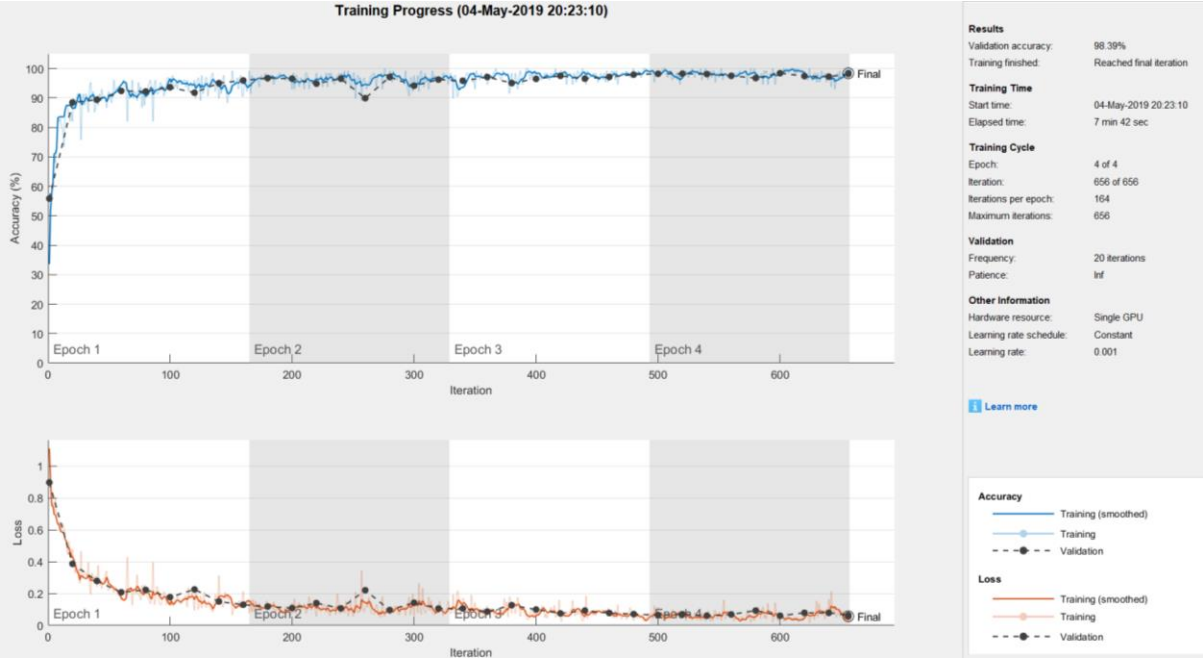
1) Optimizer: adam learning rate: 0.01



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	16.41%	55.84%	1.1341	1.2308	0.0100
1	20	00:00:17	59.38%	67.19%	0.8560	0.7466	0.0100
1	40	00:00:27	60.16%	66.34%	0.7985	0.6874	0.0100
1	50	00:00:28	77.34%		0.5666		0.0100
1	60	00:00:37	78.91%	76.02%	0.6196	0.5523	0.0100
1	80	00:00:48	79.69%	83.07%	0.4355	0.4200	0.0100
1	100	00:00:59	94.53%	91.69%	0.1505	0.2218	0.0100
1	120	00:01:09	92.97%	93.58%	0.2235	0.1827	0.0100
1	140	00:01:20	89.84%	92.46%	0.2210	0.1904	0.0100
1	150	00:01:21	90.63%		0.1803		0.0100
1	160	00:01:30	95.31%	93.89%	0.1414	0.1765	0.0100
2	180	00:01:41	89.84%	92.38%	0.2189	0.1887	0.0100
2	200	00:01:52	90.63%	94.12%	0.2855	0.1578	0.0100
2	220	00:02:04	96.88%	95.07%	0.0974	0.1427	0.0100
2	240	00:02:15	96.09%	96.94%	0.0999	0.0996	0.0100
2	250	00:02:16	99.22%		0.0483		0.0100
2	260	00:02:25	95.31%	97.19%	0.1223	0.0880	0.0100
2	280	00:02:36	96.09%	96.59%	0.0763	0.1052	0.0100
2	300	00:02:47	96.88%	97.22%	0.1137	0.0964	0.0100
2	320	00:02:57	96.09%	96.91%	0.0921	0.0987	0.0100
3	340	00:03:08	96.09%	96.64%	0.2821	0.0967	0.0100
3	350	00:03:09	98.44%		0.0606		0.0100
3	360	00:03:18	98.44%	94.70%	0.0686	0.1395	0.0100
3	380	00:03:29	96.88%	96.41%	0.0932	0.1002	0.0100
3	400	00:03:39	95.31%	97.28%	0.0880	0.0852	0.0100
3	420	00:03:50	99.22%	97.70%	0.0388	0.0782	0.0100
3	440	00:04:00	98.44%	94.99%	0.0901	0.1301	0.0100
3	450	00:04:02	95.31%		0.0921		0.0100
3	460	00:04:11	96.88%	97.81%	0.0790	0.0711	0.0100
3	480	00:04:22	99.22%	97.26%	0.0241	0.0826	0.0100
4	500	00:04:32	95.31%	97.92%	0.1345	0.0612	0.0100
4	520	00:04:43	99.22%	97.31%	0.1231	0.0788	0.0100
4	540	00:04:54	97.66%	97.07%	0.0489	0.0848	0.0100
4	550	00:04:55	98.44%		0.0619		0.0100
4	560	00:05:05	96.88%	97.76%	0.0654	0.0766	0.0100
4	580	00:05:16	98.44%	95.50%	0.0521	0.1470	0.0100
4	600	00:05:26	97.66%	97.58%	0.0468	0.0776	0.0100
4	620	00:05:37	98.44%	97.76%	0.0545	0.0707	0.0100
4	640	00:05:47	96.09%	97.42%	0.0809	0.0727	0.0100
4	650	00:05:49	98.44%		0.0570		0.0100
4	656	00:05:57	98.44%	96.88%	0.0386	0.0923	0.0100

```
accuracy =

    0.9746
```

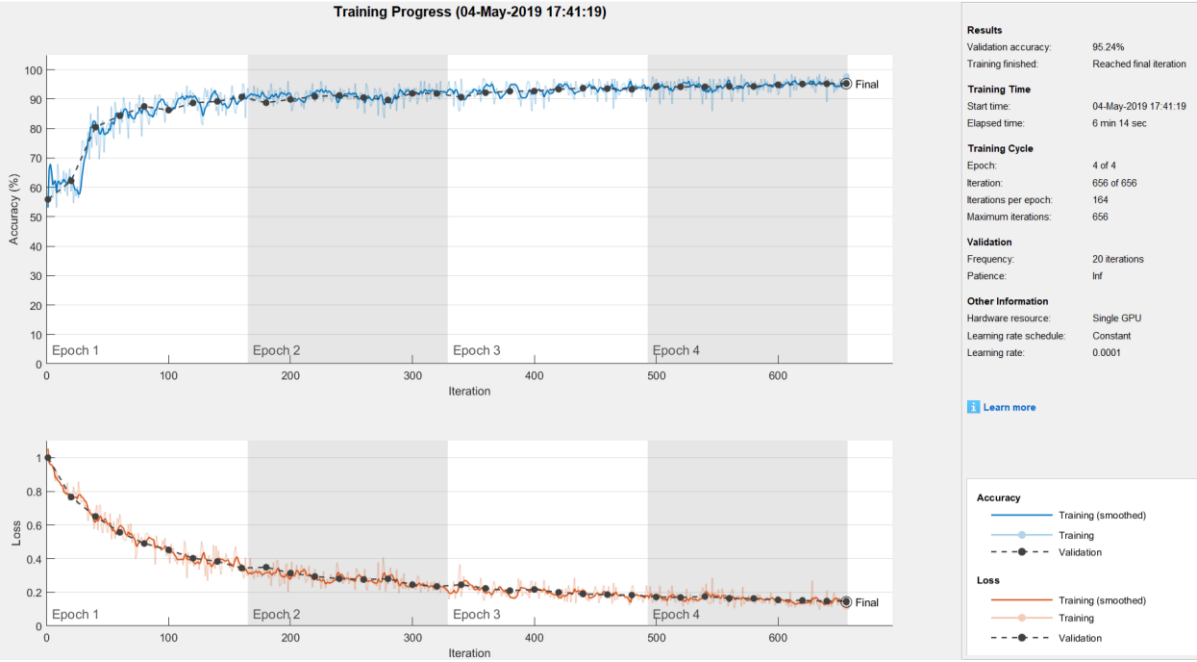
2) Optimizer: adam learning rate: 0.001



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:06	33.59%	55.84%	1.1099	0.8972	0.0010
1	20	00:00:16	82.03%	88.48%	0.4806	0.3866	0.0010
1	40	00:00:26	90.63%	89.38%	0.2441	0.2792	0.0010
1	50	00:00:28	92.19%		0.2249		0.0010
1	60	00:00:37	94.53%	92.37%	0.1601	0.2080	0.0010
1	80	00:00:47	91.41%	92.19%	0.2226	0.2228	0.0010
1	100	00:00:58	96.09%	93.61%	0.1402	0.1772	0.0010
1	120	00:01:08	96.09%	91.78%	0.0944	0.2257	0.0010
1	140	00:01:19	89.84%	95.02%	0.3133	0.1502	0.0010
1	150	00:01:20	93.75%		0.1728		0.0010
1	160	00:01:30	90.63%	95.94%	0.2136	0.1300	0.0010
2	180	00:01:44	95.31%	96.73%	0.1347	0.1185	0.0010
2	200	00:02:00	92.97%	96.53%	0.1382	0.1087	0.0010
2	220	00:02:18	98.44%	94.86%	0.0561	0.1399	0.0010
2	240	00:02:35	97.66%	96.52%	0.0852	0.1068	0.0010
2	250	00:02:37	98.44%		0.0545		0.0010
2	260	00:02:50	95.31%	89.91%	0.1083	0.2206	0.0010
2	280	00:03:04	99.22%	97.09%	0.0774	0.0964	0.0010
2	300	00:03:18	97.66%	94.14%	0.0606	0.1422	0.0010
2	320	00:03:32	96.09%	96.23%	0.0937	0.1057	0.0010
3	340	00:03:46	94.53%	95.83%	0.1710	0.1064	0.0010
3	350	00:03:48	92.97%		0.1431		0.0010
3	360	00:04:00	94.53%	97.11%	0.1224	0.0870	0.0010
3	380	00:04:15	98.44%	95.06%	0.0766	0.1267	0.0010
3	400	00:04:29	97.66%	96.47%	0.0570	0.0998	0.0010
3	420	00:04:45	99.22%	97.49%	0.0367	0.0753	0.0010
3	440	00:04:59	98.44%	96.54%	0.0553	0.0934	0.0010
3	450	00:05:02	96.09%		0.0961		0.0010
3	460	00:05:14	96.88%	97.13%	0.0701	0.0778	0.0010
3	480	00:05:31	100.00%	97.91%	0.0298	0.0709	0.0010
4	500	00:05:46	98.44%	98.20%	0.1145	0.0647	0.0010
4	520	00:06:02	97.66%	98.26%	0.0602	0.0700	0.0010
4	540	00:06:16	98.44%	98.10%	0.0609	0.0611	0.0010
4	550	00:06:18	96.88%		0.0847		0.0010
4	560	00:06:31	97.66%	97.51%	0.0500	0.0688	0.0010
4	580	00:06:46	96.88%	96.73%	0.0820	0.0930	0.0010
4	600	00:07:00	97.66%	98.39%	0.0623	0.0607	0.0010
4	620	00:07:14	97.66%	97.41%	0.0811	0.0790	0.0010
4	640	00:07:28	98.44%	97.22%	0.1647	0.0786	0.0010
4	650	00:07:30	96.09%		0.0540		0.0010
4	656	00:07:42	99.22%	98.13%	0.0389	0.0651	0.0010

