# Capstone_Project

November 5, 2020

# 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
import pandas as pd
```

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.

[ ]:
```
# Downlaoding the Dataset

! wget http://ufldl.stanford.edu/housenumbers/train.tar.gz
! wget http://ufldl.stanford.edu/housenumbers/test.tar.gz
```

```
--2020-11-05 17:49:05--  http://ufldl.stanford.edu/housenumbers/train.tar.gz
Resolving ufldl.stanford.edu (ufldl.stanford.edu)... 171.64.68.10
Connecting to ufldl.stanford.edu (ufldl.stanford.edu)|171.64.68.10|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 404141560 (385M) [application/x-gzip]
Saving to: train.tar.gz

train.tar.gz        100%[===================>] 385.42M  3.50MB/s    in 54s

2020-11-05 17:49:59 (7.18 MB/s) - train.tar.gz saved [404141560/404141560]

--2020-11-05 17:49:59--  http://ufldl.stanford.edu/housenumbers/test.tar.gz
Resolving ufldl.stanford.edu (ufldl.stanford.edu)... 171.64.68.10
Connecting to ufldl.stanford.edu (ufldl.stanford.edu)|171.64.68.10|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 276555967 (264M) [application/x-gzip]
Saving to: test.tar.gz

test.tar.gz         100%[===================>] 263.74M  8.91MB/s    in 1m 53s

2020-11-05 17:51:52 (2.34 MB/s) - test.tar.gz saved [276555967/276555967]
```

[ ]:
```
# Extracting Test Images

! tar -xzf test.tar.gz
```

[ ]:
```
# Extracting Train Images

! tar -xzf train.tar.gz
```

[ ]:
```
! pip install mat73
```

```
Collecting mat73
  Downloading https://files.pythonhosted.org/packages/95/f4/175fd094d338c5eb3b33
8268a301a1109c72f9ed92fa6783cdbded5de40b/mat73-0.45-py3-none-any.whl
```

```
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages
(from mat73) (2.10.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
(from mat73) (1.18.5)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from h5py->mat73) (1.15.0)
Installing collected packages: mat73
Successfully installed mat73-0.45
```

[ ]:
```python
from mat73 import loadmat
```

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

MATRIX = {}

MATRIX["train"] = loadmat('train/digitStruct.mat')
MATRIX["test"] = loadmat('test/digitStruct.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

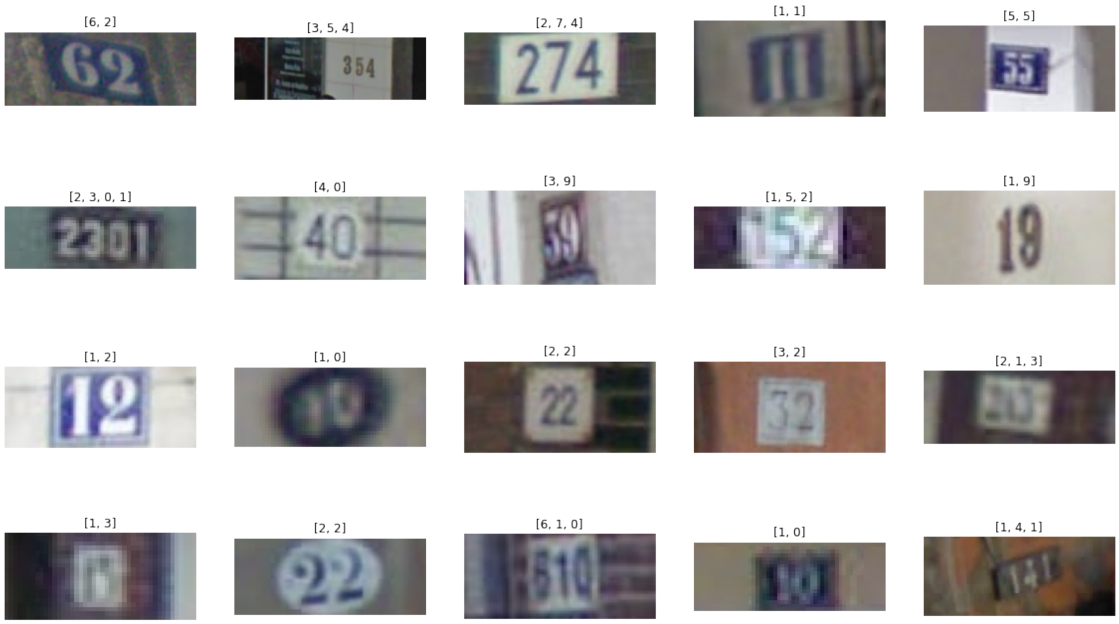
- Extract the training and testing images and labels separately from the train and test dictionaries loaded for you.
- Select a random sample of images and corresponding labels from the dataset (at least 10), and display them in a figure.
- Convert the training and test images to grayscale by taking 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.

[ ]:
```python
SIZE = {}

SIZE["train"] = len(MATRIX["train"]["digitStruct"]["name"])
SIZE["test"] = len(MATRIX["test"]["digitStruct"]["name"])
```

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from skimage import transform
from skimage.color import rgb2gray
from collections import Counter
```

[ ]:
```python
plt.rcParams["figure.figsize"] = 20, 12

for idx in range(20):
  plt.subplot(4, 5, idx+1)
  e = np.random.randint(0, SIZE["train"])
  img = plt.imread("train/{}.png".format(e+1))
  plt.imshow(img)
```

```
    plt.axis('off')
    plt.title((np.array(MATRIX["train"]["digitStruct"]["bbox"][e]["label"],␣
 ↪dtype=np.int16)%10).tolist())
plt.show()

plt.rcParams["figure.figsize"] = 5, 4
```




[ ]:
```
"""
The function does these operation on the follwoing sequences:
+ For Image:
  + Get the best crop position so that all the digits are included.
  + Convert image into gray scale.
  + Resize the image into (32, 32)
+ For Label:
  + Return Labels as a list
"""

data_point = lambda img, data: (
                                transform.resize(image = rgb2gray(img[np.
 ↪abs(data.top).min().astype(np.int16):(np.abs(data.top) + np.abs(data.
 ↪height)).max().astype(np.int16),
                                                                      np.
 ↪abs(data.left).min().astype(np.int16):(np.abs(data.left) + np.abs(data.
 ↪width)).max().astype(np.int16)]),
                                                  output_shape=(32, 32)),

                                [abs(int(x)%10) for x in data.label]
```

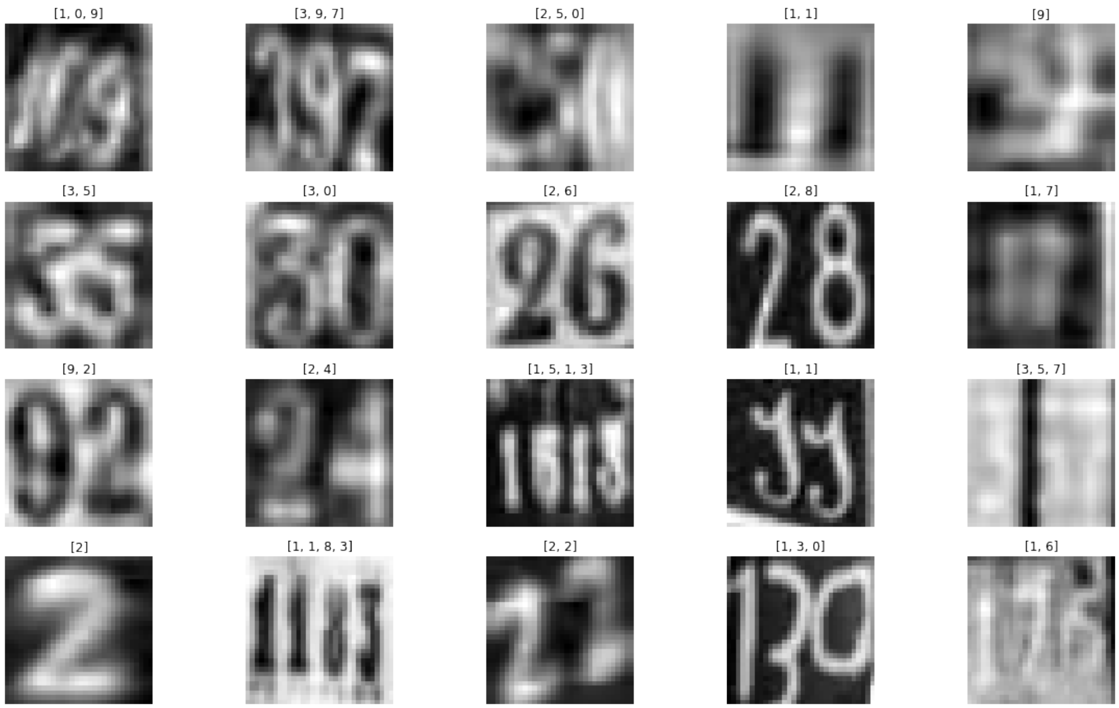
```
                              ) \
                              if type(data.label) == list else \
                              (
                                transform.resize(image = rgb2gray(
                                                                              ␣
↪img[int(abs(data.top)):int(abs(data.top) + abs(data.height)),
                                                                                  ␣
↪int(abs(data.left)):int(abs(data.left) + abs(data.width))]
                                                                             ),
                                                 output_shape=(32, 32)),

                                [abs(int(data.label)%10)]
                              )
```

[ ]:
```
def fill_arrays(path):
    """
    This function takes the path of the folder where iamges are located.
    It prepares the data so that it can be used for data anlaysis and mdoelling.
    """

    image_array = []
    label_array = []
    i = 0

    while(i < SIZE[path]):
      img = plt.imread("{}/{}".format(path,␣
↪MATRIX[path]["digitStruct"]["name"][i]))
      img, labels = data_point(img, MATRIX[path]["digitStruct"]["bbox"][i])

      image_array.append(img)
      label_array.append(labels)

      i += 1

    return  image_array, label_array
```

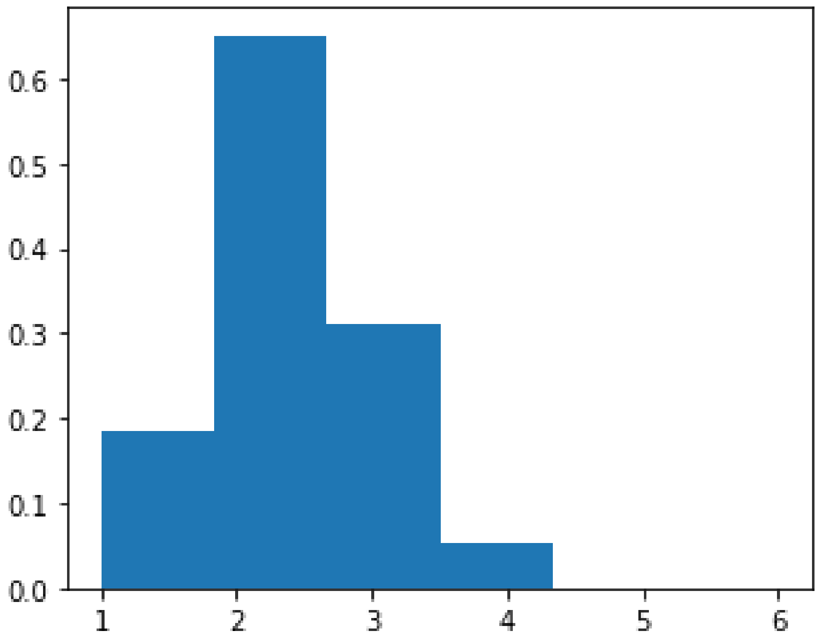
[ ]:
```
train_images, train_labels = fill_arrays("train")
test_images, test_labels = fill_arrays("test")
```

[ ]:
```
plt.rcParams["figure.figsize"] = 20, 12

for idx in range(20):
  plt.subplot(4, 5, idx+1)
  e = np.random.randint(0, SIZE["train"])
  plt.imshow(train_images[e],cmap="binary")
  plt.axis('off')
  plt.title(train_labels[e])
plt.show()
```

```python
plt.rcParams["figure.figsize"] = 5, 4
```

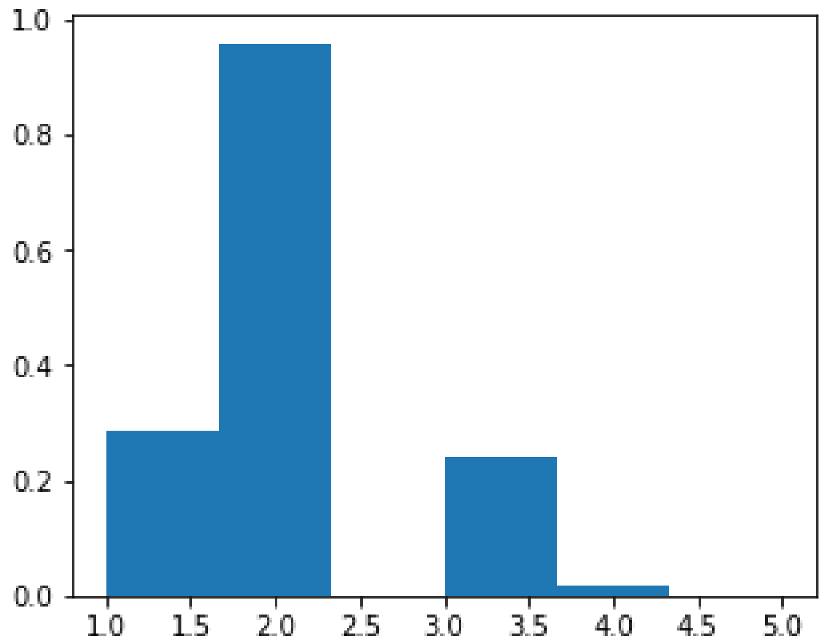



[ ]:
```python
# Frequency of x digit numbers in training set.

train_dist = list(map(lambda x: len(x), train_labels))

plt.hist(train_dist, bins=6, density=True)
plt.show()

Counter(train_dist)
```




[ ]: Counter({1: 5137, 2: 18130, 3: 8691, 4: 1434, 5: 9, 6: 1})

[ ]:
```
# Frequency of x digit numbers in training set.

test_dist = list(map(lambda x: len(x), test_labels))

plt.hist(test_dist, bins=6, density=True, )
plt.show()

Counter(test_dist)
```




[ ]: `Counter({1: 2483, 2: 8356, 3: 2081, 4: 146, 5: 2})`

[ ]:
```
# NOTE: The dataset is hughely imbalanced with repsect to the number of digits␣
↪in the images.
```

[ ]:
```
from sklearn.preprocessing import MultiLabelBinarizer
```

[ ]:
```
X_train = np.array(train_images)
X_test = np.array(test_images)


# Create MultiLabelBinarizer object
one_hot = MultiLabelBinarizer()
one_hot.fit(train_labels)

Y_train = one_hot.transform(train_labels)
Y_test = one_hot.transform(test_labels)
```

### 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.

[ ]:
```
from tensorflow.keras import layers, models, optimizers, metrics, losses,␣
↪callbacks
```

[ ]:
```
def SequentialModel(input_shape):
    """
    NOTE: Since there aree more than one diigts prense tin each aimge, hecne I␣
↪am going to threat this problem as
          multi label classification where an image can have more than one␣
↪labels. For this case, Softamax activation
          doesn't make any sense in the final layer. It is sutaible for␣
↪multiclass classifcation but not for multilabel
          classifcation. Therfore, I am going to use Sigmoid activation in␣
↪final layer whith 10 units which enocodes
          the probability of each digit by a Bernoulli varaible.
          Same reason goes for the model loss as well. It doesn't make any␣
↪sense to use Categeorical cross entropy which is
          sutaible for multi-c;ass classfication, but not for multi-label␣
↪classfication.
    """
    model = models.Sequential([
                              layers.InputLayer(input_shape=input_shape),
                              layers.Flatten(),
                              layers.Dense(60, activation="relu"),
                              layers.Dense(100, activation="relu"),
                              layers.Dense(100, activation="relu"),
                              layers.Dense(10, activation="sigmoid")
    ])

    model.compile(optimizer = "adam",
                  loss = losses.BinaryCrossentropy(),
                  metrics = [metrics.BinaryAccuracy("accuracy"), metrics.
↪Precision(name="precision"), metrics.Recall(name="recall")])

    return model
```

[ ]:
```
model_1 = SequentialModel(input_shape=(32, 32))

model_1.summary()
```

```
Model: "sequential_11"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_11 (Flatten)         (None, 1024)              0
_________________________________________________________________
dense_42 (Dense)             (None, 60)                61500
_________________________________________________________________
dense_43 (Dense)             (None, 100)               6100
_________________________________________________________________
dense_44 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_45 (Dense)             (None, 10)                1010
=================================================================
Total params: 78,710
Trainable params: 78,710
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
! rm -rf sequential_model_checkpoint
```

[ ]:
```
# Callbacks

seq_path = "sequential_model_checkpoints/checkpoint"
ckpt_seq = callbacks.ModelCheckpoint(filepath=seq_path,
                                     monitor="val_accuracy",
                                     verbose=True,
                                     mode="max",
                                     save_freq="epoch",
                                     save_weights_only=True,
                                     save_best_only=True)

csvl_seq = callbacks.CSVLogger(filename="sequential_model_CSV.csv")
```

[ ]:
```
# Model Training

history = model_1.fit(x = X_train,
                      y = Y_train,
                      batch_size=300,
                      epochs=50,
                      validation_split = 0.15,
                      callbacks = [ckpt_seq, csvl_seq],
                      verbose = 2)
```

```
Epoch 1/50

Epoch 00001: val_accuracy improved from -inf to 0.79563, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 1s - loss: 0.5136 - accuracy: 0.7918 - precision: 0.2496 - recall:
```

```
0.0134 - val_loss: 0.5065 - val_accuracy: 0.7956 - val_precision: 0.0000e+00 -
val_recall: 0.0000e+00
Epoch 2/50

Epoch 00002: val_accuracy did not improve from 0.79563
95/95 - 0s - loss: 0.4992 - accuracy: 0.7972 - precision: 0.0000e+00 - recall:
0.0000e+00 - val_loss: 0.4955 - val_accuracy: 0.7956 - val_precision: 0.0000e+00
- val_recall: 0.0000e+00
Epoch 3/50

Epoch 00003: val_accuracy improved from 0.79563 to 0.79765, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.4852 - accuracy: 0.7977 - precision: 0.7317 - recall:
0.0036 - val_loss: 0.4768 - val_accuracy: 0.7976 - val_precision: 0.7563 -
val_recall: 0.0145
Epoch 4/50

Epoch 00004: val_accuracy improved from 0.79765 to 0.80347, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.4655 - accuracy: 0.8034 - precision: 0.6883 - recall:
0.0561 - val_loss: 0.4605 - val_accuracy: 0.8035 - val_precision: 0.6546 -
val_recall: 0.0812
Epoch 5/50

Epoch 00005: val_accuracy improved from 0.80347 to 0.80868, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.4538 - accuracy: 0.8086 - precision: 0.6706 - recall:
0.1099 - val_loss: 0.4502 - val_accuracy: 0.8087 - val_precision: 0.6972 -
val_recall: 0.1129
Epoch 6/50

Epoch 00006: val_accuracy improved from 0.80868 to 0.81371, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.4400 - accuracy: 0.8142 - precision: 0.6970 - recall:
0.1480 - val_loss: 0.4395 - val_accuracy: 0.8137 - val_precision: 0.6933 -
val_recall: 0.1587
Epoch 7/50

Epoch 00007: val_accuracy improved from 0.81371 to 0.81642, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.4290 - accuracy: 0.8187 - precision: 0.7048 - recall:
0.1820 - val_loss: 0.4339 - val_accuracy: 0.8164 - val_precision: 0.7056 -
val_recall: 0.1746
Epoch 8/50

Epoch 00008: val_accuracy improved from 0.81642 to 0.82032, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.4196 - accuracy: 0.8223 - precision: 0.7077 - recall:
```

```
0.2111 - val_loss: 0.4236 - val_accuracy: 0.8203 - val_precision: 0.7132 -
val_recall: 0.2020
Epoch 9/50

Epoch 00009: val_accuracy did not improve from 0.82032
95/95 - 0s - loss: 0.4122 - accuracy: 0.8255 - precision: 0.7103 - recall:
0.2352 - val_loss: 0.4201 - val_accuracy: 0.8194 - val_precision: 0.6414 -
val_recall: 0.2640
Epoch 10/50

Epoch 00010: val_accuracy improved from 0.82032 to 0.82548, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.4062 - accuracy: 0.8278 - precision: 0.7126 - recall:
0.2528 - val_loss: 0.4120 - val_accuracy: 0.8255 - val_precision: 0.6705 -
val_recall: 0.2873
Epoch 11/50

Epoch 00011: val_accuracy improved from 0.82548 to 0.82766, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.4024 - accuracy: 0.8297 - precision: 0.7169 - recall:
0.2648 - val_loss: 0.4099 - val_accuracy: 0.8277 - val_precision: 0.6712 -
val_recall: 0.3072
Epoch 12/50

Epoch 00012: val_accuracy improved from 0.82766 to 0.82834, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3980 - accuracy: 0.8314 - precision: 0.7153 - recall:
0.2797 - val_loss: 0.4055 - val_accuracy: 0.8283 - val_precision: 0.7001 -
val_recall: 0.2800
Epoch 13/50

Epoch 00013: val_accuracy improved from 0.82834 to 0.82930, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3942 - accuracy: 0.8331 - precision: 0.7205 - recall:
0.2890 - val_loss: 0.4005 - val_accuracy: 0.8293 - val_precision: 0.7163 -
val_recall: 0.2727
Epoch 14/50

Epoch 00014: val_accuracy improved from 0.82930 to 0.83331, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3917 - accuracy: 0.8341 - precision: 0.7182 - recall:
0.2992 - val_loss: 0.3941 - val_accuracy: 0.8333 - val_precision: 0.7453 -
val_recall: 0.2801
Epoch 15/50

Epoch 00015: val_accuracy did not improve from 0.83331
95/95 - 0s - loss: 0.3900 - accuracy: 0.8348 - precision: 0.7206 - recall:
0.3024 - val_loss: 0.4027 - val_accuracy: 0.8278 - val_precision: 0.6751 -
```

```
val_recall: 0.3033
Epoch 16/50

Epoch 00016: val_accuracy improved from 0.83331 to 0.83349, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3861 - accuracy: 0.8370 - precision: 0.7273 - recall:
0.3136 - val_loss: 0.3902 - val_accuracy: 0.8335 - val_precision: 0.7273 -
val_recall: 0.2964
Epoch 17/50

Epoch 00017: val_accuracy improved from 0.83349 to 0.83365, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3844 - accuracy: 0.8374 - precision: 0.7266 - recall:
0.3174 - val_loss: 0.3906 - val_accuracy: 0.8336 - val_precision: 0.7290 -
val_recall: 0.2961
Epoch 18/50

Epoch 00018: val_accuracy improved from 0.83365 to 0.83720, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3805 - accuracy: 0.8390 - precision: 0.7314 - recall:
0.3257 - val_loss: 0.3861 - val_accuracy: 0.8372 - val_precision: 0.7161 -
val_recall: 0.3370
Epoch 19/50

Epoch 00019: val_accuracy did not improve from 0.83720
95/95 - 0s - loss: 0.3779 - accuracy: 0.8406 - precision: 0.7368 - recall:
0.3325 - val_loss: 0.3884 - val_accuracy: 0.8362 - val_precision: 0.7080 -
val_recall: 0.3376
Epoch 20/50

Epoch 00020: val_accuracy improved from 0.83720 to 0.83732, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3769 - accuracy: 0.8410 - precision: 0.7342 - recall:
0.3382 - val_loss: 0.3890 - val_accuracy: 0.8373 - val_precision: 0.7247 -
val_recall: 0.3290
Epoch 21/50

Epoch 00021: val_accuracy did not improve from 0.83732
95/95 - 0s - loss: 0.3733 - accuracy: 0.8424 - precision: 0.7383 - recall:
0.3452 - val_loss: 0.3869 - val_accuracy: 0.8359 - val_precision: 0.7256 -
val_recall: 0.3169
Epoch 22/50

Epoch 00022: val_accuracy improved from 0.83732 to 0.84129, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3728 - accuracy: 0.8430 - precision: 0.7392 - recall:
0.3488 - val_loss: 0.3792 - val_accuracy: 0.8413 - val_precision: 0.7305 -
val_recall: 0.3541
```

```
Epoch 23/50

Epoch 00023: val_accuracy did not improve from 0.84129
95/95 - 0s - loss: 0.3719 - accuracy: 0.8434 - precision: 0.7387 - recall:
0.3525 - val_loss: 0.3791 - val_accuracy: 0.8402 - val_precision: 0.7325 -
val_recall: 0.3431
Epoch 24/50

Epoch 00024: val_accuracy improved from 0.84129 to 0.84139, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3685 - accuracy: 0.8450 - precision: 0.7450 - recall:
0.3583 - val_loss: 0.3754 - val_accuracy: 0.8414 - val_precision: 0.7275 -
val_recall: 0.3580
Epoch 25/50

Epoch 00025: val_accuracy improved from 0.84139 to 0.84281, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3666 - accuracy: 0.8461 - precision: 0.7448 - recall:
0.3666 - val_loss: 0.3754 - val_accuracy: 0.8428 - val_precision: 0.7361 -
val_recall: 0.3598
Epoch 26/50

Epoch 00026: val_accuracy did not improve from 0.84281
95/95 - 0s - loss: 0.3642 - accuracy: 0.8471 - precision: 0.7485 - recall:
0.3706 - val_loss: 0.3741 - val_accuracy: 0.8417 - val_precision: 0.7257 -
val_recall: 0.3628
Epoch 27/50

Epoch 00027: val_accuracy improved from 0.84281 to 0.84346, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3623 - accuracy: 0.8484 - precision: 0.7526 - recall:
0.3759 - val_loss: 0.3716 - val_accuracy: 0.8435 - val_precision: 0.7379 -
val_recall: 0.3630
Epoch 28/50

Epoch 00028: val_accuracy improved from 0.84346 to 0.84502, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3599 - accuracy: 0.8488 - precision: 0.7505 - recall:
0.3807 - val_loss: 0.3689 - val_accuracy: 0.8450 - val_precision: 0.7403 -
val_recall: 0.3722
Epoch 29/50

Epoch 00029: val_accuracy did not improve from 0.84502
95/95 - 0s - loss: 0.3576 - accuracy: 0.8501 - precision: 0.7547 - recall:
0.3861 - val_loss: 0.3705 - val_accuracy: 0.8438 - val_precision: 0.7186 -
val_recall: 0.3877
Epoch 30/50
```

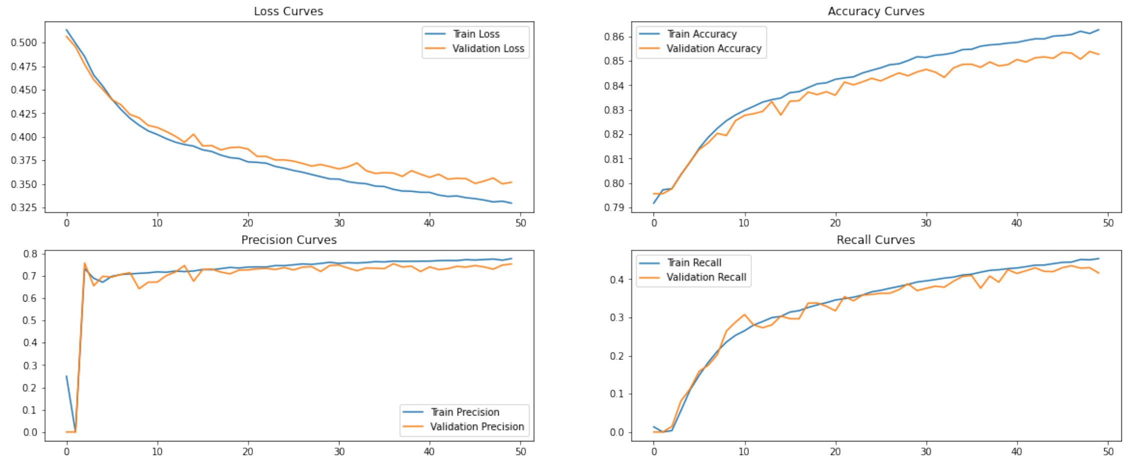
```
Epoch 00030: val_accuracy improved from 0.84502 to 0.84542, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3552 - accuracy: 0.8516 - precision: 0.7599 - recall:
0.3922 - val_loss: 0.3683 - val_accuracy: 0.8454 - val_precision: 0.7455 -
val_recall: 0.3699
Epoch 31/50

Epoch 00031: val_accuracy improved from 0.84542 to 0.84646, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3550 - accuracy: 0.8514 - precision: 0.7550 - recall:
0.3953 - val_loss: 0.3658 - val_accuracy: 0.8465 - val_precision: 0.7473 -
val_recall: 0.3757
Epoch 32/50

Epoch 00032: val_accuracy did not improve from 0.84646
95/95 - 0s - loss: 0.3523 - accuracy: 0.8522 - precision: 0.7578 - recall:
0.3987 - val_loss: 0.3680 - val_accuracy: 0.8454 - val_precision: 0.7343 -
val_recall: 0.3813
Epoch 33/50

Epoch 00033: val_accuracy did not improve from 0.84646
95/95 - 0s - loss: 0.3509 - accuracy: 0.8526 - precision: 0.7566 - recall:
0.4025 - val_loss: 0.3721 - val_accuracy: 0.8432 - val_precision: 0.7218 -
val_recall: 0.3788
Epoch 34/50

Epoch 00034: val_accuracy improved from 0.84646 to 0.84704, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3501 - accuracy: 0.8532 - precision: 0.7587 - recall:
0.4051 - val_loss: 0.3640 - val_accuracy: 0.8470 - val_precision: 0.7342 -
val_recall: 0.3943
Epoch 35/50

Epoch 00035: val_accuracy improved from 0.84704 to 0.84851, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3476 - accuracy: 0.8546 - precision: 0.7627 - recall:
0.4105 - val_loss: 0.3611 - val_accuracy: 0.8485 - val_precision: 0.7332 -
val_recall: 0.4068
Epoch 36/50

Epoch 00036: val_accuracy improved from 0.84851 to 0.84857, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3471 - accuracy: 0.8547 - precision: 0.7615 - recall:
0.4127 - val_loss: 0.3618 - val_accuracy: 0.8486 - val_precision: 0.7314 -
val_recall: 0.4094
Epoch 37/50

Epoch 00037: val_accuracy did not improve from 0.84857
```

```
95/95 - 0s - loss: 0.3441 - accuracy: 0.8560 - precision: 0.7651 - recall:
0.4180 - val_loss: 0.3614 - val_accuracy: 0.8473 - val_precision: 0.7533 -
val_recall: 0.3760
Epoch 38/50

Epoch 00038: val_accuracy improved from 0.84857 to 0.84951, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3424 - accuracy: 0.8565 - precision: 0.7641 - recall:
0.4227 - val_loss: 0.3579 - val_accuracy: 0.8495 - val_precision: 0.7388 -
val_recall: 0.4079
Epoch 39/50

Epoch 00039: val_accuracy did not improve from 0.84951
95/95 - 0s - loss: 0.3421 - accuracy: 0.8567 - precision: 0.7642 - recall:
0.4243 - val_loss: 0.3640 - val_accuracy: 0.8479 - val_precision: 0.7422 -
val_recall: 0.3919
Epoch 40/50

Epoch 00040: val_accuracy did not improve from 0.84951
95/95 - 0s - loss: 0.3410 - accuracy: 0.8572 - precision: 0.7648 - recall:
0.4273 - val_loss: 0.3602 - val_accuracy: 0.8484 - val_precision: 0.7185 -
val_recall: 0.4247
Epoch 41/50

Epoch 00041: val_accuracy improved from 0.84951 to 0.85049, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3410 - accuracy: 0.8575 - precision: 0.7651 - recall:
0.4290 - val_loss: 0.3569 - val_accuracy: 0.8505 - val_precision: 0.7393 -
val_recall: 0.4147
Epoch 42/50

Epoch 00042: val_accuracy did not improve from 0.85049
95/95 - 0s - loss: 0.3381 - accuracy: 0.8583 - precision: 0.7672 - recall:
0.4325 - val_loss: 0.3602 - val_accuracy: 0.8495 - val_precision: 0.7272 -
val_recall: 0.4218
Epoch 43/50

Epoch 00043: val_accuracy improved from 0.85049 to 0.85121, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3366 - accuracy: 0.8590 - precision: 0.7679 - recall:
0.4364 - val_loss: 0.3550 - val_accuracy: 0.8512 - val_precision: 0.7317 -
val_recall: 0.4294
Epoch 44/50

Epoch 00044: val_accuracy improved from 0.85121 to 0.85157, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3371 - accuracy: 0.8589 - precision: 0.7675 - recall:
0.4367 - val_loss: 0.3559 - val_accuracy: 0.8516 - val_precision: 0.7415 -
```

```
val_recall: 0.4202
Epoch 45/50

Epoch 00045: val_accuracy did not improve from 0.85157
95/95 - 0s - loss: 0.3353 - accuracy: 0.8601 - precision: 0.7716 - recall:
0.4403 - val_loss: 0.3556 - val_accuracy: 0.8510 - val_precision: 0.7384 -
val_recall: 0.4196
Epoch 46/50

Epoch 00046: val_accuracy improved from 0.85157 to 0.85342, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3343 - accuracy: 0.8603 - precision: 0.7699 - recall:
0.4436 - val_loss: 0.3504 - val_accuracy: 0.8534 - val_precision: 0.7452 -
val_recall: 0.4297
Epoch 47/50

Epoch 00047: val_accuracy did not improve from 0.85342
95/95 - 0s - loss: 0.3328 - accuracy: 0.8607 - precision: 0.7719 - recall:
0.4445 - val_loss: 0.3529 - val_accuracy: 0.8531 - val_precision: 0.7391 -
val_recall: 0.4349
Epoch 48/50

Epoch 00048: val_accuracy did not improve from 0.85342
95/95 - 0s - loss: 0.3308 - accuracy: 0.8620 - precision: 0.7744 - recall:
0.4512 - val_loss: 0.3562 - val_accuracy: 0.8506 - val_precision: 0.7289 -
val_recall: 0.4287
Epoch 49/50

Epoch 00049: val_accuracy improved from 0.85342 to 0.85378, saving model to
sequential_model_checkpoints/checkpoint
95/95 - 0s - loss: 0.3316 - accuracy: 0.8611 - precision: 0.7693 - recall:
0.4503 - val_loss: 0.3500 - val_accuracy: 0.8538 - val_precision: 0.7474 -
val_recall: 0.4298
Epoch 50/50

Epoch 00050: val_accuracy did not improve from 0.85378
95/95 - 0s - loss: 0.3295 - accuracy: 0.8627 - precision: 0.7762 - recall:
0.4535 - val_loss: 0.3516 - val_accuracy: 0.8526 - val_precision: 0.7522 -
val_recall: 0.4159
```

[ ]:
```python
# Reults Visualization

plt.rcParams["figure.figsize"] = 20, 8

plt.subplot(2, 2, 1)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
```

```
plt.legend()
plt.title("Loss Curves")

plt.subplot(2, 2, 2)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.legend()
plt.title("Accuracy Curves")


plt.subplot(2, 2, 3)
plt.plot(history.history["precision"], label="Train Precision")
plt.plot(history.history["val_precision"], label="Validation Precision")
plt.legend()
plt.title("Precision Curves")


plt.subplot(2, 2, 4)
plt.plot(history.history["recall"], label="Train Recall")
plt.plot(history.history["val_recall"], label="Validation Recall")
plt.legend()
plt.title("Recall Curves")

plt.show()
```




[ ]:
```
# Evalaution on Test Set

seq_res = model_1.evaluate(x = X_test,
                           y = Y_test,
                           return_dict=True,
                           verbose=False)
```

```
for key in seq_res.keys():
    if key != "loss":
      print("{}: {:0.6f}%".format(key.capitalize(), 100*seq_res[key]))
    else:
      print("{}: {:0.6f}".format(key.capitalize(), seq_res[key]))
```

```
Loss: 0.366178
Accuracy: 85.306132%
Precision: 68.388259%
Recall: 40.295342%
```

### 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.

[ ]:
```
def CNNModel(input_shape):
    """
    NOTE: Since there aree more than one diigts prense tin each aimge, hecne I␣
 ↪am going to threat this problem as
          multi label classification where an image can have more than one␣
 ↪labels. For this case, Softamax activation
          doesn't make any sense in the final layer. It is sutaible for␣
 ↪multiclass classifcation but not for multilabel
          classifcation. Therfore, I am going to use Sigmoid activation in␣
 ↪final layer whith 10 units which enocodes
          the probability of each digit by a Bernoulli varaible.
          Same reason goes for the model loss as well. It doesn't make any␣
 ↪sense to use Categeorical cross entropy which is
          sutaible for multi-class classfication, but not for multi-label␣
 ↪classfication.
    """
    model = models.Sequential([
                                layers.InputLayer(input_shape=input_shape),
```

```
                              layers.Conv2D(32, kernel_size=(3, 3),␣
↪activation="relu"),
                              layers.Dropout(rate=0.2),
                              layers.BatchNormalization(),
                              layers.MaxPool2D(pool_size=(2,2)),
                              layers.Conv2D(64, kernel_size=(3, 3),␣
↪activation="relu"),
                              layers.Dropout(rate=0.2),
                              layers.BatchNormalization(),
                              layers.MaxPool2D(pool_size=(2,2)),
                              layers.Flatten(),
                              layers.Dense(20, activation="relu"),
                              layers.Dense(10, activation="sigmoid")
    ])

    model.compile(optimizer = "adam",
                  loss = losses.BinaryCrossentropy(),
                  metrics = [metrics.BinaryAccuracy("accuracy"), metrics.
↪Precision(name="precision"), metrics.Recall(name="recall")])

    return model
```

[ ]:

```
model_2 = CNNModel(input_shape=(32, 32, 1))

model_2.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 30, 30, 32)        320
_________________________________________________________________
dropout_4 (Dropout)          (None, 30, 30, 32)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
dropout_5 (Dropout)          (None, 13, 13, 64)        0
_________________________________________________________________
batch_normalization_5 (Batch (None, 13, 13, 64)        256
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
flatten_13 (Flatten)         (None, 2304)              0
_________________________________________________________________
```

```
dense_48 (Dense)             (None, 20)                46100
_________________________________________________________________
dense_49 (Dense)             (None, 10)                210
=================================================================
Total params: 65,510
Trainable params: 65,318
Non-trainable params: 192
_________________________________________________________________
```

[ ]:
```python
# Callbacks

cnn_path = "cnn_model_checkpoints/checkpoint"
ckpt_cnn = callbacks.ModelCheckpoint(filepath=cnn_path,
                                     monitor="val_accuracy",
                                     verbose=True,
                                     mode="max",
                                     save_freq="epoch",
                                     save_weights_only=True,
                                     save_best_only=True)

csvl_cnn = callbacks.CSVLogger(filename="cnn_model_CSV.csv")

estop_cnn = callbacks.EarlyStopping(monitor='val_accuracy',
                                    patience=5,
                                    verbose=0,
                                    mode='max',
                                    restore_best_weights=False)
```

[ ]:
```python
history = model_2.fit(x = X_train[..., np.newaxis],
                      y = Y_train,
                      batch_size=100,
                      epochs=50,
                      validation_split = 0.15,
                      callbacks = [ckpt_cnn, csvl_cnn, estop_cnn],
                      verbose = 2)
```

```
Epoch 1/50

Epoch 00001: val_accuracy improved from -inf to 0.63055, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.4372 - accuracy: 0.8119 - precision: 0.5869 - recall:
0.2439 - val_loss: 0.8093 - val_accuracy: 0.6306 - val_precision: 0.2531 -
val_recall: 0.4141
Epoch 2/50

Epoch 00002: val_accuracy improved from 0.63055 to 0.84568, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.3449 - accuracy: 0.8547 - precision: 0.7319 - recall:
```

```
0.4472 - val_loss: 0.4141 - val_accuracy: 0.8457 - val_precision: 0.6647 -
val_recall: 0.4943
Epoch 3/50

Epoch 00003: val_accuracy improved from 0.84568 to 0.86961, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.3052 - accuracy: 0.8722 - precision: 0.7681 - recall:
0.5295 - val_loss: 0.3067 - val_accuracy: 0.8696 - val_precision: 0.7944 -
val_recall: 0.4883
Epoch 4/50

Epoch 00004: val_accuracy did not improve from 0.86961
284/284 - 2s - loss: 0.2806 - accuracy: 0.8841 - precision: 0.7885 - recall:
0.5855 - val_loss: 0.3400 - val_accuracy: 0.8648 - val_precision: 0.7493 -
val_recall: 0.5086
Epoch 5/50

Epoch 00005: val_accuracy improved from 0.86961 to 0.87909, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.2654 - accuracy: 0.8907 - precision: 0.7982 - recall:
0.6170 - val_loss: 0.3064 - val_accuracy: 0.8791 - val_precision: 0.8212 -
val_recall: 0.5220
Epoch 6/50

Epoch 00006: val_accuracy improved from 0.87909 to 0.88092, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.2521 - accuracy: 0.8968 - precision: 0.8097 - recall:
0.6421 - val_loss: 0.2896 - val_accuracy: 0.8809 - val_precision: 0.7640 -
val_recall: 0.6039
Epoch 7/50

Epoch 00007: val_accuracy improved from 0.88092 to 0.88711, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.2425 - accuracy: 0.9009 - precision: 0.8161 - recall:
0.6599 - val_loss: 0.2743 - val_accuracy: 0.8871 - val_precision: 0.7912 -
val_recall: 0.6081
Epoch 8/50

Epoch 00008: val_accuracy did not improve from 0.88711
284/284 - 2s - loss: 0.2351 - accuracy: 0.9045 - precision: 0.8235 - recall:
0.6734 - val_loss: 0.2940 - val_accuracy: 0.8807 - val_precision: 0.7732 -
val_recall: 0.5891
Epoch 9/50

Epoch 00009: val_accuracy improved from 0.88711 to 0.89026, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.2276 - accuracy: 0.9080 - precision: 0.8288 - recall:
0.6884 - val_loss: 0.2734 - val_accuracy: 0.8903 - val_precision: 0.8189 -
```

```
val_recall: 0.5946
Epoch 10/50

Epoch 00010: val_accuracy improved from 0.89026 to 0.89174, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.2201 - accuracy: 0.9113 - precision: 0.8348 - recall:
0.7011 - val_loss: 0.2653 - val_accuracy: 0.8917 - val_precision: 0.7523 -
val_recall: 0.7012
Epoch 11/50

Epoch 00011: val_accuracy improved from 0.89174 to 0.89677, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.2134 - accuracy: 0.9148 - precision: 0.8425 - recall:
0.7135 - val_loss: 0.2585 - val_accuracy: 0.8968 - val_precision: 0.8228 -
val_recall: 0.6307
Epoch 12/50

Epoch 00012: val_accuracy improved from 0.89677 to 0.90273, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.2078 - accuracy: 0.9164 - precision: 0.8443 - recall:
0.7207 - val_loss: 0.2434 - val_accuracy: 0.9027 - val_precision: 0.8076 -
val_recall: 0.6879
Epoch 13/50

Epoch 00013: val_accuracy did not improve from 0.90273
284/284 - 2s - loss: 0.2019 - accuracy: 0.9190 - precision: 0.8493 - recall:
0.7302 - val_loss: 0.2571 - val_accuracy: 0.8996 - val_precision: 0.8306 -
val_recall: 0.6392
Epoch 14/50

Epoch 00014: val_accuracy improved from 0.90273 to 0.90573, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.1981 - accuracy: 0.9208 - precision: 0.8526 - recall:
0.7370 - val_loss: 0.2338 - val_accuracy: 0.9057 - val_precision: 0.8199 -
val_recall: 0.6904
Epoch 15/50

Epoch 00015: val_accuracy did not improve from 0.90573
284/284 - 2s - loss: 0.1929 - accuracy: 0.9234 - precision: 0.8575 - recall:
0.7461 - val_loss: 0.2526 - val_accuracy: 0.9021 - val_precision: 0.8276 -
val_recall: 0.6580
Epoch 16/50

Epoch 00016: val_accuracy improved from 0.90573 to 0.90854, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.1886 - accuracy: 0.9253 - precision: 0.8616 - recall:
0.7523 - val_loss: 0.2285 - val_accuracy: 0.9085 - val_precision: 0.8377 -
val_recall: 0.6853
```

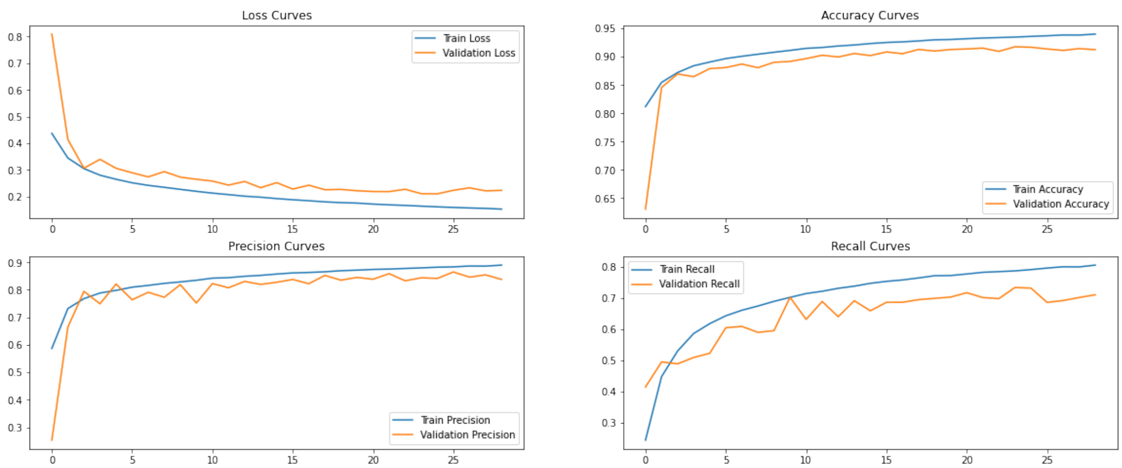
```
Epoch 17/50

Epoch 00017: val_accuracy did not improve from 0.90854
284/284 - 2s - loss: 0.1848 - accuracy: 0.9264 - precision: 0.8632 - recall:
0.7568 - val_loss: 0.2432 - val_accuracy: 0.9055 - val_precision: 0.8224 -
val_recall: 0.6855
Epoch 18/50

Epoch 00018: val_accuracy improved from 0.90854 to 0.91287, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.1806 - accuracy: 0.9280 - precision: 0.8657 - recall:
0.7633 - val_loss: 0.2261 - val_accuracy: 0.9129 - val_precision: 0.8524 -
val_recall: 0.6938
Epoch 19/50

Epoch 00019: val_accuracy did not improve from 0.91287
284/284 - 2s - loss: 0.1777 - accuracy: 0.9300 - precision: 0.8696 - recall:
0.7704 - val_loss: 0.2274 - val_accuracy: 0.9101 - val_precision: 0.8352 -
val_recall: 0.6981
Epoch 20/50

Epoch 00020: val_accuracy did not improve from 0.91287
284/284 - 2s - loss: 0.1760 - accuracy: 0.9305 - precision: 0.8717 - recall:
0.7710 - val_loss: 0.2223 - val_accuracy: 0.9128 - val_precision: 0.8452 -
val_recall: 0.7021
Epoch 21/50

Epoch 00021: val_accuracy improved from 0.91287 to 0.91381, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.1723 - accuracy: 0.9319 - precision: 0.8743 - recall:
0.7760 - val_loss: 0.2193 - val_accuracy: 0.9138 - val_precision: 0.8386 -
val_recall: 0.7161
Epoch 22/50

Epoch 00022: val_accuracy improved from 0.91381 to 0.91527, saving model to
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.1695 - accuracy: 0.9331 - precision: 0.8756 - recall:
0.7813 - val_loss: 0.2191 - val_accuracy: 0.9153 - val_precision: 0.8589 -
val_recall: 0.7005
Epoch 23/50

Epoch 00023: val_accuracy did not improve from 0.91527
284/284 - 2s - loss: 0.1673 - accuracy: 0.9340 - precision: 0.8778 - recall:
0.7836 - val_loss: 0.2278 - val_accuracy: 0.9095 - val_precision: 0.8328 -
val_recall: 0.6972
Epoch 24/50

Epoch 00024: val_accuracy improved from 0.91527 to 0.91766, saving model to
```

```
cnn_model_checkpoints/checkpoint
284/284 - 2s - loss: 0.1646 - accuracy: 0.9348 - precision: 0.8799 - recall:
0.7859 - val_loss: 0.2113 - val_accuracy: 0.9177 - val_precision: 0.8438 -
val_recall: 0.7327
Epoch 25/50

Epoch 00025: val_accuracy did not improve from 0.91766
284/284 - 2s - loss: 0.1619 - accuracy: 0.9361 - precision: 0.8824 - recall:
0.7900 - val_loss: 0.2107 - val_accuracy: 0.9168 - val_precision: 0.8415 -
val_recall: 0.7308
Epoch 26/50

Epoch 00026: val_accuracy did not improve from 0.91766
284/284 - 2s - loss: 0.1598 - accuracy: 0.9372 - precision: 0.8835 - recall:
0.7950 - val_loss: 0.2240 - val_accuracy: 0.9138 - val_precision: 0.8650 -
val_recall: 0.6851
Epoch 27/50

Epoch 00027: val_accuracy did not improve from 0.91766
284/284 - 2s - loss: 0.1577 - accuracy: 0.9385 - precision: 0.8867 - recall:
0.7990 - val_loss: 0.2333 - val_accuracy: 0.9112 - val_precision: 0.8462 -
val_recall: 0.6909
Epoch 28/50

Epoch 00028: val_accuracy did not improve from 0.91766
284/284 - 2s - loss: 0.1562 - accuracy: 0.9384 - precision: 0.8863 - recall:
0.7987 - val_loss: 0.2220 - val_accuracy: 0.9144 - val_precision: 0.8545 -
val_recall: 0.7007
Epoch 29/50

Epoch 00029: val_accuracy did not improve from 0.91766
284/284 - 2s - loss: 0.1535 - accuracy: 0.9401 - precision: 0.8899 - recall:
0.8044 - val_loss: 0.2237 - val_accuracy: 0.9126 - val_precision: 0.8379 -
val_recall: 0.7094
```

