Software’s/Tools required

- Python (preferably version 3 and above) - https://www.python.org/downloads/
- pip - The Python Package Installer - https://pip.pypa.io/en/stable/installing/
- SciPy ecosystem (numpy, scipy, matplotlib, pandas) - https://www.scipy.org/install.html
- scikit - https://scikit-learn.org/stable/install.html
- h5py - https://pypi.org/project/h5py/
- TensorFlow - https://www.tensorflow.org/install
- Keras - https://keras.io/#installation
- Spyder - https://docs.spyder-ide.org/installation.html

Tensor Flow

What is Tensor Flow?

Google's TensorFlow is an open-source and most popular deep learning library for research and product development. Currently, the most famous deep learning library in the world is Google's TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations.
Example: To give a concrete example, Google users can experience a faster and more refined search with AI. If the user types a keyword at the search bar, Google provides a recommendation about what could be the next word.

Google wants to use machine learning to take advantage of their massive datasets to give users the best experience. Three different groups use machine learning:

- Researchers
- Data scientists
- Programmers.

They can all use the same toolset to collaborate with each other and improve their efficiency.
Google does not just have any data; they have the world's most massive computer, so Tensor Flow was built to scale. TensorFlow is a library developed by the Google Brain Team to accelerate machine learning and deep neural network research.
It was built to run on multiple CPUs or GPUs and even mobile operating systems, and it has several wrappers in several languages like Python, C++ or Java.

History of TensorFlow

A couple of years ago, deep learning started to outperform all other machine learning algorithms when giving a massive amount of data. Google saw it could use these deep neural networks to improve its services:

- Gmail
- Photo
- Google search engine
- Voice Assistance

They build a framework called **Tensorflow** to let researchers and developers work together on an AI model. Once developed and scaled, it allows lots of people to use it.
It was first made public in late 2015, while the first stable version appeared in 2017. It is open source under Apache Open Source license. You can use it, modify it and redistribute the modified version for a fee without paying anything to Google.

TensorFlow Architecture

Tensorflow architecture works in three parts:

- Preprocessing the data
- Build the model
- Train and estimate the model

It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**. You can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.

This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.

Where can Tensorflow run?

TensorFlow's hardware, and software requirements can be classified into

Development Phase: This is when you train the model. Training is usually done on your Desktop or laptop.

Run Phase or Inference Phase: Once training is done Tensorflow can be run on many different platforms. You can run it on

- Desktop running Windows, macOS or Linux
- Cloud as a web service
- Mobile devices like iOS and Android

You can train it on multiple machines then you can run it on a different machine, once you have the trained model.

The model can be trained and used on GPUs as well as CPUs. GPUs were initially designed for video games. In late 2010, Stanford researchers found that GPU was also very good at matrix operations and algebra so that it makes them very fast for doing these kinds of calculations. Deep learning relies on a lot of matrix operations. TensorFlow is very fast at computing the matrix operations because it is written in C++. Although it is implemented in C++, TensorFlow can be accessed and controlled by other languages mainly, Python.

Finally, a significant feature of TensorFlow is the TensorBoard. The TensorBoard enables to monitor graphically and visually what TensorFlow is doing.

Introduction to Components of TensorFlow

Tensor

Tensorflow's name is directly derived from its core framework: **Tensor**. In Tensorflow, all the computations involve tensors. A tensor is a **vector** or **matrix** of n-dimensions that represents all types of data. All values in a tensor hold identical data type with a known (or partially known) **shape**. The shape of the data is the dimensionality of the matrix or array.

A tensor can be originated from the input data or the result of a computation. In TensorFlow, all the operations are conducted inside a **graph**. The graph is a set of computation that takes place successively. Each operation is called an **op node** and are connected to each other.
The graph outlines the ops and connections between the nodes. However, it does not display the values. The edge of the nodes is the tensor, i.e., a way to populate the operation with data.

Graphs

TensorFlow makes use of a graph framework. The graph gathers and describes all the series computations done during the training. The graph has lots of advantages:

- It was done to run on multiple CPUs or GPUs and even mobile operating system
- The portability of the graph allows to preserve the computations for immediate or later use. The graph can be saved to be executed in the future.
- All the computations in the graph are done by connecting tensors together
-
    - A tensor has a node and an edge. The node carries the mathematical operation and produces an endpoints outputs. The edges the edges explain the input/output relationships between nodes.

Why is TensorFlow popular?

TensorFlow is the best library of all because it is built to be accessible for everyone. Tensorflow library incorporates different API to build at scale deep learning architecture like CNN or RNN. TensorFlow is based on graph computation; it allows the developer to visualize the construction of the neural network with Tensorboad. This tool is helpful to debug the program. Finally, Tensorflow is built to be deployed at scale. It runs on CPU and GPU.
Tensorflow attracts the largest popularity on GitHub compare to the other deep learning framework.

What Algorithms it Supports?

Machine Learning Algorithms: K-Means Clustering, Random Forests, Support Vector Machines, Gaussian Mixture Modeling, Linear/Logistic Regression
Deep Learning Algorithms: Feed Forward Neural Network (FFNN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short Term Memory (LSTM) Networks, Generative Adversarial Networks (GANs).

More Details can be found at
https://www.guru99.com/tensorflow-tutorial.html

Keras :
Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. The details about keras is available at https://keras.io/.
Best Features of Keras:

- *Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).*
- Supports convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

About Keras models:

There are two main types of models available in Keras: the Sequential model and the Model class used with the functional API.

Sequential model: The Sequential model is a linear stack of layers. The sequential model supports architecture for feedforward neural network, convolutional neural network and recurrent neural network (includes LSTM). Importing the sequential model of keras, user has to design the base architecture by specifying the following information.

- Type of layer: Feedforward/Convolutional/ Recurrent/ LSTM
- Dense of each layer
- Input Dimension/ Input Shape
- Activation function
- User also can specify kernel and bias initialization (optional)

After the architecture is ready, the user can *compile the model* specifying the loss, accuracy and optimizer options.

After compiling the model, user has to *fit* the input data and labels with the designed model.

Basic Steps to design a neural network model:

Step-1: Database structuring or preprocessing: Database labeling (user can use any existing python library like numpy, sklearn, scipy etc or can also do the same in MATLAB, can import the same into python environment.)

Step-2: Model design: The user can design a model by following the steps bellow,
model=sequential() # To import a sequential model to the workspace
Before importing the model the user have to import the followings to the workspace.
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
After importing the sequential layer, the user has to design the network using following instructions

model.add(Dense(512, input_dim=784, activation='sigmoid')) # 1st hidden layer (512: no of neurons in 1st hidden layer)
model.add(Dense(2, activation='softmax'))
Step-3: Compile the model using following instruction:
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy']) # compile the model
Step-4: Train the model:
model.fit(data, label, epochs=4, batch_size=200)
Step-5: Evaluate the model:
score = model.evaluate(test_data, test_labels, verbose=1) #evaluate the model

Details about the available options for optimizer and activation function can be found from the following links

Activation function (Sigmoid, Tanh, RELU etc.,): https://keras.io/activations/
Optimizer (SGD, ADAM, RMSPROP etc.,): https://keras.io/optimizers/

The MNIST database (Modified National Institute of Standards and Technology database)

- Database of handwritten digits (0-9) that is commonly used for training various image processing system
- 60,000 training images and 10,000 testing images each of 28x28 pixel in grayscale
- 500 different writers including American Census Bureau employees and American high school students

Lab Experiments

Classification problem (MNIST Database)

1) Design a FeedForward Neural Network (FFNN) architecture having 1 hidden layer, to solve a 2 class problem using MNIST Database.

2) Design a Feedforward Neural Network architecture having more than one hidden layer, to solve a 2 class problem using MNIST Database.

3) Design a Feedforward Neural Network architecture having 1 hidden layer, to solve a 10 class problem using MNIST Database.

4) Design a Feedforward Neural Network architecture having more than one hidden layer, to solve a 10 class problem using MNIST Database.

5) Design a Convolutional Neural Network (CNN) architecture having more than one hidden layer, to solve a 10 class problem using MNIST Database.

6) Randomly choose some test image and perform translation, rotation and scaling to it. Observe the performance of the trained FFNN and CNN models using the modified test data.

7) Randomly choose some training and testing images, and perform translation, rotation and scaling to it. Train the FFNN and CNN model using modified data and observe the performance of the FFNN and CNN model using the modified test data.

Prediction problem (Time series data)

Observations:-

Classification problem (MNIST Database)

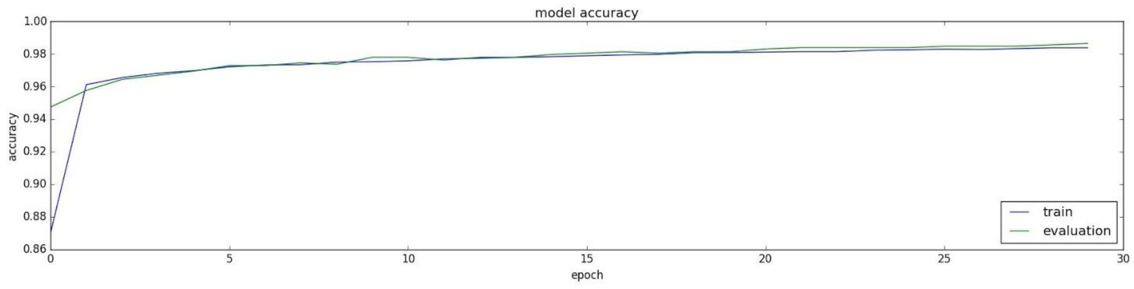
1) Design a FeedForward Neural Network (FFNN) architecture having 1 hidden layer, to solve a 2 class problem using MNIST Database.

