Capstone Project

January 18, 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 [6]: 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.

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.

```
In [7]: # Run this cell to load the dataset

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

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

1.2 1. Inspect and preprocess the dataset

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

```
In [8]: import matplotlib.pyplot as plt
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
```

```
        from tensorflow.keras.layers import Dropout, BatchNormalization
        from tensorflow.keras import regularizers
        import random
```

```
In [9]: train_images, train_labels = train['X'], train['y']
```

```
In [10]: test_images, test_labels = test['X'], test['y']
```

```
In [11]: train_labels[0]
```

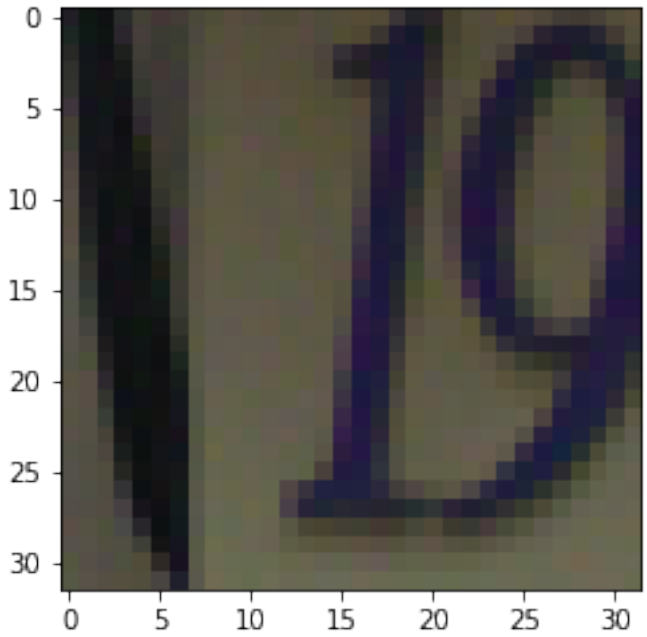
```
Out[11]: array([1], dtype=uint8)
```

```
In [12]: train_images.shape,train_labels.shape
```

```
Out[12]: ((32, 32, 3, 73257), (73257, 1))
```

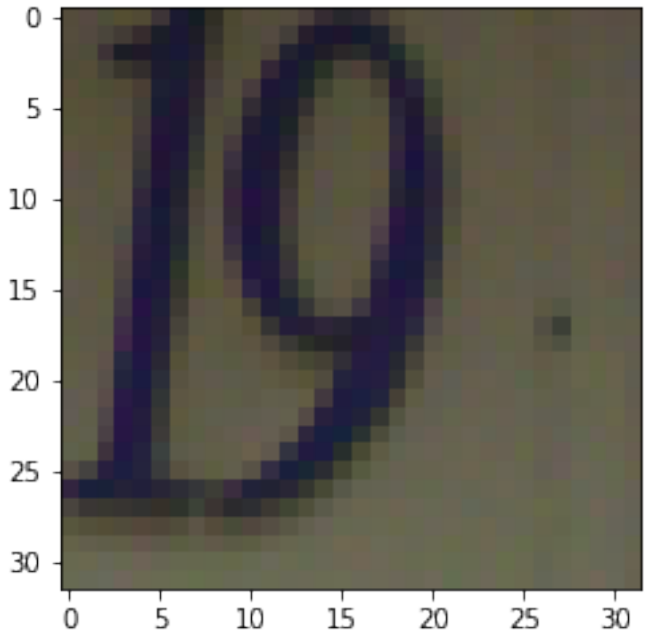
```
In [13]: # Display one of the images

         i=0
         img=train_images[:,:,:,i]
         plt.imshow(img)
         print(f'label: {train_labels[i]}')
```

```
label: [1]
```



In [14]:
```
# Display one of the images

i=1
img=train_images[:,:,:,i]
plt.imshow(img)
print(f'label: {train_labels[i]}')
```

```
label: [9]
```



In [15]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
import numpy as np

fig = plt.figure(figsize=(4., 4.))
grid = ImageGrid(fig, 111,  # similar to subplot(111)
                 nrows_ncols=(2, 5),  # creates 2x2 grid of axes
                 axes_pad=0.1,  # pad between axes in inch.
                 )

for ax, im_num in zip(grid, range(10)):
    # Iterating over the grid returns the Axes.
    ax.imshow(train_images[:,:,:,im_num])

plt.show()
print(f'label: \n {train_labels[0:10]}')
```

```
label:
 [[1]
 [9]
 [2]
 [3]
 [2]
 [5]
 [9]
 [3]
 [3]
 [1]]
```

```
In [16]: X_train = np.moveaxis(train_images, -1, 0)
         X_test = np.moveaxis(test_images, -1 , 0)

In [17]: X_train_gs = np.mean(X_train, 3).reshape(73257, 32, 32, 1)/255
         X_test_gs = np.mean(X_test,3).reshape(26032, 32,32 ,1)/255

In [18]: import matplotlib.pyplot as plt
         from mpl_toolkits.axes_grid1 import ImageGrid
         import numpy as np

         fig = plt.figure(figsize=(4., 4.))
         grid = ImageGrid(fig, 111,  # similar to subplot(111)
                          nrows_ncols=(2, 5),  # creates 2x2 grid of axes
                          axes_pad=0.1,  # pad between axes in inch.
                          )

         for ax, im_num in zip(grid, range(10)):
             # Iterating over the grid returns the Axes.
             ax.imshow(X_train_gs[im_num,:,:,0],cmap=plt.get_cmap('gray'))

         plt.show()
         print(f'label: \n {train_labels[0:10]}')
```

```
label:
 [[1]
 [9]
 [2]
 [3]
 [2]
 [5]
 [9]
 [3]
 [3]
 [1]]
```

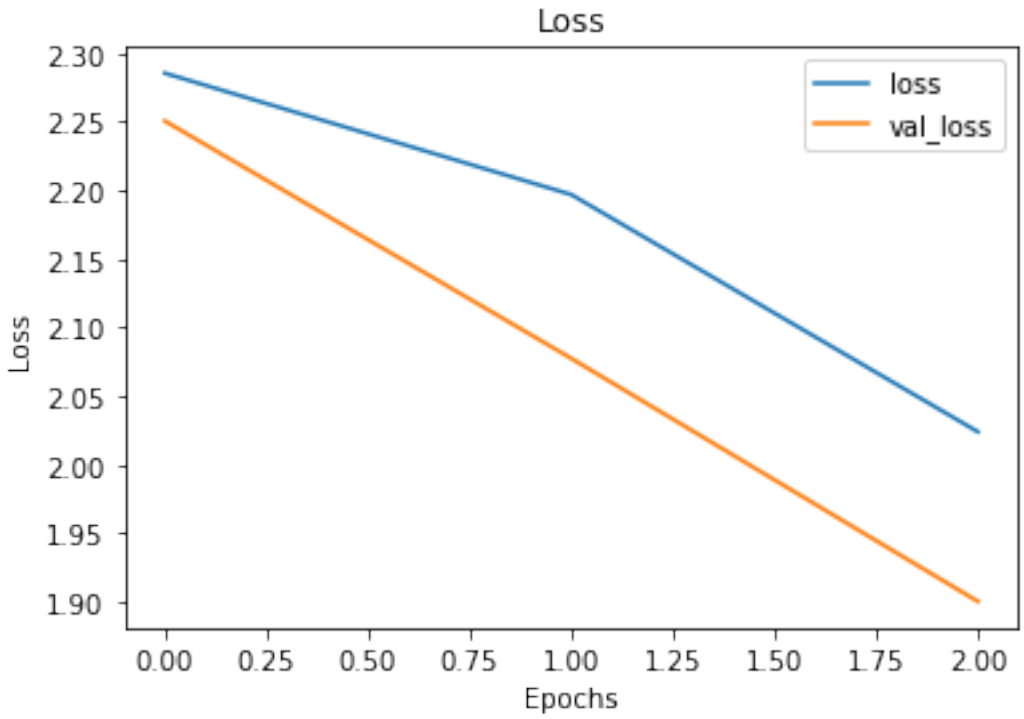
```
In [19]: from sklearn.preprocessing import OneHotEncoder

         enc = OneHotEncoder().fit(train_labels)
         y_train_oh = enc.transform(train_labels).toarray()
         y_test_oh = enc.transform(test_labels).toarray()
```

1.3 2. MLP neural network classifier

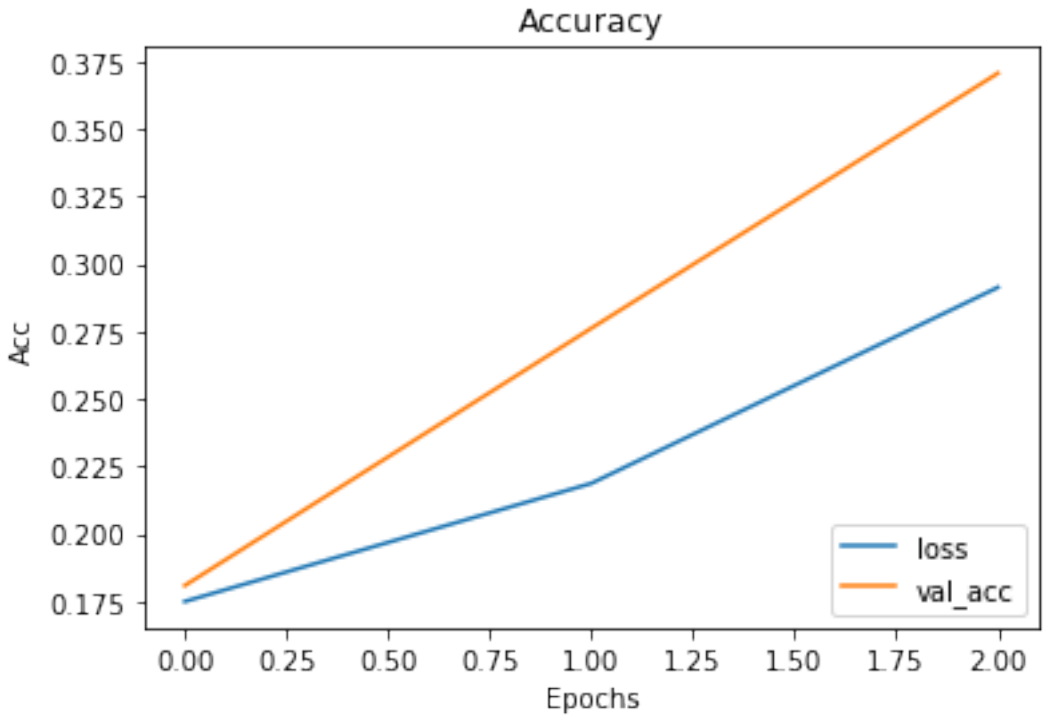
- Build an MLP classifier model using the Sequential API. Your model should use only Flatten and Dense layers, with the final layer having a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different MLP architectures. *Hint: to achieve a reasonable 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.

```
In [2]: def mlp_classifier(input_shape, wd, rate):
            """
            input_shape: input shape
            wd: weightdecay
            rate : regulizer rate
            """
            model = Sequential([
                Dense(round(32*32*0.1),activation='relu',kernel_regularizer=regularizers.l2(wd
                Flatten(),
                Dropout(rate),
                Dense(30,activation='relu',kernel_regularizer=regularizers.l2(wd)),
                BatchNormalization(),
                Dropout(rate),
                Dense(10,activation='softmax')
            ])
            return model

In [21]: model = mlp_classifier(X_train_gs[0].shape,1e-4,0.3)

In [22]: model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32, 32, 102)       204
_________________________________________________________________
flatten (Flatten)            (None, 104448)            0
_________________________________________________________________
dropout (Dropout)            (None, 104448)            0
_________________________________________________________________
dense_1 (Dense)              (None, 30)                3133470
_________________________________________________________________
batch_normalization (BatchNo (None, 30)                120
_________________________________________________________________
dropout_1 (Dropout)          (None, 30)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                310
=================================================================
Total params: 3,134,104
Trainable params: 3,134,044
Non-trainable params: 60
_________________________________________________________________


In [23]: model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

In [24]: # history = model.fit(x=X_train_gs, y=y_train_oh, epochs=3, validation_split=0.30,bat
```

```
In [25]: from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

         checkpoint = ModelCheckpoint(filepath = 'firstmodel/sequential', save_best_only=True,
         earlystop = EarlyStopping(patience=5, monitor='loss')

In [27]: history = model.fit(x=X_train_gs, y=y_train_oh, epochs=3, validation_split=0.15,batch

Train on 47617 samples, validate on 25640 samples
Epoch 1/3

Epoch 00001: val_loss improved from inf to 2.25071, saving model to firstmodel/sequential
47617/47617 - 698s - loss: 2.2855 - accuracy: 0.1748 - val_loss: 2.2507 - val_accuracy: 0.1807
Epoch 2/3

Epoch 00002: val_loss improved from 2.25071 to 2.07752, saving model to firstmodel/sequential
47617/47617 - 673s - loss: 2.1970 - accuracy: 0.2186 - val_loss: 2.0775 - val_accuracy: 0.2762
Epoch 3/3

Epoch 00003: val_loss improved from 2.07752 to 1.90032, saving model to firstmodel/sequential
47617/47617 - 671s - loss: 2.0240 - accuracy: 0.2913 - val_loss: 1.9003 - val_accuracy: 0.3707


In [36]: import pickle
         import os
         if not os.path.exists('model_history'):
             os.makedirs('model_history/')
         with open('model_history/seq_model.pkl','wb') as f:
             pickle.dump(history.history,f)

In [38]: import pandas as pd
         pd.DataFrame(history.history)

Out[38]:        loss  accuracy  val_loss  val_accuracy
         0  2.285521  0.174833  2.250710      0.180733
         1  2.196981  0.218640  2.077520      0.276170
         2  2.024030  0.291325  1.900321      0.370749

In [39]: plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.xlabel('Epochs')
         plt.ylabel('Loss')
         plt.legend(['loss','val_loss'], loc='upper right')
         plt.title("Loss")

Out[39]: Text(0.5, 1.0, 'Loss')
```




```
In [41]: plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.xlabel('Epochs')
         plt.ylabel('Acc')
         plt.legend(['loss','val_acc'], loc='lower right')
         plt.title("Accuracy")

Out[41]: Text(0.5, 1.0, 'Accuracy')
```




```
In [62]: !ls firstmodel/

checkpoint  sequential.data-00000-of-00001  sequential.index
```

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 [70]: def get_conv_model(input_shape):
             model_conv = Sequential([
```

```
            Conv2D(filters= 16, kernel_size= 3, activation='relu', input_shape=input_shape
            MaxPooling2D(pool_size= (2,2)),
            Conv2D(filters= 32, kernel_size = 3, padding='valid', activation='relu'),
            MaxPooling2D(pool_size = (3,3)),
            BatchNormalization(),
            Conv2D(filters= 32, kernel_size = 3, padding='valid', activation='relu'),
            Dropout(0.5),
            Flatten(),
            Dense(64, activation='relu'),
            Dense(32, activation='relu'),
            tf.keras.layers.Dropout(0.3),
            Dense(10, activation='softmax')
        ])
        return model_conv
```

```
In [71]: model_conv = get_conv_model(input_shape=X_train_gs[0].shape)
         model_conv.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 30, 30, 16)        160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 16)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 13, 13, 32)        4640
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 32)          128
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 2, 2, 32)          9248
_________________________________________________________________
dropout_6 (Dropout)          (None, 2, 2, 32)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_10 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_7 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_11 (Dense)             (None, 10)                330
=================================================================
Total params: 24,842
Trainable params: 24,778
```

```
Non-trainable params: 64
_________________________________________________________________
```

```
In [72]: checkpoint1 = ModelCheckpoint(filepath='secondmodel/cnnweights', save_best_only=True,
         earlystop1 = EarlyStopping(monitor='loss',patience=7, verbose=1)
```

```
In [73]: # import warnings
         # warnings.filterwarnings("ignore")
```

```
In [ ]: model_conv.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy

        history_conv = model_conv.fit(X_train_gs, y_train_oh, callbacks=[checkpoint1, earlysto
```

```
In [59]: pd.DataFrame(history_conv.history)
```

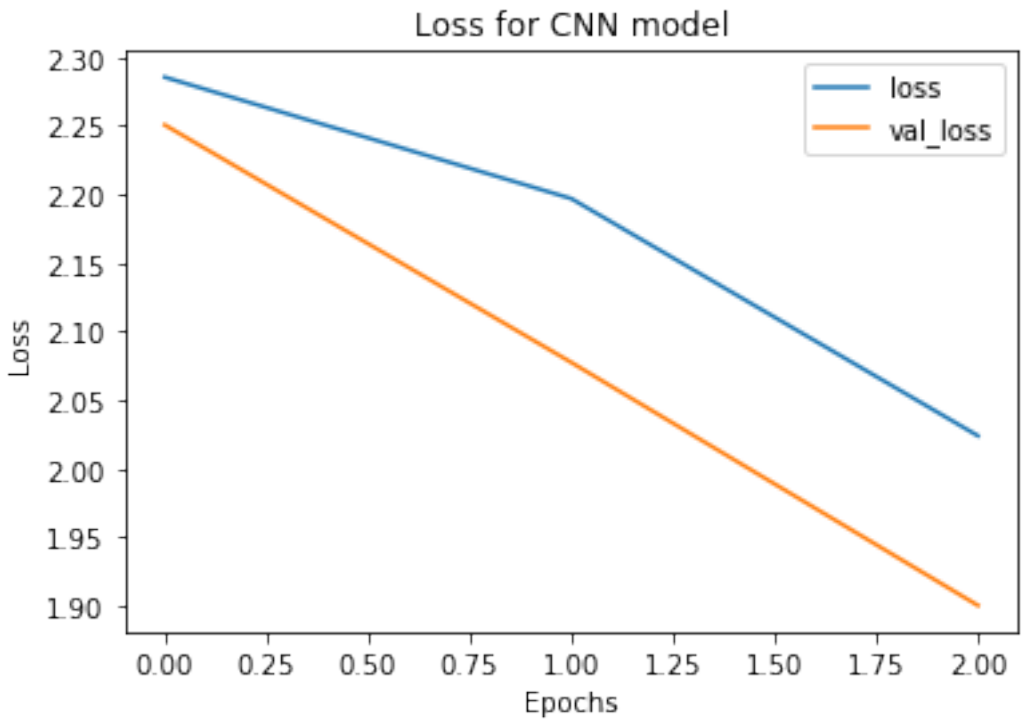
Out[59]:

	loss	accuracy	val_loss	val_accuracy
0	0.672345	0.798477	0.575622	0.825638
1	0.654156	0.805124	0.698639	0.790220
2	0.649545	0.807472	0.505571	0.849570

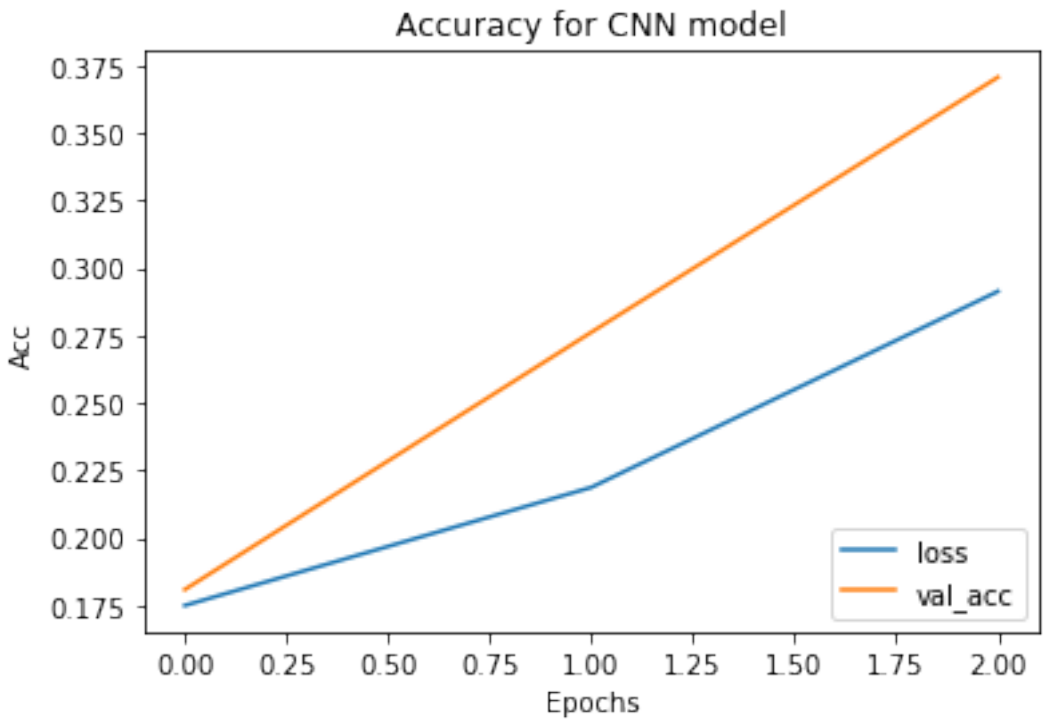
```
In [54]: import pickle
         import os
         if not os.path.exists('model_history'):
             os.makedirs('model_history')
         with open('model_history/cnn_model.pkl','wb') as f:
             pickle.dump(history_conv.history,f)
```

```
In [55]: plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])
         plt.xlabel('Epochs')
         plt.ylabel('Loss')
         plt.legend(['loss','val_loss'], loc='upper right')
         plt.title("Loss for CNN model")
```

```
Out[55]: Text(0.5, 1.0, 'Loss for CNN model')
```




```
In [56]: plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.xlabel('Epochs')
         plt.ylabel('Acc')
         plt.legend(['loss','val_acc'], loc='lower right')
         plt.title("Accuracy for CNN model")

Out[56]: Text(0.5, 1.0, 'Accuracy for CNN model')
```



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 [8]: model_seq = mlp_classifier(X_train_gs[0].shape,1e-4,0.3)
        model_seq.load_weights('firstmodel')

In [ ]: num_test_images = X_test.shape[0]

        random_inx = np.random.choice(num_test_images, 5)
        random_test_images = X_test_gs[random_inx, ...]
        random_test_labels = y_test[random_inx]

        predictions = model_seq.predict(random_test_images)

        fig, axes = plt.subplots(5, 2, figsize=(16, 12))
        fig.subplots_adjust(hspace=0.4, wspace=-0.2)

        for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images, ra
            axes[i, 0].imshow(np.squeeze(image))
```

```
            axes[i, 0].get_xaxis().set_visible(False)
            axes[i, 0].get_yaxis().set_visible(False)
            axes[i, 0].text(10., -1.5, f'Digit {label}')
            axes[i, 1].bar(np.arange(1,11), prediction)
            axes[i, 1].set_xticks(np.arange(1,11))
            axes[i, 1].set_title("Categorical distribution for Sequential Model prediction")

        plt.show()
```

In [9]:
```
model_cnn = get_conv_model(X_train_gs[0].shape)
model_cnn.load_weights('secondmodel')
```

In [ ]:
```
num_test_images = X_test.shape[0]

random_inx = np.random.choice(num_test_images, 5)
random_test_images = X_test_gs[random_inx, ...]
random_test_labels = y_test[random_inx]

predictions = model_cnn.predict(random_test_images)

fig, axes = plt.subplots(5, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)

for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images, ra
    axes[i, 0].imshow(np.squeeze(image))
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Digit {label}')
    axes[i, 1].bar(np.arange(1,11), prediction)
    axes[i, 1].set_xticks(np.arange(1,11))
    axes[i, 1].set_title("Categorical distribution for CNN Model prediction")

plt.show()
```

In [ ]:

In [ ]:

In [ ]: