**Q2: Implementation Task:**   **[15 marks]**

You are provided WITH three data files named as data\_batch\_1, data\_batch\_2 and test\_batch. The first two files are for training & the last file is testing you model.

Read the file with the code snippet below:

import numpy as np

def unpickle(file):

import pickle

with open(file, 'rb') as fo:

dict = pickle.load(fo, encoding='bytes')

return dict

X = unpickle("data\_batch\_1") # if you wanna load data\_batch\_1 file.

# Note the function unpickle returns the dictionary

# Extract data as:

X\_train = X[b'data']

y\_train = np.array(X[b'labels']).reshape((-1,1))

X\_train.shape, y\_train.shape

The provided data is a part of benchmark dataset of computer vision called **CIFAR-10.** Each image is colored (3 channels, Red, Green & Blue i.e. RGB) having resolution of . Each image is flattened to have a vector of number of unsigned 8bit integer (uint8). However, you may reshape the image for visualization. Consider the code snippet below:

def visualize(X,idx):

img = X[b'data'][idx]

img = img.reshape((3, -1))

img1 = np.zeros((32,32,3), dtype="uint8")

for i in range(len(img)):

img1[:,:,i] = img[i].reshape((32,32))

import matplotlib.pyplot as plt

plt.imshow(img1)

title = f"Label: {X[b'labels'][idx]}, file: {X[b'filenames'][idx]}"

plt.title(title)

plt.show()

visualize(X, 15)

A picture containing graphical user interface

Description automatically generated

You have to train the deep neural network and evaluate the accuracy. You may consider the following steps:

1. Preprocessing: Label Encoding, Data normalization (if required) [3 marks]
2. Model selection (TensorFlow) & Hyper-parameters tuning. (Perform at least 5 experiments for with different parameters and your justification and make a clear table of the hyper-parameters) [8 marks]
3. Evaluate the best model with the test data using accuracy metric. [4 marks]

# Appendix:

MNIST Dataset Classification Model with TensorFlow: Note: Conv2D and Maxpooling layers are part of post mid course, so you are not allowed to use these layer. You may use Dense Layers.

import warnings

warnings.filterwarnings('ignore')

import numpy as np

np.random.seed(123) # for reproducibility

from keras.models import Sequential

from keras.layers import Flatten, MaxPool2D, Conv2D, Dense, Reshape, Dropout

from keras.utils import np\_utils

Using TensorFlow backend.

from keras.datasets import mnist

# Load pre-shuffled MNIST data into train and test sets

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

X\_train = X\_train.reshape(X\_train.shape[0], 28, 28, 1)

X\_test = X\_test.reshape(X\_test.shape[0], 28, 28, 1)

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

X\_train /= 255

X\_test /= 255

Y\_train = np\_utils.to\_categorical(y\_train, 10)

Y\_test = np\_utils.to\_categorical(y\_test, 10)

model = Sequential()

model.add(Conv2D(32, 3, 3, activation='relu', input\_shape=(28,28,1)))

model.add(Conv2D(32, 3, 3, activation='relu'))

model.add(MaxPool2D(pool\_size=(2,2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='adam',

metrics=['accuracy'])

model.fit(X\_train, Y\_train, batch\_size=32, epochs=10, verbose=1)

Pytorch Implementation of MNIST Classification Model:

import torch

import torch.nn as nn

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt

input\_size = 784 # 28x28

hidden\_size = 500

num\_classes = 10

num\_epochs = 2

batch\_size = 100

learning\_rate = 0.001

# Data loader

train\_loader = torch.utils.data.DataLoader(dataset=train\_dataset,

batch\_size=batch\_size,

shuffle=True)

test\_loader = torch.utils.data.DataLoader(dataset=test\_dataset,

batch\_size=batch\_size,

shuffle=False)

# Fully connected neural network with one hidden layer

class NeuralNet(nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_size, num\_classes):

super(NeuralNet, self).\_\_init\_\_()

self.input\_size = input\_size

self.l1 = nn.Linear(input\_size, hidden\_size)

self.relu = nn.ReLU()

self.l2 = nn.Linear(hidden\_size, num\_classes)

def forward(self, x):

out = self.l1(x)

out = self.relu(out)

out = self.l2(out)

# no activation and no softmax at the end

return out

# Loss and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate)

n\_total\_steps = len(train\_loader)

for epoch in range(num\_epochs):

for i, (images, labels) in enumerate(train\_loader):

# origin shape: [100, 1, 28, 28]

# resized: [100, 784]

images = images.reshape(-1, 28\*28).to(device)

labels = labels.to(device)

# Forward pass

outputs = model(images)

loss = criterion(outputs, labels)

# Backward and optimize

optimizer.zero\_grad()

loss.backward()

optimizer.step()

with torch.no\_grad():

n\_correct = 0

n\_samples = 0

for images, labels in test\_loader:

images = images.reshape(-1, 28\*28).to(device)

labels = labels.to(device)

outputs = model(images)

# max returns (value ,index)

\_, predicted = torch.max(outputs.data, 1)

n\_samples += labels.size(0)

n\_correct += (predicted == labels).sum().item()

acc = 100.0 \* n\_correct / n\_samples

print(f'Accuracy of the network on the 10000 test images: {acc} %')