Capstone Project

October 27, 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 (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 [1]: import tensorflow as tf
        from scipy.io import loadmat
```

```
In [2]: from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Conv2D, Flatten, BatchNormalization, MaxPool2D, Den
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        %matplotlib inline
```


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 [3]: # 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 [4]: X_train = train['X']
        X_test = test['X']
        y_train = train['y']
```

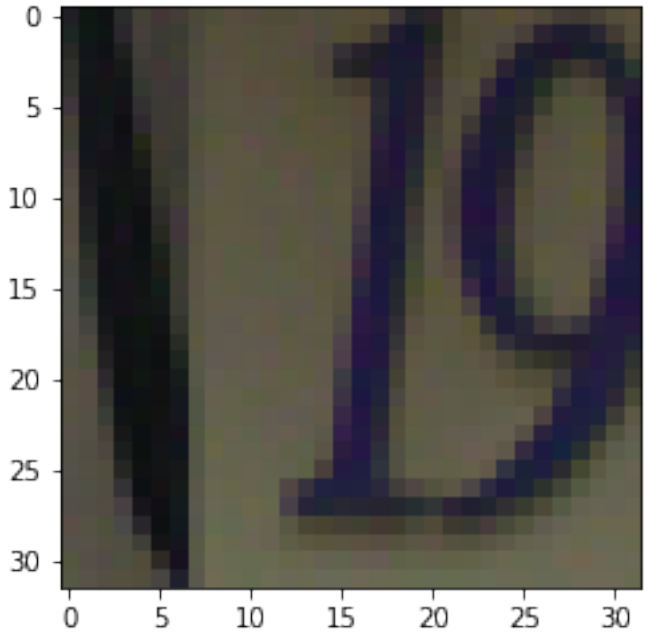
```
        y_test = test['y']

        X_train.shape, X_test.shape
```

```
Out[4]: ((32, 32, 3, 73257), (32, 32, 3, 26032))
```

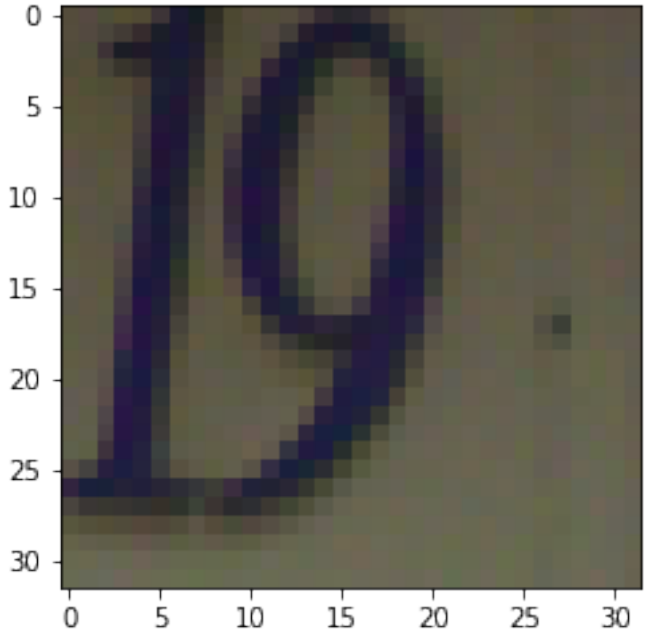
```
In [5]: X_train = np.moveaxis(X_train, -1, 0)
        X_test = np.moveaxis(X_test, -1 , 0)
        X_train.shape, X_test.shape
```

```
Out[5]: ((73257, 32, 32, 3), (26032, 32, 32, 3))
```