[ ]:

```
# Reults Visualization

plt.rcParams["figure.figsize"] = 20, 8

plt.subplot(2, 2, 1)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
plt.title("Loss Curves")

plt.subplot(2, 2, 2)
plt.plot(history.history["accuracy"], label="Train Accuracy")
```

```python
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.legend()
plt.title("Accuracy Curves")


plt.subplot(2, 2, 3)
plt.plot(history.history["precision"], label="Train Precision")
plt.plot(history.history["val_precision"], label="Validation Precision")
plt.legend()
plt.title("Precision Curves")


plt.subplot(2, 2, 4)
plt.plot(history.history["recall"], label="Train Recall")
plt.plot(history.history["val_recall"], label="Validation Recall")
plt.legend()
plt.title("Recall Curves")

plt.show()
```




[ ]:

```python
# Evalaution on Test Set

cnn_res = model_2.evaluate(x = X_test[..., np.newaxis],
                           y = Y_test,
                           return_dict=True,
                           verbose=False)

for key in cnn_res.keys():
    if key != "loss":
      print("{}: {:0.6f}%".format(key.capitalize(), 100*cnn_res[key]))
    else:
```

```
    print("{}: {:0.6f}".format(key.capitalize(), cnn_res[key]))
```

```
Loss: 0.230679
Accuracy: 91.500604%
Precision: 81.986928%
Recall: 70.090562%
```

### 1.5 4. Get model predictions

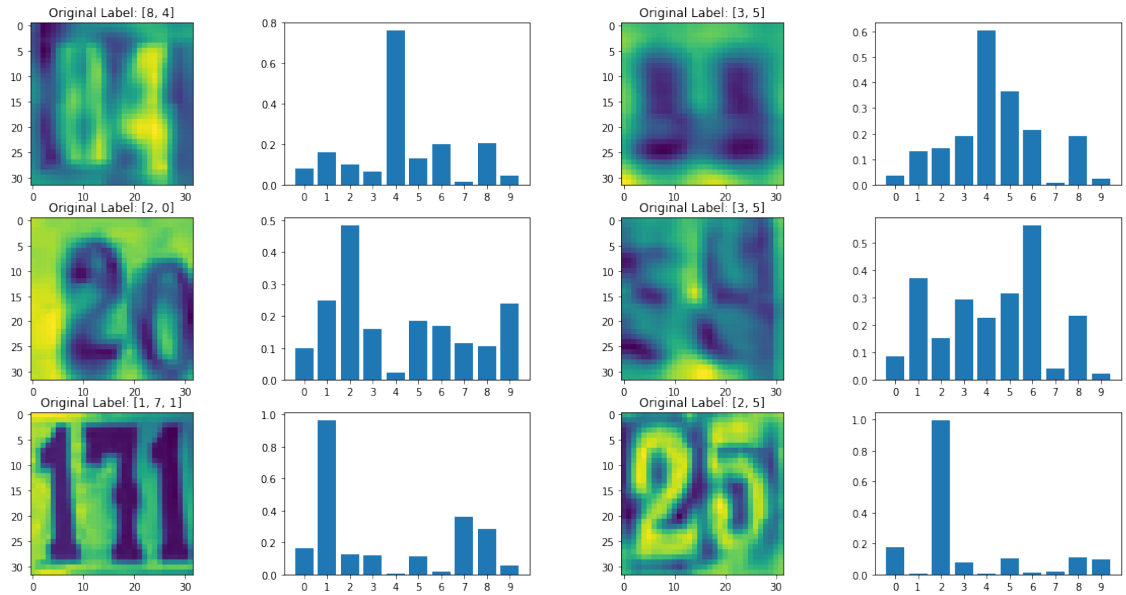
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the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

[ ]:
```
# Loading best Sequential Model

best_seq = SequentialModel(input_shape=(32, 32))
best_seq.load_weights(seq_path)
```

[ ]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7fc2a80d7048>
```

[ ]:
```
# NOTE: I have done this exercise for MultiLabel Classification.
# Hence, the reviewer should see probability across multiple digits.
```

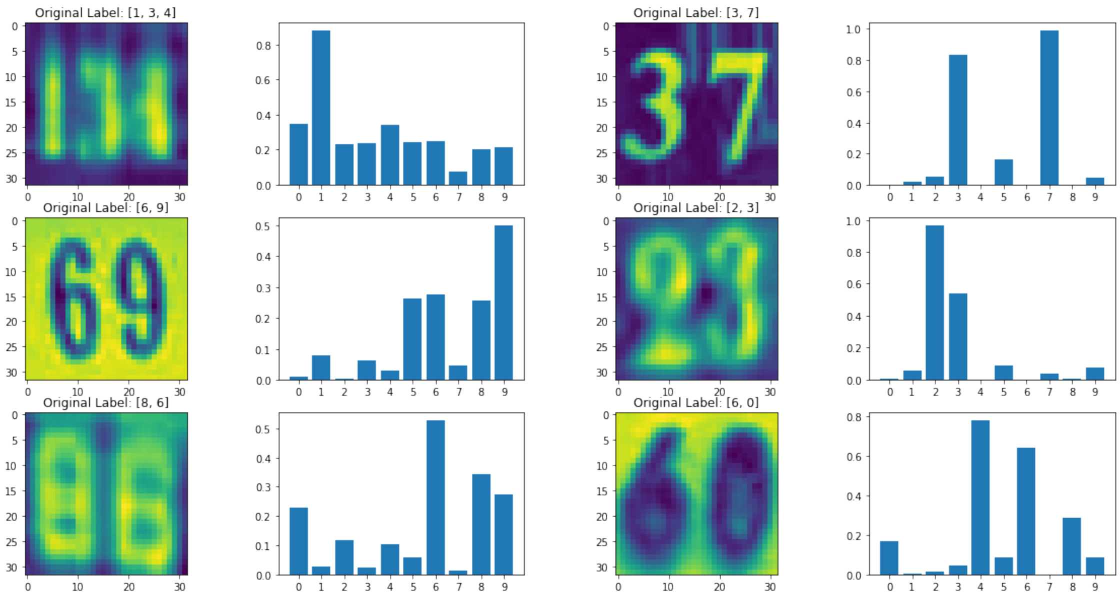
[ ]:
```
# Predictions for best Sequential Model

plt.rcParams["figure.figsize"] = 20, 10

idx = np.random.randint(0, SIZE["train"], 6)

for i in range(6):
  plt.subplot(3, 4, 2*i+1)
  plt.imshow(train_images[idx[i]])
  plt.title("Original Label: {}".format(train_labels[idx[i]]))

  plt.subplot(3, 4, 2*i + 2)
  plt.bar(x = np.arange(10), height=best_seq.predict(train_images[idx[i]][np.
 ↪newaxis, ...])[0])
  plt.xticks(np.arange(10))

plt.show()
```




[ ]:
```
# Loading best CNN Model

best_cnn = CNNModel(input_shape=(32, 32, 1))
best_cnn.load_weights(cnn_path)
```

[ ]: <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fc31608e908>

[ ]:
```
# Predictions for best CNN Model

plt.rcParams["figure.figsize"] = 20, 10

idx = np.random.randint(0, SIZE["train"], 6)

for i in range(6):
  plt.subplot(3, 4, 2*i+1)
  plt.imshow(train_images[idx[i]])
  plt.title("Original Label: {}".format(train_labels[idx[i]]))

  plt.subplot(3, 4, 2*i + 2)
  plt.bar(x = np.arange(10), height=best_seq.predict(train_images[idx[i]][np.
↪newaxis, ..., np.newaxis])[0])
  plt.xticks(np.arange(10))

plt.show()
```


Original Label: [1, 3, 4]
Original Label: [3, 7]
Original Label: [6, 9]
Original Label: [2, 3]
Original Label: [8, 6]
Original Label: [6, 0]

[ ]: