

## ▼ Capstone Project

### Image classifier for the SVHN dataset

#### 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.

#### 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.

#### Let's get started!

We'll start by running some imports, and loading the dataset. For this project you are free to make further imports throughout the notebook as you wish.

```
import tensorflow as tf
from scipy.io import loadmat
import numpy as np
```

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

# Run this cell to connect to your Drive folder

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

🔗 Drive already mounted at /content/gdrive; to attempt to forcibly remount, call `drive.mount('/content/gdrive')`

# Load the dataset from your Drive folder

```
train = loadmat('gdrive/My Drive/train_32x32.mat')
test = loadmat('gdrive/My Drive/test_32x32.mat')
```

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

## ▼ 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.

```
x_train,y_train = train['X'],train['y']
x_test,y_test = test['X'],test['y']
y_train[y_train==10]=0
y_test[y_test==10]=0
```

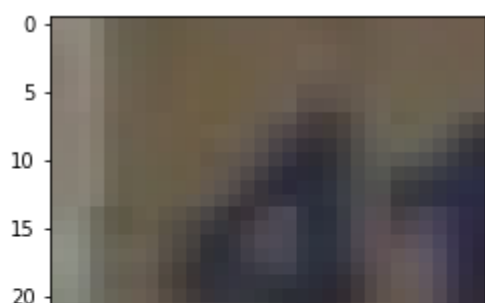
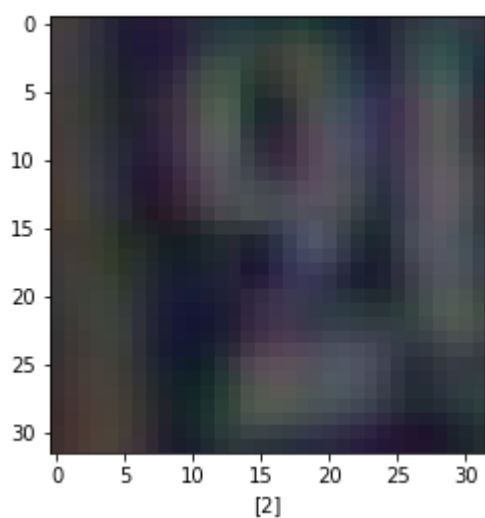
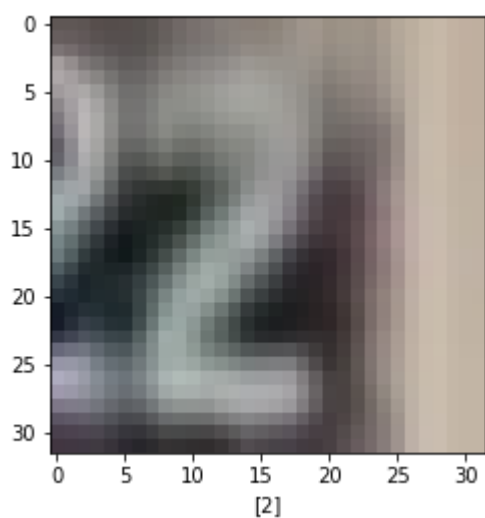
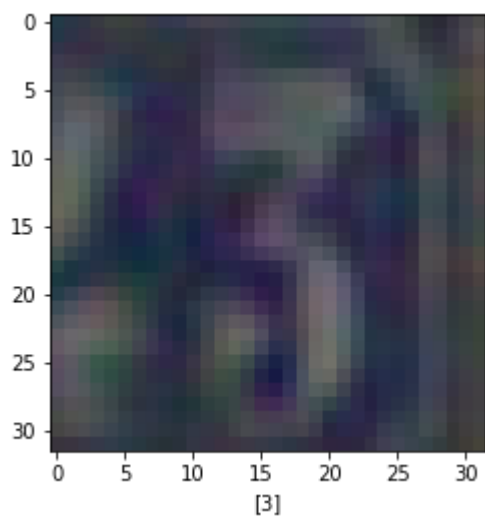
```
print("x_train.shape: ",x_train.shape," y_train.shape",y_train.shape)
x_train=np.moveaxis(x_train,3,0)
x_test =np.moveaxis(x_test,3,0)
print("x_train.shape: ",x_train.shape," y_train.shape",y_train.shape)
```

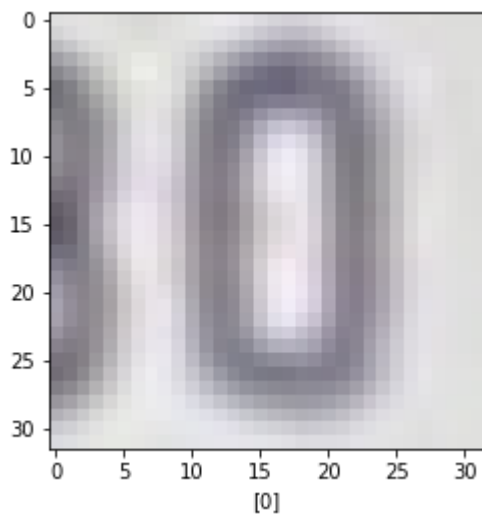
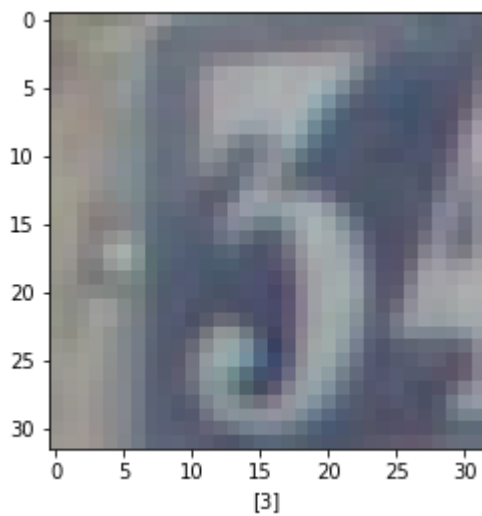
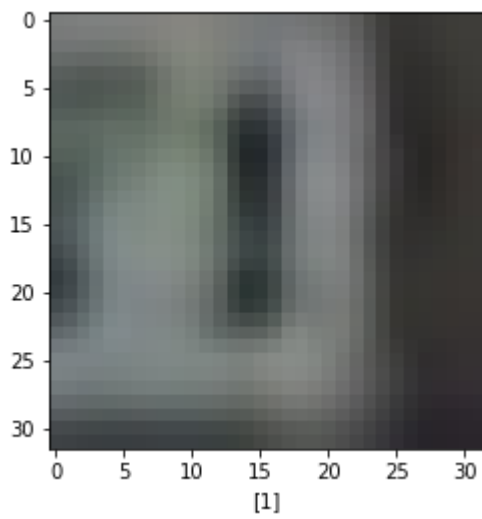
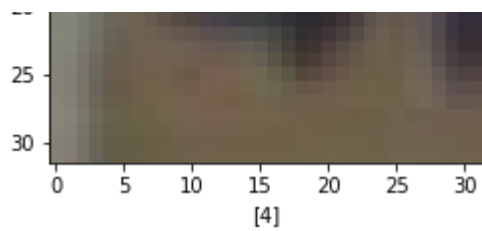
```
↳ x_train.shape: (32, 32, 3, 73257) y_train.shape (73257, 1)  
   x_train.shape: (73257, 32, 32, 3) y_train.shape (73257, 1)
```

```
for i in range(10):  
    random_idx = np.random.choice(x_train.shape[0])  
    plt.imshow(x_train[random_idx])  
    plt.xlabel(y_train[random_idx])  
    plt.show()
```

```
↳
```

```
/usr/local/lib/python3.6/dist-packages/matplotlib/text.py:1165: FutureWarning: elementwise  
if s != self._text:
```





```
x_train = np.mean(x_train,3).reshape(73257, 32, 32, 1)
```

```
x_test = np.mean(x_test,3).reshape(26032, 32, 32, 1)
```

```
x_test = np.mean(x_test, 0).reshape(20000, 32, 32, 1)
```

```
x_plot = np.mean(x_train, 3)
```

```
-- 
```

```
for _ in range(10):
```

```
    random_idx = np.random.choice(x_plot.shape[0])
```

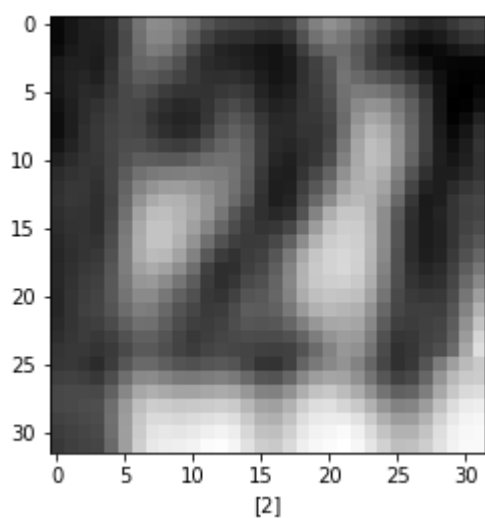
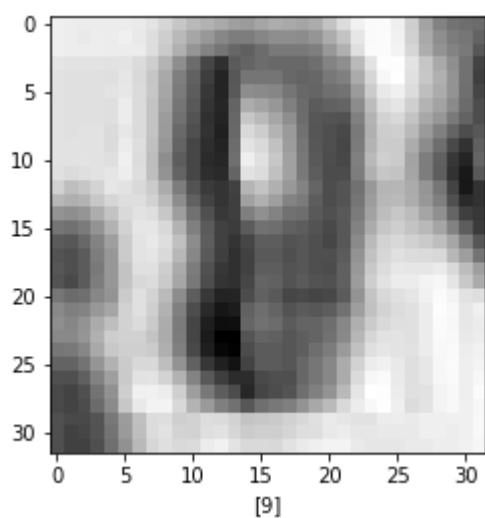
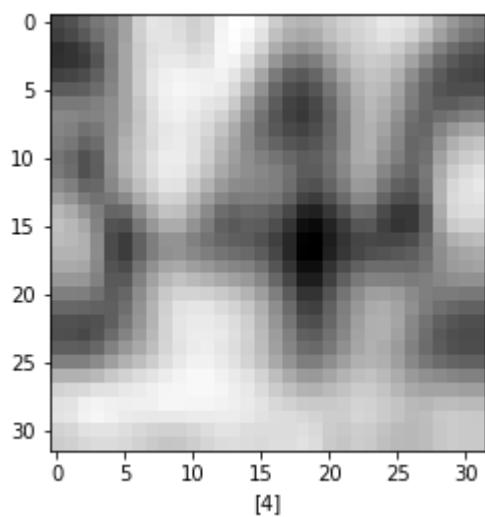
```
    plt.imshow(x_plot[random_idx], cmap='gray')
```

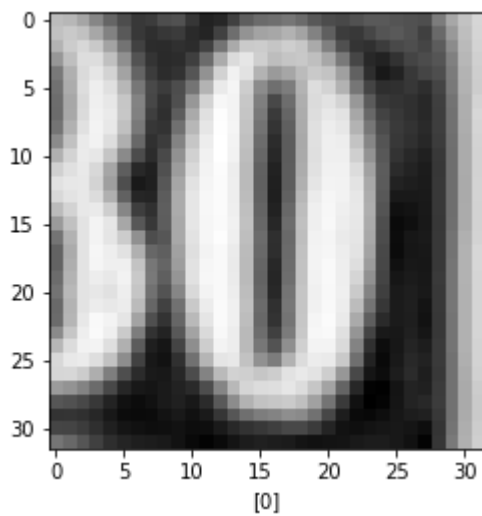
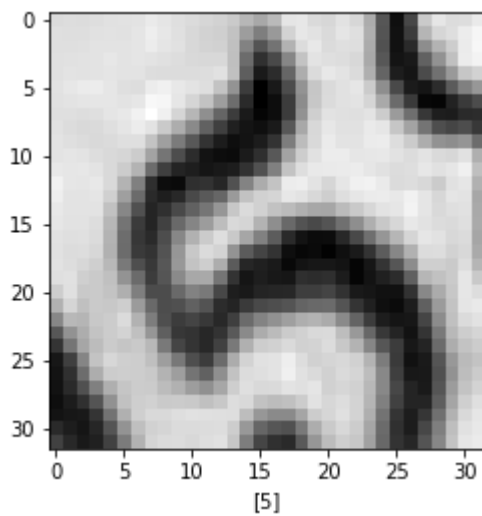
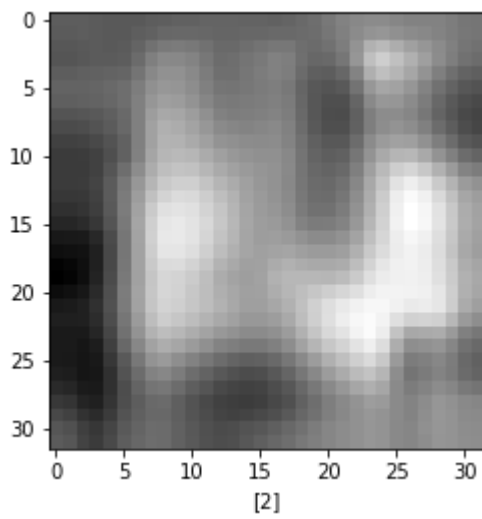
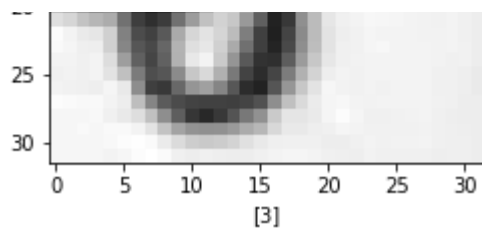
```
    plt.xlabel(y_train[random_idx])
```

```
    plt.show()
```



```
/usr/local/lib/python3.6/dist-packages/matplotlib/text.py:1165: FutureWarning: elementwise  
if s != self._text:
```





```
x_train = x_train/255
```

```
x_test = x_test/255
```



```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)
set(y_train)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

## ▼ 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.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=x_train[0].shape),
    tf.keras.layers.Dense(32,activation='relu'),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(10,activation='softmax')
])
model.summary()
```

```
↳
```

Model: "sequential\_25"

Layer (type)	Output Shape	Param #
=====		
flatten_25 (Flatten)	(None, 1024)	0
dense_89 (Dense)	(None, 32)	32800
dense_90 (Dense)	(None, 64)	2112

```
model.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(learn
checkpoint = tf.keras.callbacks.ModelCheckpoint('MLP3/check_points_best_only1/',monitor='val_
earllystop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',patience=3)
callbacks=[checkpoint,earllystop]
```

Total params: 44,522

```
history = model.fit(x_train,y_train,epochs=30,callbacks=callbacks,validation_data=(x_test,y_t
```



```

Epoch 1/30
2290/2290 [=====] - 10s 4ms/step - loss: 1.1535 - accuracy: 0.6
Epoch 2/30
2290/2290 [=====] - 10s 4ms/step - loss: 1.1205 - accuracy: 0.6
Epoch 3/30
2290/2290 [=====] - 10s 4ms/step - loss: 1.0943 - accuracy: 0.6
Epoch 4/30
2290/2290 [=====] - 10s 4ms/step - loss: 1.0679 - accuracy: 0.6
Epoch 5/30
2290/2290 [=====] - 10s 4ms/step - loss: 1.0472 - accuracy: 0.6
Epoch 6/30
2290/2290 [=====] - 10s 4ms/step - loss: 1.0287 - accuracy: 0.6
Epoch 7/30
2290/2290 [=====] - 10s 4ms/step - loss: 1.0116 - accuracy: 0.6
Epoch 8/30
2290/2290 [=====] - 10s 4ms/step - loss: 0.9941 - accuracy: 0.6
Epoch 9/30
2290/2290 [=====] - 10s 4ms/step - loss: 0.9781 - accuracy: 0.7
Epoch 10/30

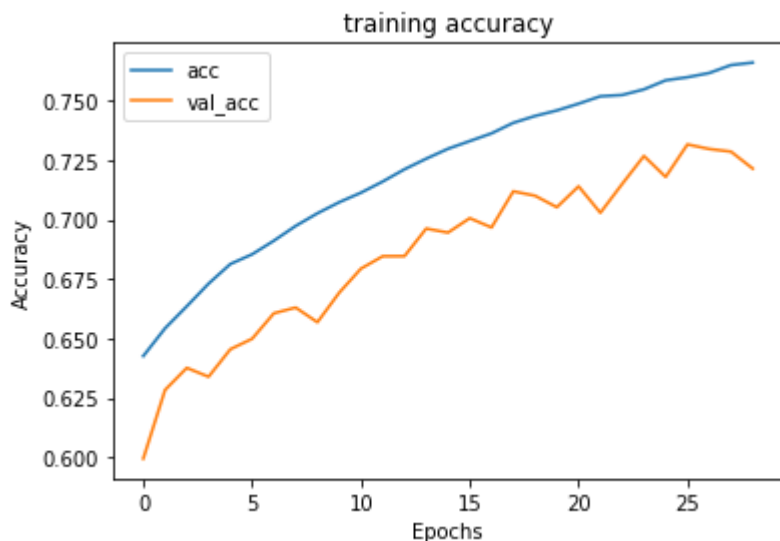
```

```

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['acc', 'val_acc'], loc='upper left')
plt.title('training accuracy')

```

☞ Text(0.5, 1.0, 'training accuracy')



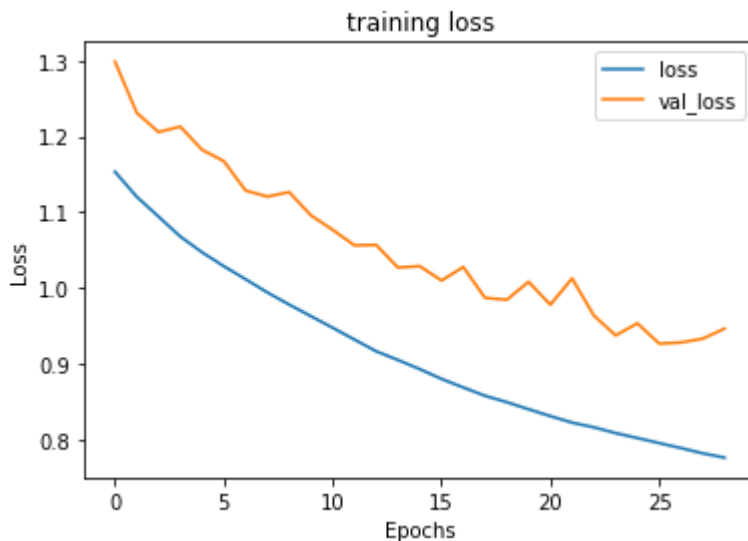
Epoch 24/30

```

import matplotlib.pyplot as plt
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('training loss')

```

↳ Text(0.5, 1.0, 'training loss')



### ▼ 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.

```
model_cnn = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32,(3,3),activation='relu',input_shape=x_train[0].shape),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Conv2D(34,(3,3),activation='relu'),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(256,activation='relu').
```

```
tf.keras.layers.Dense(10,activation='softmax')
])
model_cnn.summary()
```

Model: "sequential\_26"

Layer (type)	Output Shape	Param #
conv2d_22 (Conv2D)	(None, 30, 30, 32)	320
max_pooling2d_22 (MaxPooling)	(None, 15, 15, 32)	0
conv2d_23 (Conv2D)	(None, 13, 13, 34)	9826
max_pooling2d_23 (MaxPooling)	(None, 6, 6, 34)	0
batch_normalization_11 (Batch Normalization)	(None, 6, 6, 34)	136
flatten_26 (Flatten)	(None, 1224)	0
dense_93 (Dense)	(None, 128)	156800
dense_94 (Dense)	(None, 256)	33024
dense_95 (Dense)	(None, 10)	2570
Total params: 202,676		
Trainable params: 202,608		
Non-trainable params: 68		

```
model_cnn.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(1e-4))
checkpoint = tf.keras.callbacks.ModelCheckpoint('CNN1/check_points_best_only1/',monitor='val_loss',save_best_only=True)
earlystop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',patience=3)
callbacks=[checkpoint,earlystop]
```

```
hist = model_cnn.fit(x_train,y_train,epochs=30,callbacks=callbacks,validation_data=(x_test,y_test))
```

Model History

```

Epoch 1/30
2290/2290 [=====] - 16s 7ms/step - loss: 1.0046 - accuracy: 0.6
Epoch 2/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.5058 - accuracy: 0.8
Epoch 3/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.4342 - accuracy: 0.8
Epoch 4/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.3895 - accuracy: 0.8
Epoch 5/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.3550 - accuracy: 0.8
Epoch 6/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.3289 - accuracy: 0.9
Epoch 7/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.3047 - accuracy: 0.9
Epoch 8/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.2852 - accuracy: 0.9
Epoch 9/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.2663 - accuracy: 0.9
Epoch 10/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.2491 - accuracy: 0.9
Epoch 11/30
2290/2290 [=====] - 16s 7ms/step - loss: 0.2342 - accuracy: 0.9

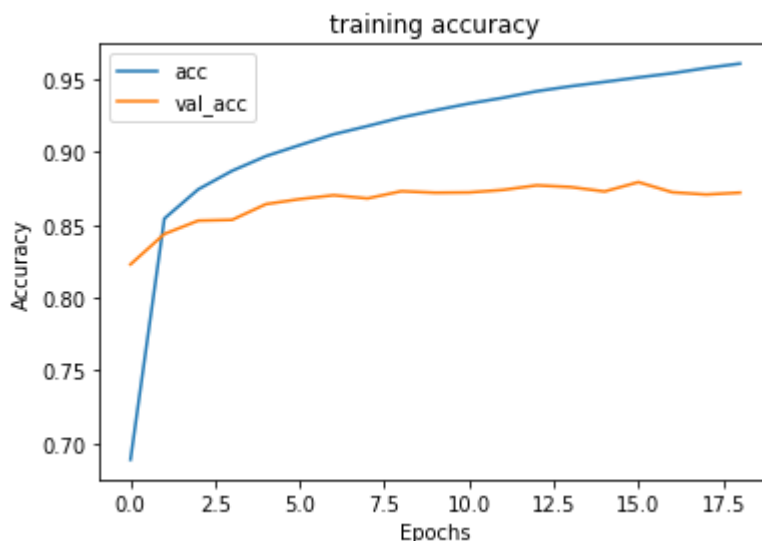
```

```

import matplotlib.pyplot as plt
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['acc', 'val_acc'], loc='upper left')
plt.title('training accuracy')

```

```
plt.text(0.5, 1.0, 'training accuracy')
```



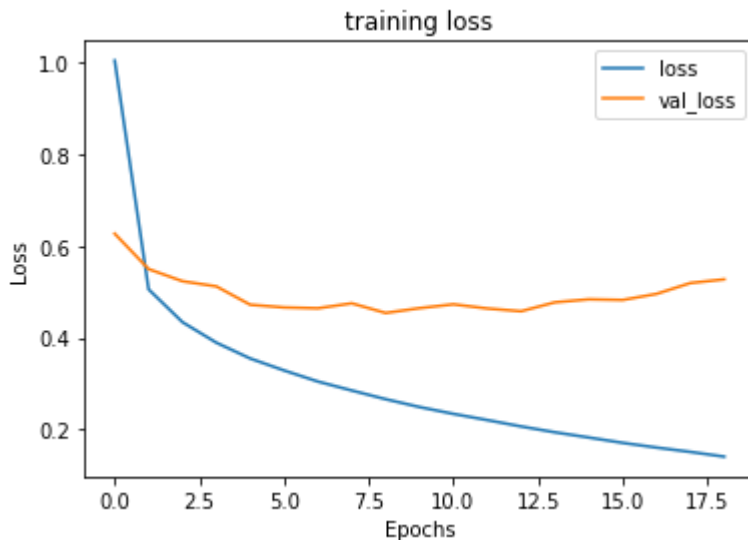
```

import matplotlib.pyplot as plt
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss', 'val_loss'], loc='upper right')

```

```
plt.title('training loss')
```

```
Text(0.5, 1.0, 'training loss')
```



## ▼ 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.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=x_train[0].shape),
    tf.keras.layers.Dense(32,activation='relu'),
    tf.keras.layers.Dense(64,activation='relu'),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(10,activation='softmax')
])
model.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(learn
```

```
model_cnn = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32,(3,3),activation='relu',input_shape=x_train[0].shape),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Conv2D(34,(3,3),activation='relu'),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128,activation='relu'),
    tf.keras.layers.Dense(10,activation='softmax')
```

```

tf.keras.layers.Dense(256,activation='relu'),
tf.keras.layers.Dense(10,activation='softmax')
])
model_cnn.compile(loss='sparse_categorical_crossentropy',optimizer=tf.keras.optimizers.Adam(1

```

```
model.load_weights('MLP3/check_points_best_only1/')
```

```
↳ <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f9a06f847f0>
```

```
model_cnn.load_weights('CNN1/check_points_best_only1/')
```

```
↳ <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f9a06f8d940>
```

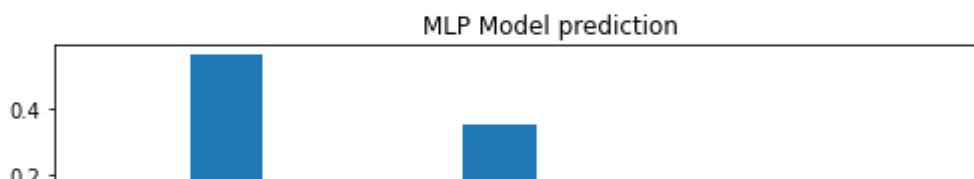
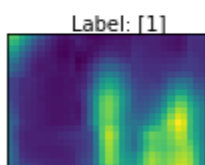
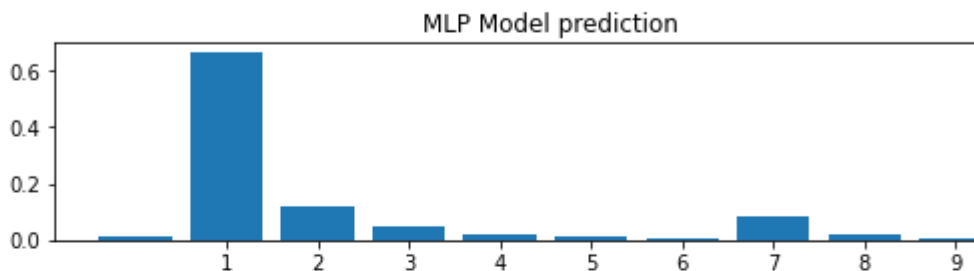
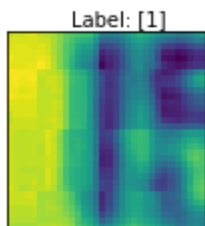
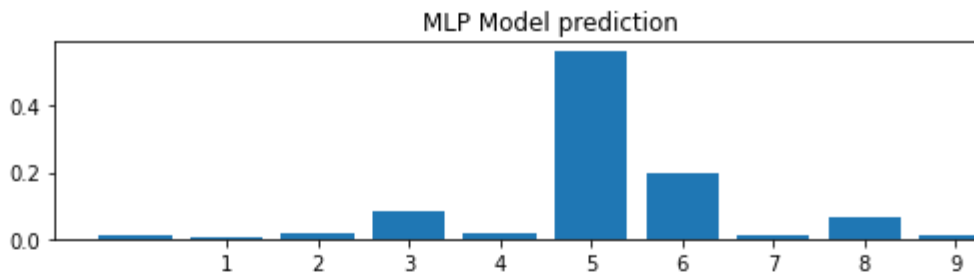
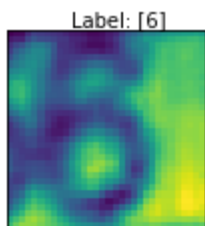
```

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, random_te
    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'Label: {le.inverse_transform([label])}')
    axes[i, 1].bar(np.arange(0,10), prediction)
    axes[i, 1].set_xticks(np.arange(0,10))
    axes[i, 1].set_title("MLP Model prediction")

```

```
↳
```





```

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_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, random_test_labels)):
    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'Label: {le.inverse_transform([label])}')
    axes[i, 1].bar(np.arange(0,10), prediction)
    axes[i, 1].set_xticks(np.arange(0,10))
    axes[i, 1].set_title("MLP Model prediction")

```



