Capstone_Project_tf_course_one

November 22, 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.

[1]:
```
import tensorflow as tf
from scipy.io import loadmat
```

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.

[2]:
```
# Run this cell to connect to your Drive folder

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

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

[3]:
```
# Load the dataset from your Drive folder

train = loadmat('/content/gdrive/MyDrive/Colab Notebooks/train_32x32.mat')
test = loadmat('/content/gdrive/MyDrive/Colab Notebooks/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.

[4]:
```
import numpy
train_data=train['X']
train_data=numpy.moveaxis(train_data,3,0)
train_data=train_data/255.
train_labels=train['y']
test_data=test['X']
test_data=numpy.moveaxis(test_data,3,0)
test_data=test_data/255.
test_labels=test['y']
print(train_data.shape)
print(test_data.shape)
print(train_labels.shape)
```

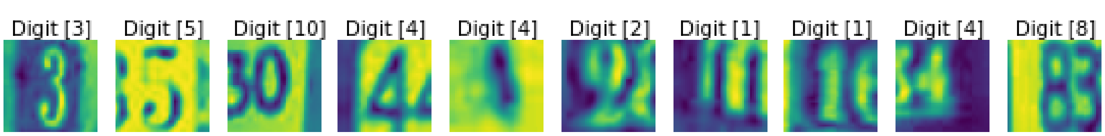
```
(73257, 32, 32, 3)
(26032, 32, 32, 3)
(73257, 1)
```

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
num_train=train_data.shape[0]
np.random.seed(0)
random_idx=np.random.choice(num_train,10)
fig,axes=plt.subplots(1,10,figsize=(10,1))
for i in range(10):
    axes[i].set_axis_off()
    axes[i].imshow(train_data[random_idx[i]])
    axes[i].text(2., -1.5, f'Digit {train_labels[random_idx[i],]}')
```

[6]:
```
train_data_gray=np.mean(train_data,axis=3)
test_data_gray=np.mean(test_data,axis=3)
train_data_gray=train_data_gray[...,np.newaxis]
test_data_gray=test_data_gray[...,np.newaxis]
print(train_data_gray.shape)
```

```
(73257, 32, 32, 1)
```

[7]:
```
np.random.seed(0)
random_idx_1=np.random.choice(num_train,10)
fig,axes=plt.subplots(1,10,figsize=(10,1))
for i in range(10):
    axes[i].set_axis_off()
    axes[i].imshow(np.squeeze(train_data_gray[random_idx_1[i]]))
    axes[i].text(2., -1.5, f'Digit {train_labels[random_idx_1[i],]}')
```



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.

[8]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D,␣
↪MaxPooling2D,Flatten,BatchNormalization,Dropout
```

[9]:
```
def get_new_model(input_shape):
    model=Sequential([
        Flatten(input_shape=(32, 32)),
        Dense(512, activation='tanh', name='Dense_1'),
        Dense(128, activation='tanh', name='Dense_2'),
        Dense(128, activation='tanh', name='Dense_3'),
        Dense(10, activation='softmax', name='final'),
    ])
    return model
```

[10]:
```
model=get_new_model(train_data_gray[0].shape)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
Dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
Dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
Dense_3 (Dense)              (None, 128)               16512
_________________________________________________________________
final (Dense)                (None, 10)                1290
=================================================================
Total params: 608,266
Trainable params: 608,266
Non-trainable params: 0
_________________________________________________________________
```

[11]:
```
opt=tf.keras.optimizers.Adam(learning_rate=0.001)
model.
 ↪compile(optimizer=opt,loss='categorical_crossentropy',metrics=['accuracy'])
```

[12]:
```
from tensorflow.keras.callbacks import ModelCheckpoint
#early_stopping=tf.keras.callbacks.EarlyStopping(patience=3)
epoch_accuracy_callback=tf.keras.callbacks.LambdaCallback(
                       on_epoch_end=lambda epoch,logs:print('\n after␣
 ↪epoch{},the accuracy is {:7.2f}'.format(epoch,logs['accuracy'])))
checkpoint_best_only_path='checkpoint_best_only/checkpoint'
checkpoint_best_only=ModelCheckpoint(filepath=checkpoint_best_only_path,
                                    save_weights_only=True,
                                    save_freq='epoch',
                                    save_best_only=True,
                                    monitor='val_accuracy',
                                   verbose=1)
```

[13]:
```
train_labels=[i if i!=10 else 0 for i in train_labels ]
train_labels=tf.keras.utils.to_categorical(np.array(train_labels))
```

[14]:
```
history=model.fit(train_data_gray,
                  train_labels,
                  validation_split=0.2,
                  epochs=30,
                  batch_size=64,
                  verbose=1,
                  callbacks=[epoch_accuracy_callback,checkpoint_best_only])
```

```
Epoch 1/30
915/916 [============================>.] - ETA: 0s - loss: 2.2277 - accuracy:
0.1881
 after epoch0,the accuracy is    0.19

Epoch 00001: val_accuracy improved from -inf to 0.24939, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 2.2275 -
accuracy: 0.1882 - val_loss: 2.1140 - val_accuracy: 0.2494
Epoch 2/30
912/916 [============================>.] - ETA: 0s - loss: 1.9169 - accuracy:
0.3169
 after epoch1,the accuracy is    0.32

Epoch 00002: val_accuracy improved from 0.24939 to 0.42540, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 1.9160 -
accuracy: 0.3172 - val_loss: 1.6446 - val_accuracy: 0.4254
Epoch 3/30
916/916 [==============================] - ETA: 0s - loss: 1.5647 - accuracy:
0.4561
```

```
 after epoch2,the accuracy is    0.46

Epoch 00003: val_accuracy improved from 0.42540 to 0.48990, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 1.5647 -
accuracy: 0.4561 - val_loss: 1.4971 - val_accuracy: 0.4899
Epoch 4/30
910/916 [============================>.] - ETA: 0s - loss: 1.4026 - accuracy:
0.5268
 after epoch3,the accuracy is    0.53

Epoch 00004: val_accuracy improved from 0.48990 to 0.57501, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 1.4015 -
accuracy: 0.5272 - val_loss: 1.3121 - val_accuracy: 0.5750
Epoch 5/30
912/916 [============================>.] - ETA: 0s - loss: 1.2684 - accuracy:
0.5859
 after epoch4,the accuracy is    0.59

Epoch 00005: val_accuracy improved from 0.57501 to 0.60579, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 1.2682 -
accuracy: 0.5860 - val_loss: 1.2273 - val_accuracy: 0.6058
Epoch 6/30
911/916 [============================>.] - ETA: 0s - loss: 1.1834 - accuracy:
0.6227
 after epoch5,the accuracy is    0.62

Epoch 00006: val_accuracy improved from 0.60579 to 0.62851, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 1.1834 -
accuracy: 0.6228 - val_loss: 1.1753 - val_accuracy: 0.6285
Epoch 7/30
910/916 [============================>.] - ETA: 0s - loss: 1.1315 - accuracy:
0.6414
 after epoch6,the accuracy is    0.64

Epoch 00007: val_accuracy did not improve from 0.62851
916/916 [==============================] - 7s 8ms/step - loss: 1.1315 -
accuracy: 0.6415 - val_loss: 1.2207 - val_accuracy: 0.6103
Epoch 8/30
913/916 [============================>.] - ETA: 0s - loss: 1.0918 - accuracy:
0.6555
 after epoch7,the accuracy is    0.66

Epoch 00008: val_accuracy improved from 0.62851 to 0.65192, saving model to
checkpoint_best_only/checkpoint
```

```
916/916 [==============================] - 7s 8ms/step - loss: 1.0921 -
accuracy: 0.6553 - val_loss: 1.1086 - val_accuracy: 0.6519
Epoch 9/30
910/916 [============================>.] - ETA: 0s - loss: 1.0583 - accuracy:
0.6660
 after epoch8,the accuracy is    0.67

Epoch 00009: val_accuracy improved from 0.65192 to 0.66510, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 1.0578 -
accuracy: 0.6661 - val_loss: 1.0705 - val_accuracy: 0.6651
Epoch 10/30
912/916 [============================>.] - ETA: 0s - loss: 1.0351 - accuracy:
0.6755
 after epoch9,the accuracy is    0.68

Epoch 00010: val_accuracy improved from 0.66510 to 0.68271, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 1.0353 -
accuracy: 0.6755 - val_loss: 1.0166 - val_accuracy: 0.6827
Epoch 11/30
911/916 [============================>.] - ETA: 0s - loss: 1.0066 - accuracy:
0.6841
 after epoch10,the accuracy is    0.68

Epoch 00011: val_accuracy did not improve from 0.68271
916/916 [==============================] - 7s 8ms/step - loss: 1.0070 -
accuracy: 0.6838 - val_loss: 1.0908 - val_accuracy: 0.6564
Epoch 12/30
915/916 [============================>.] - ETA: 0s - loss: 0.9894 - accuracy:
0.6910
 after epoch11,the accuracy is    0.69

Epoch 00012: val_accuracy did not improve from 0.68271
916/916 [==============================] - 7s 8ms/step - loss: 0.9893 -
accuracy: 0.6911 - val_loss: 1.0090 - val_accuracy: 0.6807
Epoch 13/30
916/916 [==============================] - ETA: 0s - loss: 0.9713 - accuracy:
0.6959
 after epoch12,the accuracy is    0.70

Epoch 00013: val_accuracy improved from 0.68271 to 0.68851, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 0.9713 -
accuracy: 0.6959 - val_loss: 0.9828 - val_accuracy: 0.6885
Epoch 14/30
911/916 [============================>.] - ETA: 0s - loss: 0.9560 - accuracy:
0.7019
```

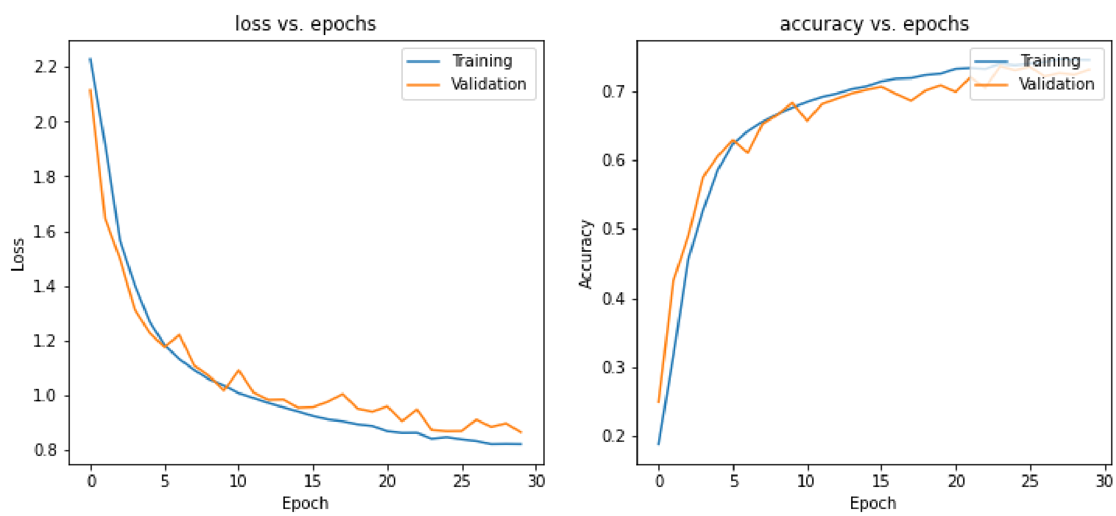
```
 after epoch13,the accuracy is    0.70

Epoch 00014: val_accuracy improved from 0.68851 to 0.69608, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 0.9554 -
accuracy: 0.7023 - val_loss: 0.9839 - val_accuracy: 0.6961
Epoch 15/30
913/916 [============================>.] - ETA: 0s - loss: 0.9404 - accuracy:
0.7063
 after epoch14,the accuracy is    0.71

Epoch 00015: val_accuracy improved from 0.69608 to 0.70141, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 10s 11ms/step - loss: 0.9401 -
accuracy: 0.7063 - val_loss: 0.9539 - val_accuracy: 0.7014
Epoch 16/30
916/916 [==============================] - ETA: 0s - loss: 0.9236 - accuracy:
0.7135
 after epoch15,the accuracy is    0.71

Epoch 00016: val_accuracy improved from 0.70141 to 0.70577, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 8s 9ms/step - loss: 0.9236 -
accuracy: 0.7135 - val_loss: 0.9561 - val_accuracy: 0.7058
Epoch 17/30
911/916 [============================>.] - ETA: 0s - loss: 0.9104 - accuracy:
0.7177
 after epoch16,the accuracy is    0.72

Epoch 00017: val_accuracy did not improve from 0.70577
916/916 [==============================] - 7s 8ms/step - loss: 0.9112 -
accuracy: 0.7175 - val_loss: 0.9760 - val_accuracy: 0.6953
Epoch 18/30
916/916 [==============================] - ETA: 0s - loss: 0.9041 - accuracy:
0.7184
 after epoch17,the accuracy is    0.72

Epoch 00018: val_accuracy did not improve from 0.70577
916/916 [==============================] - 7s 8ms/step - loss: 0.9041 -
accuracy: 0.7184 - val_loss: 1.0041 - val_accuracy: 0.6857
Epoch 19/30
910/916 [============================>.] - ETA: 0s - loss: 0.8932 - accuracy:
0.7227
 after epoch18,the accuracy is    0.72

Epoch 00019: val_accuracy did not improve from 0.70577
916/916 [==============================] - 7s 8ms/step - loss: 0.8931 -
accuracy: 0.7229 - val_loss: 0.9507 - val_accuracy: 0.7006
```

```
Epoch 20/30
915/916 [============================>.] - ETA: 0s - loss: 0.8870 - accuracy:
0.7248
 after epoch19,the accuracy is    0.72

Epoch 00020: val_accuracy improved from 0.70577 to 0.70782, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 7s 8ms/step - loss: 0.8871 -
accuracy: 0.7248 - val_loss: 0.9393 - val_accuracy: 0.7078
Epoch 21/30
916/916 [==============================] - ETA: 0s - loss: 0.8680 - accuracy:
0.7319
 after epoch20,the accuracy is    0.73

Epoch 00021: val_accuracy did not improve from 0.70782
916/916 [==============================] - 8s 8ms/step - loss: 0.8680 -
accuracy: 0.7319 - val_loss: 0.9589 - val_accuracy: 0.6983
Epoch 22/30
916/916 [==============================] - ETA: 0s - loss: 0.8616 - accuracy:
0.7331
 after epoch21,the accuracy is    0.73

Epoch 00022: val_accuracy improved from 0.70782 to 0.71963, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 8s 8ms/step - loss: 0.8616 -
accuracy: 0.7331 - val_loss: 0.9047 - val_accuracy: 0.7196
Epoch 23/30
914/916 [============================>.] - ETA: 0s - loss: 0.8619 - accuracy:
0.7317
 after epoch22,the accuracy is    0.73

Epoch 00023: val_accuracy did not improve from 0.71963
916/916 [==============================] - 7s 8ms/step - loss: 0.8620 -
accuracy: 0.7316 - val_loss: 0.9474 - val_accuracy: 0.7040
Epoch 24/30
912/916 [============================>.] - ETA: 0s - loss: 0.8406 - accuracy:
0.7390
 after epoch23,the accuracy is    0.74

Epoch 00024: val_accuracy improved from 0.71963 to 0.73567, saving model to
checkpoint_best_only/checkpoint
916/916 [==============================] - 8s 8ms/step - loss: 0.8406 -
accuracy: 0.7391 - val_loss: 0.8724 - val_accuracy: 0.7357
Epoch 25/30
912/916 [============================>.] - ETA: 0s - loss: 0.8469 - accuracy:
0.7366
 after epoch24,the accuracy is    0.74
```

```
Epoch 00025: val_accuracy did not improve from 0.73567
916/916 [==============================] - 7s 8ms/step - loss: 0.8465 -
accuracy: 0.7368 - val_loss: 0.8678 - val_accuracy: 0.7297
Epoch 26/30
914/916 [============================>.] - ETA: 0s - loss: 0.8385 - accuracy:
0.7392
 after epoch25,the accuracy is    0.74

Epoch 00026: val_accuracy did not improve from 0.73567
916/916 [==============================] - 7s 8ms/step - loss: 0.8385 -
accuracy: 0.7393 - val_loss: 0.8684 - val_accuracy: 0.7347
Epoch 27/30
910/916 [============================>.] - ETA: 0s - loss: 0.8325 - accuracy:
0.7411
 after epoch26,the accuracy is    0.74

Epoch 00027: val_accuracy did not improve from 0.73567
916/916 [==============================] - 7s 8ms/step - loss: 0.8321 -
accuracy: 0.7413 - val_loss: 0.9103 - val_accuracy: 0.7209
Epoch 28/30
913/916 [============================>.] - ETA: 0s - loss: 0.8203 - accuracy:
0.7460
 after epoch27,the accuracy is    0.75

Epoch 00028: val_accuracy did not improve from 0.73567
916/916 [==============================] - 7s 8ms/step - loss: 0.8206 -
accuracy: 0.7458 - val_loss: 0.8836 - val_accuracy: 0.7256
Epoch 29/30
909/916 [============================>.] - ETA: 0s - loss: 0.8209 - accuracy:
0.7449
 after epoch28,the accuracy is    0.74

Epoch 00029: val_accuracy did not improve from 0.73567
916/916 [==============================] - 7s 8ms/step - loss: 0.8214 -
accuracy: 0.7449 - val_loss: 0.8965 - val_accuracy: 0.7232
Epoch 30/30
911/916 [============================>.] - ETA: 0s - loss: 0.8208 - accuracy:
0.7445
 after epoch29,the accuracy is    0.74

Epoch 00030: val_accuracy did not improve from 0.73567
916/916 [==============================] - 8s 8ms/step - loss: 0.8207 -
accuracy: 0.7446 - val_loss: 0.8641 - val_accuracy: 0.7307
```

[15]:
```
import matplotlib.pyplot as plt
import pandas as pd
fig=plt.figure(figsize=(12,5))
```

```
fig.add_subplot(121)

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')

fig.add_subplot(122)

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')

plt.show()
```



[16]:
```
test_labels=[i if i!=10 else 0 for i in test_labels ]
test_labels=tf.keras.utils.to_categorical(np.array(test_labels))
model.evaluate(test_data_gray,test_labels)
```

```
814/814 [==============================] - 2s 2ms/step - loss: 0.9387 -
accuracy: 0.7120
```

[16]: [0.9386504292488098, 0.7120466828346252]

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.

[17]:
```
model_cnn=Sequential([
                          ␣
 ↪Conv2D(16,kernel_size=3,activation='tanh',input_shape=(32,32,1)),
                          MaxPooling2D(pool_size=3),
                          Dense(64,activation='tanh'),
                          Flatten(),
                          Dense(10,activation='softmax')
])
```

[18]:
```
model_cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 16)        160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 10, 10, 16)        0
_________________________________________________________________
dense (Dense)                (None, 10, 10, 64)        1088
_________________________________________________________________
flatten_1 (Flatten)          (None, 6400)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                64010
=================================================================
Total params: 65,258
Trainable params: 65,258
Non-trainable params: 0
_________________________________________________________________
```

[19]:
```
opt=tf.keras.optimizers.Adam(learning_rate=0.001)
model_cnn.
 ↪compile(optimizer=opt,loss='categorical_crossentropy',metrics=['accuracy'])
epoch_accuracy_callback_cnn=tf.keras.callbacks.LambdaCallback(
                        on_epoch_end=lambda epoch,logs:print('\n after␣
 ↪epoch{},the accuracy is {:7.2f}'.format(epoch,logs['accuracy'])))
checkpoint_best_only_cnn_path='checkpoint_best_only_cnn/checkpoint'
checkpoint_best_only_cnn=ModelCheckpoint(filepath=checkpoint_best_only_cnn_path,
                                     save_weights_only=True,
                                     save_freq='epoch',
                                     save_best_only=True,
                                     monitor='val_accuracy',
                                    verbose=1)
```

[20]:
```
history_cnn=model_cnn.fit(train_data_gray,
              train_labels,
              validation_split=0.2,
              epochs=30,
              batch_size=64,
              verbose=1,
              callbacks=[epoch_accuracy_callback_cnn,checkpoint_best_only_cnn])
```

```
Epoch 1/30
915/916 [============================>.] - ETA: 0s - loss: 1.3787 - accuracy:
0.5619
 after epoch0,the accuracy is    0.56

Epoch 00001: val_accuracy improved from -inf to 0.73956, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 28ms/step - loss: 1.3785 -
accuracy: 0.5620 - val_loss: 0.9244 - val_accuracy: 0.7396
Epoch 2/30
915/916 [============================>.] - ETA: 0s - loss: 0.7919 - accuracy:
0.7742
 after epoch1,the accuracy is    0.77

Epoch 00002: val_accuracy improved from 0.73956 to 0.79928, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 28ms/step - loss: 0.7920 -
accuracy: 0.7742 - val_loss: 0.7415 - val_accuracy: 0.7993
Epoch 3/30
915/916 [============================>.] - ETA: 0s - loss: 0.6879 - accuracy:
0.8076
 after epoch2,the accuracy is    0.81

Epoch 00003: val_accuracy improved from 0.79928 to 0.80685, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 27ms/step - loss: 0.6882 -
```

```
accuracy: 0.8076 - val_loss: 0.7129 - val_accuracy: 0.8069
Epoch 4/30
915/916 [============================>.] - ETA: 0s - loss: 0.6601 - accuracy:
0.8154
 after epoch3,the accuracy is    0.82

Epoch 00004: val_accuracy improved from 0.80685 to 0.80699, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 27ms/step - loss: 0.6599 -
accuracy: 0.8154 - val_loss: 0.7055 - val_accuracy: 0.8070
Epoch 5/30
916/916 [==============================] - ETA: 0s - loss: 0.6415 - accuracy:
0.8214
 after epoch4,the accuracy is    0.82

Epoch 00005: val_accuracy improved from 0.80699 to 0.81402, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 27ms/step - loss: 0.6415 -
accuracy: 0.8214 - val_loss: 0.6915 - val_accuracy: 0.8140
Epoch 6/30
916/916 [==============================] - ETA: 0s - loss: 0.6297 - accuracy:
0.8242
 after epoch5,the accuracy is    0.82

Epoch 00006: val_accuracy improved from 0.81402 to 0.81600, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 27ms/step - loss: 0.6297 -
accuracy: 0.8242 - val_loss: 0.6887 - val_accuracy: 0.8160
Epoch 7/30
914/916 [============================>.] - ETA: 0s - loss: 0.6198 - accuracy:
0.8272
 after epoch6,the accuracy is    0.83

Epoch 00007: val_accuracy improved from 0.81600 to 0.81743, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 26s 28ms/step - loss: 0.6201 -
accuracy: 0.8272 - val_loss: 0.6863 - val_accuracy: 0.8174
Epoch 8/30
914/916 [============================>.] - ETA: 0s - loss: 0.6114 - accuracy:
0.8283
 after epoch7,the accuracy is    0.83

Epoch 00008: val_accuracy improved from 0.81743 to 0.82064, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 27ms/step - loss: 0.6111 -
accuracy: 0.8284 - val_loss: 0.6753 - val_accuracy: 0.8206
Epoch 9/30
914/916 [============================>.] - ETA: 0s - loss: 0.6034 - accuracy:
```

```
0.8334
 after epoch8,the accuracy is    0.83

Epoch 00009: val_accuracy did not improve from 0.82064
916/916 [==============================] - 25s 28ms/step - loss: 0.6036 -
accuracy: 0.8334 - val_loss: 0.6874 - val_accuracy: 0.8168
Epoch 10/30
914/916 [============================>.] - ETA: 0s - loss: 0.5937 - accuracy:
0.8345
 after epoch9,the accuracy is    0.83

Epoch 00010: val_accuracy improved from 0.82064 to 0.82098, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 28ms/step - loss: 0.5936 -
accuracy: 0.8346 - val_loss: 0.6734 - val_accuracy: 0.8210
Epoch 11/30
915/916 [============================>.] - ETA: 0s - loss: 0.5863 - accuracy:
0.8371
 after epoch10,the accuracy is    0.84

Epoch 00011: val_accuracy did not improve from 0.82098
916/916 [==============================] - 25s 27ms/step - loss: 0.5863 -
accuracy: 0.8371 - val_loss: 0.6799 - val_accuracy: 0.8181
Epoch 12/30
916/916 [==============================] - ETA: 0s - loss: 0.5815 - accuracy:
0.8378
 after epoch11,the accuracy is    0.84

Epoch 00012: val_accuracy improved from 0.82098 to 0.82159, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 27ms/step - loss: 0.5815 -
accuracy: 0.8378 - val_loss: 0.6682 - val_accuracy: 0.8216
Epoch 13/30
914/916 [============================>.] - ETA: 0s - loss: 0.5744 - accuracy:
0.8379
 after epoch12,the accuracy is    0.84

Epoch 00013: val_accuracy improved from 0.82159 to 0.82651, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 25s 28ms/step - loss: 0.5744 -
accuracy: 0.8379 - val_loss: 0.6607 - val_accuracy: 0.8265
Epoch 14/30
915/916 [============================>.] - ETA: 0s - loss: 0.5687 - accuracy:
0.8417
 after epoch13,the accuracy is    0.84

Epoch 00014: val_accuracy did not improve from 0.82651
916/916 [==============================] - 26s 28ms/step - loss: 0.5686 -
```

```
accuracy: 0.8417 - val_loss: 0.6660 - val_accuracy: 0.8245
Epoch 15/30
915/916 [============================>.] - ETA: 0s - loss: 0.5639 - accuracy:
0.8419
 after epoch14,the accuracy is    0.84

Epoch 00015: val_accuracy did not improve from 0.82651
916/916 [==============================] - 26s 28ms/step - loss: 0.5638 -
accuracy: 0.8419 - val_loss: 0.6683 - val_accuracy: 0.8242
Epoch 16/30
915/916 [============================>.] - ETA: 0s - loss: 0.5602 - accuracy:
0.8439
 after epoch15,the accuracy is    0.84

Epoch 00016: val_accuracy did not improve from 0.82651
916/916 [==============================] - 25s 28ms/step - loss: 0.5603 -
accuracy: 0.8439 - val_loss: 0.6660 - val_accuracy: 0.8253
Epoch 17/30
915/916 [============================>.] - ETA: 0s - loss: 0.5545 - accuracy:
0.8448
 after epoch16,the accuracy is    0.84

Epoch 00017: val_accuracy did not improve from 0.82651
916/916 [==============================] - 25s 28ms/step - loss: 0.5543 -
accuracy: 0.8449 - val_loss: 0.6637 - val_accuracy: 0.8227
Epoch 18/30
915/916 [============================>.] - ETA: 0s - loss: 0.5499 - accuracy:
0.8462
 after epoch17,the accuracy is    0.85

Epoch 00018: val_accuracy did not improve from 0.82651
916/916 [==============================] - 25s 28ms/step - loss: 0.5498 -
accuracy: 0.8462 - val_loss: 0.6761 - val_accuracy: 0.8247
Epoch 19/30
915/916 [============================>.] - ETA: 0s - loss: 0.5450 - accuracy:
0.8466
 after epoch18,the accuracy is    0.85

Epoch 00019: val_accuracy improved from 0.82651 to 0.82658, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 27s 29ms/step - loss: 0.5450 -
accuracy: 0.8465 - val_loss: 0.6624 - val_accuracy: 0.8266
Epoch 20/30
915/916 [============================>.] - ETA: 0s - loss: 0.5411 - accuracy:
0.8473
 after epoch19,the accuracy is    0.85

Epoch 00020: val_accuracy did not improve from 0.82658
```

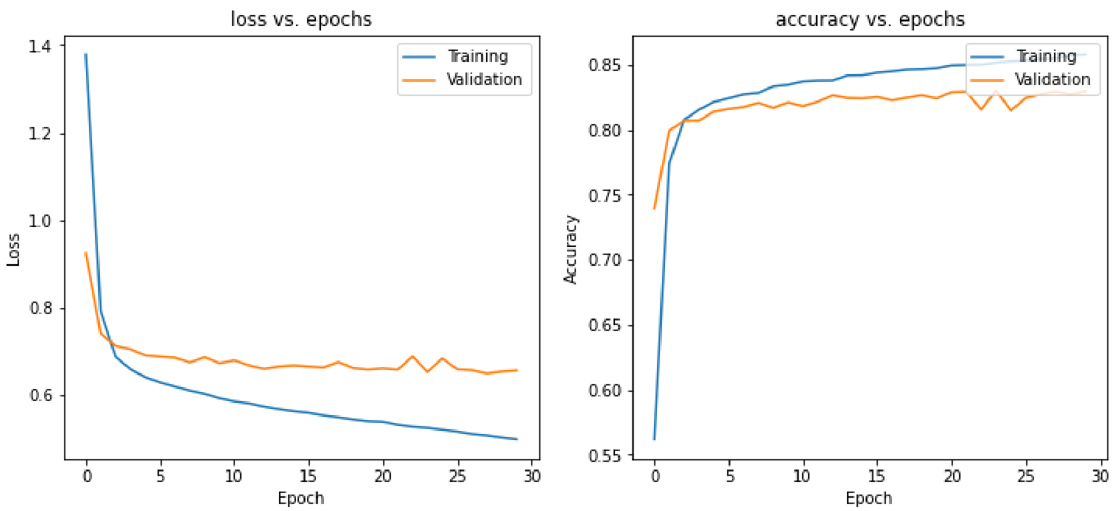
```
916/916 [==============================] - 26s 28ms/step - loss: 0.5410 -
accuracy: 0.8473 - val_loss: 0.6591 - val_accuracy: 0.8242
Epoch 21/30
915/916 [============================>.] - ETA: 0s - loss: 0.5397 - accuracy:
0.8495
 after epoch20,the accuracy is    0.85

Epoch 00021: val_accuracy improved from 0.82658 to 0.82890, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 26s 28ms/step - loss: 0.5396 -
accuracy: 0.8495 - val_loss: 0.6617 - val_accuracy: 0.8289
Epoch 22/30
915/916 [============================>.] - ETA: 0s - loss: 0.5327 - accuracy:
0.8498
 after epoch21,the accuracy is    0.85

Epoch 00022: val_accuracy improved from 0.82890 to 0.82924, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 26s 28ms/step - loss: 0.5327 -
accuracy: 0.8498 - val_loss: 0.6589 - val_accuracy: 0.8292
Epoch 23/30
914/916 [============================>.] - ETA: 0s - loss: 0.5287 - accuracy:
0.8500
 after epoch22,the accuracy is    0.85

Epoch 00023: val_accuracy did not improve from 0.82924
916/916 [==============================] - 26s 28ms/step - loss: 0.5284 -
accuracy: 0.8500 - val_loss: 0.6889 - val_accuracy: 0.8155
Epoch 24/30
914/916 [============================>.] - ETA: 0s - loss: 0.5253 - accuracy:
0.8517
 after epoch23,the accuracy is    0.85

Epoch 00024: val_accuracy improved from 0.82924 to 0.83019, saving model to
checkpoint_best_only_cnn/checkpoint
916/916 [==============================] - 26s 28ms/step - loss: 0.5258 -
accuracy: 0.8515 - val_loss: 0.6538 - val_accuracy: 0.8302
Epoch 25/30
916/916 [==============================] - ETA: 0s - loss: 0.5216 - accuracy:
0.8526
 after epoch24,the accuracy is    0.85

Epoch 00025: val_accuracy did not improve from 0.83019
916/916 [==============================] - 26s 28ms/step - loss: 0.5216 -
accuracy: 0.8526 - val_loss: 0.6843 - val_accuracy: 0.8148
Epoch 26/30
915/916 [============================>.] - ETA: 0s - loss: 0.5171 - accuracy:
0.8532
```

```
 after epoch25,the accuracy is    0.85

Epoch 00026: val_accuracy did not improve from 0.83019
916/916 [==============================] - 26s 28ms/step - loss: 0.5172 -
accuracy: 0.8532 - val_loss: 0.6598 - val_accuracy: 0.8245
Epoch 27/30
914/916 [============================>.] - ETA: 0s - loss: 0.5117 - accuracy:
0.8546
 after epoch26,the accuracy is    0.85

Epoch 00027: val_accuracy did not improve from 0.83019
916/916 [==============================] - 26s 28ms/step - loss: 0.5118 -
accuracy: 0.8545 - val_loss: 0.6577 - val_accuracy: 0.8273
Epoch 28/30
915/916 [============================>.] - ETA: 0s - loss: 0.5081 - accuracy:
0.8548
 after epoch27,the accuracy is    0.85

Epoch 00028: val_accuracy did not improve from 0.83019
916/916 [==============================] - 26s 28ms/step - loss: 0.5084 -
accuracy: 0.8547 - val_loss: 0.6503 - val_accuracy: 0.8294
Epoch 29/30
914/916 [============================>.] - ETA: 0s - loss: 0.5034 - accuracy:
0.8572
 after epoch28,the accuracy is    0.86

Epoch 00029: val_accuracy did not improve from 0.83019
916/916 [==============================] - 26s 28ms/step - loss: 0.5037 -
accuracy: 0.8571 - val_loss: 0.6546 - val_accuracy: 0.8273
Epoch 30/30
914/916 [============================>.] - ETA: 0s - loss: 0.4995 - accuracy:
0.8578
 after epoch29,the accuracy is    0.86

Epoch 00030: val_accuracy did not improve from 0.83019
916/916 [==============================] - 26s 29ms/step - loss: 0.4998 -
accuracy: 0.8578 - val_loss: 0.6571 - val_accuracy: 0.8294
```