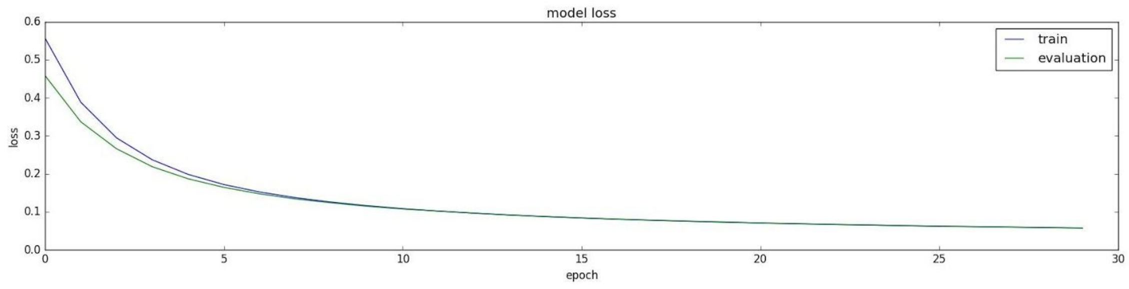
1) 2 class chosen: 0 and 8 (Image size= 28*28)

Neural Network Structure: 1 hidden layer (512 neurons, sigmoid activation function), Output layer (2 neurons, softmax activation function), Optimizer: SGD, epochs=30, batch_size=200, validation_split=0.10

```
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
dense_11 (Dense) (None, 512) 401920 (28*28*512+512)
_________________________________________________________________
dense_12 (Dense) (None, 2) 1026 (512*512+2)
=================================================================
Total params: 402,946
Trainable params: 402,946
Non-trainable params: 0
```

Results:
Test loss: 0.05008049780348489
Test accuracy: 0.989252814738997





Python code:**Mnist_ffnn.py**

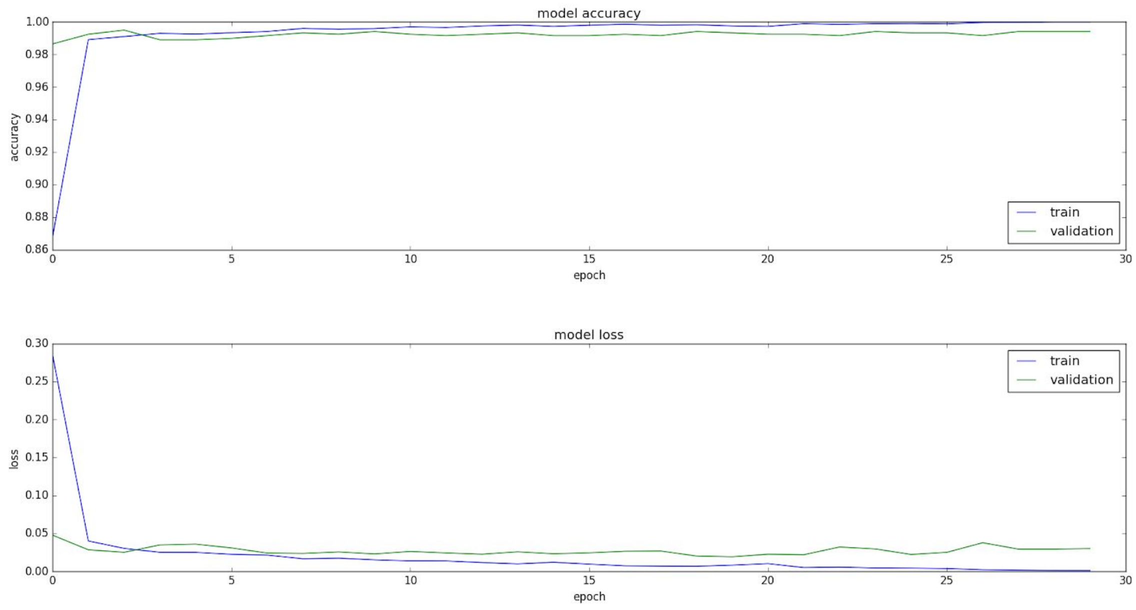
2) Design a Feedforward Neural Network architecture having more than one hidden layer, to solve a 2 class problem using MNIST Database.

2) 2 class chosen: 0 and 8 (Image size= 28*28)

Neural Network Structure: 4 hidden layer (1024,512, 256, 64 neurons, sigmoid activation function), Output layer (2 neurons, softmax activation function), Optimizer: adam, epochs=30, batch_size=200, validation_split=0.10

```
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
dense_1 (Dense) (None, 1024) 803840
_________________________________________________________________
dense_2 (Dense) (None, 512) 524800
_________________________________________________________________
dense_3 (Dense) (None, 256) 131328
_________________________________________________________________
dense_4 (Dense) (None, 64) 16448
_________________________________________________________________
dense_5 (Dense) (None, 2) 130
=================================================================
```

Results:
Test loss: 0.03376440113662845
Test accuracy: 0.9928352098259979



Python code:**Mnist_dffnn.py**

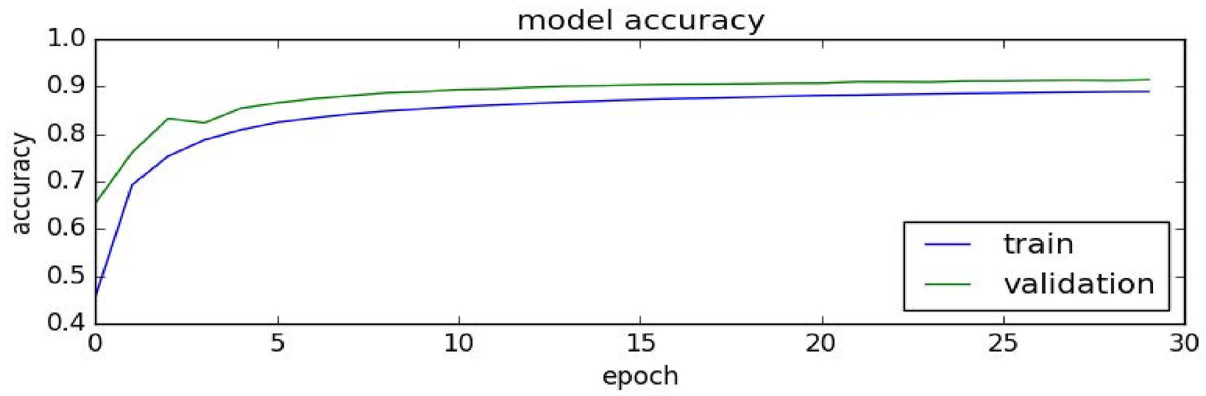
3) Design a Feedforward Neural Network architecture having 1 hidden layer, to solve a 10 class problem using MNIST Database.

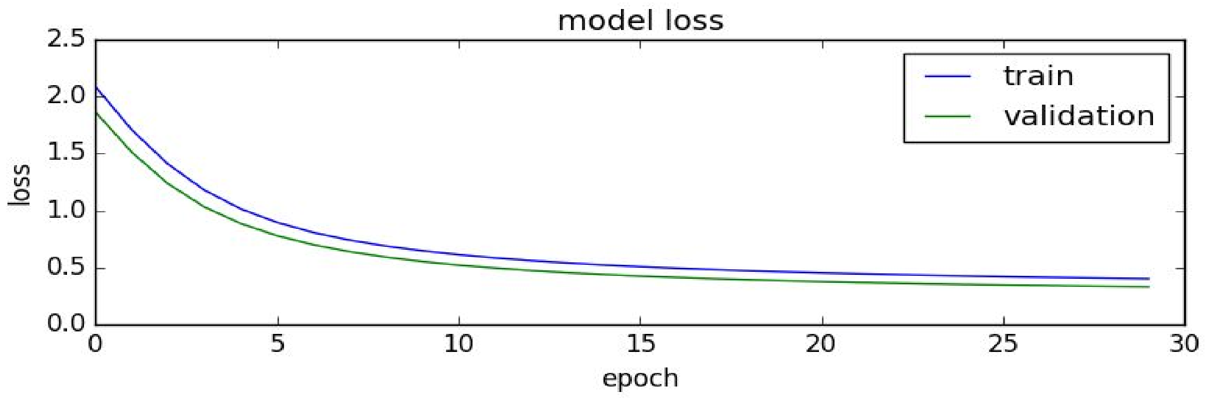
3) 10 class chosen: 0 to 9 (Image size= 28*28)

Neural Network Structure: 1 hidden layer (512 neurons, sigmoid activation function), Output layer (10 neurons, softmax activation function), Optimizer: SGD, epochs=30, batch_size=200, validation_split=0.10

```
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
dense_1 (Dense) (None, 512) 401920
_________________________________________________________________
dense_2 (Dense) (None, 10) 5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

Results:
Test loss: 0.37369379016160964
Test accuracy: 0.8993





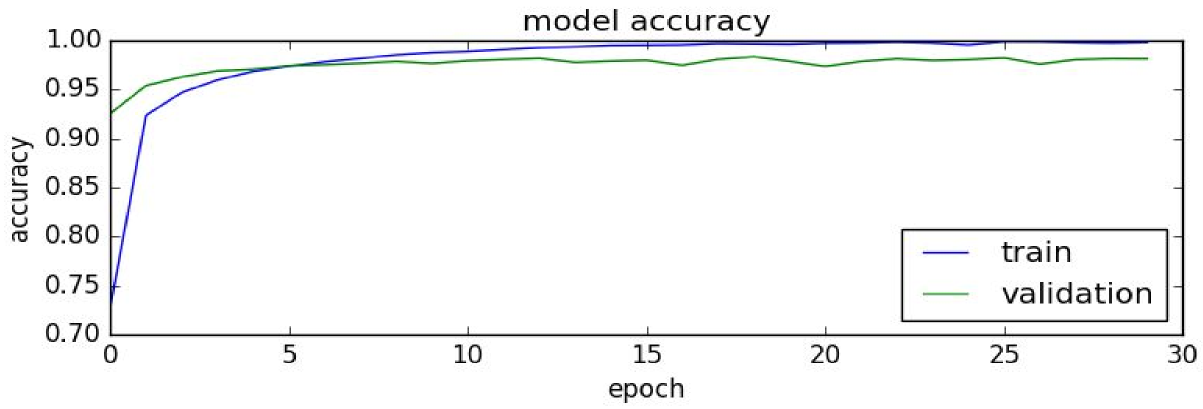
Python code: **Mnist_ffnn_allclass.py**

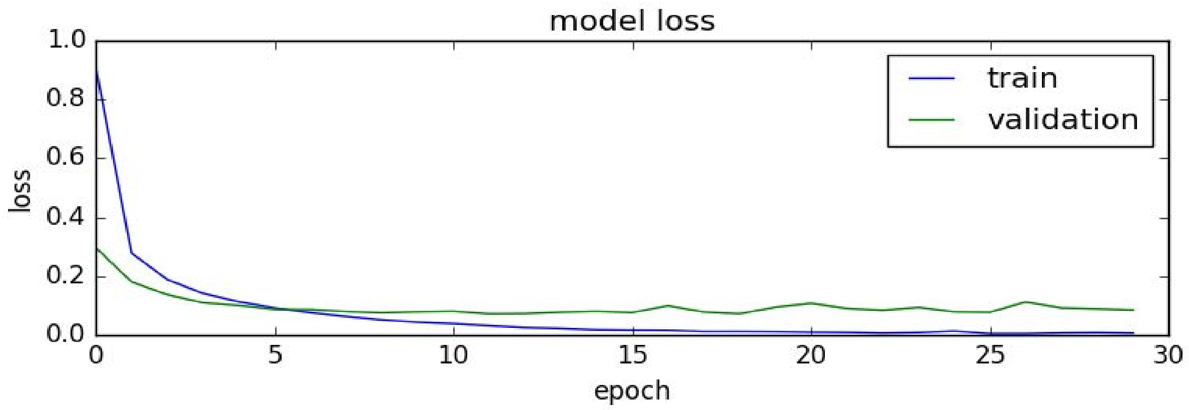
4) Design a Feedforward Neural Network architecture having more than one hidden layer, to solve a 10 class problem using MNIST Database.4) 10 class chosen: 0 to 9 (Image size= 28*28)

Neural Network Structure: 4 hidden layer (1024,512, 256, 64 neurons, sigmoid activation function), Output layer (10 neurons, softmax activation function), Optimizer: adam, epochs=30, batch_size=200, validation_split=0.10

```
Layer (type) Output Shape Param #
=================================================================
dense_1 (Dense) (None, 1024) 803840
_________________________________________________________________
dense_2 (Dense) (None, 512) 524800
_________________________________________________________________
dense_3 (Dense) (None, 256) 131328
_________________________________________________________________
dense_4 (Dense) (None, 64) 16448
_________________________________________________________________
dense_5 (Dense) (None, 10) 650
=================================================================
```

Results:
Test loss: 0.09666743702506647
Test accuracy: 0.9811





Python code: **Mnist_dffnn_allclass.py** Model: **dffnn10class.h5**

5) Design a Convolutional Neural Network (CNN) architecture having more than one hidden layer, to solve a 10 class problem using MNIST Database.

5) 10 class chosen: 0 to 9 (Image size= 28*28)

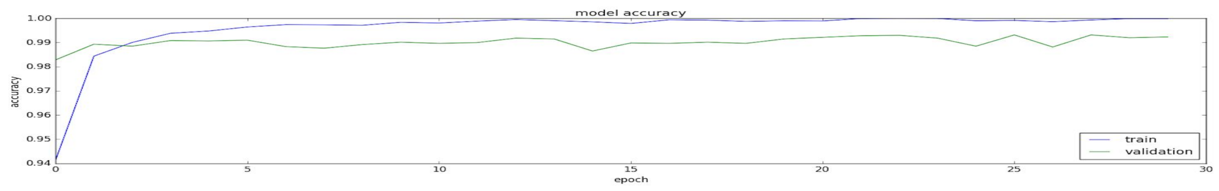
Neural Network Structure: Layer 1: 2D conv layer (32 filter, 3*3 filter size, relu activation function), Layer 2: 2D conv layer (64 filter, 3*3 filter size, relu activation function), Layer 3: Max pooling layer (2*2 filter size), Layer 4: Fully connected layer (128 neurons, relu activation function), Output layer: Fully connected layer (10 neurons, softmax activation function) ,Optimizer: adam, epochs=30, batch_size=200, validation_split=0.10

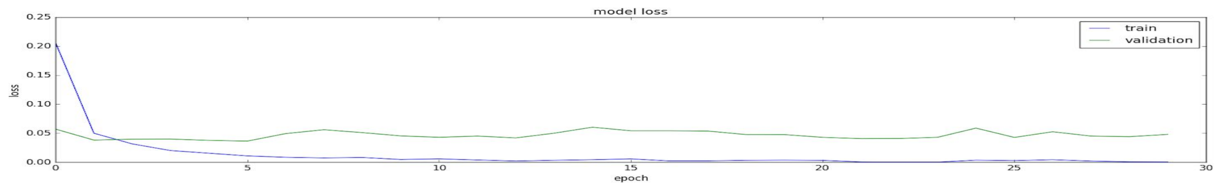
```
_________________________________________________________________
Layer (type)              Output Shape           Param #
=================================================================
conv2d_5 (Conv2D)          (None, 26, 26, 32)       320       (3*3*32+32)
_________________________________________________________________
conv2d_6 (Conv2D)          (None, 24, 24, 64)       18496     (3*3*64+64)
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 64)       0
_________________________________________________________________
flatten_3 (Flatten)        (None, 9216)             0
_________________________________________________________________
dense_15 (Dense)           (None, 128)              1179776
_________________________________________________________________
dense_16 (Dense)           (None, 10)               1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
```

Results: (Python code: **Mnist_cnn_allclass.py** Model: **cnn_10class.h5)**

Test loss: 0.04662909052905507

Test accuracy: 0.9916





6) Randomly choose some test image and perform translation, rotation and scaling to it. Observe the performance of the trained FFNN and CNN models using the modified test data.

6) Randomly selected 3000 images from the test set of MNIST dataset.

1000 images are rotated with angle ranging from -60 to 60
1000 images are scaled with a factor 0.3 to 0.98
1000 images are translated in X and Y direction with a shift pixel of 0 to 4

FeedForward Network Architecture:

```
Layer (type)              Output Shape            Param #
=================================================================
dense_17 (Dense)          (None, 1024)            803840
_________________________________________________________________
dense_18 (Dense)          (None, 512)             524800
_________________________________________________________________
dense_19 (Dense)          (None, 256)             131328
_________________________________________________________________
dense_20 (Dense)          (None, 64)              16448
_________________________________________________________________
dense_21 (Dense)          (None, 10)              650
=================================================================
Total params: 1,477,066
Trainable params: 1,477,066
Non-trainable params: 0
_________________________________________________________________
```

Results (FFNN):

```
Test loss: 0.7132828288909281
Test accuracy: 0.8937
```

CNN Architecture:

```
_________________________________________________________________
Layer (type)              Output Shape            Param #
=================================================================
conv2d_5 (Conv2D)         (None, 26, 26, 32)      320
_________________________________________________________________
```

```
conv2d_6 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               1179776
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

Results (CNN):
Test loss: 0.5919975529133299
Test accuracy: 0.9289

Matlab and Python code:
cnn_modified_image_test.py
ffnn_modified_image_test.py
trans_scal_rotate_mnist_test.m

Model used:

cnn_10class.h5
dffnn10class.h5

7) Randomly choose some training and testing images, and perform translation, rotation and scaling to it. Train the FFNN and CNN model using modified data and observe the performance of the FFNN and CNN model using the modified test data.

7)

Randomly selected 30000 images from the train set of MNIST dataset.

10000 images are rotated with angle ranging from -60 to 60
10000 images are scaled with a factor 0.3 to 0.98
10000 images are translated in X and Y direction with a shift pixel of 0 to 4

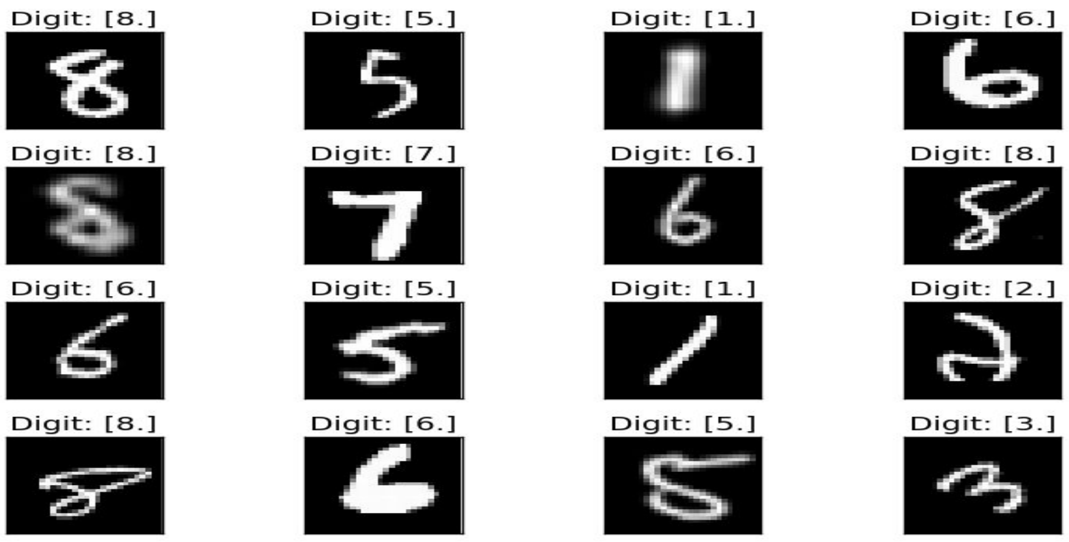
Neural Network Structure (FFNN): 4 hidden layer (1024,512, 256, 64 neurons, sigmoid activation function), Output layer (10 neurons, softmax activation function), Optimizer: adam, epochs=30, batch_size=200, validation_split=0.10

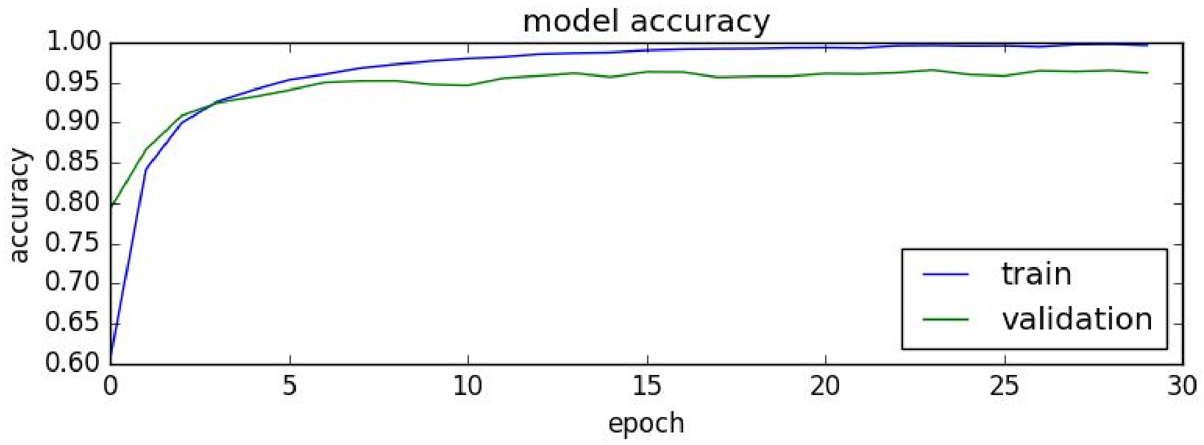
```
Layer (type) Output Shape Param #
=================================================================
dense_1 (Dense) (None, 1024) 803840
_________________________________________________________________
dense_2 (Dense) (None, 512) 524800
_________________________________________________________________
dense_3 (Dense) (None, 256) 131328
_________________________________________________________________
dense_4 (Dense) (None, 64) 16448
_________________________________________________________________
dense_5 (Dense) (None, 10) 650
=================================================================
```

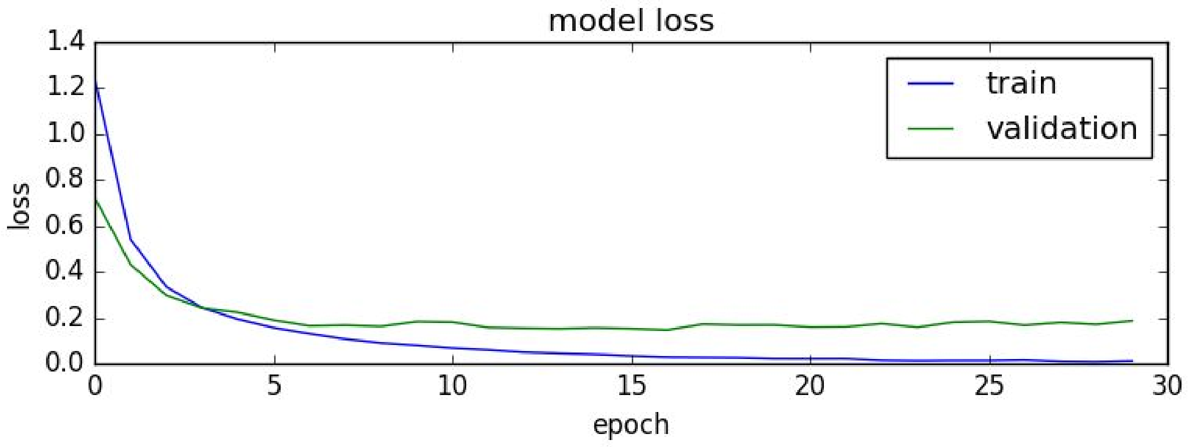
Results:
Test loss: 0.1387768559299875
Test accuracy: 0.9703

Matlab and Python code:
trans_scal_rotate_mnist_train.m
Mnist_cnn_modified_train_test.py
Mnist_dffnn_allclass_modified_train_test.py
Output Model:
cnn_10class_modified.h5
dffnn10class_modified.h5

Modified train set:



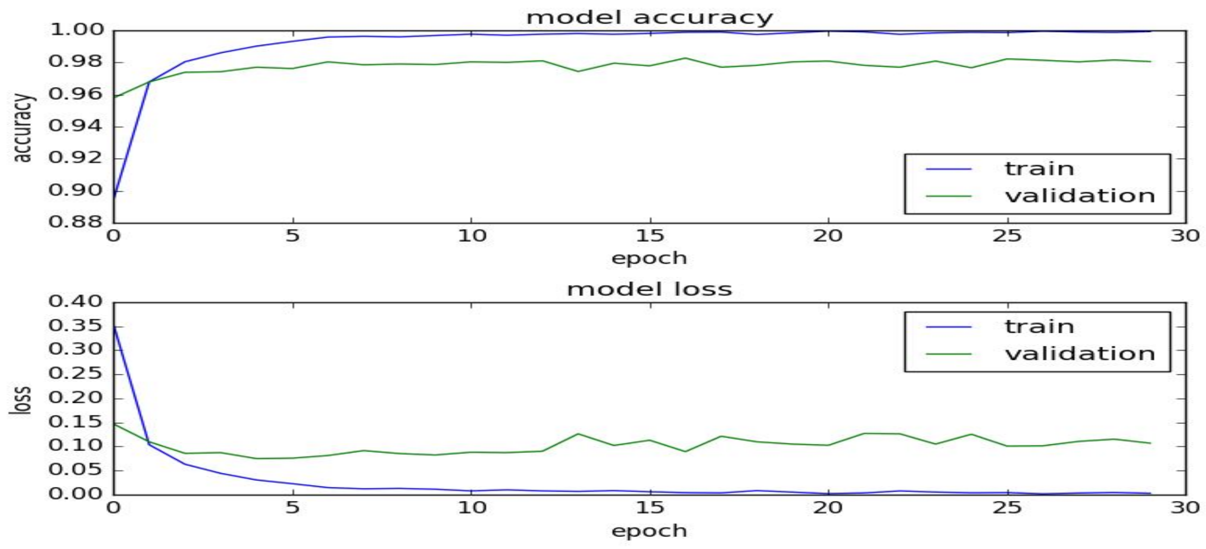




Neural Network Structure (CNN): Layer 1: 2D conv layer (32 filter, 3*3 filter size, relu activation function), Layer 2: 2D conv layer (64 filter, 3*3 filter size, relu activation function), Layer 3: Max pooling layer (2*2 filter size), Layer 4: Fully connected layer (128 neurons, relu activation function), Output layer: Fully connected layer (10 neurons, softmax activation function) ,Optimizer: adam, epochs=30, batch_size=200, validation_split=0.10

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               1179776
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
```

Results:
Test loss: 0.0896888452693864
Test accuracy: 0.9844


model accuracy
accuracy
epoch
train
validation
model loss
loss
epoch
train
validation