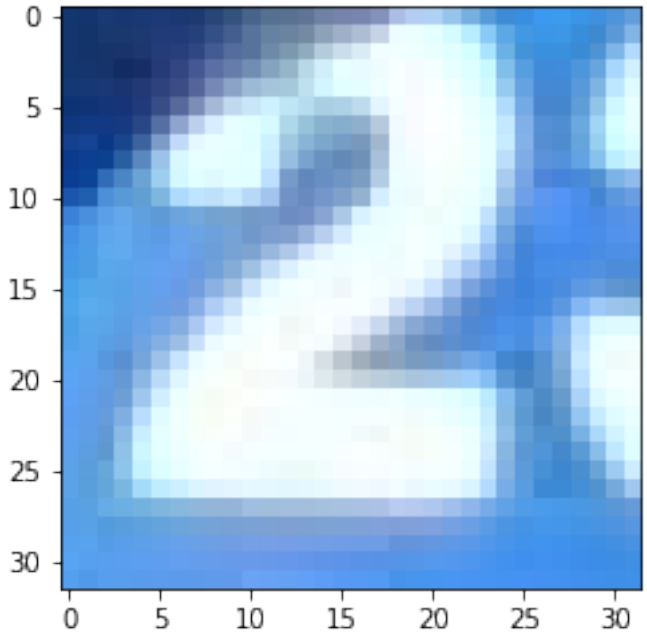
```
In [6]: for i in range(10):
            plt.imshow(X_train[i, :, :, :,])
            plt.show()
            print(y_train[i])
```



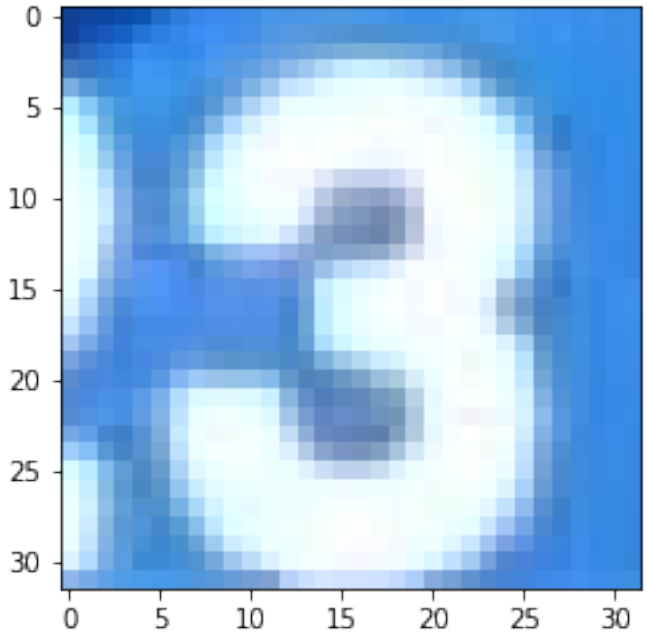

```
[1]
```



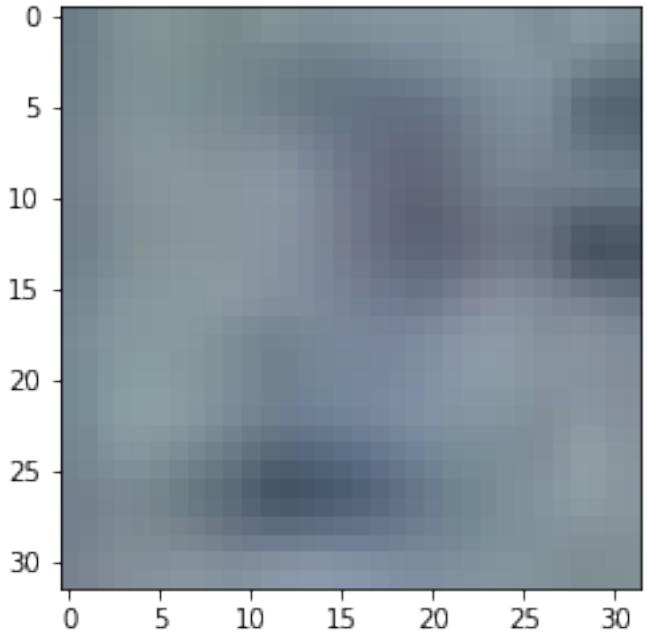
[9]



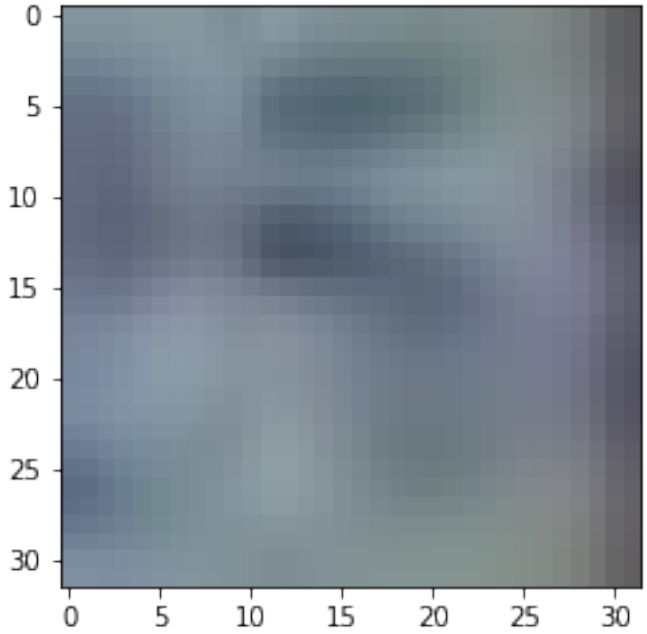
[2]



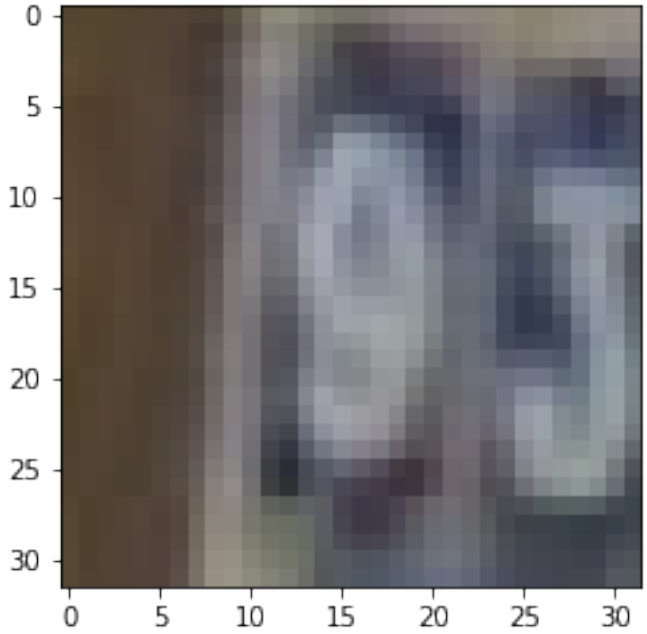
[3]



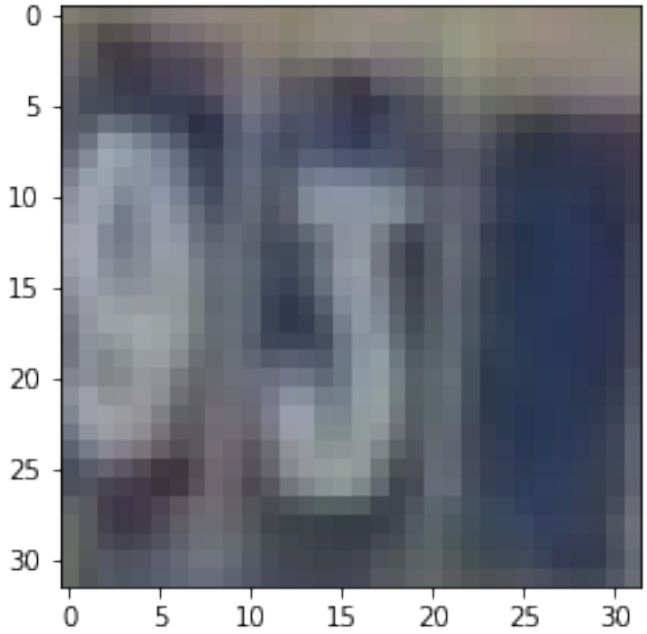
[2]



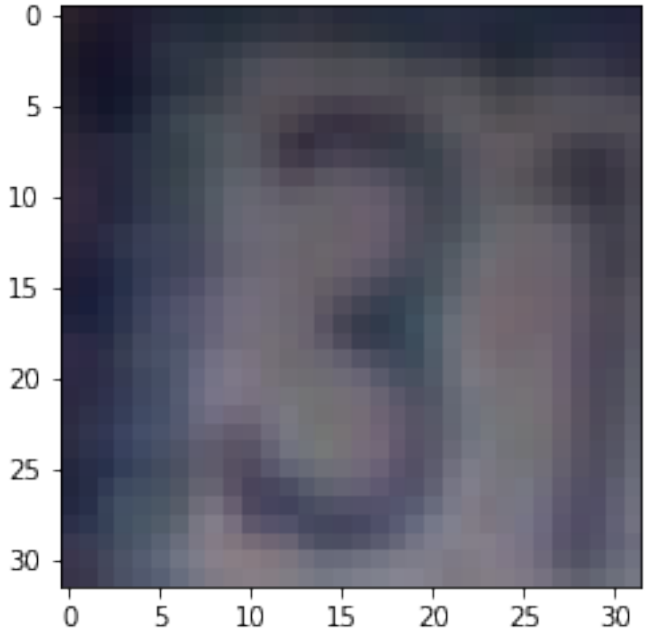
[5]



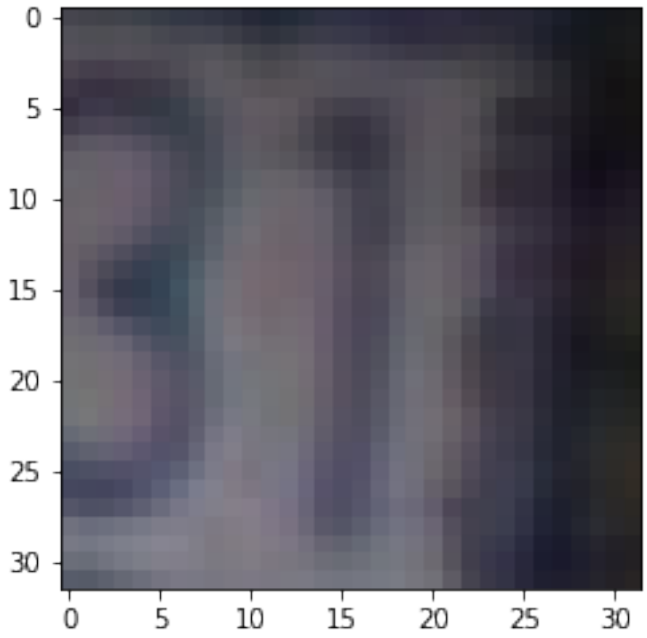
[9]



[3]



[3]



[1]

```
In [7]: 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 [8]: from sklearn.preprocessing import OneHotEncoder

        enc = OneHotEncoder().fit(y_train)
        y_train_oh = enc.transform(y_train).toarray()
        y_test_oh = enc.transform(y_test).toarray()

In [ ]:
```

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

In [ ]:
```
checkpoint = ModelCheckpoint(filepath = 'MLP', save_best_only=True, save_weights_only=
earlystop = EarlyStopping(patience=5, monitor='loss')
```

In [ ]:
```
model = Sequential([
    Flatten(input_shape=X_train[0].shape),
    Dense(128*4, activation='relu'),
    BatchNormalization(),
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dense(32, activation='relu'),
    BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    Dense(16, activation='relu'),
    BatchNormalization(),
    Dense(10, activation='softmax')
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1573376
_________________________________________________________________
batch_normalization (BatchNo (None, 512)               2048
_________________________________________________________________
dense_1 (Dense)              (None, 64)                32832
_________________________________________________________________
batch_normalization_1 (Batch (None, 64)                256
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
batch_normalization_2 (Batch (None, 32)                128
_________________________________________________________________
```

```
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 16)                528
_________________________________________________________________
batch_normalization_3 (Batch (None, 16)                64
_________________________________________________________________
dense_4 (Dense)              (None, 10)                170
=================================================================
Total params: 1,611,482
Trainable params: 1,610,234
Non-trainable params: 1,248
_________________________________________________________________
```

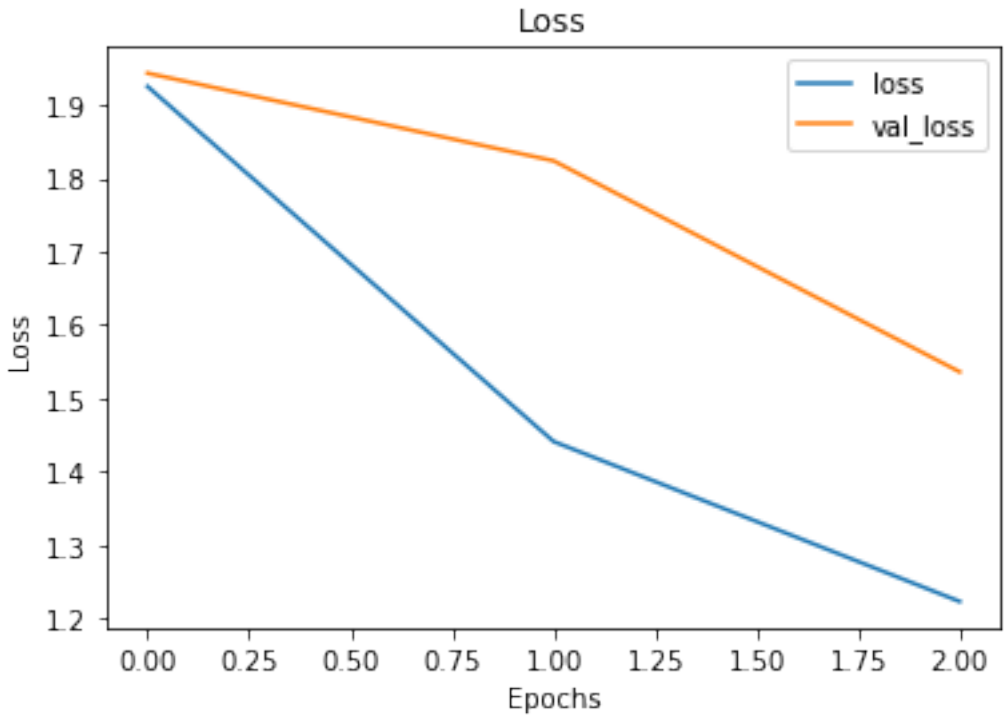
```
In [ ]: model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
```

```
In [ ]: history = model.fit(X_train, y_train_oh, callbacks=[checkpoint, earlystop], batch_size=
```

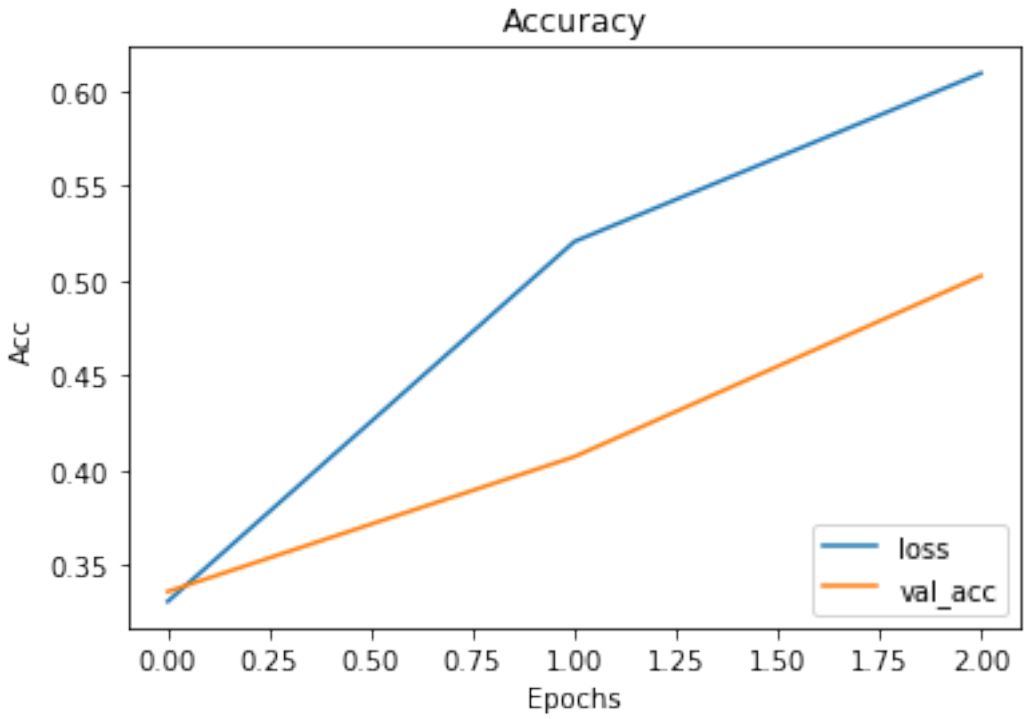
```
Train on 73257 samples, validate on 26032 samples
Epoch 1/3
73216/73257 [============================>.] - ETA: 0s - loss: 1.9251 - acc: 0.3309
Epoch 00001: val_loss improved from inf to 1.94311, saving model to MLP
73257/73257 [==============================] - 84s 1ms/sample - loss: 1.9250 - acc: 0.3309 - va
Epoch 2/3
73088/73257 [============================>.] - ETA: 0s - loss: 1.4409 - acc: 0.5206
Epoch 00002: val_loss improved from 1.94311 to 1.82360, saving model to MLP
73257/73257 [==============================] - 78s 1ms/sample - loss: 1.4407 - acc: 0.5207 - va
Epoch 3/3
73216/73257 [============================>.] - ETA: 0s - loss: 1.2228 - acc: 0.6094- ETA: 1s -
Epoch 00003: val_loss improved from 1.82360 to 1.53572, saving model to MLP
73257/73257 [==============================] - 75s 1ms/sample - loss: 1.2225 - acc: 0.6094 - va
```

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




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

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



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 [ ]: model2 = Sequential([
            Conv2D(filters= 32, kernel_size= 3, activation='relu', input_shape=X_train[0].shape
            MaxPool2D(pool_size= (3,3), strides=1),
            Conv2D(filters= 16, kernel_size = 3, activation='relu'),
            MaxPool2D(pool_size = (2,2), strides = 1),
            BatchNormalization(),
```

```
            Conv2D(filters= 8, kernel_size = 3, activation='relu'),
            tf.keras.layers.Dropout(0.3),
            Flatten(),
            Dense(16, activation='relu'),
            Dense(16, activation='relu'),
            tf.keras.layers.Dropout(0.3),
            Dense(10, activation='softmax')
        ])

        model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 16)        4624
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 25, 25, 16)        0
_________________________________________________________________
batch_normalization_4 (Batch (None, 25, 25, 16)        64
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 23, 23, 8)         1160
_________________________________________________________________
dropout_1 (Dropout)          (None, 23, 23, 8)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4232)              0
_________________________________________________________________
dense_5 (Dense)              (None, 16)                67728
_________________________________________________________________
dense_6 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                170
=================================================================
Total params: 74,914
Trainable params: 74,882
Non-trainable params: 32
_________________________________________________________________
```

```
In [ ]: model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
        check = ModelCheckpoint(filepath='CNNweights', save_best_only=True, save_weights_only=
        early = EarlyStopping(monitor='loss',patience=7, verbose=1)
```

```
In [ ]: history = model2.fit(X_train, y_train_oh, callbacks=[check, early], batch_size=256, val

In [ ]:

In [ ]:

In [ ]:
```

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 [ ]: model.load_weights('MLP')

In [ ]: model2.load_weights('CNNweights')

In [ ]: import random

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

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

        predictions = model.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. Model prediction")

        plt.show()

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

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

```
predictions = model2.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. Model prediction")

plt.show()
```

In [ ]: