Capstone Project

Image classifier for the SVHN dataset

Instructions

In this notebook, you will create a neural network that classifies real-world images digits. You will use concepts from throughout this course in building, training, testing, validating and saving your Tensorflow classifier model.

This project is peer-assessed. Within this notebook you will find instructions in each section for how to complete the project. Pay close attention to the instructions as the peer review will be carried out according to a grading rubric that checks key parts of the project instructions. Feel free to add extra cells into the notebook as required.

How to submit

When you have completed the Capstone project notebook, you will submit a pdf of the notebook for peer review. First ensure that the notebook has been fully executed from beginning to end, and all of the cell outputs are visible. This is important, as the grading rubric depends on the reviewer being able to view the outputs of your notebook. Save the notebook as a pdf (you could download the notebook with File -> Download .ipynb, open the notebook locally, and then File -> Download as -> PDF via LaTeX), and then submit this pdf for review.

Let's get started!

We'll start by running some imports, and loading the dataset. For this project you are free to make further imports throughout the notebook as you wish.

```
import tensorflow as tf
from scipy.io import loadmat
```

```
#! pip install tensorflow==2
print(tf.__version__)
```

```
2.0.0
```

For the capstone project, you will use the [SVHN dataset](). This is an image dataset of over 600,000 digit images in all, and is a harder dataset than MNIST as the numbers appear in the context of natural scene images. SVHN is obtained from house numbers in Google Street View images.

- Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng. "Reading Digits in Natural Images with Unsupervised Feature Learning". NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

The train and test datasets required for this project can be downloaded from [here]() and [here](). Once unzipped, you will have two files: `train_32x32.mat` and `test_32x32.mat`. You should store these files in Drive for use in this Colab notebook.

Your goal is to develop an end-to-end workflow for building, training, validating, evaluating and saving a neural network that classifies a real-world image into one of ten classes.

```
# Run this cell to connect to your Drive folder

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mo
```

```
# Load the dataset from your Drive folder

train = loadmat('gdrive/MyDrive/content/train_32x32.mat')
test = loadmat('gdrive/MyDrive/content/test_32x32.mat')
```

Inspect Images on Gray Scale

Both `train` and `test` are dictionaries with keys `X` and `y` for the input images and labels respectively.

1. Inspect and preprocess the dataset

- Extract the training and testing images and labels separately from the train and test dictionaries loaded for you.
- Select a random sample of images and corresponding labels from the dataset (at least 10), and display them in a figure.
- Convert the training and test images to grayscale by taking the average across all colour channels for each pixel. *Hint: retain the channel dimension, which will now have size 1.*
- Select a random sample of the grayscale images and corresponding labels from the dataset (at least 10), and display them in a figure.

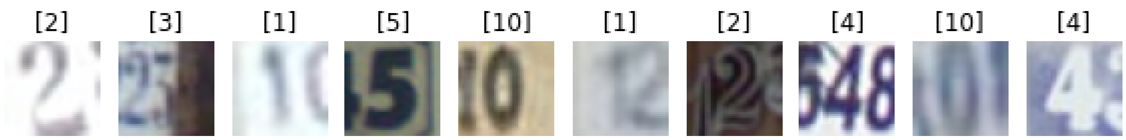
```
import numpy as np
import pandas as pd
```

```
import pandas as pd
from tensorflow.keras.utils import to_categorical


print('#########Print the Keys in the dictionary')
print(train.keys())


# Extract the training data and corresponding targets
train_data = np.array(train['X'])
train_targets = np.array(train['y'])


test_data = np.array(test['X'])
test_targets = np.array(test['y'])

#train_data = train_data/255
#test_data = test_data/255


distinctTypes=10
#train_targets = to_categorical(train_targets)
#test_targets = to_categorical(test_targets)




print('\n#########Traning data and targets shape are as follows')
print(train_data.shape)
print(train_targets.shape)

print('\n##########Reorder the Axis appropriately to extract images and print shape again')
train_data = np.moveaxis(train_data, -1, 0)
test_data = np.moveaxis(test_data, -1, 0)

print(train_data.shape)
print(train_targets.shape)
```

```
#########Print the Keys in the dictionary
dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])

#########Traning data and targets shape are as follows
(32, 32, 3, 73257)
(73257, 1)

##########Reorder the Axis appropriately to extract images and print shape again
(73257, 32, 32, 3)
(73257, 1)
```

```
#Using this for printint color images
from tensorflow.keras.preprocessing import image
%matplotlib inline
import matplotlib.pyplot as plt
def printImagesV1(noofImages):
```

```
def printImagesV1(noofImages):
  train_data_random =  np.random.choice(train_data.shape[0], noofImages)
  drfig,draxis = plt.subplots(1, noofImages, figsize=(10, 1))
  for i in range(noofImages):
    draxis[i].set_axis_off()
    draxis[i].imshow(np.squeeze(train_data[train_data_random[i]]), cmap="gray")
    draxis[i].set_title("{}".format(train_targets[train_data_random[i]]))

printImagesV1(10)
```




```
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing import image
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np


train_data_modified = np.mean(train_data, axis=-1, keepdims=True)/255
test_targets_modified = np.mean(train_targets, axis=-1, keepdims=True)/255

def printImagesOnGrayV1(noofImages):
  train_data_random =  np.random.choice(train_data_modified.shape[0], noofImages)
  drfig,dtaxis = plt.subplots(1, noofImages, figsize=(noofImages, 1))
  for index in range(noofImages):
    dtaxis[index].set_axis_off()
    dtaxis[index].imshow(np.squeeze(train_data_modified[train_data_random[index]]), cmap="gra
    dtaxis[index].set_title("{}".format(test_targets_modified[train_data_random[index]]))


printImagesOnGrayV1(10)
```

```
# training  and testing data has to be  'float64' type

train_data = train_data.astype('float64')
test_data = test_data.astype('float64')


# training  and testing targets has to be  'int64' type

train_targets = train_targets.astype('int64')
test_targets = test_targets.astype('int64')
```

```
# normalizatioin needed to avoid vanishing gradient and fast convergence

train_data /= 255.0
test_data /= 255.0


# Assign a value to each output/target in  categorical feature.
from sklearn.preprocessing import LabelBinarizer
lb = LabelBinarizer()
train_targets = lb.fit_transform(train_targets)
test_targets = lb.fit_transform(test_targets)


from tensorflow.keras.callbacks import ModelCheckpoint
def get_checkpoint_every_epoch(checkpoint_path):
    #checkpoint_path = 'checkpoints_every_epoch/checkpoint_{epoch:03d}'
    checkpoint = ModelCheckpoint(filepath=checkpoint_path,frequencey='epoch',save_weights_onl
    return checkpoint


def get_early_stopping():
    earlystop = tf.keras.callbacks.EarlyStopping(
        monitor='val_accuracy', patience=3)
    return earlystop



def get_checkpoint_best_only(checkpoint_best_path):
    #checkpoint_best_path='checkpoints_best_only/checkpoint'
    checkpoint_best = ModelCheckpoint(filepath=checkpoint_best_path,
            save_weights_only=True,
            save_freq='epoch',
            monitor='val_accuracy',
            save_best_only=True,
            verbose=1)
    return checkpoint_best
```

2. MPL neural network classifier

- Build an MLP classifier model using the Sequential API. Your model should use only Flatten and Dense layers, with the final layer having a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different MLP architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 4 or 5 layers.*
- Print out the model summary (using the summary() method)
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.

- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- As a guide, you should aim to achieve a final categorical cross entropy training loss of less than 1.0 (the validation loss might be higher).
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D

def get_MPL(input_shape):
    model_ret = Sequential([

        Flatten(),
        Dense(128, activation='relu',input_shape=input_shape),
        Dense(128,activation='relu'),
        Dense(128,activation='relu'),
        Dense(128,activation='relu'),
        Dense(128,activation='relu'),
        Dense(128,activation='relu'),
        Dense(128,activation='relu'),
        Dense(128,activation='relu'),

        Dense(128,activation='relu'),
        Dense(10,activation='softmax'),
    ])
    model_ret.compile(optimizer='adam',
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])

    return model_ret


model = get_MPL(train_data.shape)
```

```
#model.summary()
```

```
print(train_data.shape)
print(train_targets.shape)
```

```
(73257, 32, 32, 3)
(73257, 10)
```

Create Checkpoints for epochs early stopp and best weights

```
checkpoint_every_epoch_mpl = get_checkpoint_every_epoch('checkpoints_every_epoch_mpl/checkpoi
checkpoint_best_only_mpl = get_checkpoint_best_only('checkpoints_best_only_mpl/checkpoint')
early_stopping = get_early_stopping()

checkpoint_every_epoch = get_checkpoint_every_epoch('checkpoints_every_epoch/checkpoint_{epoc
checkpoint_best_only = get_checkpoint_best_only('checkpoints_best_only/checkpoint')
```

**Summary of MPL Model **

```
callbacks = [checkpoint_every_epoch_mpl, checkpoint_best_only_mpl]
history_mpl = model.fit(train_data,train_targets, epochs=70, batch_size=250, verbose=2, valic
```

```
Epoch 00060: saving model to checkpoints_every_epoch_mpl/checkpoint_060

Epoch 00060: val_accuracy did not improve from 0.78979
58605/58605 - 7s - loss: 0.3824 - accuracy: 0.8730 - val_loss: 0.8265 - val_accuracy:
Epoch 61/70

Epoch 00061: saving model to checkpoints_every_epoch_mpl/checkpoint_061

Epoch 00061: val_accuracy did not improve from 0.78979
58605/58605 - 6s - loss: 0.3937 - accuracy: 0.8694 - val_loss: 0.8628 - val_accuracy:
Epoch 62/70

Epoch 00062: saving model to checkpoints_every_epoch_mpl/checkpoint_062

Epoch 00062: val_accuracy did not improve from 0.78979
58605/58605 - 6s - loss: 0.3876 - accuracy: 0.8712 - val_loss: 0.8727 - val_accuracy:
Epoch 63/70

Epoch 00063: saving model to checkpoints_every_epoch_mpl/checkpoint_063


Epoch 00063: val_accuracy did not improve from 0.78979
58605/58605 - 6s - loss: 0.3822 - accuracy: 0.8714 - val_loss: 0.8773 - val_accuracy:
Epoch 64/70

Epoch 00064: saving model to checkpoints_every_epoch_mpl/checkpoint_064

Epoch 00064: val_accuracy did not improve from 0.78979
58605/58605 - 6s - loss: 0.3787 - accuracy: 0.8739 - val_loss: 0.8861 - val_accuracy:
Epoch 65/70

Epoch 00065: saving model to checkpoints_every_epoch_mpl/checkpoint_065

Epoch 00065: val_accuracy did not improve from 0.78979
58605/58605 - 6s - loss: 0.3656 - accuracy: 0.8783 - val_loss: 0.8887 - val_accuracy:
Epoch 66/70
```

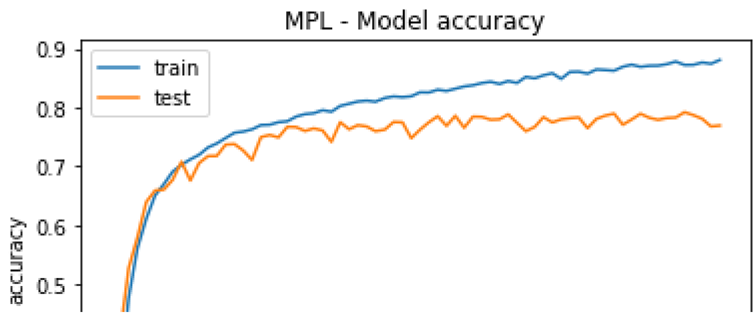
```
Epoch 00066: saving model to checkpoints_every_epoch_mpl/checkpoint_066

Epoch 00066: val_accuracy improved from 0.78979 to 0.79211, saving model to checkpoir
58605/58605 - 6s - loss: 0.3797 - accuracy: 0.8725 - val_loss: 0.8671 - val_accuracy:
Epoch 67/70

Epoch 00067: saving model to checkpoints_every_epoch_mpl/checkpoint_067

Epoch 00067: val_accuracy did not improve from 0.79211
58605/58605 - 6s - loss: 0.3804 - accuracy: 0.8727 - val_loss: 0.8825 - val_accuracy:
Epoch 68/70

Epoch 00068: saving model to checkpoints_every_epoch_mpl/checkpoint_068

Epoch 00068: val_accuracy did not improve from 0.79211
58605/58605 - 6s - loss: 0.3689 - accuracy: 0.8767 - val_loss: 0.8521 - val_accuracy:
Epoch 69/70

Epoch 00069: saving model to checkpoints_every_epoch_mpl/checkpoint_069

Epoch 00069: val_accuracy did not improve from 0.79211
58605/58605 - 6s - loss: 0.3707 - accuracy: 0.8746 - val_loss: 0.9473 - val_accuracy:
Epoch 70/70
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            multiple                  0
_________________________________________________________________
dense (Dense)                multiple                  393344
_________________________________________________________________
dense_1 (Dense)              multiple                  16512
_________________________________________________________________
dense_2 (Dense)              multiple                  16512
_________________________________________________________________
dense_3 (Dense)              multiple                  16512
_________________________________________________________________
dense_4 (Dense)              multiple                  16512
_________________________________________________________________
dense_5 (Dense)              multiple                  16512
_________________________________________________________________
dense_6 (Dense)              multiple                  16512
_________________________________________________________________
dense_7 (Dense)              multiple                  16512
_________________________________________________________________
dense_8 (Dense)              multiple                  16512
_________________________________________________________________
dense_9 (Dense)              multiple                  1290
=================================================================
Total params: 526,730
Trainable params: 526,730
Non-trainable params: 0
_________________________________________________________________
```

Plot Graph for Loss Accuracy FOR MPL Model

```
# PLOT THE GRAPH FOR - history- PUT  ACCURACY VS VAL ACCURACY AGAINST EPOCH
def plotAccuracyVsEpochForTrainMPL():
  plt.plot(history_mpl.history['accuracy'])
  plt.plot(history_mpl.history['val_accuracy'])
  plt.title('MPL - Model accuracy')
  plt.ylabel('accuracy')
  plt.xlabel('epoch')
  plt.legend(['train', 'test'], loc='upper left')
  plt.show()

# PLOT THE GRAPH FOR - history- PUT  LOSS VS VAL LOSS AGAINST EPOCH
def plotAccuracyVsEpochForTestMPL():
  plt.plot(history_mpl.history['loss'])
  plt.plot(history_mpl.history['val_loss'])
  plt.title('MPL - Model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'test'], loc='upper left')
  plt.show()

#CALL ABOVE API TO PLOT FOR TRAIN DATA
plotAccuracyVsEpochForTrainMPL()

#CALL ABOVE API TO PLOT FOR TEST DATA
plotAccuracyVsEpochForTestMPL()
```



3. CNN neural network classifier

- Build a CNN classifier model using the Sequential API. Your model should use the Conv2D, MaxPool2D, BatchNormalization, Flatten, Dense and Dropout layers. The final layer should again have a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different CNN architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 2 or 3 convolutional layers and 2 fully connected layers.)*
- The CNN model should use fewer trainable parameters than your MLP model.
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- You should aim to beat the MLP model performance with fewer parameters!
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras.layers import Dropout

def get_CNN(input_shape, dropout_rate, weight_decay):
  model =  Sequential([
    Conv2D(32, (3, 3), padding='same',
                       activation='relu',
                       input_shape=(32, 32, 3)),
    BatchNormalization(),
    Conv2D(32, (3, 3), padding='same',
                    activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(weight_decay),

    Conv2D(64, (3, 3), padding='same',
                       activation='relu'),
```

```
        BatchNormalization(),
        Conv2D(64, (3, 3), padding='same',
                           activation='relu'),
        MaxPooling2D((2, 2)),
        Dropout(weight_decay),

        Conv2D(128, (3, 3), padding='same',
                              activation='relu'),
        BatchNormalization(),
        Conv2D(128, (3, 3), padding='same',
                           activation='relu'),
        MaxPooling2D((2, 2)),
        Dropout(weight_decay),

        Flatten(),
        Dense(128, activation='relu'),
        Dropout(weight_decay),
        Dense(10,  activation='softmax')
      ])
      return model
```

```
reg_model = get_CNN(train_data.shape, 0.3, 0.001)
reg_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
batch_normalization_6 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_7 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_9 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
```

```
batch_normalization_8 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_10 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               262272
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                1290
=================================================================
Total params: 551,466
Trainable params: 551,018
Non-trainable params: 448
_________________________________________________________________
```

```
from tensorflow import keras
def compile_model(model):
  opt = keras.optimizers.Adam(learning_rate=0.0001)
  model.compile(optimizer=opt,loss="categorical_crossentropy",metrics=["acc"])
```

```
compile_model(reg_model)
```

```
callbacks = [checkpoint_every_epoch, checkpoint_best_only]
history_cnn = reg_model.fit(train_data,train_targets, epochs=10, batch_size=250, verbose=2, \
```

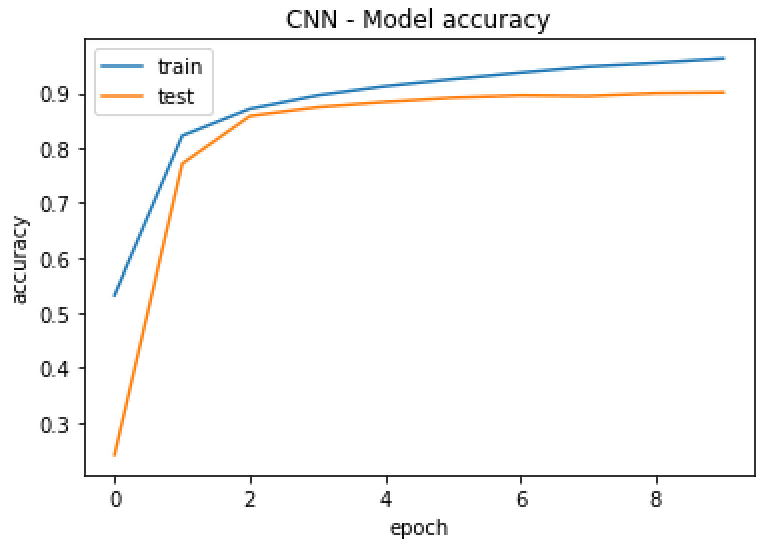
```
Train on 58605 samples, validate on 14652 samples
Epoch 1/10

Epoch 00001: saving model to checkpoints_every_epoch/checkpoint_001
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 563s - loss: 1.4080 - acc: 0.5326 - val_loss: 2.0798 - val_acc: 0.2410
Epoch 2/10

Epoch 00002: saving model to checkpoints_every_epoch/checkpoint_002
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 560s - loss: 0.5831 - acc: 0.8226 - val_loss: 0.7664 - val_acc: 0.7713
Epoch 3/10

Epoch 00003: saving model to checkpoints_every_epoch/checkpoint_003
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 562s - loss: 0.4328 - acc: 0.8716 - val_loss: 0.4728 - val_acc: 0.8583
Epoch 4/10

Epoch 00004: saving model to checkpoints_every_epoch/checkpoint_004
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 563s - loss: 0.3541 - acc: 0.8961 - val_loss: 0.4216 - val_acc: 0.8746
Epoch 5/10
```

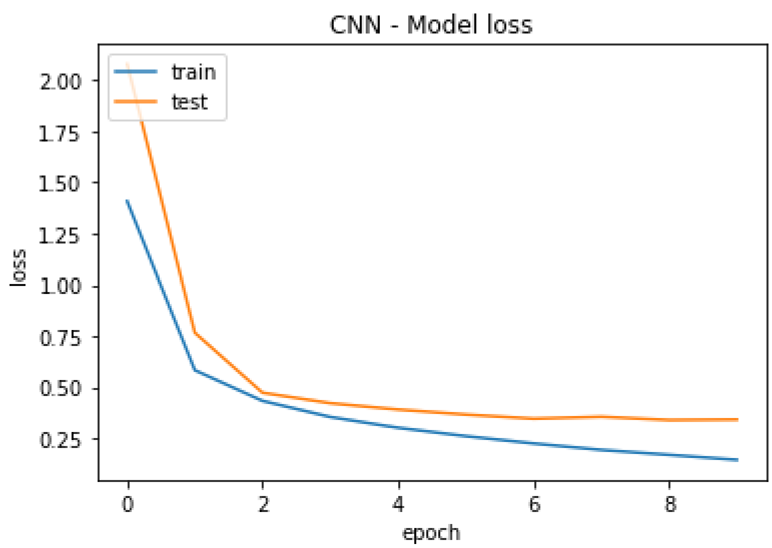
```
Epoch 00005: saving model to checkpoints_every_epoch/checkpoint_005
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 563s - loss: 0.3010 - acc: 0.9128 - val_loss: 0.3903 - val_acc: 0.8842
Epoch 6/10

Epoch 00006: saving model to checkpoints_every_epoch/checkpoint_006
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 561s - loss: 0.2600 - acc: 0.9258 - val_loss: 0.3657 - val_acc: 0.8920
Epoch 7/10

Epoch 00007: saving model to checkpoints_every_epoch/checkpoint_007
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 563s - loss: 0.2243 - acc: 0.9376 - val_loss: 0.3457 - val_acc: 0.8962
Epoch 8/10

Epoch 00008: saving model to checkpoints_every_epoch/checkpoint_008
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 565s - loss: 0.1927 - acc: 0.9487 - val_loss: 0.3551 - val_acc: 0.8949
Epoch 9/10

Epoch 00009: saving model to checkpoints_every_epoch/checkpoint_009
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 564s - loss: 0.1687 - acc: 0.9555 - val_loss: 0.3390 - val_acc: 0.9000
Epoch 10/10

Epoch 00010: saving model to checkpoints_every_epoch/checkpoint_010
WARNING:tensorflow:Can save best model only with val_accuracy available, skipping.
58605/58605 - 561s - loss: 0.1442 - acc: 0.9636 - val_loss: 0.3414 - val_acc: 0.9014
```

PLOT GRAPH FOR LOSS ACCURRACY EPOCHS

```
#PLOTTING FOR CNN- ACCUARCY VS VAL ACCURACY AGAINST EPOCH
def plotLossAccuracyEpochForTrain():
  plt.plot(history_cnn.history['acc'])
  plt.plot(history_cnn.history['val_acc'])
  plt.title('CNN - Model accuracy')
  plt.ylabel('accuracy')
  plt.xlabel('epoch')
  plt.legend(['train', 'test'], loc='upper left')
  plt.show()

#PLOTTING FOR CNN- LOSS VS VAL LOSS AGAINST EPOCH
def plotLossAccuracyEpochForTest():
  plt.plot(history_cnn.history['loss'])
  plt.plot(history_cnn.history['val_loss'])
  plt.title('CNN - Model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'test'], loc='upper left')
  plt.show()

#CALL ABOVE API TO PLOT FOR TRAIN DATA
plotLossAccuracyEpochForTrain()
```

```
#CALL ABOVE API TO PLOT FOR TEST DATA
plotLossAccuracyEpochForTest()
```





4. Get model predictions

- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the final model prediction given by the label with maximum probability.

> Model has been trained with Both MLP and CNN. Now start writing functions to load the model with weights

Function to Load MPL with latest weights

```
def load_MPL_weight_Latest(model):
    #model = get_new_model(x_train[0].shape)
    latest_checkpoint_dir = tf.train.latest_checkpoint('checkpoints_every_epoch_mpl', latest_
    model.load_weights(latest_checkpoint_dir)
    return model
```

Function to load MPL with BEST Weights

```
def load_MPL_weight_Best(model):
    #model = get_new_model(x_train[0].shape)
    checkpoint_path = 'checkpoints_best_only_mpl/checkpoint'
    model.load_weights(checkpoint_path)
    return model
```

Function to load CNN with latest weights

```
def load_CNN_weight_Latest(model):
    #model = get_new_model(x_train[0].shape)
    latest_checkpoint_dir = tf.train.latest_checkpoint('checkpoints_every_epoch', latest_file
    model.load_weights(latest_checkpoint_dir)
    return model
```

Function to load CNN with Best Weights

```
def load_CNN_weight_Best(model):
    #model = get_new_model(x_train[0].shape)
    checkpoint_path = 'checkpoints_best_only/checkpoint'
    model.load_weights(checkpoint_path)
    return model
```

Utility function to show test images in a given index range

```
def printTestImages(noofImages):
    plt.cla()
    index = 0
    print('\n#########Showing Images')
    for i in range(noofImages):
      img = test_data[i]
      plt.imshow(img, cmap="Greys")
      plt.show()
      plt.title(test_targets[i])
      plt.show()
```

Function to show image of a given index in test_data. Input is index

```
def printGivenImages(index):
    plt.cla()

    img = test_data[index]
    plt.imshow(img, cmap="Greys")
    #plt.show()
    plt.title(test_targets[index])
    plt.show()
```

PREDICT WITH MPL

GET THE INSTANCE OF MPL AND LOAD BEST WEIGHT. COMPILE AND PREDICT test_data. OUTPUT is predict_targets

```
    model = get_MPL(test_data.shape)
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    model = load_MPL_weight_Best(model)
    predict_targets  = model.predict(test_data,verbose=False)
    #print(predict_targets[3])
    #print(test_targets[3])
    #printGivenImages(3)
```

FUNCTION SHOWS THE DETAILS OF MLP PREDICTED DATA ALONG WITH IMAGE.

predict_targets from above will be used in this method to show details. THIS IS FOR MPL ONLY

```
def showPredictedDataAndOrigianl():
  noofImages  = 5
  predict_targets_random =  np.random.choice(predict_targets.shape[0], noofImages)
  for i in range(noofImages):
    print('\n#########Showing Image for ')
    count = np.argmax(predict_targets[i])
```

```
    count = np.argmax(predict_targets[i])
    print('Result from Prediction As ',count)
    print('Actual from Data       ',np.argmax(test_targets[i]))
    printGivenImages(i)
```

CALL showPredictedDataAndOrigianl() function to show the predcited data and origianal output in test_targets

```
showPredictedDataAndOrigianl()
```

PREDICT WITH CNN

GET THE INSTANCE OF CNN AND LOAD BEST WEIGHT. COMPILE AND PREDICT test_data. OUTPUT is predict_targets_cnn

```
reg_model = get_CNN(train_data.shape, 0.3, 0.001)

reg_model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
reg_model = load_CNN_weight_Latest(reg_model)
predict_targets_cnn  = reg_model.predict(test_data,verbose=False)
    #print(predict_targets_cnn[3])
    #print(test_targets[3])
    #printGivenImages(3)
```

FUNCTION SHOWS THE DETAILS OF CNN PREDICTED DATA ALONG WITH IMAGE.

predict_targets from above will be used in this method to show details. THIS IS FOR CNN ONLY

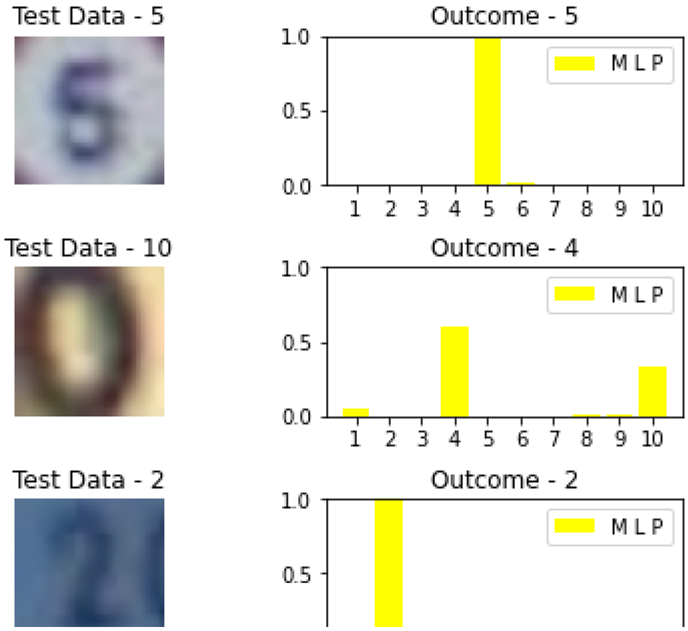
```
def showPredictedDataAndOriginal_CNN():
  for i in range(5):
    print('\n#########Showing Image for ')
    print('Result from Prediction ',np.argmax(predict_targets_cnn[i]))
    print('Actual from Data       ',np.argmax(test_targets[i]))
    printGivenImages(i)
```

CALL showPredictedDataAndOriginal_CNN() function to show the predicted data from CNN and origianal output in test_targets

```
showPredictedDataAndOriginal_CNN()
```

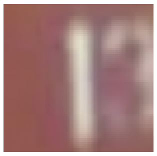
** Distribution Chart**

```
# FUNCTION TO SHOW DISTRIBUTION OF OUTCOME AND TEST
from matplotlib.gridspec import GridSpec

def categoricalOutputDistributionMLPV1(noOfSample):

  selectIndexRandom = np.random.choice(test_data.shape[0], noOfSample)
  wrapLaoutFig = plt.figure(constrained_layout=True, figsize=(8, 8))

  gridObject = wrapLaoutFig.add_gridspec(ncols=3, nrows=noOfSample)
  for index in range(noOfSample):
    currentSelIndex = selectIndexRandom[index]
    ax = wrapLaoutFig.add_subplot(gridObject[index,0])
    ax.set_axis_off()

    ax.imshow(test_data[currentSelIndex])

    ax.set_title("Test Data - {}".format(np.argmax(test_targets[currentSelIndex])+1))
    #print("Test Date - {}",np.argmax(test_targets[currentSelIndex]))

    ax = wrapLaoutFig.add_subplot(gridObject[index,1])

    xaix=[1,2,3,4,5,6,7,8,9,10]
    ax.bar(xaix, predict_targets[currentSelIndex], color=(1., 1., 0., 1.), label="Multi Layer

    ax.legend()

    ax.set_xticks(xaix)
    ax.set_ylim((0, 1))
    ax.set_title("Outcome - {}".format(np.argmax(predict_targets[currentSelIndex])+1))



# Show prediction
categoricalOutputDistributionV1(5)
```

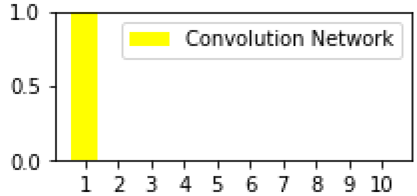



```
# FUNCTION TO SHOW DISTRIBUTION OF OUTCOME AND TEST DATA FOR CNN
from matplotlib.gridspec import GridSpec

def categoricalOutputDistributionCNNV1(noOfSample):
  selectIndexRandom = np.random.choice(test_data.shape[0], noOfSample)
  wrapLaoutFig = plt.figure(constrained_layout=True, figsize=(8, 8))
  gridObject = wrapLaoutFig.add_gridspec(ncols=3, nrows=noOfSample)
  for index in range(noOfSample):
    currentSelIndex = selectIndexRandom[index]
    drawaxis = wrapLaoutFig.add_subplot(gridObject[index,0])
    drawaxis.set_title("Test Data - {}".format(np.argmax(test_targets[currentSelIndex])+1))
    drawaxis.set_axis_off()
    drawaxis.imshow(test_data[currentSelIndex])



    drawaxis = wrapLaoutFig.add_subplot(gridObject[index,1])
    drawaxis.set_title("Outcome - {}".format(np.argmax(predict_targets_cnn[currentSelIndex])+
    xaix=[1,2,3,4,5,6,7,8,9,10]
    drawaxis.set_xticks(xaix)
    drawaxis.set_ylim((0, 1))


    drawaxis.bar(xaix, predict_targets_cnn[currentSelIndex], color=(1., 1., 0., 1.), label="(
    drawaxis.legend()



categoricalOutputDistributionCNNV1(5)
```

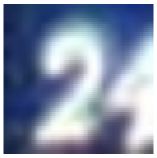
Test Data - 1



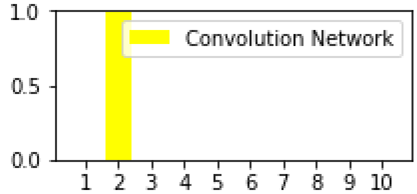
Outcome - 1



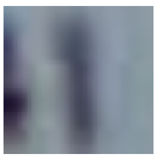
Test Data - 2



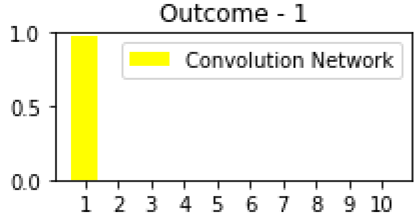
Outcome - 2



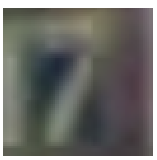
Test Data - 1



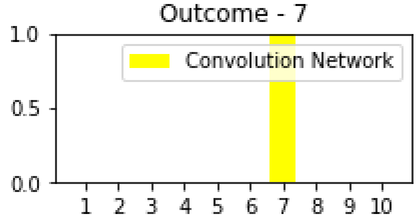
Outcome - 1



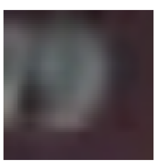
Test Data - 7



Outcome - 7



Test Data - 9



Outcome - 9