Capstone_Project

February 1, 2021

1 Capstone Project

1.1 Image classifier for the SVHN dataset

1.1.1 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.

1.1.2 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.

1.1.3 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 tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization,␣
 ↪Dropout, Conv2D, MaxPool2D
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from scipy.io import loadmat
import matplotlib.pyplot as plt
import numpy as np
```

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')
```

```
Mounted at /content/gdrive
```

[ ]:
```
# Load the dataset from your Drive folder

train = loadmat('gdrive/MyDrive/Colab Notebooks/data/train_32x32.mat')
test = loadmat('gdrive/MyDrive/Colab Notebooks/data/test_32x32.mat')
```

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

1.2 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.

[ ]:
```
X_train = train['X']
y_train = train['y']
X_test = test['X']
y_test = test['y']
```

[ ]:
```
# Checking the shapes

print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
print('X_test shape:', X_test.shape)
```

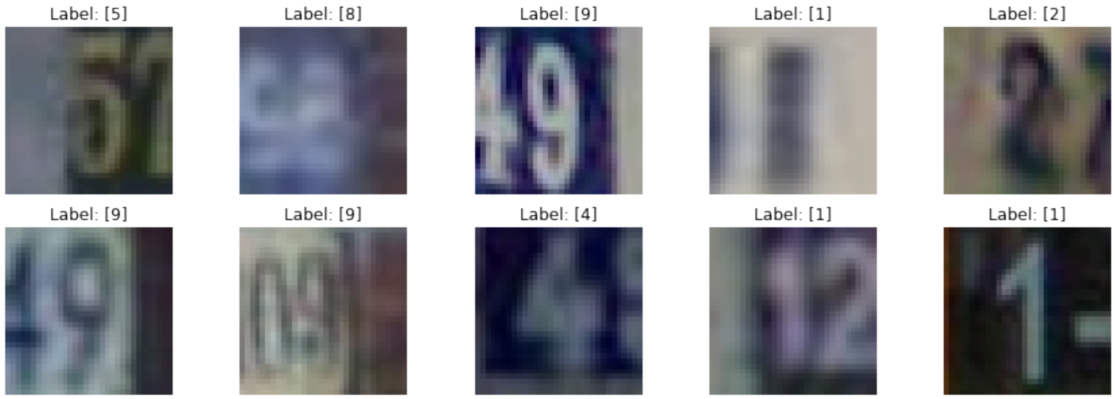
```
print('y_test shape:', y_test.shape)
```

```
X_train shape: (32, 32, 3, 73257)
y_train shape: (73257, 1)
X_test shape: (32, 32, 3, 26032)
y_test shape: (26032, 1)
```

[ ]:
```
# For training we need the batch dimension in the first position

X_train = np.moveaxis(X_train, -1, 0)
X_test = np.moveaxis(X_test, -1, 0)

print('X_train shape:', X_train.shape)
print('X_test shape:', X_train.shape)
```

```
X_train shape: (73257, 32, 32, 3)
X_test shape: (73257, 32, 32, 3)
```

[ ]:
```
%matplotlib inline

random_sample = np.random.randint(0, 73256, 10)
# Creates an random array to select images from the dataset

fig, axs = plt.subplots(2,5, figsize=(15,5))

j = 0

for i,v in enumerate(random_sample):
  if i>=5:
    j = 1
    i -= 5
  axs[j][i].set_axis_off()
  axs[j][i].imshow(X_train[v,:,:,:])
  caption = 'Label: ' + str(y_train[v,:])
  axs[j][i].set_title(caption)

fig.show()
```



[ ]:
```
# Now we convert the training an testing images to grayscale

X_train = X_train.mean(axis=3, keepdims=True)
X_test = X_test.mean(axis=3, keepdims=True)
```

[ ]:
```
# If we set the last layer to have a softmax activation with 10 units,
# and we use sparse categorical crossentropy, this will try to categorize
# the labels in the range from 0 to 9, but the tags are given from 1 to 10,
# therefore, we need to change de 10s by 0s to keep the tags between the
# needed range.

y_train = np.where(y_train==10, 0, y_train)
y_test = np.where(y_test==10, 0, y_test)
```

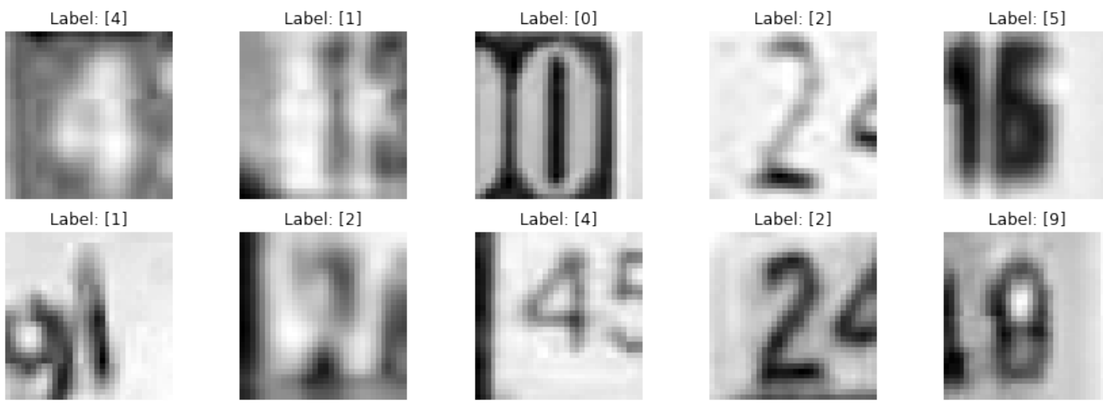
[ ]:
```
# We divide by 255 to get all the values between 0 and 1.

X_train = X_train  / 255
X_test = X_test / 255
```