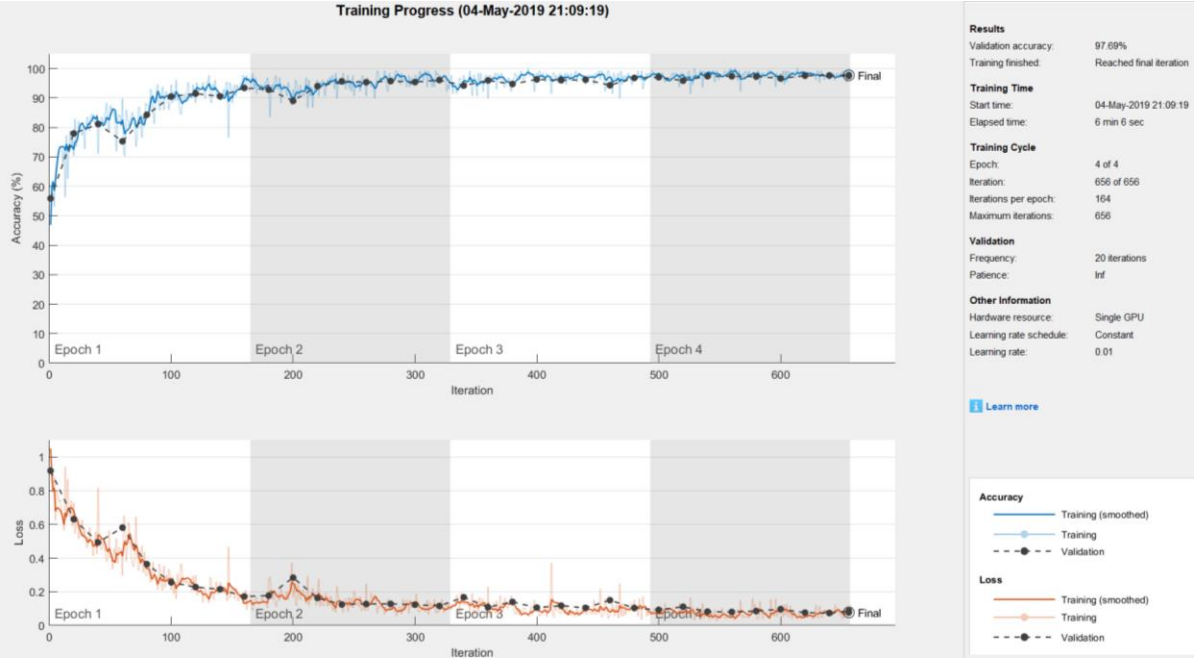
```
accuracy =

    0.9839
```

3) Optimizer: adam learning rate: 0.0001



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:06	53.13%	55.90%	1.0538	0.9999	1.0000e-04
1	20	00:00:16	58.59%	62.22%	0.7455	0.7662	1.0000e-04
1	40	00:00:26	76.56%	80.47%	0.6848	0.6505	1.0000e-04
1	50	00:00:27	81.25%		0.5542		1.0000e-04
1	60	00:00:36	78.13%	84.32%	0.6638	0.5561	1.0000e-04
1	80	00:00:46	86.72%	87.49%	0.4731	0.4899	1.0000e-04
1	100	00:00:56	85.94%	86.26%	0.4247	0.4521	1.0000e-04
1	120	00:01:06	92.19%	88.69%	0.3724	0.4023	1.0000e-04
1	140	00:01:16	87.50%	89.22%	0.3895	0.3830	1.0000e-04
1	150	00:01:17	89.84%		0.3433		1.0000e-04
1	160	00:01:26	91.41%	90.70%	0.3703	0.3437	1.0000e-04
2	180	00:01:37	89.06%	88.76%	0.3592	0.3487	1.0000e-04
2	200	00:01:48	90.63%	89.90%	0.2843	0.3140	1.0000e-04
2	220	00:01:58	89.06%	90.88%	0.3373	0.2938	1.0000e-04
2	240	00:02:08	92.19%	91.23%	0.2575	0.2808	1.0000e-04
2	250	00:02:09	90.63%		0.2811		1.0000e-04
2	260	00:02:18	87.50%	90.52%	0.2894	0.2756	1.0000e-04
2	280	00:02:28	91.41%	89.70%	0.2466	0.2799	1.0000e-04
2	300	00:02:38	92.19%	91.93%	0.2681	0.2447	1.0000e-04
2	320	00:02:48	95.31%	91.92%	0.2130	0.2343	1.0000e-04
3	340	00:02:59	96.88%	90.64%	0.1727	0.2445	1.0000e-04
3	350	00:03:00	93.75%		0.2121		1.0000e-04
3	360	00:03:10	95.31%	92.16%	0.1973	0.2229	1.0000e-04
3	380	00:03:20	95.31%	92.68%	0.1652	0.2095	1.0000e-04
3	400	00:03:30	92.19%	92.71%	0.2203	0.2165	1.0000e-04
3	420	00:03:41	95.31%	93.36%	0.1575	0.1995	1.0000e-04
3	440	00:03:51	92.19%	93.71%	0.2231	0.1897	1.0000e-04
3	450	00:03:52	91.41%		0.2353		1.0000e-04
3	460	00:04:01	96.09%	93.57%	0.1661	0.1861	1.0000e-04
3	480	00:04:12	90.63%	93.33%	0.2343	0.1831	1.0000e-04
4	500	00:04:22	96.88%	94.18%	0.1425	0.1718	1.0000e-04
4	520	00:04:33	94.53%	94.17%	0.1664	0.1688	1.0000e-04
4	540	00:04:47	95.31%	94.17%	0.1597	0.1748	1.0000e-04
4	550	00:04:49	95.31%		0.1705		1.0000e-04
4	560	00:05:00	94.53%	94.34%	0.1500	0.1638	1.0000e-04
4	580	00:05:16	92.97%	94.31%	0.1730	0.1631	1.0000e-04
4	600	00:05:31	91.41%	94.84%	0.1780	0.1543	1.0000e-04
4	620	00:05:46	93.75%	95.13%	0.1690	0.1512	1.0000e-04
4	640	00:06:01	89.06%	95.20%	0.2101	0.1475	1.0000e-04
4	650	00:06:03	93.75%		0.1673		1.0000e-04
4	656	00:06:14	97.66%	95.33%	0.1144	0.1472	1.0000e-04

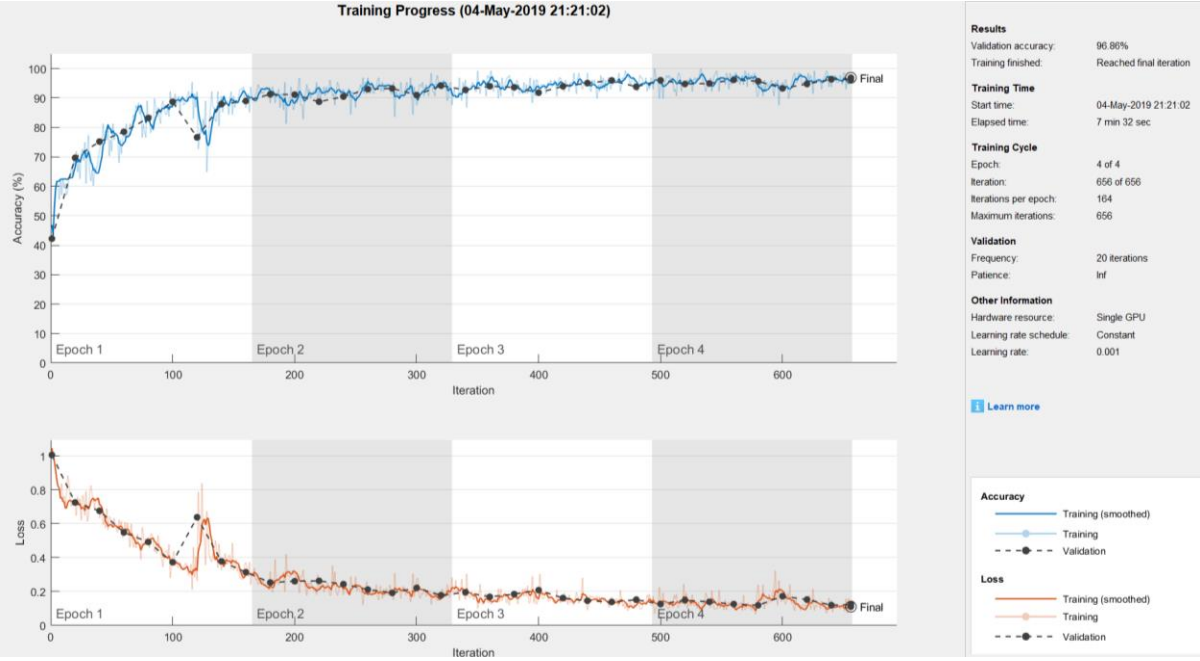
4) Optimizer: sgdm learning rate: 0.01



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	46.88%	55.87%	1.0511	0.9188	0.0100
1	20	00:00:17	70.31%	77.94%	0.7281	0.6311	0.0100
1	40	00:00:28	71.09%	81.14%	0.8171	0.4933	0.0100
1	50	00:00:30	87.50%		0.4026		0.0100
1	60	00:00:39	89.84%	75.29%	0.2957	0.5811	0.0100
1	80	00:00:50	87.50%	84.29%	0.2945	0.3637	0.0100
1	100	00:01:02	94.53%	90.48%	0.2088	0.2558	0.0100
1	120	00:01:13	88.28%	91.48%	0.3000	0.2276	0.0100
1	140	00:01:24	89.84%	90.56%	0.2569	0.2147	0.0100
1	150	00:01:25	88.28%		0.2238		0.0100
1	160	00:01:35	96.09%	93.40%	0.1341	0.1719	0.0100
2	180	00:01:46	94.53%	92.81%	0.2687	0.1768	0.0100
2	200	00:01:57	92.97%	89.00%	0.1690	0.2834	0.0100
2	220	00:02:08	96.09%	93.99%	0.1862	0.1628	0.0100
2	240	00:02:18	94.53%	95.62%	0.1177	0.1246	0.0100
2	250	00:02:20	90.63%		0.2035		0.0100
2	260	00:02:29	90.63%	95.32%	0.2474	0.1273	0.0100
2	280	00:02:40	96.09%	95.72%	0.1055	0.1280	0.0100
2	300	00:02:51	97.66%	95.37%	0.1075	0.1231	0.0100
2	320	00:03:02	96.09%	96.08%	0.0926	0.1156	0.0100
3	340	00:03:13	97.66%	94.19%	0.1208	0.1671	0.0100
3	350	00:03:14	93.75%		0.1467		0.0100
3	360	00:03:24	89.84%	95.96%	0.2282	0.1077	0.0100
3	380	00:03:36	96.09%	94.71%	0.1173	0.1395	0.0100
3	400	00:03:47	98.44%	96.30%	0.0908	0.1054	0.0100
3	420	00:03:58	97.66%	96.00%	0.0745	0.1175	0.0100
3	440	00:04:09	94.53%	96.20%	0.0956	0.1040	0.0100
3	450	00:04:11	97.66%		0.1031		0.0100
3	460	00:04:21	96.88%	94.30%	0.0957	0.1497	0.0100
3	480	00:04:32	96.88%	96.79%	0.1522	0.1036	0.0100
4	500	00:04:42	98.44%	97.04%	0.0734	0.0926	0.0100
4	520	00:04:53	98.44%	95.86%	0.0611	0.1107	0.0100
4	540	00:05:03	98.44%	97.39%	0.0628	0.0818	0.0100
4	550	00:05:05	97.66%		0.0492		0.0100
4	560	00:05:14	98.44%	97.36%	0.0518	0.0796	0.0100
4	580	00:05:24	97.66%	97.26%	0.0572	0.0852	0.0100
4	600	00:05:35	98.44%	96.63%	0.0655	0.0962	0.0100
4	620	00:05:45	98.44%	97.56%	0.0440	0.0759	0.0100
4	640	00:05:56	96.09%	97.62%	0.0909	0.0734	0.0100
4	650	00:05:57	96.88%		0.0540		0.0100
4	656	00:06:06	98.44%	97.44%	0.0998	0.0856	0.0100

```
accuracy =

    0.9769
```

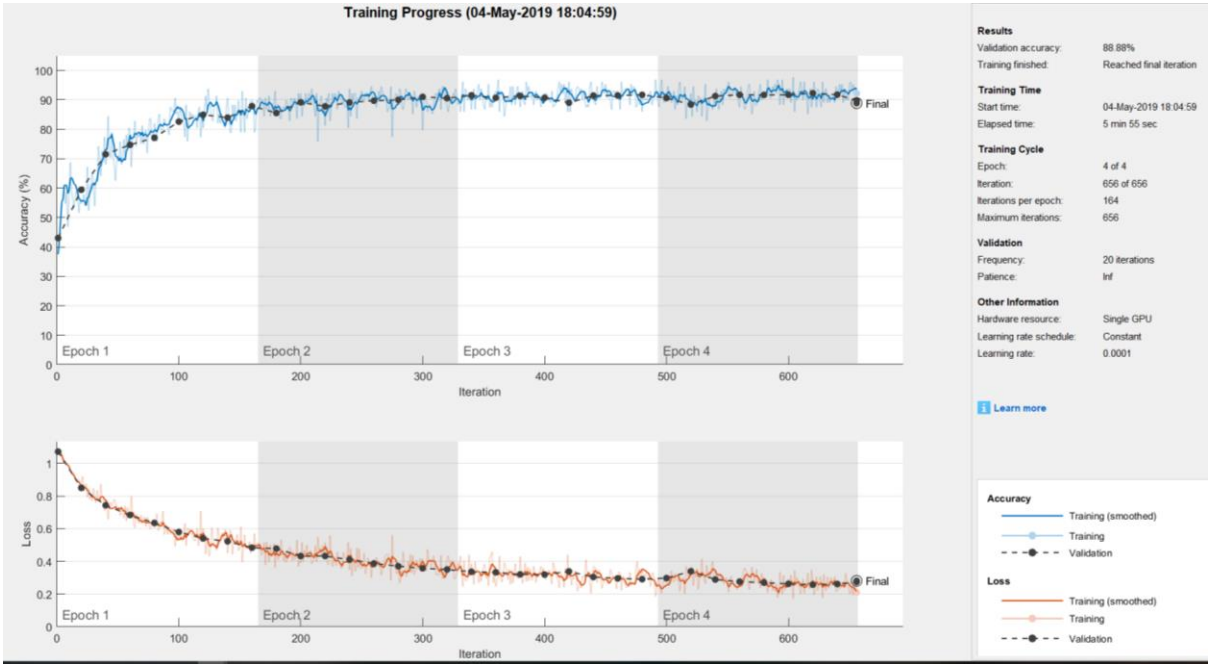
5) Optimizer: sgdm learning rate: 0.001



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	46.88%	42.21%	1.0458	1.0054	0.0010
1	20	00:00:19	67.19%	69.68%	0.7239	0.7253	0.0010
1	40	00:00:29	71.88%	75.17%	0.6332	0.6759	0.0010
1	50	00:00:31	71.88%		0.6026		0.0010
1	60	00:00:40	77.34%	78.49%	0.5571	0.5486	0.0010
1	80	00:00:50	82.81%	83.17%	0.5195	0.4925	0.0010
1	100	00:01:01	86.72%	88.67%	0.4082	0.3729	0.0010
1	120	00:01:14	92.19%	76.61%	0.2632	0.6383	0.0010
1	140	00:01:27	90.63%	87.90%	0.3278	0.3780	0.0010
1	150	00:01:29	90.63%		0.3273		0.0010
1	160	00:01:41	86.72%	88.98%	0.3715	0.3142	0.0010
2	180	00:01:55	88.28%	91.26%	0.3179	0.2530	0.0010
2	200	00:02:08	87.50%	91.12%	0.3066	0.2609	0.0010
2	220	00:02:22	93.75%	88.71%	0.2150	0.2628	0.0010
2	240	00:02:36	92.97%	90.46%	0.1803	0.2434	0.0010
2	250	00:02:37	86.72%		0.2866		0.0010
2	260	00:02:49	96.88%	92.99%	0.1790	0.2120	0.0010
2	280	00:03:03	91.41%	93.22%	0.1856	0.1919	0.0010
2	300	00:03:17	87.50%	90.97%	0.2729	0.2210	0.0010
2	320	00:03:31	93.75%	94.18%	0.1903	0.1780	0.0010
3	340	00:03:44	96.09%	92.71%	0.1526	0.1954	0.0010
3	350	00:03:46	99.22%		0.0994		0.0010
3	360	00:03:58	96.88%	93.97%	0.1447	0.1687	0.0010
3	380	00:04:12	93.75%	93.60%	0.1979	0.1849	0.0010
3	400	00:04:26	92.19%	91.79%	0.1639	0.2082	0.0010
3	420	00:04:39	92.97%	93.96%	0.1626	0.1617	0.0010
3	440	00:04:53	98.44%	95.03%	0.0911	0.1456	0.0010
3	450	00:04:55	92.19%		0.1679		0.0010
3	460	00:05:07	94.53%	95.94%	0.1674	0.1385	0.0010
3	480	00:05:21	94.53%	93.78%	0.1419	0.1519	0.0010
4	500	00:05:37	94.53%	95.99%	0.1444	0.1260	0.0010
4	520	00:05:50	95.31%	94.74%	0.1172	0.1488	0.0010
4	540	00:06:06	95.31%	94.91%	0.1140	0.1392	0.0010
4	550	00:06:08	95.31%		0.1355		0.0010
4	560	00:06:20	96.09%	96.14%	0.1045	0.1259	0.0010
4	580	00:06:34	90.63%	95.67%	0.2392	0.1191	0.0010
4	600	00:06:48	95.31%	93.26%	0.1726	0.1726	0.0010
4	620	00:07:03	93.75%	94.73%	0.1608	0.1525	0.0010
4	640	00:07:17	93.75%	96.32%	0.1487	0.1196	0.0010
4	650	00:07:19	98.44%		0.0943		0.0010
4	656	00:07:32	96.88%	96.27%	0.1309	0.1226	0.0010

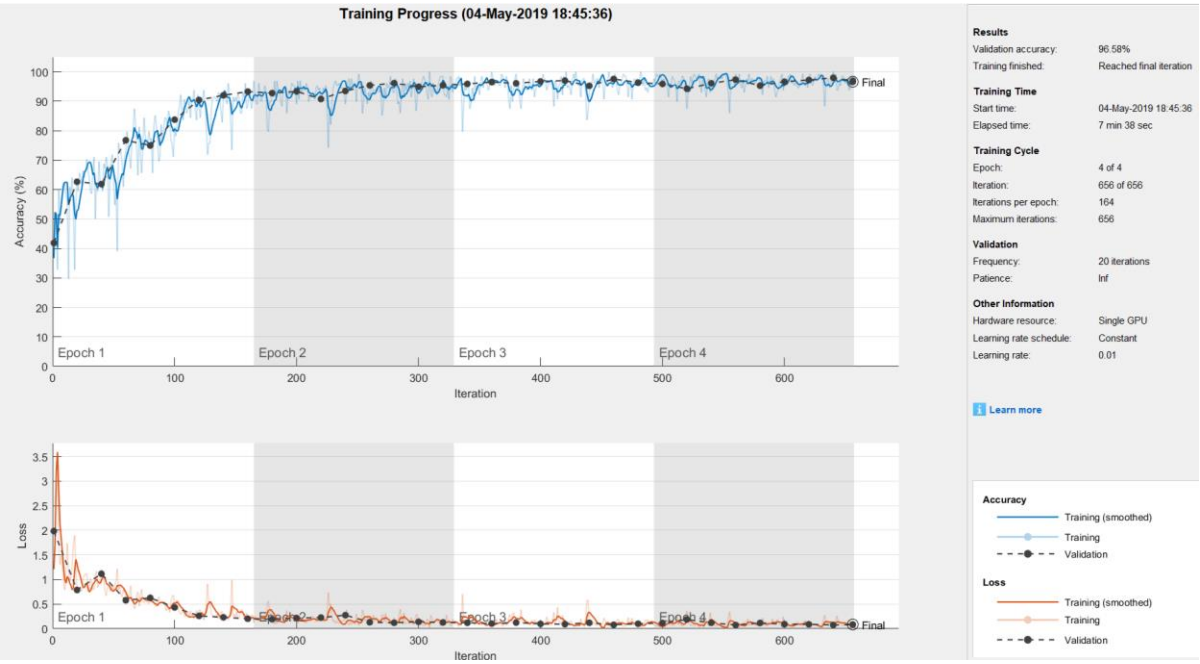
```
accuracy =

    0.9686
```

6) Optimizer: sgdm learning rate: 0.0001



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	37.50%	43.00%	1.0890	1.0720	1.0000e-04
1	20	00:00:17	54.69%	59.42%	0.8656	0.8498	1.0000e-04
1	40	00:00:27	67.97%	71.50%	0.7719	0.7434	1.0000e-04
1	50	00:00:29	71.09%		0.7078		1.0000e-04
1	60	00:00:38	79.69%	74.71%	0.6677	0.6838	1.0000e-04
1	80	00:00:49	82.03%	77.11%	0.6057	0.6350	1.0000e-04
1	100	00:01:00	82.03%	82.57%	0.5895	0.5796	1.0000e-04
1	120	00:01:11	80.47%	84.93%	0.6022	0.5407	1.0000e-04
1	140	00:01:23	90.63%	83.93%	0.4503	0.5223	1.0000e-04
1	150	00:01:24	84.38%		0.4926		1.0000e-04
1	160	00:01:34	89.84%	87.86%	0.5000	0.4837	1.0000e-04
2	180	00:01:46	85.16%	85.52%	0.5021	0.4779	1.0000e-04
2	200	00:01:56	92.19%	89.16%	0.3900	0.4325	1.0000e-04
2	220	00:02:08	81.25%	87.79%	0.5155	0.4328	1.0000e-04
2	240	00:02:20	93.75%	89.11%	0.3470	0.4103	1.0000e-04
2	250	00:02:22	92.97%		0.3562		1.0000e-04
2	260	00:02:30	89.84%	89.70%	0.4037	0.3875	1.0000e-04
2	280	00:02:41	89.06%	89.97%	0.3606	0.3704	1.0000e-04
2	300	00:02:51	79.69%	91.00%	0.4163	0.3574	1.0000e-04
2	320	00:03:01	87.50%	90.56%	0.4135	0.3501	1.0000e-04
3	340	00:03:12	86.72%	91.53%	0.3852	0.3364	1.0000e-04
3	350	00:03:13	88.28%		0.3699		1.0000e-04
3	360	00:03:23	92.97%	90.70%	0.2456	0.3328	1.0000e-04
3	380	00:03:33	87.50%	91.31%	0.4299	0.3202	1.0000e-04
3	400	00:03:43	92.19%	90.69%	0.3043	0.3191	1.0000e-04
3	420	00:03:53	96.09%	89.00%	0.2958	0.3384	1.0000e-04
3	440	00:04:03	89.84%	91.44%	0.3663	0.3042	1.0000e-04
3	450	00:04:05	95.31%		0.2812		1.0000e-04
3	460	00:04:14	92.97%	91.48%	0.2966	0.2957	1.0000e-04
3	480	00:04:24	85.94%	91.69%	0.3530	0.2906	1.0000e-04
4	500	00:04:34	90.63%	90.62%	0.2741	0.2975	1.0000e-04
4	520	00:04:45	92.19%	88.39%	0.3424	0.3393	1.0000e-04
4	540	00:04:55	89.06%	91.27%	0.2977	0.2887	1.0000e-04
4	550	00:04:56	94.53%		0.2530		1.0000e-04
4	560	00:05:05	87.50%	91.70%	0.2968	0.2759	1.0000e-04
4	580	00:05:15	91.41%	91.67%	0.3008	0.2696	1.0000e-04
4	600	00:05:26	90.63%	91.71%	0.2921	0.2624	1.0000e-04
4	620	00:05:36	91.41%	92.26%	0.2807	0.2575	1.0000e-04
4	640	00:05:46	93.75%	91.76%	0.2175	0.2615	1.0000e-04
4	650	00:05:47	95.31%		0.2003		1.0000e-04
4	656	00:05:55	92.19%	89.73%	0.2101	0.2740	1.0000e-04

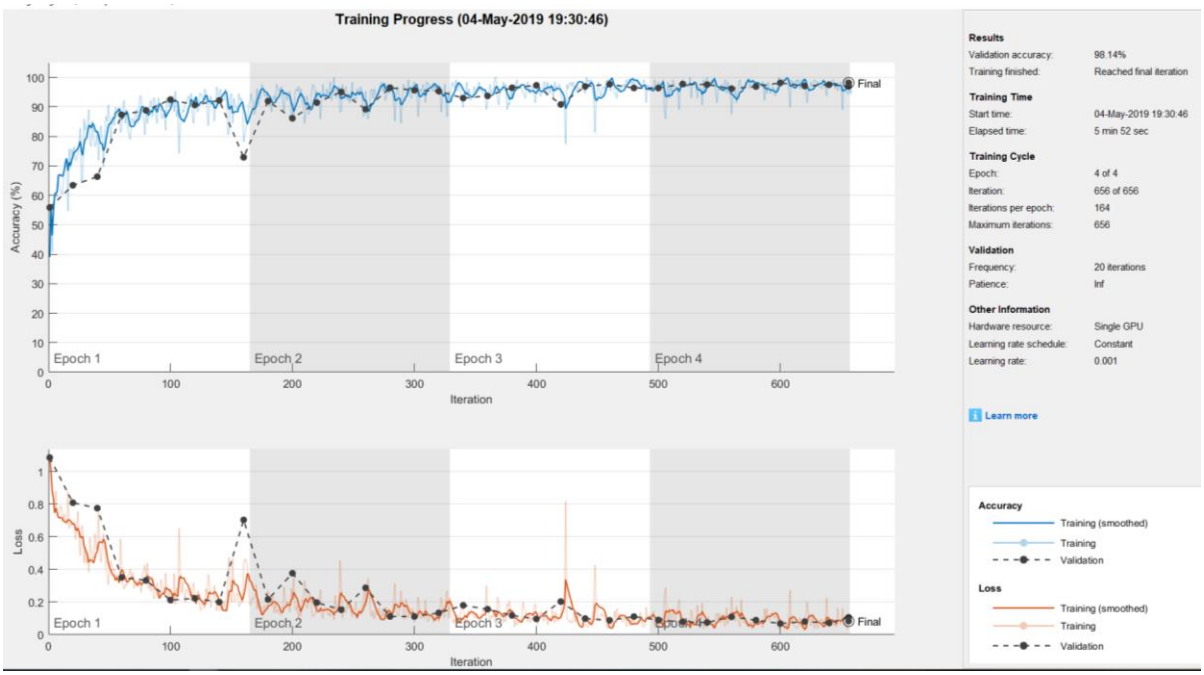
7) Optimizer: rmspro learning rate: 0.01



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	36.72%	41.88%	1.2092	1.9852	0.0100
1	20	00:00:18	63.28%	62.64%	0.7888	0.7870	0.0100
1	40	00:00:29	61.72%	61.87%	1.2031	1.1155	0.0100
1	50	00:00:31	62.50%		0.7805		0.0100
1	60	00:00:40	72.66%	76.74%	0.6734	0.5776	0.0100
1	80	00:00:50	76.56%	74.94%	0.5712	0.6267	0.0100
1	100	00:01:04	81.25%	83.73%	0.5146	0.4326	0.0100
1	120	00:01:17	89.06%	90.42%	0.3718	0.2598	0.0100
1	140	00:01:31	90.63%	91.99%	0.2616	0.2309	0.0100
1	150	00:01:33	91.41%		0.2428		0.0100
1	160	00:01:46	88.28%	93.27%	0.2420	0.2026	0.0100
2	180	00:02:01	93.75%	92.76%	0.1595	0.1922	0.0100
2	200	00:02:16	89.06%	93.31%	0.2312	0.2162	0.0100
2	220	00:02:30	90.63%	90.76%	0.2149	0.2256	0.0100
2	240	00:02:43	89.06%	93.48%	0.2187	0.2744	0.0100
2	250	00:02:45	96.09%		0.1320		0.0100
2	260	00:02:59	90.63%	95.37%	0.2254	0.1308	0.0100
2	280	00:03:15	92.97%	96.12%	0.2202	0.1201	0.0100
2	300	00:03:30	95.31%	94.89%	0.1338	0.1396	0.0100
2	320	00:03:43	98.44%	95.41%	0.0635	0.1272	0.0100
3	340	00:03:57	95.31%	95.88%	0.1043	0.1195	0.0100
3	350	00:03:59	92.19%		0.1911		0.0100
3	360	00:04:11	100.00%	96.49%	0.0485	0.1036	0.0100
3	380	00:04:25	99.22%	96.07%	0.0665	0.1237	0.0100
3	400	00:04:39	97.66%	96.70%	0.0692	0.0974	0.0100
3	420	00:04:53	95.31%	97.00%	0.0863	0.0914	0.0100
3	440	00:05:06	92.19%	95.21%	0.1884	0.1199	0.0100
3	450	00:05:08	96.88%		0.0758		0.0100
3	460	00:05:20	96.09%	97.53%	0.0875	0.0791	0.0100
3	480	00:05:33	96.09%	96.30%	0.0867	0.1005	0.0100
4	500	00:05:47	96.88%	95.87%	0.0924	0.1146	0.0100
4	520	00:06:01	87.50%	94.23%	0.3649	0.1870	0.0100
4	540	00:06:16	94.53%	96.08%	0.1979	0.1204	0.0100
4	550	00:06:18	99.22%		0.0608		0.0100
4	560	00:06:30	99.22%	97.31%	0.0362	0.0744	0.0100
4	580	00:06:43	96.88%	95.28%	0.1031	0.1169	0.0100
4	600	00:06:57	96.09%	96.62%	0.0736	0.0916	0.0100
4	620	00:07:11	97.66%	97.20%	0.0601	0.0890	0.0100
4	640	00:07:25	100.00%	97.94%	0.0313	0.0751	0.0100
4	650	00:07:27	96.09%		0.0742		0.0100
4	656	00:07:38	97.66%	96.89%	0.0715	0.0815	0.0100

```
accuracy =

    0.9658
```

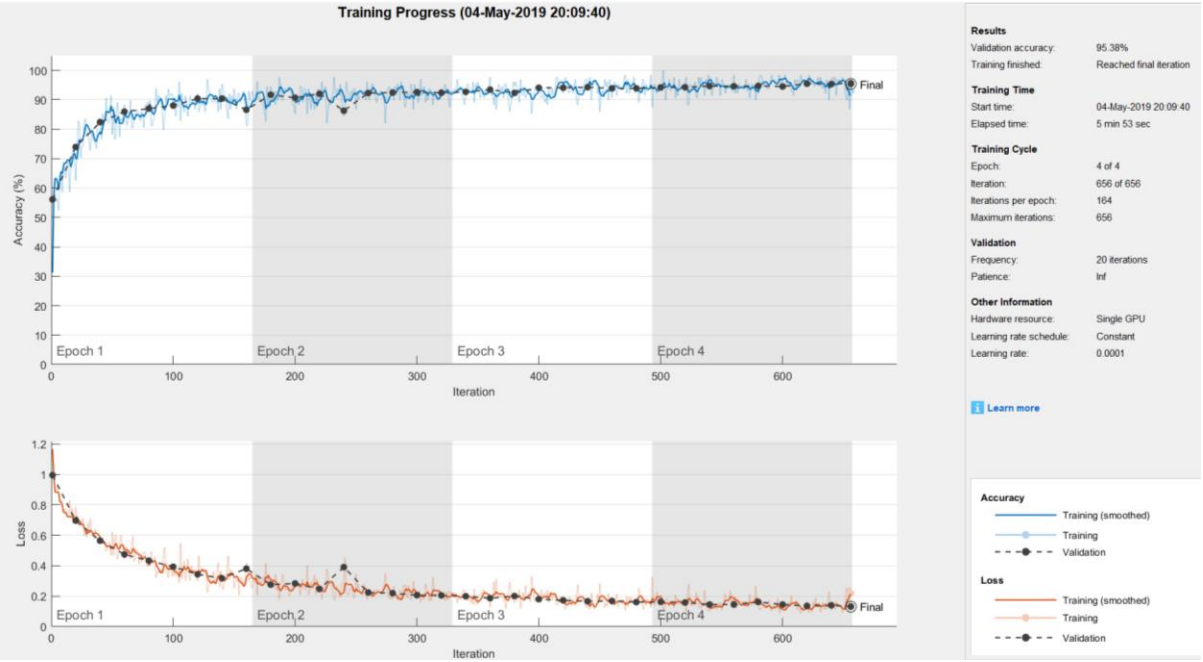
8) Optimizer: rmspro learning rate: 0.001



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	39.06%	55.84%	1.0735	1.0862	0.0010
1	20	00:00:18	73.44%	63.39%	0.6788	0.8092	0.0010
1	40	00:00:28	82.03%	66.31%	0.5481	0.7756	0.0010
1	50	00:00:29	85.94%		0.4014		0.0010
1	60	00:00:38	82.81%	87.31%	0.4080	0.3508	0.0010
1	80	00:00:49	89.84%	88.79%	0.3074	0.3330	0.0010
1	100	00:00:59	92.97%	92.46%	0.2080	0.2121	0.0010
1	120	00:01:10	92.19%	90.64%	0.2173	0.2227	0.0010
1	140	00:01:20	93.75%	92.20%	0.1592	0.1995	0.0010
1	150	00:01:22	94.53%		0.1978		0.0010
1	160	00:01:31	78.13%	72.89%	0.4301	0.7037	0.0010
2	180	00:01:41	90.63%	91.96%	0.2687	0.2164	0.0010
2	200	00:01:52	88.28%	86.11%	0.2199	0.3766	0.0010
2	220	00:02:02	95.31%	91.44%	0.1345	0.1967	0.0010
2	240	00:02:13	88.28%	95.07%	0.2946	0.1527	0.0010
2	250	00:02:14	92.19%		0.1546		0.0010
2	260	00:02:23	95.31%	89.19%	0.1747	0.2872	0.0010
2	280	00:02:33	96.09%	96.48%	0.1129	0.1115	0.0010
2	300	00:02:44	98.44%	95.64%	0.0750	0.1114	0.0010
2	320	00:02:54	96.88%	95.32%	0.0822	0.1335	0.0010
3	340	00:03:05	89.84%	92.98%	0.1857	0.1796	0.0010
3	350	00:03:06	94.53%		0.1247		0.0010
3	360	00:03:16	92.19%	93.73%	0.1582	0.1561	0.0010
3	380	00:03:26	92.97%	96.47%	0.1517	0.1182	0.0010
3	400	00:03:37	94.53%	97.29%	0.1245	0.0963	0.0010
3	420	00:03:47	94.53%	90.81%	0.1135	0.2029	0.0010
3	440	00:03:58	94.53%	96.92%	0.1163	0.0987	0.0010
3	450	00:03:59	96.88%		0.0922		0.0010
3	460	00:04:08	97.66%	97.68%	0.0657	0.0879	0.0010
3	480	00:04:19	96.88%	96.41%	0.0963	0.1106	0.0010
4	500	00:04:29	98.44%	96.39%	0.0432	0.0898	0.0010
4	520	00:04:40	99.22%	97.76%	0.0531	0.0787	0.0010
4	540	00:04:50	100.00%	97.56%	0.0443	0.0745	0.0010
4	550	00:04:52	96.88%		0.1310		0.0010
4	560	00:05:01	87.50%	96.18%	0.2320	0.1089	0.0010
4	580	00:05:11	97.66%	96.90%	0.0994	0.0888	0.0010
4	600	00:05:22	98.44%	98.14%	0.0739	0.0680	0.0010
4	620	00:05:32	99.22%	97.09%	0.0270	0.0801	0.0010
4	640	00:05:42	98.44%	97.52%	0.0493	0.0719	0.0010
4	650	00:05:44	96.09%		0.0876		0.0010
4	656	00:05:52	95.31%	96.98%	0.1085	0.1056	0.0010

```
accuracy =

    0.9814
```

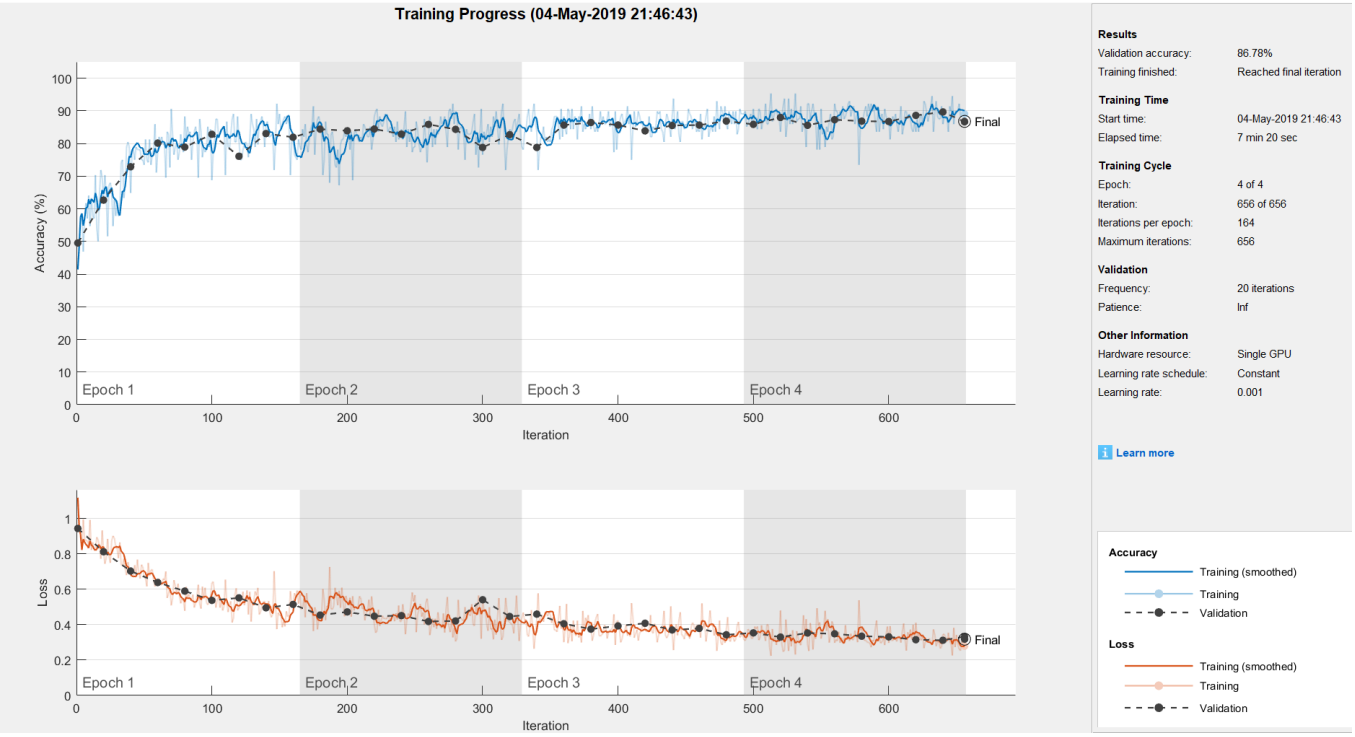
9) Optimizer: rmspro learning rate: 0.0001



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	31.25%	56.11%	1.1649	0.9956	1.0000e-04
1	20	00:00:16	62.50%	73.94%	0.7855	0.6970	1.0000e-04
1	40	00:00:27	82.81%	82.38%	0.5459	0.5646	1.0000e-04
1	50	00:00:28	78.13%		0.6230		1.0000e-04
1	60	00:00:37	88.28%	85.94%	0.4423	0.4747	1.0000e-04
1	80	00:00:47	88.28%	87.01%	0.3972	0.4333	1.0000e-04
1	100	00:00:58	89.06%	88.02%	0.3744	0.3934	1.0000e-04
1	120	00:01:08	92.19%	90.39%	0.2992	0.3438	1.0000e-04
1	140	00:01:19	95.31%	90.41%	0.2611	0.3175	1.0000e-04
1	150	00:01:20	93.75%		0.2506		1.0000e-04
1	160	00:01:29	81.25%	86.60%	0.4190	0.3807	1.0000e-04
2	180	00:01:40	94.53%	91.70%	0.2693	0.2761	1.0000e-04
2	200	00:01:50	85.16%	90.68%	0.2930	0.2834	1.0000e-04
2	220	00:02:00	92.19%	92.03%	0.2430	0.2478	1.0000e-04
2	240	00:02:11	84.38%	86.27%	0.2969	0.3909	1.0000e-04
2	250	00:02:13	86.72%		0.3464		1.0000e-04
2	260	00:02:22	90.63%	92.29%	0.2331	0.2241	1.0000e-04
2	280	00:02:32	93.75%	92.44%	0.1816	0.2197	1.0000e-04
2	300	00:02:43	91.41%	92.59%	0.2227	0.2065	1.0000e-04
2	320	00:02:53	93.75%	92.38%	0.1928	0.2052	1.0000e-04
3	340	00:03:04	91.41%	92.67%	0.2138	0.1992	1.0000e-04
3	350	00:03:05	96.09%		0.1465		1.0000e-04
3	360	00:03:14	97.66%	93.43%	0.1272	0.1868	1.0000e-04
3	380	00:03:24	94.53%	92.32%	0.1579	0.2005	1.0000e-04
3	400	00:03:35	95.31%	94.04%	0.1628	0.1801	1.0000e-04
3	420	00:03:46	92.19%	94.09%	0.1904	0.1728	1.0000e-04
3	440	00:03:57	92.97%	94.27%	0.1671	0.1665	1.0000e-04
3	450	00:03:58	85.94%		0.2824		1.0000e-04
3	460	00:04:08	90.63%	93.87%	0.1930	0.1680	1.0000e-04
3	480	00:04:18	93.75%	93.89%	0.1686	0.1611	1.0000e-04
4	500	00:04:29	94.53%	94.19%	0.2056	0.1627	1.0000e-04
4	520	00:04:39	96.09%	94.26%	0.1353	0.1584	1.0000e-04
4	540	00:04:50	97.66%	94.67%	0.1031	0.1448	1.0000e-04
4	550	00:04:51	96.88%		0.1070		1.0000e-04
4	560	00:05:01	98.44%	94.60%	0.1125	0.1455	1.0000e-04
4	580	00:05:11	92.19%	94.70%	0.1799	0.1618	1.0000e-04
4	600	00:05:22	96.88%	94.51%	0.1199	0.1450	1.0000e-04
4	620	00:05:32	94.53%	95.48%	0.1654	0.1356	1.0000e-04
4	640	00:05:43	96.88%	95.33%	0.1065	0.1398	1.0000e-04
4	650	00:05:44	96.88%		0.1170		1.0000e-04
4	656	00:05:53	92.97%	95.63%	0.2185	0.1324	1.0000e-04

```
accuracy =

    0.9538
```

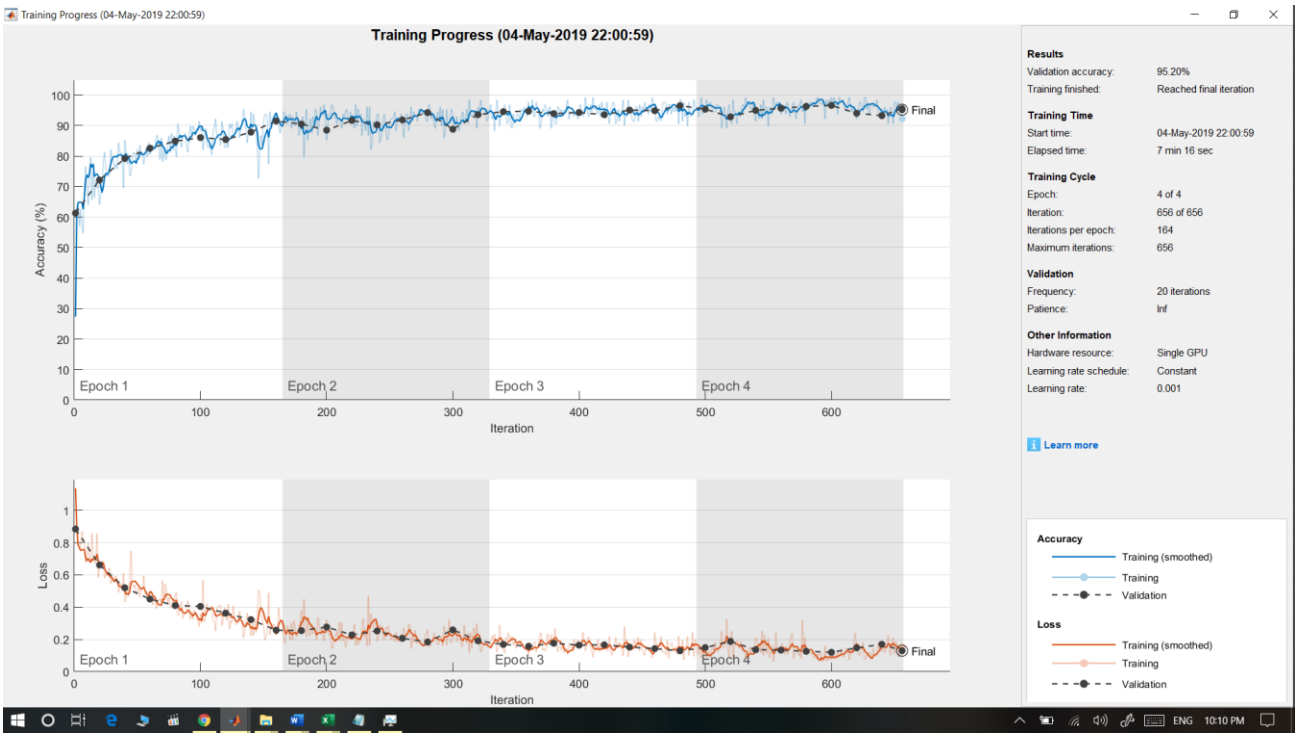
10) Using one convolutional2Dlayer:



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	41.41%	49.53%	1.1183	0.9443	0.0010
1	20	00:00:16	62.50%	62.67%	0.8154	0.8128	0.0010
1	40	00:00:26	64.84%	72.87%	0.7019	0.7038	0.0010
1	50	00:00:28	73.44%		0.6726		0.0010
1	60	00:00:37	77.34%	80.12%	0.6454	0.6391	0.0010
1	80	00:00:47	88.28%	78.90%	0.4788	0.5902	0.0010
1	100	00:00:57	68.75%	82.83%	0.6363	0.5378	0.0010
1	120	00:01:08	81.25%	76.09%	0.4916	0.5520	0.0010
1	140	00:01:20	85.94%	83.04%	0.4969	0.4957	0.0010
1	150	00:01:22	89.06%		0.4529		0.0010
1	160	00:01:34	82.03%	81.91%	0.5177	0.5149	0.0010
2	180	00:01:48	85.94%	84.40%	0.4467	0.4545	0.0010
2	200	00:02:04	83.59%	83.90%	0.5323	0.4725	0.0010
2	220	00:02:18	88.28%	84.44%	0.4010	0.4483	0.0010
2	240	00:02:33	84.38%	82.88%	0.4199	0.4515	0.0010
2	250	00:02:35	78.91%		0.5677		0.0010
2	260	00:02:47	82.03%	85.86%	0.4840	0.4186	0.0010
2	280	00:03:01	89.84%	84.34%	0.3059	0.4213	0.0010
2	300	00:03:15	76.56%	78.79%	0.5615	0.5412	0.0010
2	320	00:03:29	84.38%	82.70%	0.4170	0.4466	0.0010
3	340	00:03:44	83.59%	78.79%	0.4297	0.4601	0.0010
3	350	00:03:46	80.47%		0.4577		0.0010
3	360	00:03:59	88.28%	85.64%	0.3012	0.4056	0.0010
3	380	00:04:12	84.38%	86.42%	0.3844	0.3751	0.0010
3	400	00:04:26	84.38%	85.67%	0.4233	0.3935	0.0010
3	420	00:04:39	85.16%	83.87%	0.3765	0.4082	0.0010
3	440	00:04:53	88.28%	85.52%	0.3534	0.3704	0.0010
3	450	00:04:54	85.94%		0.3640		0.0010
3	460	00:05:06	83.59%	85.64%	0.4006	0.3792	0.0010
3	480	00:05:20	85.94%	86.87%	0.3749	0.3432	0.0010
4	500	00:05:34	90.63%	85.87%	0.3332	0.3541	0.0010
4	520	00:05:47	83.59%	87.98%	0.4805	0.3302	0.0010
4	540	00:06:00	92.19%	85.58%	0.2906	0.3533	0.0010
4	550	00:06:02	90.63%		0.3218		0.0010
4	560	00:06:14	92.19%	87.29%	0.2979	0.3489	0.0010
4	580	00:06:27	86.72%	86.84%	0.3784	0.3353	0.0010
4	600	00:06:41	87.50%	86.73%	0.2976	0.3315	0.0010
4	620	00:06:54	83.59%	88.61%	0.4112	0.3154	0.0010
4	640	00:07:07	85.94%	89.67%	0.3465	0.3122	0.0010
4	650	00:07:09	89.84%		0.3426		0.0010
4	656	00:07:20	89.06%	86.63%	0.2788	0.3326	0.0010

```
accuracy =

    0.8678
```

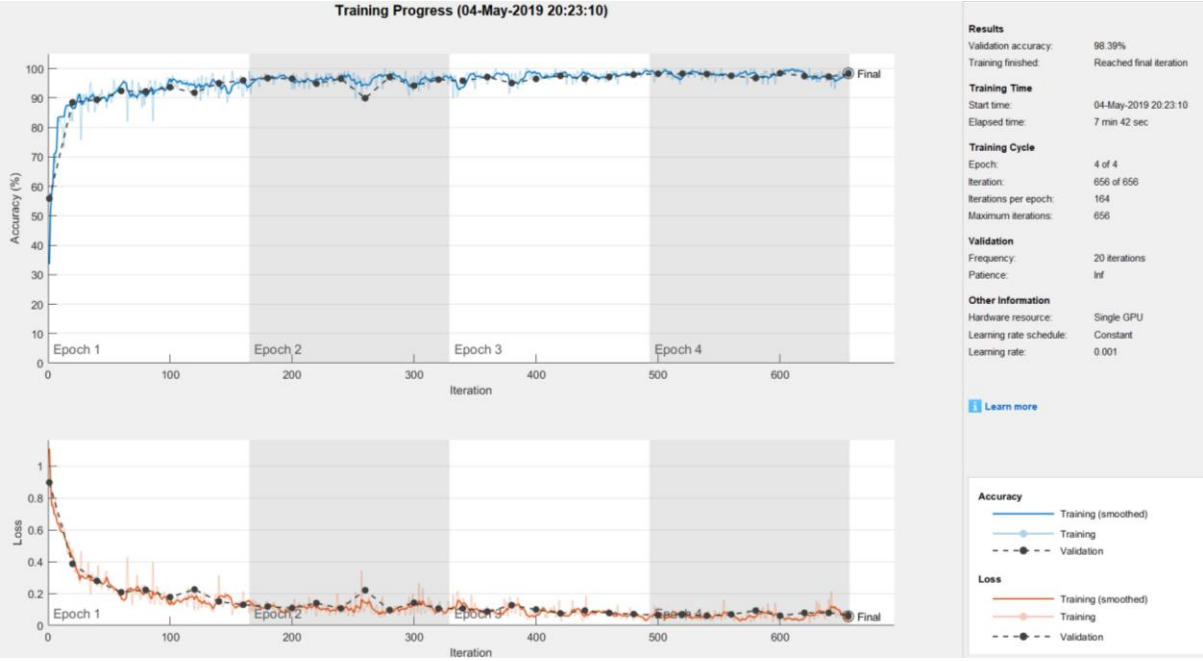
11) Using two convolutional2Dlayer:



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	27.34%	61.31%	1.1371	0.8839	0.0010
1	20	00:00:16	64.06%	72.14%	0.6549	0.6613	0.0010
1	40	00:00:26	85.16%	79.19%	0.4329	0.5202	0.0010
1	50	00:00:28	82.03%		0.4997		0.0010
1	60	00:00:37	80.47%	82.57%	0.4984	0.4501	0.0010
1	80	00:00:47	82.03%	84.89%	0.4835	0.4103	0.0010
1	100	00:00:57	88.28%	86.06%	0.3586	0.4044	0.0010
1	120	00:01:08	82.81%	85.42%	0.3730	0.3627	0.0010
1	140	00:01:19	94.53%	87.84%	0.2215	0.3225	0.0010
1	150	00:01:20	91.41%		0.2808		0.0010
1	160	00:01:32	93.75%	91.48%	0.2505	0.2570	0.0010
2	180	00:01:46	89.06%	90.49%	0.3792	0.2544	0.0010
2	200	00:02:00	85.16%	88.49%	0.3236	0.2766	0.0010
2	220	00:02:14	91.41%	91.69%	0.2480	0.2279	0.0010
2	240	00:02:27	93.75%	90.24%	0.1886	0.2524	0.0010
2	250	00:02:29	91.41%		0.2207		0.0010
2	260	00:02:41	91.41%	91.86%	0.2058	0.2068	0.0010
2	280	00:02:55	91.41%	94.24%	0.2576	0.1836	0.0010
2	300	00:03:09	93.75%	88.81%	0.2122	0.2592	0.0010
2	320	00:03:23	96.09%	93.58%	0.1799	0.1910	0.0010
3	340	00:03:36	95.31%	94.54%	0.1421	0.1688	0.0010
3	350	00:03:38	92.19%		0.1452		0.0010
3	360	00:03:50	97.66%	94.70%	0.0993	0.1573	0.0010
3	380	00:04:04	92.19%	93.92%	0.1876	0.1760	0.0010
3	400	00:04:18	95.31%	94.28%	0.1657	0.1642	0.0010
3	420	00:04:31	95.31%	93.58%	0.1661	0.1666	0.0010
3	440	00:04:45	95.31%	95.07%	0.1366	0.1523	0.0010
3	450	00:04:47	96.09%		0.1402		0.0010
3	460	00:04:59	96.88%	94.87%	0.2414	0.1425	0.0010
3	480	00:05:13	98.44%	96.53%	0.1167	0.1298	0.0010
4	500	00:05:26	94.53%	95.36%	0.1532	0.1492	0.0010
4	520	00:05:40	91.41%	92.86%	0.2170	0.1872	0.0010
4	540	00:05:53	91.41%	94.98%	0.2118	0.1360	0.0010
4	550	00:05:55	89.84%		0.2073		0.0010
4	560	00:06:07	93.75%	95.69%	0.1507	0.1336	0.0010
4	580	00:06:21	97.66%	96.28%	0.1037	0.1261	0.0010
4	600	00:06:34	99.22%	96.60%	0.0744	0.1198	0.0010
4	620	00:06:48	92.19%	94.03%	0.1526	0.1476	0.0010
4	640	00:07:02	95.31%	93.24%	0.1250	0.1695	0.0010
4	650	00:07:04	97.66%		0.1104		0.0010
4	656	00:07:16	92.19%	95.57%	0.1538	0.1292	0.0010

```
accuracy =

    0.9520
```

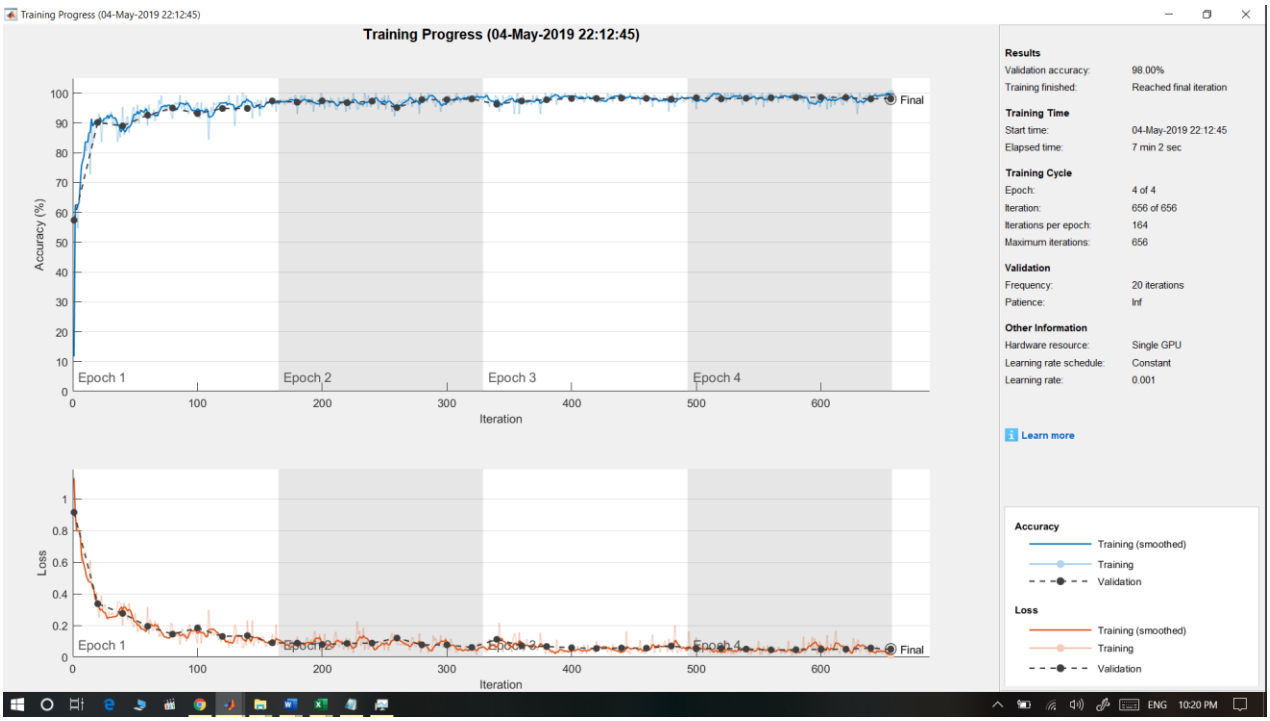
12) Using three convolutional2Dlayer:



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:06	33.59%	55.84%	1.1099	0.8972	0.0010
1	20	00:00:16	82.03%	88.48%	0.4806	0.3866	0.0010
1	40	00:00:26	90.63%	89.38%	0.2441	0.2792	0.0010
1	50	00:00:28	92.19%		0.2249		0.0010
1	60	00:00:37	94.53%	92.37%	0.1601	0.2080	0.0010
1	80	00:00:47	91.41%	92.19%	0.2226	0.2228	0.0010
1	100	00:00:58	96.09%	93.61%	0.1402	0.1772	0.0010
1	120	00:01:08	96.09%	91.78%	0.0944	0.2257	0.0010
1	140	00:01:19	89.84%	95.02%	0.3133	0.1502	0.0010
1	150	00:01:20	93.75%		0.1728		0.0010
1	160	00:01:30	90.63%	95.94%	0.2136	0.1300	0.0010
2	180	00:01:44	95.31%	96.73%	0.1347	0.1185	0.0010
2	200	00:02:00	92.97%	96.53%	0.1382	0.1087	0.0010
2	220	00:02:18	98.44%	94.86%	0.0561	0.1399	0.0010
2	240	00:02:35	97.66%	96.52%	0.0852	0.1068	0.0010
2	250	00:02:37	98.44%		0.0545		0.0010
2	260	00:02:50	95.31%	89.91%	0.1083	0.2206	0.0010
2	280	00:03:04	99.22%	97.09%	0.0774	0.0964	0.0010
2	300	00:03:18	97.66%	94.14%	0.0606	0.1422	0.0010
2	320	00:03:32	96.09%	96.23%	0.0937	0.1057	0.0010
3	340	00:03:46	94.53%	95.83%	0.1710	0.1064	0.0010
3	350	00:03:48	92.97%		0.1431		0.0010
3	360	00:04:00	94.53%	97.11%	0.1224	0.0870	0.0010
3	380	00:04:15	98.44%	95.06%	0.0766	0.1267	0.0010
3	400	00:04:29	97.66%	96.47%	0.0570	0.0998	0.0010
3	420	00:04:45	99.22%	97.49%	0.0367	0.0753	0.0010
3	440	00:04:59	98.44%	96.54%	0.0553	0.0934	0.0010
3	450	00:05:02	96.09%		0.0961		0.0010
3	460	00:05:14	96.88%	97.13%	0.0701	0.0778	0.0010
3	480	00:05:31	100.00%	97.91%	0.0298	0.0709	0.0010
4	500	00:05:46	98.44%	98.20%	0.1145	0.0647	0.0010
4	520	00:06:02	97.66%	98.26%	0.0602	0.0700	0.0010
4	540	00:06:16	98.44%	98.10%	0.0609	0.0611	0.0010
4	550	00:06:18	96.88%		0.0847		0.0010
4	560	00:06:31	97.66%	97.51%	0.0500	0.0688	0.0010
4	580	00:06:46	96.88%	96.73%	0.0820	0.0930	0.0010
4	600	00:07:00	97.66%	98.39%	0.0623	0.0607	0.0010
4	620	00:07:14	97.66%	97.41%	0.0811	0.0790	0.0010
4	640	00:07:28	98.44%	97.22%	0.1647	0.0786	0.0010
4	650	00:07:30	96.09%		0.0540		0.0010
4	656	00:07:42	99.22%	98.13%	0.0389	0.0651	0.0010

```
accuracy =

    0.9839
```

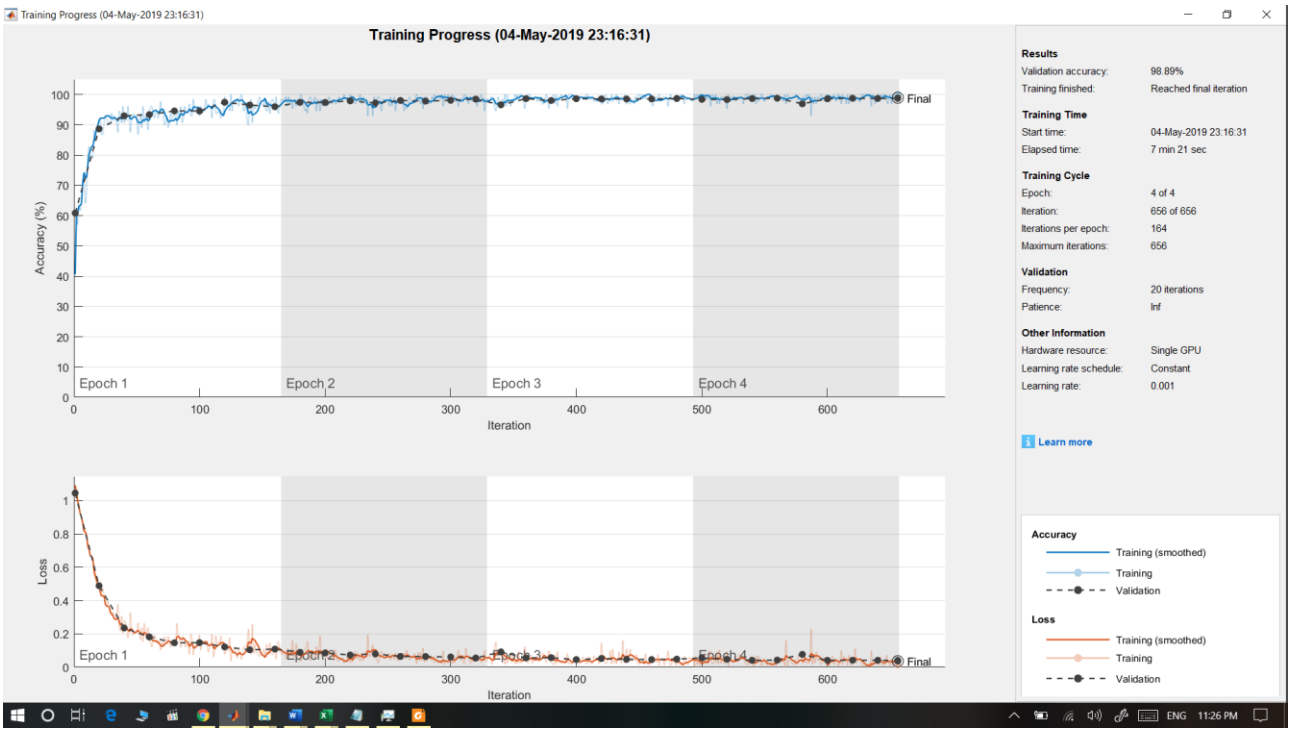
13) Using four convolutional2Dlayer:



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	11.72%	57.39%	1.1333	0.9161	0.0010
1	20	00:00:19	89.06%	90.18%	0.3235	0.3374	0.0010
1	40	00:00:30	95.31%	88.98%	0.2012	0.2760	0.0010
1	50	00:00:31	89.84%		0.2782		0.0010
1	60	00:00:40	89.06%	92.58%	0.3171	0.1944	0.0010
1	80	00:00:51	91.41%	95.01%	0.1923	0.1448	0.0010
1	100	00:01:02	96.88%	93.12%	0.0867	0.1840	0.0010
1	120	00:01:13	94.53%	94.84%	0.1499	0.1303	0.0010
1	140	00:01:24	96.88%	94.87%	0.2319	0.1353	0.0010
1	150	00:01:26	92.19%		0.1200		0.0010
1	160	00:01:35	95.31%	97.38%	0.1433	0.0904	0.0010
2	180	00:01:46	92.97%	96.92%	0.1286	0.0878	0.0010
2	200	00:01:57	99.22%	97.40%	0.0415	0.0790	0.0010
2	220	00:02:08	96.09%	96.77%	0.1078	0.0860	0.0010
2	240	00:02:18	97.66%	97.29%	0.0992	0.0880	0.0010
2	250	00:02:20	99.22%		0.0430		0.0010
2	260	00:02:29	96.88%	95.13%	0.0808	0.1201	0.0010
2	280	00:02:40	98.44%	97.80%	0.0650	0.0770	0.0010
2	300	00:02:51	98.44%	97.79%	0.0577	0.0771	0.0010
2	320	00:03:02	99.22%	98.06%	0.0511	0.0606	0.0010
3	340	00:03:16	98.44%	96.26%	0.0775	0.1119	0.0010
3	350	00:03:19	99.22%		0.0413		0.0010
3	360	00:03:32	96.88%	97.34%	0.0759	0.0714	0.0010
3	380	00:03:46	96.88%	97.76%	0.0558	0.0666	0.0010
3	400	00:04:00	99.22%	98.19%	0.0229	0.0583	0.0010
3	420	00:04:14	96.88%	98.21%	0.1003	0.0543	0.0010
3	440	00:04:28	99.22%	98.29%	0.0355	0.0572	0.0010
3	450	00:04:30	98.44%		0.0532		0.0010
3	460	00:04:42	98.44%	98.20%	0.0382	0.0562	0.0010
3	480	00:04:56	96.88%	97.97%	0.0679	0.0695	0.0010
4	500	00:05:10	94.53%	98.44%	0.1254	0.0525	0.0010
4	520	00:05:25	98.44%	98.04%	0.0280	0.0552	0.0010
4	540	00:05:39	97.66%	98.27%	0.0587	0.0503	0.0010
4	550	00:05:41	97.66%		0.1001		0.0010
4	560	00:05:53	99.22%	98.44%	0.0392	0.0454	0.0010
4	580	00:06:07	100.00%	98.49%	0.0145	0.0459	0.0010
4	600	00:06:21	96.88%	98.59%	0.0697	0.0476	0.0010
4	620	00:06:35	100.00%	98.52%	0.0186	0.0492	0.0010
4	640	00:06:49	96.88%	98.03%	0.0704	0.0571	0.0010
4	650	00:06:51	99.22%		0.0358		0.0010
4	656	00:07:02	100.00%	98.24%	0.0113	0.0478	0.0010

```
accuracy =

    0.9800
```

14) Using five convolutional2Dlayer:



Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	1	00:00:07	40.63%	60.82%	1.0921	1.0455	0.0010
1	20	00:00:17	91.41%	88.62%	0.5066	0.4887	0.0010
1	40	00:00:28	90.63%	92.93%	0.2787	0.2351	0.0010
1	50	00:00:30	92.19%		0.2075		0.0010
1	60	00:00:39	94.53%	93.33%	0.1551	0.1813	0.0010
1	80	00:00:50	93.75%	94.57%	0.1572	0.1460	0.0010
1	100	00:01:00	92.97%	94.52%	0.1934	0.1468	0.0010
1	120	00:01:11	99.22%	97.41%	0.0762	0.1213	0.0010
1	140	00:01:22	96.09%	96.50%	0.1316	0.1038	0.0010
1	150	00:01:23	96.88%		0.0804		0.0010
1	160	00:01:33	95.31%	95.94%	0.1120	0.1081	0.0010
2	180	00:01:44	99.22%	97.38%	0.0367	0.0885	0.0010
2	200	00:01:54	97.66%	97.37%	0.0696	0.0846	0.0010
2	220	00:02:09	100.00%	97.88%	0.0262	0.0724	0.0010
2	240	00:02:23	96.09%	97.19%	0.1032	0.0791	0.0010
2	250	00:02:26	98.44%		0.0603		0.0010
2	260	00:02:38	97.66%	98.01%	0.0824	0.0640	0.0010
2	280	00:02:52	98.44%	97.80%	0.0554	0.0634	0.0010
2	300	00:03:07	96.88%	98.07%	0.0632	0.0604	0.0010
2	320	00:03:20	97.66%	98.46%	0.0765	0.0545	0.0010
3	340	00:03:35	96.88%	96.57%	0.0828	0.0913	0.0010
3	350	00:03:37	97.66%		0.0660		0.0010
3	360	00:03:49	97.66%	98.61%	0.0620	0.0549	0.0010
3	380	00:04:03	98.44%	98.00%	0.0270	0.0564	0.0010
3	400	00:04:17	99.22%	98.59%	0.0331	0.0449	0.0010
3	420	00:04:31	96.88%	98.52%	0.1452	0.0516	0.0010
3	440	00:04:45	96.88%	98.51%	0.1095	0.0462	0.0010
3	450	00:04:47	100.00%		0.0164		0.0010
3	460	00:05:00	96.88%	98.51%	0.0696	0.0451	0.0010
3	480	00:05:14	100.00%	98.66%	0.0113	0.0477	0.0010
4	500	00:05:28	99.22%	98.38%	0.0224	0.0551	0.0010
4	520	00:05:42	95.31%	98.29%	0.0930	0.0468	0.0010
4	540	00:05:56	97.66%	98.64%	0.0470	0.0407	0.0010
4	550	00:05:59	97.66%		0.0484		0.0010
4	560	00:06:10	99.22%	98.74%	0.0417	0.0420	0.0010
4	580	00:06:25	99.22%	96.91%	0.0232	0.0765	0.0010
4	600	00:06:39	99.22%	98.57%	0.0260	0.0403	0.0010
4	620	00:06:53	99.22%	98.73%	0.0534	0.0413	0.0010
4	640	00:07:07	100.00%	98.66%	0.0191	0.0409	0.0010
4	650	00:07:09	100.00%		0.0100		0.0010
4	656	00:07:21	100.00%	98.68%	0.0113	0.0403	0.0010

```
accuracy =

    0.9889
```

Appendix B

Code (Python):

We have used python 3.7, to find the number of spots and mixing up or currently ordering data that might lead our network astray in training. First, we have loaded all the required libraries among them numpy as np is used to deal with arrays and cv2 for mainly resizing images. We have also loaded Keras to preprocess the image using the ImageDataGenerator and Keras models to align the images in sequence, Keras layers we need to tune at the time of Convolutional Neural Networks (CNNs) training and Keras call back for the early stopping of training model.

```
import keras as k
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

After that using the For loop we have extracted the image in three bins i.e. we have made three folders and after writing algorithm for it automatically sends the image with 0 spot in 0 spot folder, images with 1 spot in 1 spot folder and images with 2 spots in 2 spot folder by doing this we have extracted and arranged the 209932 images in three folder. There are total 99,721 images with 0 spot which is about 47.74% of total images, 100,608 images with 1 spot which is about 48.06% of total images, 9,604 images with 2 spots which is about 4.57% of total images. Then we have arranged the testing images in an array.

Then by creating one-hot labels by using k.utils.to_categorical (labels, num_classes=3). Once we done with the creating labels then we create the hyperparameters for creating training and validation dataset where the batch size is 2048. The batch size should always be in binary form like in the multiple of i.e. 2, 4, 16, 32, 64, 128, 256, 512, 1024, **2048** and seed size is 1729 and then we train the images using the ImageDatagenerator where we did the rescaling from 0 to 255 and set the validation split as 0.2. After that we have trained the data the target size is 24 because the pixel of images are 24x24 so it cannot exceed that and here in this case we have set the subset as “training” for training data and at the time of testing data the size is same as above but for this case we have set the subset as “validation” for validation data. After that we did the

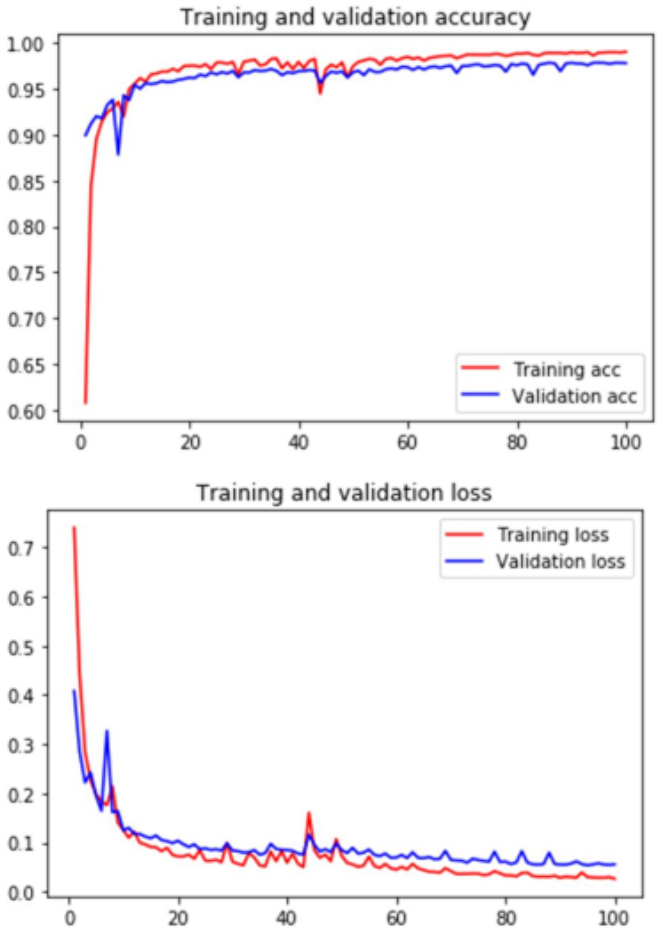
training and validation and found out that 167498 images belonging to 3 classes.

For Model Hyper parameters we have 3 channels with a batch size of 2048 and 3 classes. We have run our model for 100 epochs and recorded the result. We have used Relu (Rectified Linear Units) as an image activation function. We have total 25875 trainable parameters and 0 non trainable parameters in our model summary. After that we have trained the data for 100 epochs using model.fit_generator. We have tested 80% of total images data and trained 20% of total images data.

We found the accuracy to be 89.94% for the first epoch and after training it for 100 epochs the final accuracy we got is 97.85% and loss values to be 0.0562.

```
Epoch 00091: val_acc did not improve from 0.97859
Epoch 92/100
 - 94s - loss: 0.0298 - acc: 0.9896 - val_loss: 0.0574 - val_acc: 0.9775

Epoch 00092: val_acc did not improve from 0.97859
Epoch 93/100
 - 93s - loss: 0.0288 - acc: 0.9901 - val_loss: 0.0620 - val_acc: 0.9759

Epoch 00093: val_acc did not improve from 0.97859
Epoch 94/100
 - 94s - loss: 0.0397 - acc: 0.9865 - val_loss: 0.0565 - val_acc: 0.9785

Epoch 00094: val_acc did not improve from 0.97859
Epoch 95/100
 - 94s - loss: 0.0301 - acc: 0.9897 - val_loss: 0.0540 - val_acc: 0.9786

Epoch 00095: val_acc did not improve from 0.97859
Epoch 96/100
 - 94s - loss: 0.0290 - acc: 0.9899 - val_loss: 0.0558 - val_acc: 0.9784

Epoch 00096: val_acc did not improve from 0.97859
Epoch 97/100
 - 94s - loss: 0.0286 - acc: 0.9902 - val_loss: 0.0581 - val_acc: 0.9774

Epoch 00097: val_acc did not improve from 0.97859
Epoch 98/100
 - 94s - loss: 0.0285 - acc: 0.9902 - val_loss: 0.0558 - val_acc: 0.9784

Epoch 00098: val_acc did not improve from 0.97859
Epoch 99/100
 - 94s - loss: 0.0296 - acc: 0.9899 - val_loss: 0.0549 - val_acc: 0.9784

Epoch 00099: val_acc did not improve from 0.97859
Epoch 100/100
 - 94s - loss: 0.0267 - acc: 0.9906 - val_loss: 0.0562 - val_acc: 0.9782

Epoch 00100: val_acc did not improve from 0.97859
```

Then we have plotted the training and validation accuracy graph and represented the Training loss values by red color and validation loss values by blue color.




```
import math                     # providing access to the mathematical
functions defined by the C standard

import matplotlib.pyplot as plt  # plotting library

import scipy                    # scientific computnig and technical
computing

import cv2                      # working with, mainly resizing, images

import numpy as np              # dealing with arrays

import glob                     # return a possibly-empty list of path names
that match pathname
```

```
import os                          # dealing with directories
import pandas as pd                # providing data structures and data
analysis tools
import tensorflow as tf
import itertools
import random
from random import shuffle         # mixing up or currently ordered data that
might lead our network astray in training.
from tqdm import tqdm              # a nice pretty percentage bar for tasks.
Thanks to viewer Daniel Bühler for this suggestion
from PIL import Image
from scipy import ndimage
from pathlib import Path
from sklearn.metrics import classification_report, confusion_matrix
from skimage import io
from sklearn import metrics
%matplotlib inline
np.random.seed(1)
import Keras as k
from Keras.preprocessing.image import ImageDataGenerator
from Keras.models import Sequential
from Keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten,
Dense
from Keras.callbacks import ModelCheckpoint, EarlyStopping
base_dir = os.getcwd()+"\\"
training_img = io.imread('images_training.tiff')
training_imgarray = np.array(training_img)
'''validation_img = io.imread('images_test.tiff')
validation_imgarray = np.array(validation_img)'''
'''
df = pd.read_csv('descriptions_training.csv', header=None)
df = df.drop(columns=[2,3,4,5])
df = df.rename(columns={0:"img",1:"label"})
```

```
for i, row in df.iterrows():
    image_array = training_imgarray[i]
    if row[1] == 0:
        scipy.misc.toimage(image_array, cmin=0.0,
cmax=...).save(base_dir+"\\train\\0\\"+str(i)+'.jpg')
    elif row[1] == 1:
        scipy.misc.toimage(image_array, cmin=0.0,
cmax=...).save(base_dir+"\\train\\1\\"+str(i)+'.jpg')
    else:
        scipy.misc.toimage(image_array, cmin=0.0,
cmax=...).save(base_dir+"\\train\\2\\"+str(i)+'.jpg')
'''
'''
# Creating one-hot labels
labels = df.label
labels = np.array(labels)
labels = labels.reshape(df.shape[0],1)
labels = k.utils.to_categorical(labels, num_classes=3)
'''
# Hyperparameters for creating training and validation dataset
batch_size = 2048
seed = 1729
# Training generator
train_dir = base_dir+"\\train\\"
train_datagen = ImageDataGenerator(
                                   rescale=1./255,
                                   fill_mode='nearest',
                                   validation_split=0.2) # set validation split
train_data = train_datagen.flow_from_directory(
                                               train_dir,
                                               target_size=(24, 24),
                                               batch_size=batch_size,
```

```
                                                seed=seed,
                                                shuffle=True,
                                                class_mode='categorical',
                                                subset='training') # set as
training data
test_data = train_datagen.flow_from_directory(
                                                train_dir, # same directory as
training data
                                                target_size=(24, 24),
                                                batch_size=batch_size,
                                                seed=seed,
                                                shuffle=True,
                                                class_mode='categorical',
                                                subset='validation') # set as
validation data
train_num = train_data.samples
validation_num = test_data.samples
# Model Hyperparameters
LR = 1e-3
channels=3
seed=1729
batch_size = 2048
num_classes = 3
epochs = 100
data_augmentation = True
num_predictions = 100
# Model Definition
model = Sequential()
model.add(Conv2D(16, (3, 3), input_shape=(24, 24, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(16, (3, 3)))
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['acc'])
model.summary()
# Start training
filepath=str(base_dir+"\\model.h5f")
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
save_best_only=True, mode='max')
# = EarlyStopping(monitor='val_acc', patience=15)
callbacks_list = [checkpoint]#, stopper]
history = model.fit_generator(train_data,
                              steps_per_epoch= train_num // batch_size,
                              epochs=epochs,
                              validation_data=test_data,
                              validation_steps= validation_num // batch_size,
                              callbacks=callbacks_list,
                              verbose = 2
                             )
acc = history.history['acc']
```

```
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.title('Training and validation accuracy')
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend()
plt.figure()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')
plt.legend()
plt.show()
```