project

February 13, 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 (File -> Download as -> PDF via LaTeX). You should 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.

```
In [4]: import tensorflow as tf
        from scipy.io import loadmat
```

```
In [5]: !ls
```

```
sample_data  test_32x32.mat  train_32x32.mat
```

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.

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.

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

```
In [7]: from google.colab import drive
        drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
In [11]: !ls /content/drive/MyDrive/data
         !cp /content/drive/MyDrive/data/svhn/*.* .
```

```
svhn
```

```
In [12]: # Run this cell to load the dataset

         train = loadmat('train_32x32.mat')
         test = loadmat('test_32x32.mat')
```

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.

1.2.1 1.1 Extract Train/Test Data and Examine

Let us read the dictionary and identify the shape of the test and training data

```
In [13]: train.keys(), test.keys(), train['X'].shape, train['y'].shape, test['X'].shape, test[
```

```
Out[13]: (dict_keys(['__header__', '__version__', '__globals__', 'X', 'y']),
          dict_keys(['__header__', '__version__', '__globals__', 'X', 'y']),
          (32, 32, 3, 73257),
          (73257, 1),
          (32, 32, 3, 26032),
          (26032, 1))
```

Observation: Interesting, shapes of images are reversed. I guess there must be more to ponder over this dataset.

Examine the number of classes in the label data

```
In [14]: import pandas as pd
         pd.Series(train['y'][:,0]).value_counts()
```

```
Out[14]: 1      13861
         2      10585
         3       8497
         4       7458
         5       6882
         6       5727
         7       5595
         8       5045
         10      4948
         9       4659
         dtype: int64
```

Observation: Labels are not $\{0 \cdots 9\}$ but $\{1 \cdots 10\}$

1.2.2 1.2 Preprocess the Data

- Normalize by dividing it by 255.0
- Reformat the labels from $\{0 \cdots 9\}$ to $\{1 \cdots 10\}$ by subtracting 1

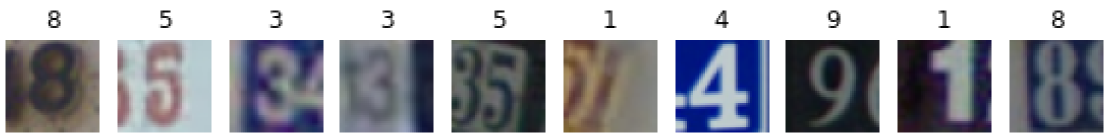
```
In [15]: X_train, y_train, X_test, y_test = train['X']/255., train['y'], test['X']/255., test[
```

```
In [16]: y_train[:,0] = y_train[:,0] - 1
         y_test[:,0] = y_test[:,0] - 1
```

1.2.3 1.3 Display Random Images

```
In [17]: import matplotlib.pyplot as plt
         import numpy as np

         def display_images(img_data, labels, is_grayscale=False, count=10):

             image_idxs = np.random.choice(np.arange(img_data.shape[-1]), count, replace=False

             fig, ax = plt.subplots(1, count, figsize=(count, 1))

             if(is_grayscale == False):

                 for i in np.arange(len(image_idxs)):
                     ax[i].set_axis_off()
                     ax[i].imshow(img_data[:, :, :, image_idxs[i]])
                     ax[i].set_title(labels[image_idxs[i]][0] + 1)

             else:
                 for i in np.arange(len(image_idxs)):
                     ax[i].set_axis_off()
                     ax[i].imshow(img_data[:, :, 0, image_idxs[i]])
                     ax[i].set_title(labels[image_idxs[i]][0] + 1)
```

```
In [18]: display_images(X_train, y_train)
```

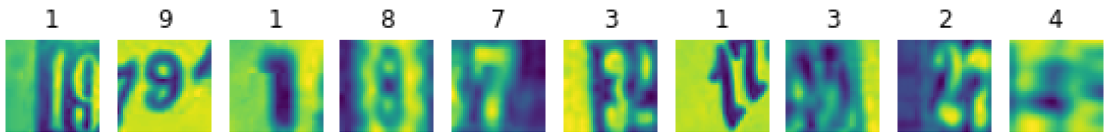


1.2.4 1.4 Convert Train and Test Images to Grayscale

```
In [19]: Xg_train = np.mean(X_train, axis=2, keepdims=True)
         Xg_test = np.mean(X_test, axis=2, keepdims=True)
         Xg_train.shape
```

```
Out[19]: (32, 32, 1, 73257)
```

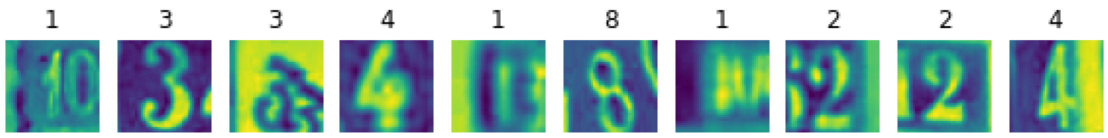
```
In [20]: display_images(Xg_train, y_train, is_grayscale=True)
```



1.2.5 1.5 Reshape the Images

```
In [21]: X_train = []
         for i in range(Xg_train.shape[-1]):
             X_train.append(Xg_train[:, :, :, i])

         X_test = []
         for i in range(Xg_test.shape[-1]):
             X_test.append(Xg_test[:, :, :, i])

         X_train = np.array(X_train)
         X_test = np.array(X_test)
```

```
In [22]: count = 10
         fig, ax = plt.subplots(1, count, figsize=(count, 1))
         image_idxs = np.random.choice(np.arange(X_train.shape[0]), count, replace=False)
         for i in np.arange(len(image_idxs)):
             ax[i].set_axis_off()
             ax[i].imshow(X_train[image_idxs[i], :, :, 0])
             ax[i].set_title(y_train[image_idxs[i]][0] + 1)
```



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.

1.3.1 2.1 Build MLP Classifier

```
In [23]: from tensorflow.keras.layers import Dense, Flatten
         from tensorflow.keras.models import Sequential
```

```
def get_mlp_model(input_shape):
    model = Sequential([
        #Dense(16, activation="relu", ),
        Flatten(input_shape=input_shape),
        Dense(
            512, activation="relu",
            kernel_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.
            bias_initializer=tf.keras.initializers.Constant(value=0.4),
        ),
        Dense(
            256, activation="relu",
            kernel_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.
            bias_initializer=tf.keras.initializers.Constant(value=0.4),
        ),
        Dense(
            128, activation="relu",
            kernel_initializer=tf.keras.initializers.Orthogonal(gain=1.0, seed=None),
            bias_initializer=tf.keras.initializers.Constant(value=0.4),
        ),
        Dense(
            64, activation="relu",
            kernel_initializer=tf.keras.initializers.Orthogonal(gain=1.0, seed=None),
            bias_initializer=tf.keras.initializers.Constant(value=0.4),
        ),
        Dense(10, activation="softmax")
    ])

    return model

model = get_mlp_model(X_train[0].shape)
```

```
In [24]: X_train.shape
```

```
Out[24]: (73257, 32, 32, 1)
```

```
In [25]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 512)               524800
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
```

```
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 10)                650
=================================================================
Total params: 697,930
Trainable params: 697,930
Non-trainable params: 0
_________________________________________________________________
```

1.3.2 2.2 Callback Functions

```
In [26]: def get_checkpoint_every_epoch(path="checkpoints_every_epoch"):
             return tf.keras.callbacks.ModelCheckpoint(
                 filepath=path+'/checkpoint_{epoch:03d}',
                 save_freq="epoch",
                 save_weights_only=True,
                 verbose=1
             )


         def get_checkpoint_best_only(where='mlp'):
             return tf.keras.callbacks.ModelCheckpoint(
                 filepath=f'checkpoints_best_only_{where}/checkpoint',
                 save_freq="epoch",
                 monitor="val_loss",
                 mode="max",
                 save_weights_only=True,
                 save_best_only=True,
                 verbose=1
             )

         def get_early_stopping():
             return tf.keras.callbacks.EarlyStopping(
                 monitor="val_loss",
                 patience=5
             )

         def get_reduce_lr():
             return tf.keras.callbacks.ReduceLROnPlateau(
                 monitor="loss",factor=0.2, verbose=1
             )
```

1.3.3 2.3 Compile and Train

```
In [27]: model.compile(
             optimizer="adam",
             loss="sparse_categorical_crossentropy",
             metrics=["accuracy"]
         )
```

```
In [28]: EPOCHS = 30
         history = model.fit(
             x=X_train,
             y=y_train,
             #validation_data=(X_test, y_test),
             validation_split=0.2,
             epochs=EPOCHS, verbose=1,
             callbacks=[
                 get_checkpoint_every_epoch(),
                 get_checkpoint_best_only(),
                 get_early_stopping(),
                 get_reduce_lr()
             ]
         )
```

```
Epoch 1/30
1832/1832 [==============================] - 6s 2ms/step - loss: 2.1634 - accuracy: 0.2138 - va

Epoch 00001: saving model to checkpoints_every_epoch/checkpoint_001

Epoch 00001: val_loss improved from -inf to 1.57263, saving model to checkpoints_best_only_mlp/
Epoch 2/30
1832/1832 [==============================] - 4s 2ms/step - loss: 1.5114 - accuracy: 0.4726 - va

Epoch 00002: saving model to checkpoints_every_epoch/checkpoint_002

Epoch 00002: val_loss did not improve from 1.57263
Epoch 3/30
1832/1832 [==============================] - 4s 2ms/step - loss: 1.2649 - accuracy: 0.5863 - va

Epoch 00003: saving model to checkpoints_every_epoch/checkpoint_003

Epoch 00003: val_loss did not improve from 1.57263
Epoch 4/30
1832/1832 [==============================] - 4s 2ms/step - loss: 1.0959 - accuracy: 0.6449 - va

Epoch 00004: saving model to checkpoints_every_epoch/checkpoint_004

Epoch 00004: val_loss did not improve from 1.57263
Epoch 5/30
1832/1832 [==============================] - 4s 2ms/step - loss: 1.0153 - accuracy: 0.6760 - va

Epoch 00005: saving model to checkpoints_every_epoch/checkpoint_005

Epoch 00005: val_loss did not improve from 1.57263
Epoch 6/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.9763 - accuracy: 0.6879 - va

Epoch 00006: saving model to checkpoints_every_epoch/checkpoint_006

Epoch 00006: val_loss did not improve from 1.57263
Epoch 7/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.9245 - accuracy: 0.7043 - va

Epoch 00007: saving model to checkpoints_every_epoch/checkpoint_007

Epoch 00007: val_loss did not improve from 1.57263
Epoch 8/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.9035 - accuracy: 0.7123 - va

Epoch 00008: saving model to checkpoints_every_epoch/checkpoint_008

Epoch 00008: val_loss did not improve from 1.57263
Epoch 9/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.8717 - accuracy: 0.7213 - va

Epoch 00009: saving model to checkpoints_every_epoch/checkpoint_009

Epoch 00009: val_loss did not improve from 1.57263
Epoch 10/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.8439 - accuracy: 0.7312 - va

Epoch 00010: saving model to checkpoints_every_epoch/checkpoint_010

Epoch 00010: val_loss did not improve from 1.57263
Epoch 11/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.8180 - accuracy: 0.7377 - va

Epoch 00011: saving model to checkpoints_every_epoch/checkpoint_011

Epoch 00011: val_loss did not improve from 1.57263
Epoch 12/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.8047 - accuracy: 0.7415 - va

Epoch 00012: saving model to checkpoints_every_epoch/checkpoint_012

Epoch 00012: val_loss did not improve from 1.57263
Epoch 13/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.7904 - accuracy: 0.7481 - va

Epoch 00013: saving model to checkpoints_every_epoch/checkpoint_013

Epoch 00013: val_loss did not improve from 1.57263
Epoch 14/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.7719 - accuracy: 0.7531 - va

Epoch 00014: saving model to checkpoints_every_epoch/checkpoint_014

Epoch 00014: val_loss did not improve from 1.57263
Epoch 15/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.7538 - accuracy: 0.7598 - va

Epoch 00015: saving model to checkpoints_every_epoch/checkpoint_015

Epoch 00015: val_loss did not improve from 1.57263
Epoch 16/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.7287 - accuracy: 0.7692 - va

Epoch 00016: saving model to checkpoints_every_epoch/checkpoint_016

Epoch 00016: val_loss did not improve from 1.57263
Epoch 17/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.7208 - accuracy: 0.7714 - va

Epoch 00017: saving model to checkpoints_every_epoch/checkpoint_017

Epoch 00017: val_loss did not improve from 1.57263
Epoch 18/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.7049 - accuracy: 0.7771 - va

Epoch 00018: saving model to checkpoints_every_epoch/checkpoint_018

Epoch 00018: val_loss did not improve from 1.57263
Epoch 19/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6969 - accuracy: 0.7781 - va

Epoch 00019: saving model to checkpoints_every_epoch/checkpoint_019

Epoch 00019: val_loss did not improve from 1.57263
Epoch 20/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6918 - accuracy: 0.7802 - va

Epoch 00020: saving model to checkpoints_every_epoch/checkpoint_020

Epoch 00020: val_loss did not improve from 1.57263
Epoch 21/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6820 - accuracy: 0.7815 - va

Epoch 00021: saving model to checkpoints_every_epoch/checkpoint_021

Epoch 00021: val_loss did not improve from 1.57263
Epoch 22/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6708 - accuracy: 0.7862 - va

Epoch 00022: saving model to checkpoints_every_epoch/checkpoint_022

Epoch 00022: val_loss did not improve from 1.57263
Epoch 23/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6623 - accuracy: 0.7876 - va

Epoch 00023: saving model to checkpoints_every_epoch/checkpoint_023

Epoch 00023: val_loss did not improve from 1.57263
Epoch 24/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6547 - accuracy: 0.7906 - va

Epoch 00024: saving model to checkpoints_every_epoch/checkpoint_024

Epoch 00024: val_loss did not improve from 1.57263
Epoch 25/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6402 - accuracy: 0.7931 - va

Epoch 00025: saving model to checkpoints_every_epoch/checkpoint_025

Epoch 00025: val_loss did not improve from 1.57263
Epoch 26/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6464 - accuracy: 0.7920 - va

Epoch 00026: saving model to checkpoints_every_epoch/checkpoint_026

Epoch 00026: val_loss did not improve from 1.57263
Epoch 27/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6314 - accuracy: 0.7985 - va

Epoch 00027: saving model to checkpoints_every_epoch/checkpoint_027

Epoch 00027: val_loss did not improve from 1.57263
Epoch 28/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6363 - accuracy: 0.7935 - va

Epoch 00028: saving model to checkpoints_every_epoch/checkpoint_028

Epoch 00028: val_loss did not improve from 1.57263
Epoch 29/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6151 - accuracy: 0.8020 - va

Epoch 00029: saving model to checkpoints_every_epoch/checkpoint_029

Epoch 00029: val_loss did not improve from 1.57263
Epoch 30/30
1832/1832 [==============================] - 4s 2ms/step - loss: 0.6193 - accuracy: 0.7985 - va

Epoch 00030: saving model to checkpoints_every_epoch/checkpoint_030

Epoch 00030: val_loss did not improve from 1.57263
```

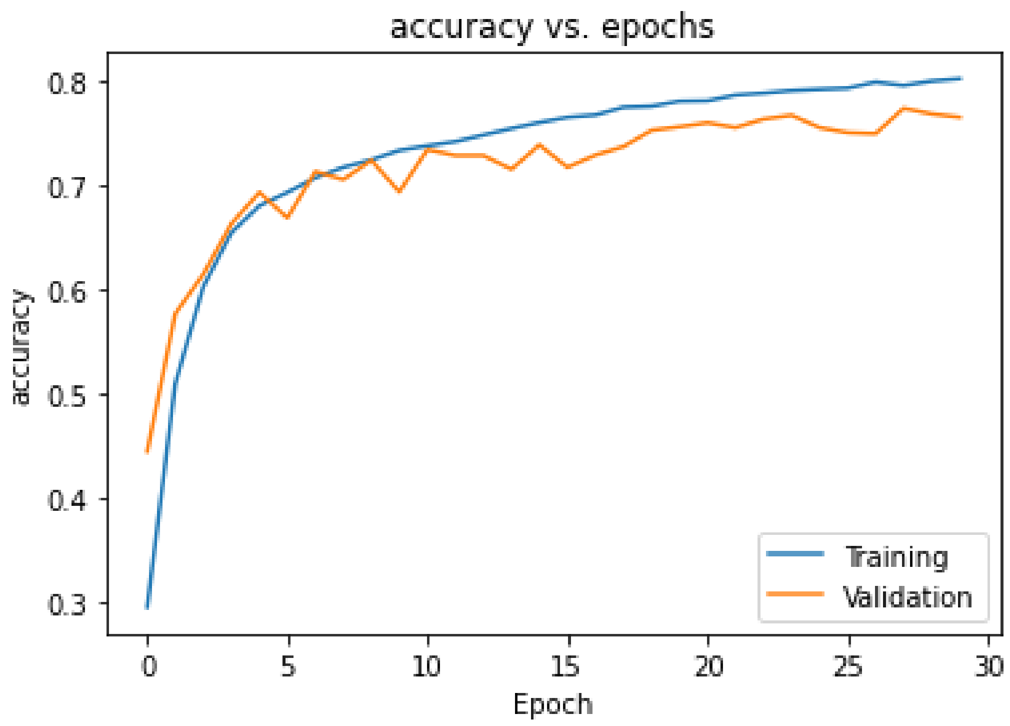
1.3.4 2.4 Plot History

```
In [29]: def plot_history(df, what):
             try:
                 plt.plot(df[[what]])
                 plt.plot(df[[f'val_{what}']])
             except KeyError:
                 plt.plot(df[what])
                 plt.plot(df[f'val_{what}'])
             plt.title(f'{what} vs. epochs')
             plt.ylabel(what)
             plt.xlabel('Epoch')
             plt.legend(['Training', 'Validation'], loc='lower right')
             plt.show()
```

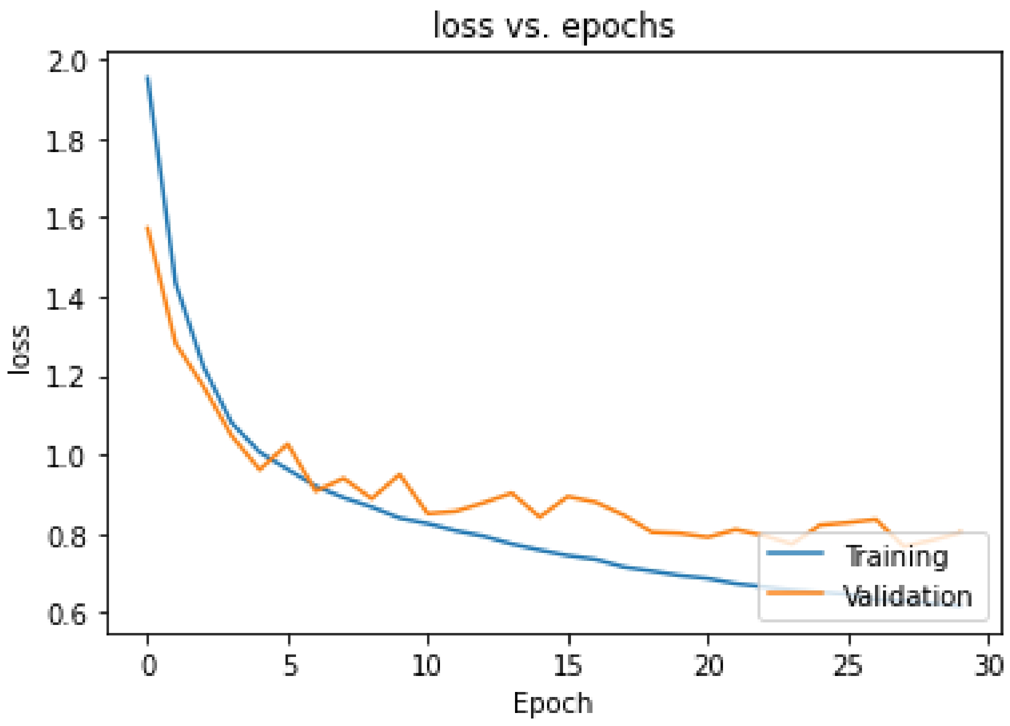
```
In [30]: import pandas as pd
         history_df = pd.DataFrame(history.history)
         history_df.head()
```

```
Out[30]:        loss  accuracy  val_loss  val_accuracy     lr
         0  1.952345  0.296067  1.572634      0.444922  0.001
         1  1.434126  0.509479  1.281786      0.576508  0.001
         2  1.222293  0.602184  1.172108      0.614251  0.001
         3  1.080987  0.653750  1.047988      0.662367  0.001
         4  1.006363  0.679720  0.962650      0.692533  0.001
```

```
In [31]: plot_history(history_df, what="accuracy")
```



```
In [32]: plot_history(history_df, what="loss")
```



1.3.5 2.5 Evaluate

```
In [33]: test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
         print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 0.915
Test accuracy: 73.90%
```

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.

```
In [47]: from tensorflow.keras.layers import BatchNormalization, Dropout
         def get_cnn_classifier(input_shape):
             model = Sequential([
                 tf.keras.layers.Conv2D(
                     512, kernel_size=(3, 3), activation='relu', padding="SAME", input_shape=i
                     #kernel_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0
                     #bias_initializer=tf.keras.initializers.Constant(value=0.4),
                 ),
                 BatchNormalization(),
                 Dropout(0.3),
                 #tf.keras.layers.MaxPooling2D(pool_size=(8, 8)),
                 tf.keras.layers.Conv2D(
                     512, kernel_size=(3, 3), activation="relu", padding="SAME",
                     #kernel_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0
                     #bias_initializer=tf.keras.initializers.Constant(value=0.4),
                 ),
                 #tf.keras.layers.MaxPooling2D(pool_size=(4, 4)),
                 BatchNormalization(),
                 Dropout(0.3),
                 tf.keras.layers.Conv2D(
                     256, kernel_size=(3, 3), activation="relu", padding="SAME",
                     #kernel_initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0
                     #bias_initializer=tf.keras.initializers.Constant(value=0.4),
                 ),
                 tf.keras.layers.MaxPooling2D(pool_size=(8, 8)),
                 Flatten(),
                 Dense(128, kernel_initializer='he_uniform', activation='relu'),
                 Dense(128, kernel_initializer='he_uniform', activation='relu'),
                 Dense(10, activation="softmax")
             ])
             return model
         cnn_model = get_cnn_classifier(X_train[0].shape)
```

```
In [48]: cnn_model.compile(
             optimizer="adam",
             loss="sparse_categorical_crossentropy",
             metrics=["accuracy"]
         )
```

```
In [49]: cnn_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 32, 32, 512)       5120
_________________________________________________________________
batch_normalization_4 (Batch (None, 32, 32, 512)       2048
_________________________________________________________________
dropout_4 (Dropout)          (None, 32, 32, 512)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 512)       2359808
_________________________________________________________________
batch_normalization_5 (Batch (None, 32, 32, 512)       2048
_________________________________________________________________
dropout_5 (Dropout)          (None, 32, 32, 512)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 256)       1179904
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_16 (Dense)             (None, 128)               524416
_________________________________________________________________
dense_17 (Dense)             (None, 128)               16512
_________________________________________________________________
```

```
dense_18 (Dense)             (None, 10)               1290
=================================================================
Total params: 4,091,146
Trainable params: 4,089,098
Non-trainable params: 2,048
_________________________________________________________________
```

```
In [50]: history = cnn_model.fit(
             x=X_train,
             y=y_train,
             #validation_data=(X_test, y_test),
             validation_split=0.2,
             epochs=EPOCHS, verbose=1,
             callbacks=[
                 #get_checkpoint_every_epoch(where="cnn"),
                 get_checkpoint_best_only(where="cnn"),
                 get_early_stopping(),
                 get_reduce_lr()
             ]
         )
```

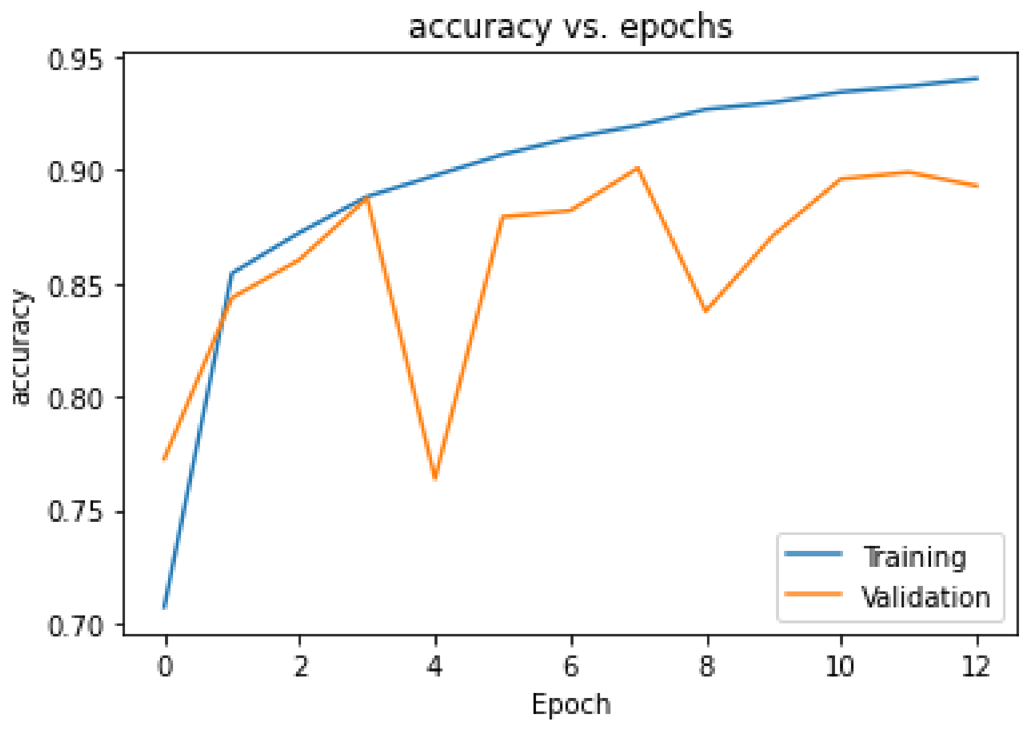
```
Epoch 1/30
1832/1832 [==============================] - 49s 26ms/step - loss: 1.9715 - accuracy: 0.5312 -

Epoch 00001: val_loss improved from -inf to 0.77842, saving model to checkpoints_best_only_cnn/
Epoch 2/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.4855 - accuracy: 0.8528 -

Epoch 00002: val_loss did not improve from 0.77842
Epoch 3/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.4189 - accuracy: 0.8729 -

Epoch 00003: val_loss did not improve from 0.77842
Epoch 4/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.3741 - accuracy: 0.8876 -

Epoch 00004: val_loss did not improve from 0.77842
Epoch 5/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.3279 - accuracy: 0.8996 -

Epoch 00005: val_loss did not improve from 0.77842
Epoch 6/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.3053 - accuracy: 0.9072 -

Epoch 00006: val_loss did not improve from 0.77842
Epoch 7/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.2789 - accuracy: 0.9157 -
```

```
Epoch 00007: val_loss did not improve from 0.77842
Epoch 8/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.2586 - accuracy: 0.9205 -

Epoch 00008: val_loss did not improve from 0.77842
Epoch 9/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.2315 - accuracy: 0.9302 -

Epoch 00009: val_loss did not improve from 0.77842
Epoch 10/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.2236 - accuracy: 0.9326 -

Epoch 00010: val_loss did not improve from 0.77842
Epoch 11/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.2081 - accuracy: 0.9370 -

Epoch 00011: val_loss did not improve from 0.77842
Epoch 12/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.1982 - accuracy: 0.9397 -

Epoch 00012: val_loss did not improve from 0.77842
Epoch 13/30
1832/1832 [==============================] - 47s 26ms/step - loss: 0.1859 - accuracy: 0.9427 -

Epoch 00013: val_loss did not improve from 0.77842
```

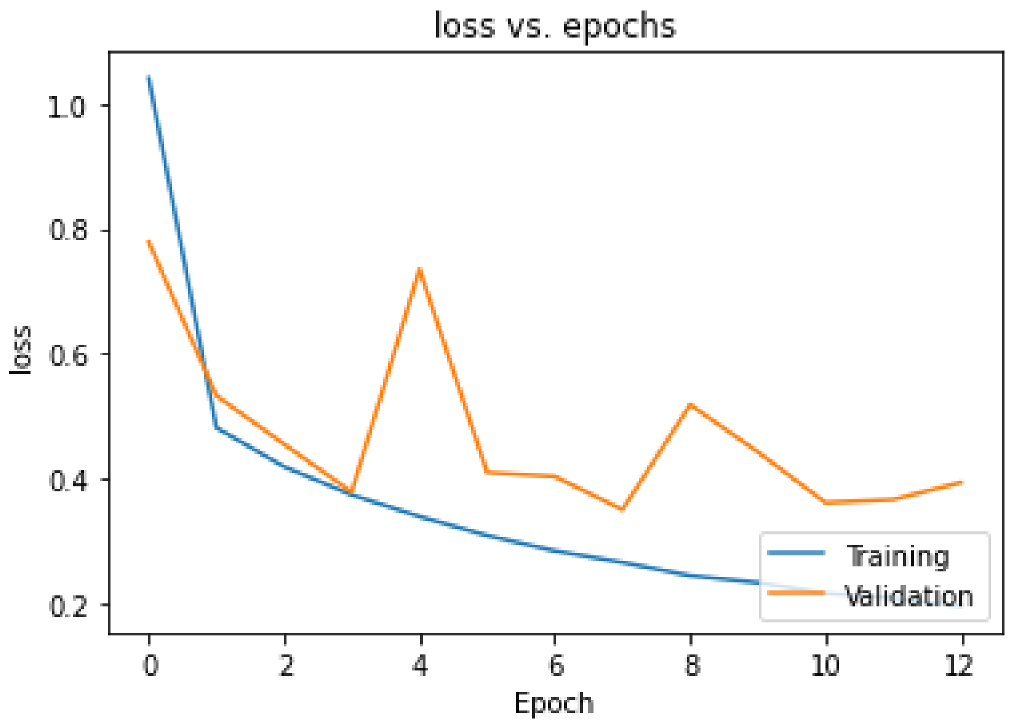
```
In [51]: cnn_history_df = pd.DataFrame(history.history)
         cnn_history_df.head()
```

```
Out[51]:        loss  accuracy  val_loss  val_accuracy     lr
         0  1.041135  0.707158  0.778419      0.772523  0.001
         1  0.481290  0.854415  0.532738      0.843571  0.001
         2  0.418784  0.872315  0.455071      0.860429  0.001
         3  0.373699  0.888252  0.378260      0.887524  0.001
         4  0.338922  0.897551  0.735163      0.763786  0.001
```

```
In [52]: plot_history(cnn_history_df, what="accuracy")
```



```
In [53]: plot_history(cnn_history_df, what="loss")
```



```
In [54]: test_loss, test_acc = cnn_model.evaluate(X_test, y_test, verbose=0)
         print("Test loss: {:.3f}\nTest accuracy: {:.2f}%".format(test_loss, 100 * test_acc))
```

```
Test loss: 0.419
Test accuracy: 88.70%
```

```
In [55]: !ls checkpoints_best_only_mlp -lh
```

```
total 8.0M
-rw-r--r-- 1 root root   77 Feb 13 17:05 checkpoint
-rw-r--r-- 1 root root 8.0M Feb 13 17:05 checkpoint.data-00000-of-00001
-rw-r--r-- 1 root root 2.4K Feb 13 17:05 checkpoint.index
```

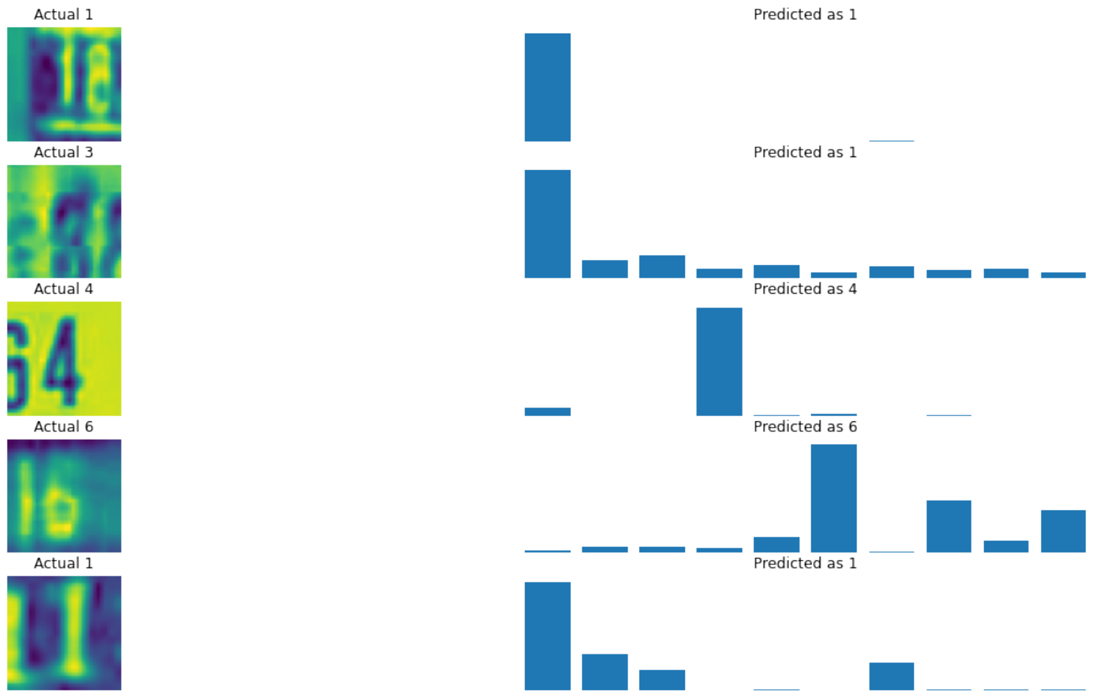
1.5 4. Get model predictions

- 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.

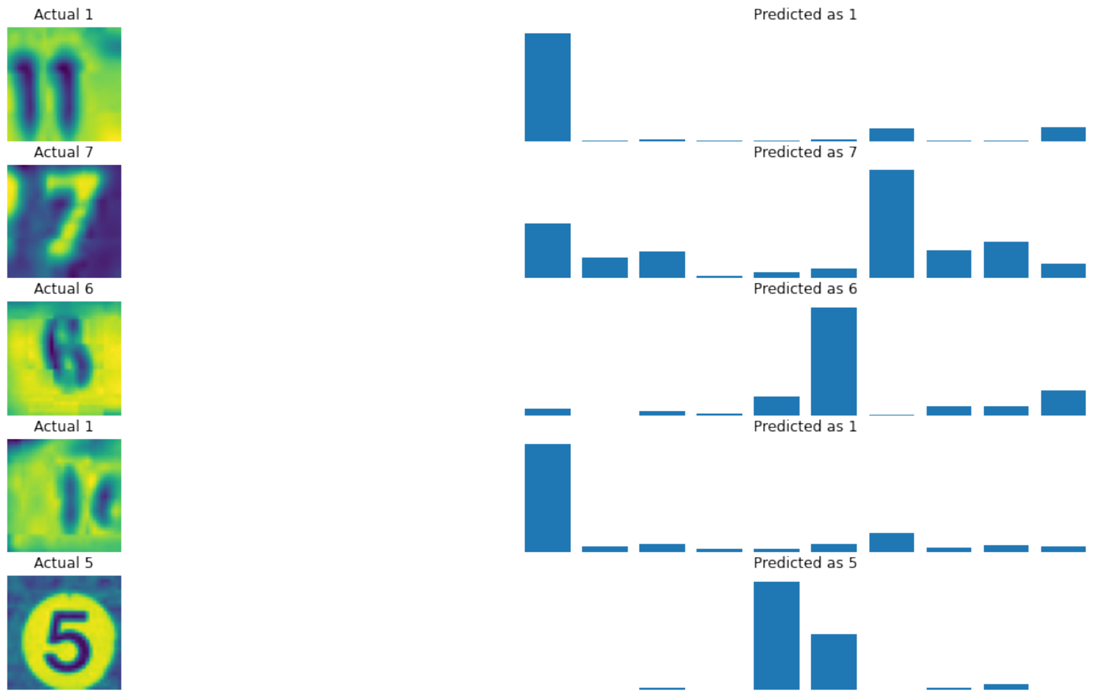
```
In [56]: def get_model_best_epoch(model, where=""):
             model.load_weights(
                 tf.train.latest_checkpoint(f'checkpoints_best_only_{where}')
             )

             return model
         mlp_model = get_model_best_epoch(get_mlp_model(X_train[0].shape), where="mlp")
         cnn_model = get_model_best_epoch(get_cnn_classifier(X_train[0].shape), where="cnn")
```

```
In [63]: def plot_predictions(model, data, labels, count=5):
             fig, ax = plt.subplots(5, 2, figsize=(count*4, 10))
             image_idxs = np.random.choice(np.arange(data.shape[0]), count, replace=False)
             for i in np.arange(len(image_idxs)):
                 ax[i, 0].set_axis_off()
                 ax[i, 1].set_axis_off()
                 ax[i, 0].imshow(data[image_idxs[i], :, :, 0])
                 ax[i, 0].set_title(f"Actual {labels[image_idxs[i]][0] + 1}")
                 flat_img = np.reshape(data[image_idxs[i], :, :, :], [1, 32, 32, 1])
                 predictions = model.predict(flat_img)
                 # print(predictions)
                 ax[i, 1].bar(np.arange(0, 10), predictions[0])
                 ax[i, 1].set_title(f"Predicted as {np.argmax(predictions) + 1}")
```

```
In [64]: plot_predictions(mlp_model, X_test, y_test)
```



```
In [65]: plot_predictions(cnn_model, X_test, y_test)
```



```
In [ ]:
```