[ ]:
```
# Checking the shape after these changes

print('X_train shape:', X_train.shape)
print('X_test shape:', X_train.shape)
```

```
X_train shape: (73257, 32, 32, 1)
X_test shape: (73257, 32, 32, 1)
```

[ ]:
```
%matplotlib inline

random_sample = np.random.randint(0, 73256, 10)
# Creates an random array to select images from the dataset

fig, axs = plt.subplots(2,5, figsize=(15,5))
```

```python
j = 0

for i,v in enumerate(random_sample):
  if i>=5:
    j = 1
    i -= 5
  axs[j][i].set_axis_off()
  axs[j][i].imshow(X_train[v,:,:,0], cmap='gray')
  caption = 'Label: ' + str(y_train[v,:])
  axs[j][i].set_title(caption)

fig.show()
```



1.3 2. MLP 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.

[ ]:
```python
MLP_model = Sequential([
                        Flatten(input_shape=X_train[0].shape),
```

```
                        Dense(1024, activation='relu', kernel_regularizer=tf.
 ↪keras.regularizers.L2(l2=0.0003)),
                        Dense(512, activation='relu'),
                        Dense(256, activation='relu'),
                        Dense(128, activation='relu'),
                        Dense(10, activation='softmax')
])
```

[ ]:
```
MLP_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              1049600
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,739,914
Trainable params: 1,739,914
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
opt = tf.keras.optimizers.Adam(learning_rate=0.0002)

MLP_model.compile(optimizer=opt, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
# Instantiation of the callbacks

checkpoint_path = 'gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/
 ↪checkpoint'
checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_loss', verbose=1,
                             save_best_only=True, save_weights_only=True)

earlystopping = EarlyStopping(monitor='val_loss', patience=5, verbose=0)
```

[ ]:
```
MLP_history = MLP_model.fit(X_train, y_train, epochs=30, batch_size=64,
                    callbacks=[checkpoint, earlystopping],
                    validation_split=0.15, verbose=1)
```

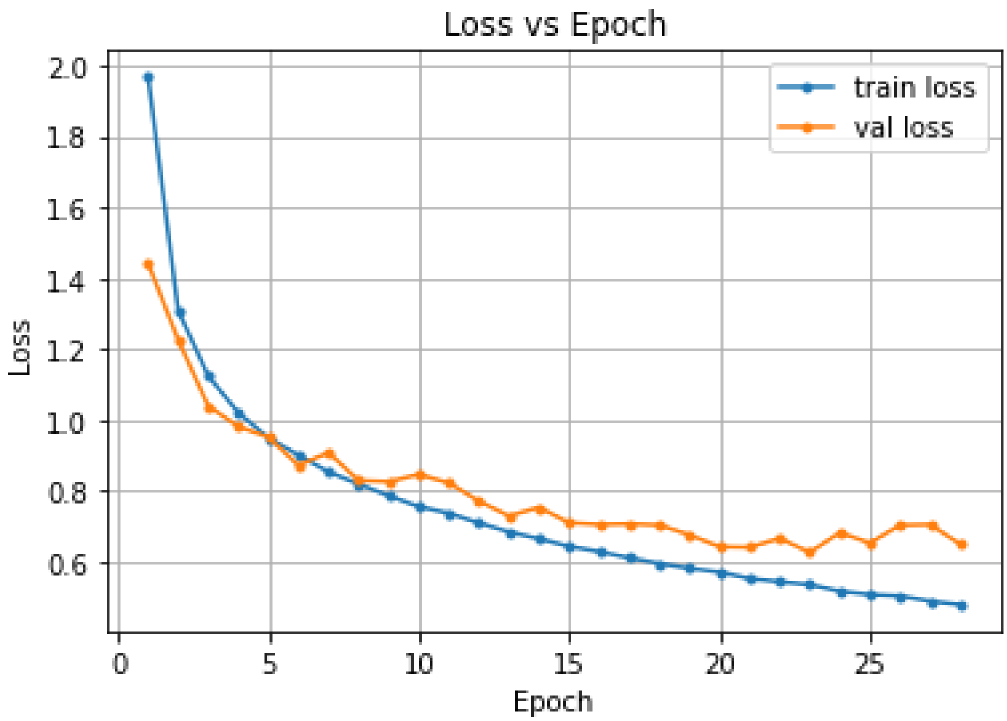
```
Epoch 1/30
973/973 [==============================] - 5s 3ms/step - loss: 2.2935 -
accuracy: 0.2613 - val_loss: 1.4424 - val_accuracy: 0.5792

Epoch 00001: val_loss improved from inf to 1.44236, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 2/30
973/973 [==============================] - 3s 3ms/step - loss: 1.3729 -
accuracy: 0.6057 - val_loss: 1.2258 - val_accuracy: 0.6475

Epoch 00002: val_loss improved from 1.44236 to 1.22584, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 3/30
973/973 [==============================] - 3s 3ms/step - loss: 1.1552 -
accuracy: 0.6781 - val_loss: 1.0397 - val_accuracy: 0.7057

Epoch 00003: val_loss improved from 1.22584 to 1.03970, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 4/30
973/973 [==============================] - 3s 3ms/step - loss: 1.0453 -
accuracy: 0.7057 - val_loss: 0.9813 - val_accuracy: 0.7220

Epoch 00004: val_loss improved from 1.03970 to 0.98135, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 5/30
973/973 [==============================] - 3s 3ms/step - loss: 0.9506 -
accuracy: 0.7390 - val_loss: 0.9519 - val_accuracy: 0.7288

Epoch 00005: val_loss improved from 0.98135 to 0.95185, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 6/30
973/973 [==============================] - 3s 3ms/step - loss: 0.9098 -
accuracy: 0.7484 - val_loss: 0.8716 - val_accuracy: 0.7615

Epoch 00006: val_loss improved from 0.95185 to 0.87165, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 7/30
973/973 [==============================] - 3s 3ms/step - loss: 0.8552 -
accuracy: 0.7634 - val_loss: 0.9079 - val_accuracy: 0.7460

Epoch 00007: val_loss did not improve from 0.87165
Epoch 8/30
973/973 [==============================] - 3s 3ms/step - loss: 0.8174 -
accuracy: 0.7773 - val_loss: 0.8297 - val_accuracy: 0.7761

Epoch 00008: val_loss improved from 0.87165 to 0.82969, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 9/30
```

```
973/973 [==============================] - 3s 3ms/step - loss: 0.7811 -
accuracy: 0.7843 - val_loss: 0.8265 - val_accuracy: 0.7760

Epoch 00009: val_loss improved from 0.82969 to 0.82645, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 10/30
973/973 [==============================] - 3s 3ms/step - loss: 0.7563 -
accuracy: 0.7927 - val_loss: 0.8465 - val_accuracy: 0.7690

Epoch 00010: val_loss did not improve from 0.82645
Epoch 11/30
973/973 [==============================] - 3s 3ms/step - loss: 0.7398 -
accuracy: 0.7978 - val_loss: 0.8229 - val_accuracy: 0.7718

Epoch 00011: val_loss improved from 0.82645 to 0.82289, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 12/30
973/973 [==============================] - 3s 3ms/step - loss: 0.7105 -
accuracy: 0.8095 - val_loss: 0.7711 - val_accuracy: 0.7921

Epoch 00012: val_loss improved from 0.82289 to 0.77105, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 13/30
973/973 [==============================] - 3s 3ms/step - loss: 0.6746 -
accuracy: 0.8188 - val_loss: 0.7295 - val_accuracy: 0.8066

Epoch 00013: val_loss improved from 0.77105 to 0.72947, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 14/30
973/973 [==============================] - 3s 3ms/step - loss: 0.6693 -
accuracy: 0.8187 - val_loss: 0.7536 - val_accuracy: 0.7973

Epoch 00014: val_loss did not improve from 0.72947
Epoch 15/30
973/973 [==============================] - 3s 3ms/step - loss: 0.6422 -
accuracy: 0.8278 - val_loss: 0.7101 - val_accuracy: 0.8124

Epoch 00015: val_loss improved from 0.72947 to 0.71012, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 16/30
973/973 [==============================] - 3s 3ms/step - loss: 0.6271 -
accuracy: 0.8314 - val_loss: 0.7052 - val_accuracy: 0.8100

Epoch 00016: val_loss improved from 0.71012 to 0.70515, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 17/30
973/973 [==============================] - 3s 3ms/step - loss: 0.6105 -
accuracy: 0.8386 - val_loss: 0.7069 - val_accuracy: 0.8125
```

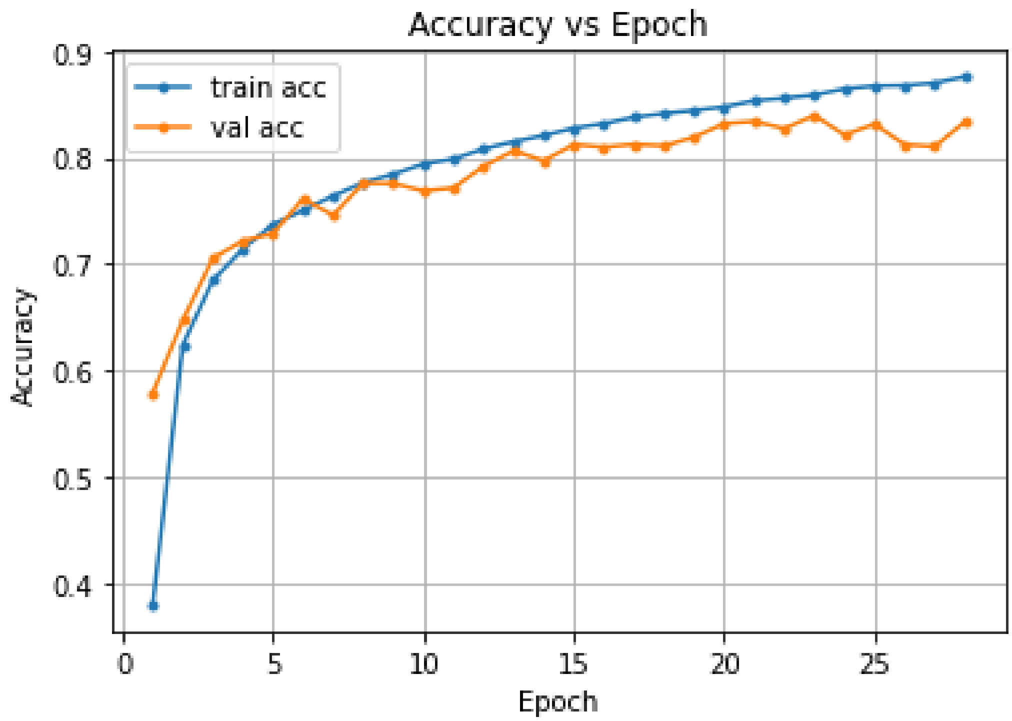
```
Epoch 00017: val_loss did not improve from 0.70515
Epoch 18/30
973/973 [==============================] - 3s 3ms/step - loss: 0.5926 -
accuracy: 0.8433 - val_loss: 0.7036 - val_accuracy: 0.8112

Epoch 00018: val_loss improved from 0.70515 to 0.70357, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 19/30
973/973 [==============================] - 3s 3ms/step - loss: 0.5773 -
accuracy: 0.8458 - val_loss: 0.6758 - val_accuracy: 0.8194

Epoch 00019: val_loss improved from 0.70357 to 0.67578, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 20/30
973/973 [==============================] - 3s 3ms/step - loss: 0.5653 -
accuracy: 0.8498 - val_loss: 0.6423 - val_accuracy: 0.8321

Epoch 00020: val_loss improved from 0.67578 to 0.64232, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 21/30
973/973 [==============================] - 3s 3ms/step - loss: 0.5500 -
accuracy: 0.8552 - val_loss: 0.6408 - val_accuracy: 0.8341

Epoch 00021: val_loss improved from 0.64232 to 0.64083, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 22/30
973/973 [==============================] - 3s 3ms/step - loss: 0.5380 -
accuracy: 0.8595 - val_loss: 0.6659 - val_accuracy: 0.8275

Epoch 00022: val_loss did not improve from 0.64083
Epoch 23/30
973/973 [==============================] - 3s 3ms/step - loss: 0.5257 -
accuracy: 0.8630 - val_loss: 0.6263 - val_accuracy: 0.8397

Epoch 00023: val_loss improved from 0.64083 to 0.62631, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/checkpoint
Epoch 24/30
973/973 [==============================] - 3s 3ms/step - loss: 0.5151 -
accuracy: 0.8643 - val_loss: 0.6823 - val_accuracy: 0.8215

Epoch 00024: val_loss did not improve from 0.62631
Epoch 25/30
973/973 [==============================] - 3s 3ms/step - loss: 0.5027 -
accuracy: 0.8702 - val_loss: 0.6530 - val_accuracy: 0.8316

Epoch 00025: val_loss did not improve from 0.62631
Epoch 26/30
```

```
973/973 [==============================] - 3s 3ms/step - loss: 0.5021 -
accuracy: 0.8674 - val_loss: 0.7034 - val_accuracy: 0.8123

Epoch 00026: val_loss did not improve from 0.62631
Epoch 27/30
973/973 [==============================] - 3s 3ms/step - loss: 0.4860 -
accuracy: 0.8719 - val_loss: 0.7050 - val_accuracy: 0.8105

Epoch 00027: val_loss did not improve from 0.62631
Epoch 28/30
973/973 [==============================] - 3s 3ms/step - loss: 0.4748 -
accuracy: 0.8769 - val_loss: 0.6496 - val_accuracy: 0.8338

Epoch 00028: val_loss did not improve from 0.62631
```

[ ]:
```
x = np.arange(1, len(MLP_history.history.get('loss'))+1)
y1 = MLP_history.history.get('loss')
y2 = MLP_history.history.get('val_loss')



fig, ax = plt.subplots()
plt.plot(x, y1, marker='.', label = 'train loss')
plt.plot(x, y2, marker='.', label = 'val loss')


ax.set(xlabel='Epoch', ylabel='Loss',
       title='Loss vs Epoch')
plt.legend()
ax.grid()
plt.show()
```



[ ]:
```
x = np.arange(1, len(MLP_history.history.get('accuracy'))+1)
y1 = MLP_history.history.get('accuracy')
y2 = MLP_history.history.get('val_accuracy')


fig, ax = plt.subplots()
plt.plot(x, y1, marker='.', label = 'train acc')
plt.plot(x, y2, marker='.', label = 'val acc')


ax.set(xlabel='Epoch', ylabel='Accuracy',
       title='Accuracy vs Epoch')
plt.legend()
ax.grid()
plt.show()
```



1.4 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.

[ ]:
```
CNN_model = Sequential([
                         Conv2D(12,(4,4),padding='SAME', activation='relu',␣
 ↪input_shape=X_train[0].shape),
                         Conv2D(24,(16,16),padding='SAME', activation='relu'),
                         MaxPool2D(5,5),
```

```
                                Flatten(),
                                BatchNormalization(),
                                Dense(128, activation='relu'),
                                Dropout(0.25),
                                Dense(128, activation='relu'),
                                Dropout(0.25),
                                Dense(64, activation='relu'),
                                Dense(10, activation='softmax')

])
```

[ ]:
```
CNN_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 12)        204
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 24)        73752
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 6, 6, 24)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 864)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 864)               3456
_________________________________________________________________
dense_5 (Dense)              (None, 128)               110720
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_8 (Dense)              (None, 10)                650
=================================================================
Total params: 213,550
Trainable params: 211,822
Non-trainable params: 1,728
_________________________________________________________________
```

[ ]:
```
opt = tf.keras.optimizers.Adam(learning_rate=0.0003)

CNN_model.compile(optimizer=opt, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

```
[ ]: # Instantiation of the callbacks for the CNN

     checkpoint_path = 'gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/
      ↪checkpoint'
     checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_loss', verbose=1,
                                  save_best_only=True, save_weights_only=True)

     earlystopping = EarlyStopping(monitor='val_loss', patience=5, verbose=0)
```

```
[ ]: CNN_history = CNN_model.fit(X_train, y_train, batch_size=64, epochs=30,
                                 verbose=1, callbacks=[checkpoint, earlystopping],
                                 validation_split=0.15)
```

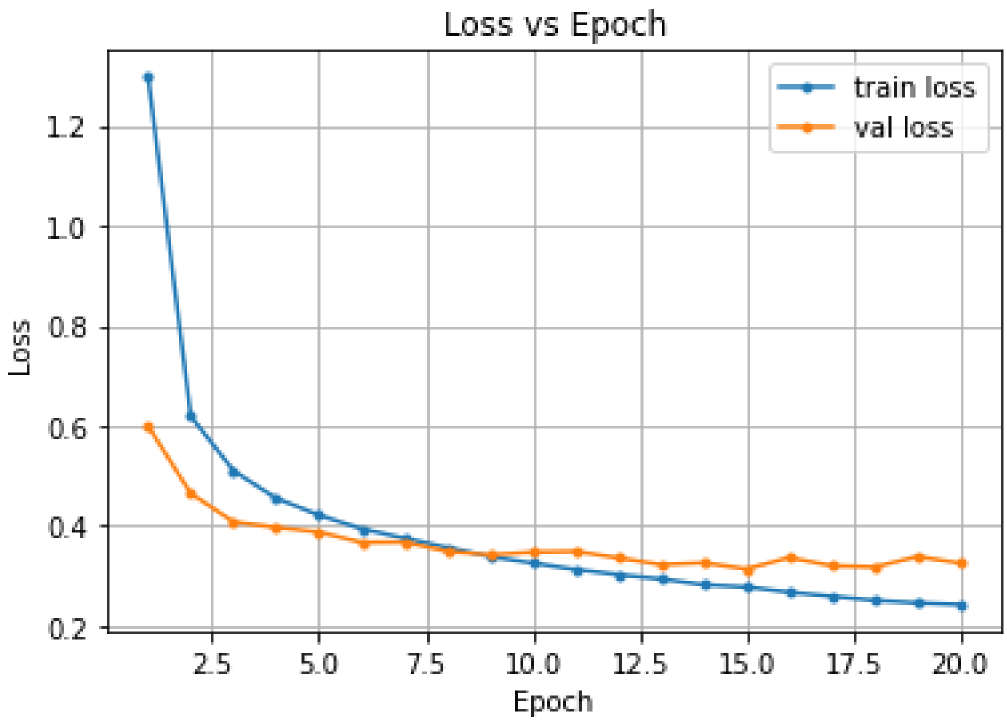
```
Epoch 1/30
973/973 [==============================] - 12s 6ms/step - loss: 1.7970 -
accuracy: 0.3594 - val_loss: 0.5994 - val_accuracy: 0.8113

Epoch 00001: val_loss improved from inf to 0.59941, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 2/30
973/973 [==============================] - 6s 6ms/step - loss: 0.6616 -
accuracy: 0.7938 - val_loss: 0.4647 - val_accuracy: 0.8601

Epoch 00002: val_loss improved from 0.59941 to 0.46467, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 3/30
973/973 [==============================] - 5s 6ms/step - loss: 0.5203 -
accuracy: 0.8406 - val_loss: 0.4078 - val_accuracy: 0.8768

Epoch 00003: val_loss improved from 0.46467 to 0.40783, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 4/30
973/973 [==============================] - 6s 6ms/step - loss: 0.4562 -
accuracy: 0.8615 - val_loss: 0.3967 - val_accuracy: 0.8821

Epoch 00004: val_loss improved from 0.40783 to 0.39674, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 5/30
973/973 [==============================] - 6s 6ms/step - loss: 0.4182 -
accuracy: 0.8720 - val_loss: 0.3874 - val_accuracy: 0.8838

Epoch 00005: val_loss improved from 0.39674 to 0.38742, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 6/30
973/973 [==============================] - 6s 6ms/step - loss: 0.3917 -
accuracy: 0.8800 - val_loss: 0.3668 - val_accuracy: 0.8885

Epoch 00006: val_loss improved from 0.38742 to 0.36675, saving model to
```

```
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 7/30
973/973 [==============================] - 6s 6ms/step - loss: 0.3758 -
accuracy: 0.8866 - val_loss: 0.3673 - val_accuracy: 0.8918

Epoch 00007: val_loss did not improve from 0.36675
Epoch 8/30
973/973 [==============================] - 6s 6ms/step - loss: 0.3551 -
accuracy: 0.8908 - val_loss: 0.3488 - val_accuracy: 0.8970

Epoch 00008: val_loss improved from 0.36675 to 0.34879, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 9/30
973/973 [==============================] - 6s 6ms/step - loss: 0.3358 -
accuracy: 0.8957 - val_loss: 0.3418 - val_accuracy: 0.9001

Epoch 00009: val_loss improved from 0.34879 to 0.34185, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 10/30
973/973 [==============================] - 6s 6ms/step - loss: 0.3185 -
accuracy: 0.9036 - val_loss: 0.3473 - val_accuracy: 0.8979

Epoch 00010: val_loss did not improve from 0.34185
Epoch 11/30
973/973 [==============================] - 6s 6ms/step - loss: 0.3118 -
accuracy: 0.9044 - val_loss: 0.3488 - val_accuracy: 0.8973

Epoch 00011: val_loss did not improve from 0.34185
Epoch 12/30
973/973 [==============================] - 6s 6ms/step - loss: 0.2989 -
accuracy: 0.9085 - val_loss: 0.3357 - val_accuracy: 0.9028

Epoch 00012: val_loss improved from 0.34185 to 0.33569, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 13/30
973/973 [==============================] - 6s 6ms/step - loss: 0.2977 -
accuracy: 0.9107 - val_loss: 0.3222 - val_accuracy: 0.9074

Epoch 00013: val_loss improved from 0.33569 to 0.32224, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 14/30
973/973 [==============================] - 6s 6ms/step - loss: 0.2759 -
accuracy: 0.9149 - val_loss: 0.3258 - val_accuracy: 0.9055

Epoch 00014: val_loss did not improve from 0.32224
Epoch 15/30
973/973 [==============================] - 6s 6ms/step - loss: 0.2705 -
accuracy: 0.9166 - val_loss: 0.3137 - val_accuracy: 0.9086
```

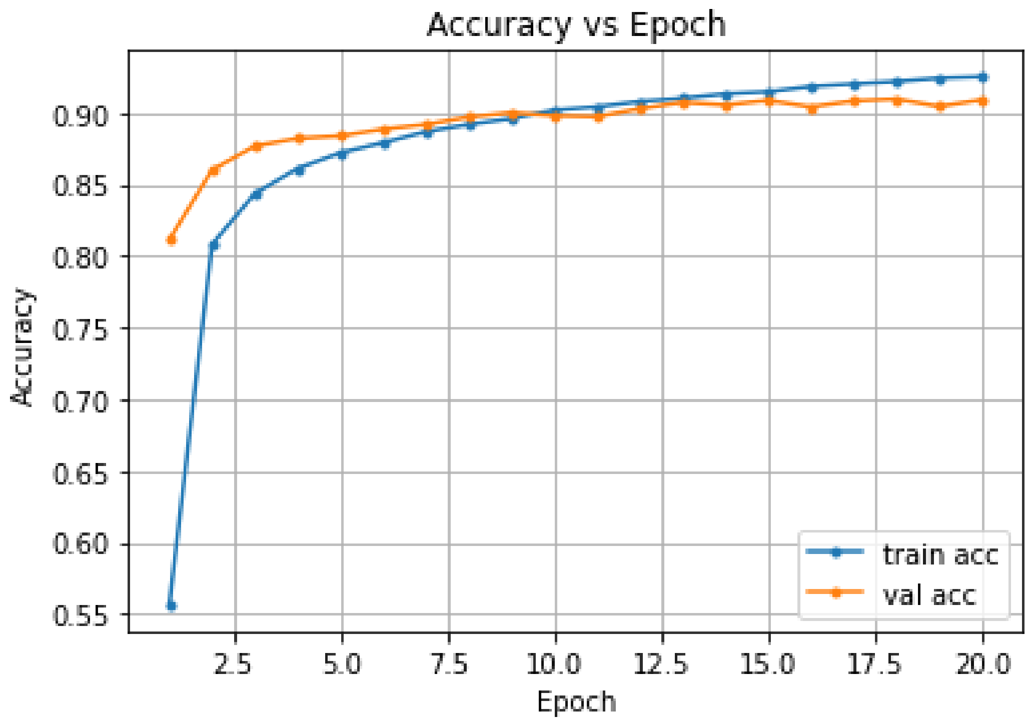
```
Epoch 00015: val_loss improved from 0.32224 to 0.31370, saving model to
gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/checkpoint
Epoch 16/30
973/973 [==============================] - 6s 6ms/step - loss: 0.2546 -
accuracy: 0.9225 - val_loss: 0.3361 - val_accuracy: 0.9042

Epoch 00016: val_loss did not improve from 0.31370
Epoch 17/30
973/973 [==============================] - 6s 6ms/step - loss: 0.2536 -
accuracy: 0.9216 - val_loss: 0.3200 - val_accuracy: 0.9083

Epoch 00017: val_loss did not improve from 0.31370
Epoch 18/30
973/973 [==============================] - 6s 6ms/step - loss: 0.2507 -
accuracy: 0.9222 - val_loss: 0.3174 - val_accuracy: 0.9096

Epoch 00018: val_loss did not improve from 0.31370
Epoch 19/30
973/973 [==============================] - 6s 6ms/step - loss: 0.2430 -
accuracy: 0.9241 - val_loss: 0.3384 - val_accuracy: 0.9044

Epoch 00019: val_loss did not improve from 0.31370
Epoch 20/30
973/973 [==============================] - 6s 6ms/step - loss: 0.2416 -
accuracy: 0.9248 - val_loss: 0.3251 - val_accuracy: 0.9091

Epoch 00020: val_loss did not improve from 0.31370
```

[ ]:

```
x = np.arange(1, len(CNN_history.history.get('loss'))+1)
y1 = CNN_history.history.get('loss')
y2 = CNN_history.history.get('val_loss')



fig, ax = plt.subplots()
plt.plot(x, y1, marker='.', label = 'train loss')
plt.plot(x, y2, marker='.', label = 'val loss')


ax.set(xlabel='Epoch', ylabel='Loss',
       title='Loss vs Epoch')
plt.legend()
ax.grid()
plt.show()
```



[ ]:
```
x = np.arange(1, len(CNN_history.history.get('accuracy'))+1)
y1 = CNN_history.history.get('accuracy')
y2 = CNN_history.history.get('val_accuracy')


fig, ax = plt.subplots()
plt.plot(x, y1, marker='.', label = 'train acc')
plt.plot(x, y2, marker='.', label = 'val acc')


ax.set(xlabel='Epoch', ylabel='Accuracy',
       title='Accuracy vs Epoch')

plt.legend()
ax.grid()
plt.show()
```



1.5 4. Get model predictions

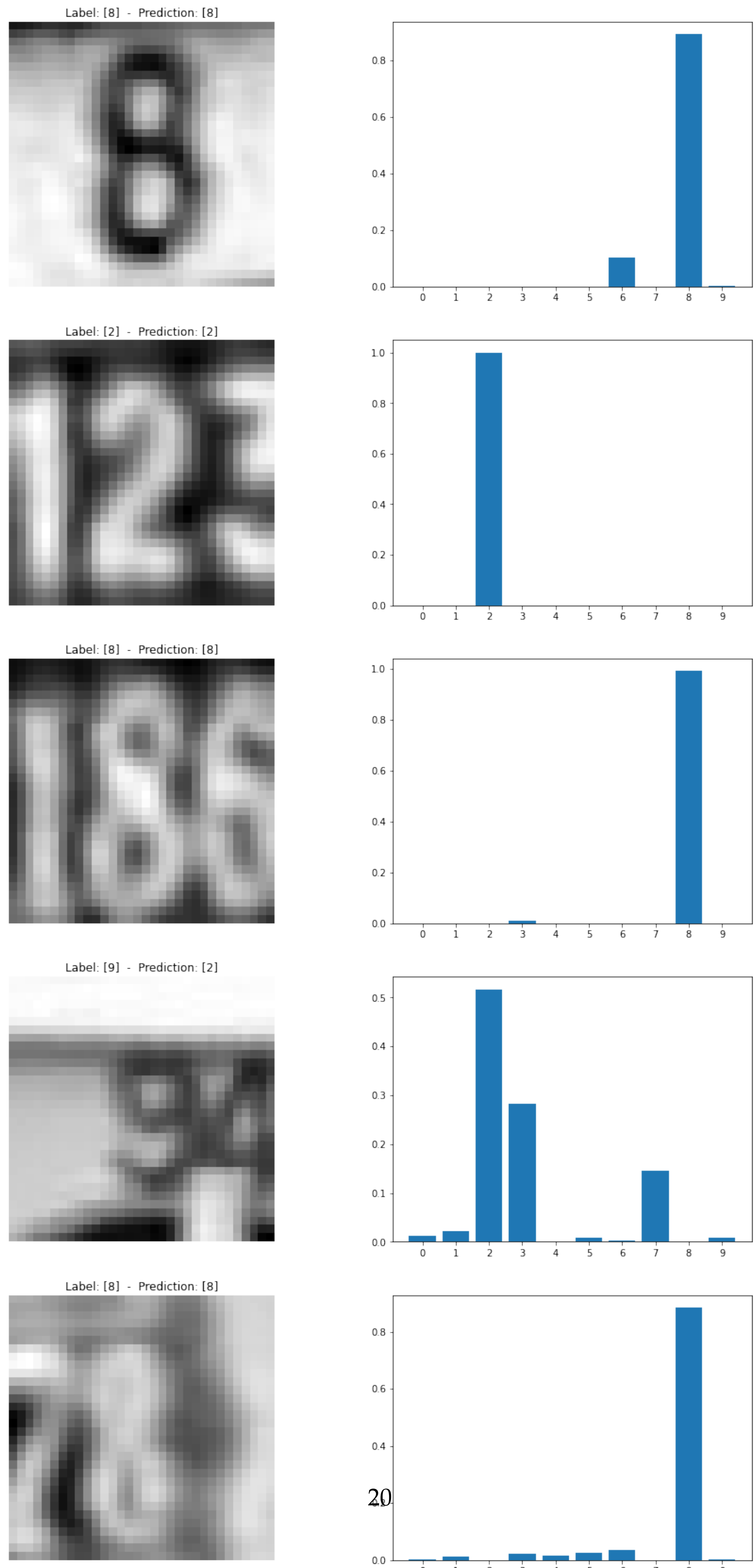
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the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

[ ]:
```
MLP_checkpoint_path = 'gdrive/MyDrive/Colab Notebooks/Capstone_Project/MPL/
↪checkpoint'
CNN_checkpoint_path = 'gdrive/MyDrive/Colab Notebooks/Capstone_Project/CNN/
↪checkpoint'


MLP_model.load_weights(MLP_checkpoint_path)
CNN_model.load_weights(CNN_checkpoint_path)
```

[ ]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7fe2aa21b0f0>
```

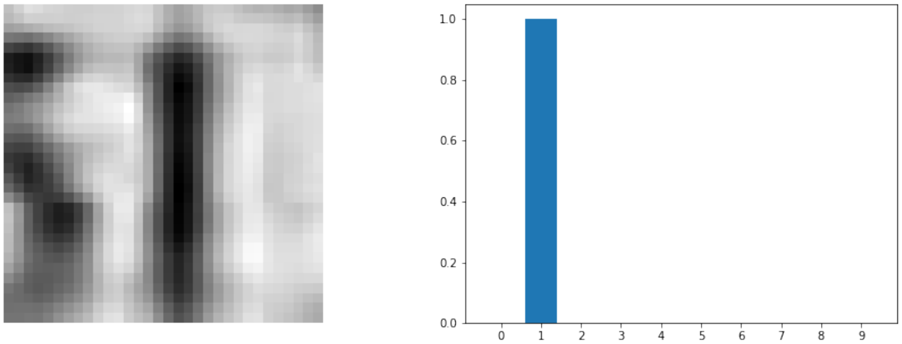
[ ]:
```
random_sample = np.random.randint(0, 26031, 5)

fig, axs = plt.subplots(5,2, figsize=(15,30))
```

```
x = np.arange(10).astype(str)


for i,v in enumerate(random_sample):
  predictions = MLP_model.predict(X_test[v,:,:,:].reshape(-1,32,32,1))
  axs[i, 0].set_axis_off()
  axs[i, 0].imshow(X_test[v,:,:,0], cmap='gray')
  caption = 'Label: ' + str(y_test[v,:]) + '  -  Prediction: [' +␣
 ↪str(predictions.argmax()) + ']'
  axs[i, 0].set_title(caption)
  axs[i, 1].bar(x, predictions.reshape(-1,))

fig.show()
```

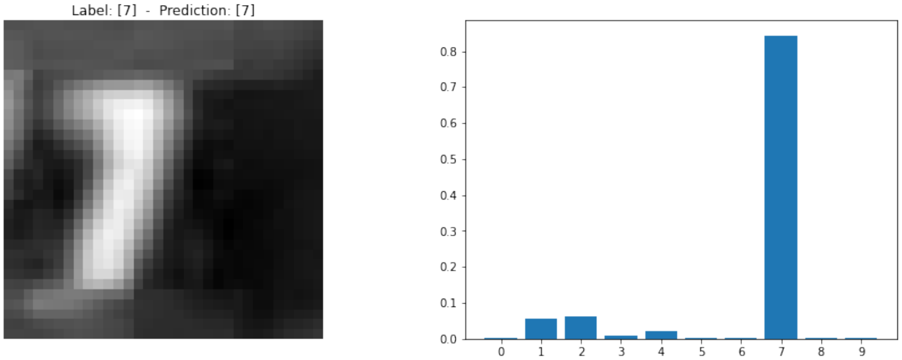

Label: [8] - Prediction: [8]
Label: [2] - Prediction: [2]
Label: [8] - Prediction: [8]
Label: [9] - Prediction: [2]
Label: [8] - Prediction: [8]

[ ]:
```
random_sample = np.random.randint(0, 26031, 5)

fig, axs = plt.subplots(5,2, figsize=(15,30))

x = np.arange(10).astype(str)


for i,v in enumerate(random_sample):
  predictions = CNN_model.predict(X_test[v,:,:,:].reshape(-1,32,32,1))
  axs[i, 0].set_axis_off()
  axs[i, 0].imshow(X_test[v,:,:,0], cmap='gray')
  caption = 'Label: ' + str(y_test[v,:]) + '  -  Prediction: [' +␣
 ↪str(predictions.argmax()) + ']'
  axs[i, 0].set_title(caption)
  axs[i, 1].bar(x, predictions.reshape(-1,))

fig.show()
```

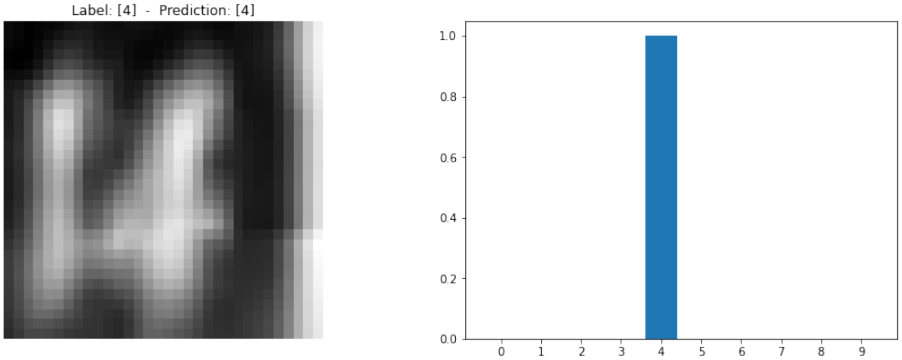
Label: [1] - Prediction: [1]



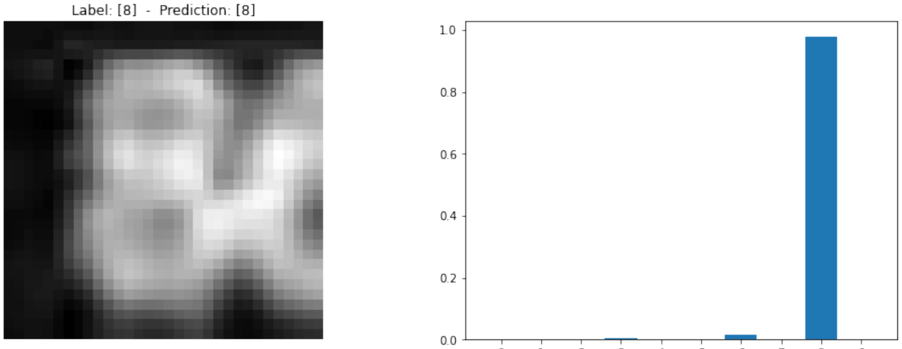
Label: [7] - Prediction: [7]



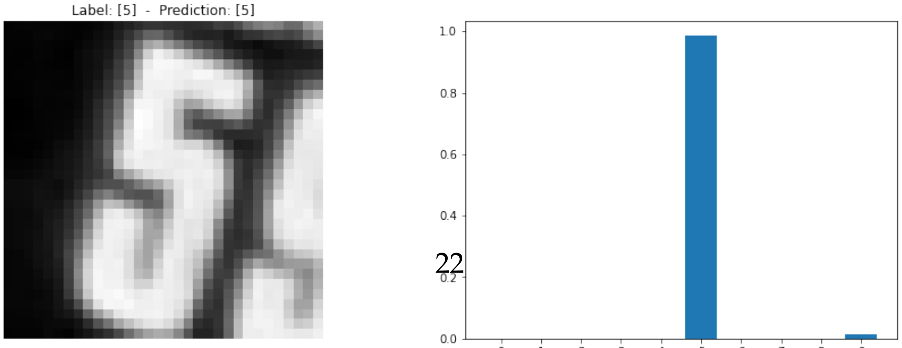
Label: [4] - Prediction: [4]



Label: [8] - Prediction: [8]



Label: [5] - Prediction: [5]



[ ]:
```python
# Finally, we will evaluate the overall performance of both models in the test
 ↪set

print(' ----- MLP Model evaluation -----')

NPL_test = MLP_model.evaluate(X_test, y_test)

print('\nLoss:{:.04f}'.format(NPL_test[0]))
print('Accuracy:{:.02f}%\n\n'.format(NPL_test[1]*100))


print(' ----- CNN Model evaluation ----- ')

CNN_test = CNN_model.evaluate(X_test, y_test)

print('\nLoss:{:.04f}'.format(CNN_test[0]))
print('Accuracy:{:.02f}%'.format(CNN_test[1]*100))
```

```
 ----- MLP Model evaluation -----
814/814 [==============================] - 1s 2ms/step - loss: 0.7119 -
accuracy: 0.8186

Loss:0.7119
Accuracy:81.86%


 ----- CNN Model evaluation -----
814/814 [==============================] - 2s 2ms/step - loss: 0.3648 -
accuracy: 0.8937

Loss:0.3648
Accuracy:89.37%
```