[21]:

```
fig_cnn=plt.figure(figsize=(12,5))

fig_cnn.add_subplot(121)

plt.plot(history_cnn.history['loss'])
plt.plot(history_cnn.history['val_loss'])
plt.title('loss vs. epochs')
plt.ylabel('Loss')
plt.xlabel('Epoch')
```

```
plt.legend(['Training', 'Validation'], loc='upper right')

fig_cnn.add_subplot(122)

plt.plot(history_cnn.history['accuracy'])
plt.plot(history_cnn.history['val_accuracy'])
plt.title('accuracy vs. epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')

plt.show()
```



[22]:
```
model_cnn.evaluate(test_data_gray,test_labels)
```

```
814/814 [==============================] - 5s 7ms/step - loss: 0.7515 -
accuracy: 0.8085
```

[22]: `[0.7514657974243164, 0.80854332447052]`

1.5 4. Get model predictions

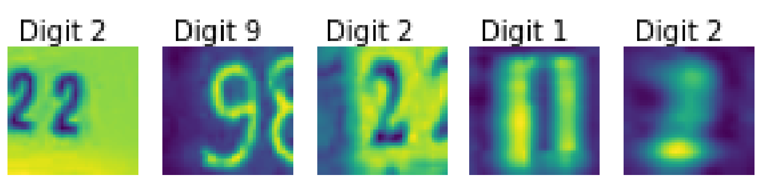
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the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

[23]:
```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
```

[31]:
```
model_mlp=get_new_model(train_data_gray[0].shape)
model.load_weights('checkpoint_best_only/checkpoint')
model_cnn2=Sequential([
                      Conv2D(16,kernel_size=3,activation='tanh',input_shape=(32,32,1)),
                      MaxPooling2D(pool_size=3),
                      Dense(64,activation='tanh'),
                      Flatten(),
                      Dense(10,activation='softmax')
])
model_cnn2.load_weights('checkpoint_best_only_cnn/checkpoint')
```

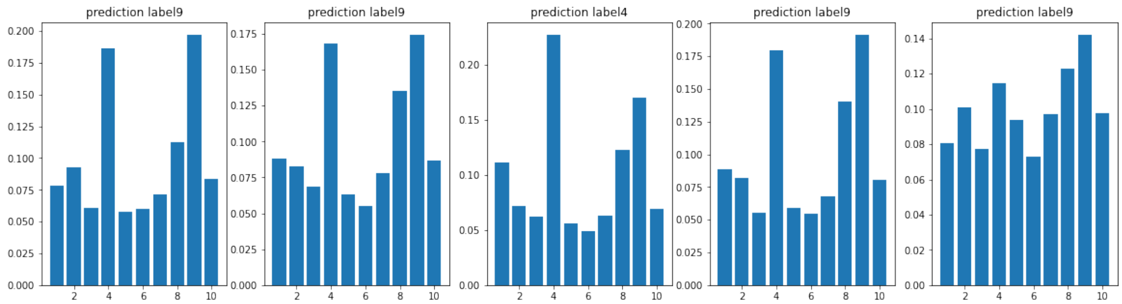
[31]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7f7178fd36d8>
```

[144]:
```
np.random.seed(0)
random_idx_2=np.random.choice(test_data_gray.shape[0],5)
fig,axes=plt.subplots(1,5,figsize=(5,1))
for i in range(5):
    axes[i].set_axis_off()
    axes[i].imshow(np.squeeze(test_data_gray[random_idx_2[i]]))
    axes[i].text(2., -1.5, f'Digit {numpy.argmax(test_labels[random_idx_2[i],])}')
```



[201]:
```
def get_all_predictions(model):
    fig, ax = plt.subplots(1,5,figsize=(20,5))
    for i in range(5):
        prob=model.predict(img_to_array(test_data_gray[random_idx_2[i]])[np.newaxis, ...]).tolist()[0]
        ax[i].bar(np.insert(np.arange(9)+1,0,10),prob)
        ax[i].set_title(f'prediction label{np.argmax(prob)}')
```

[203]:
```
get_all_predictions(model_mlp)
```



[204]:
```
get_all_predictions(model_cnn2)
```