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

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

SVHN overview image 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 [2]:
```
# 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. 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 [3]:

```
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D, Bat
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
tf.config.set_visible_devices([], 'GPU')
```

In [4]:

```
# Load the train and test dictionaries
train = loadmat('data/train_32x32.mat')
test = loadmat('data/test_32x32.mat')

train_images = train['X']
train_labels = train['y']

train_images = np.transpose(train_images, (3, 0, 1, 2))

test_images = test['X']
test_labels = test['y']

test_images = np.transpose(test_images, (3, 0, 1, 2))

train_labels = np.where(train_labels == 10, 0, train_labels)
test_labels = np.where(test_labels == 10, 0, test_labels)

print(train_images.shape)
print(train_labels.shape)
```

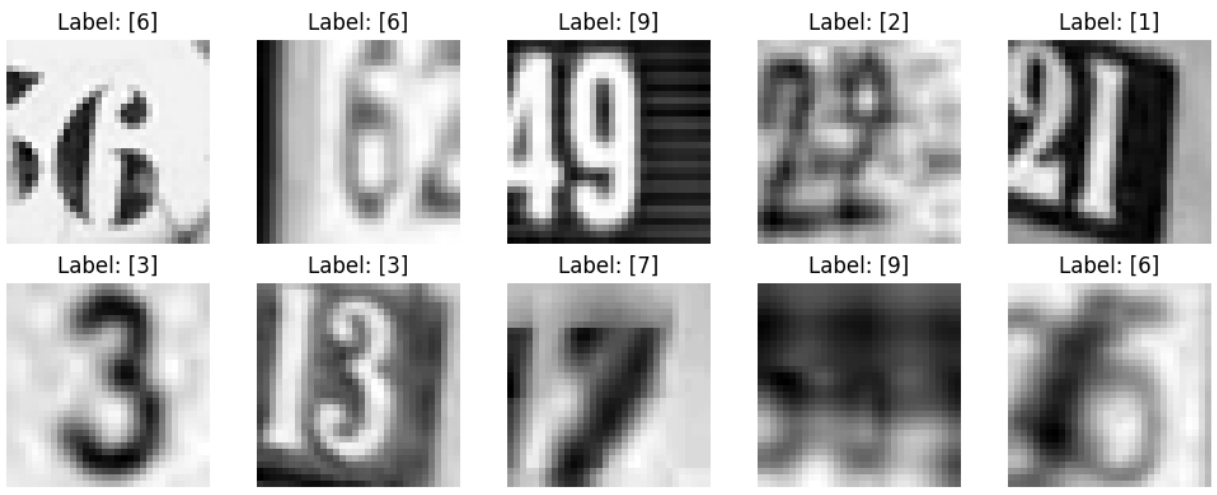
```
(73257, 32, 32, 3)
(73257, 1)
```

In [5]:
```
#Display random images
random_indices = np.random.choice(train_images.shape[0], 10, replace=False)
fig, axs = plt.subplots(2, 5, figsize=(10, 4))

for i, ax in enumerate(axs.flat):
    # Get the image and corresponding label
    image = train_images[random_indices[i]]
    label = train_labels[random_indices[i]]

    ax.imshow(image)
    ax.axis('off')
    ax.set_title(f'Label: {label}')

plt.tight_layout()
plt.show()
```

In [6]:
```
#Convert to greyscale
train_images_gray = np.mean(train_images, axis=3, keepdims=True)
test_images_gray = np.mean(test_images, axis=3, keepdims=True)

train_images_gray = train_images_gray / 255.
test_images_gray = test_images_gray / 255.

print(train_images_gray.shape)
print(train_labels.shape)
```

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

In [7]:
```
#Display the greyscale images
random_indices = np.random.choice(train_images_gray.shape[0], 10, replace=Fal
fig, axs = plt.subplots(2, 5, figsize=(10, 4))

for i, ax in enumerate(axs.flat):
    image_gray = train_images_gray[random_indices[i]]
    label = train_labels[random_indices[i]]

    ax.imshow(image_gray[:, :, 0], cmap='gray')
    ax.axis('off')
    ax.set_title(f'Label: {label}')

plt.tight_layout()
plt.show()
```



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 [8]:
```
def get_model(input_shape):
    model = Sequential([
        Flatten(input_shape = input_shape),
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(128, activation = 'relu'),
        Dense(10, activation = 'softmax')
    ])
    return model

model = get_model(train_images_gray[0].shape)

model.compile(optimizer = 'sgd',
                 loss = 'sparse_categorical_crossentropy',
                 metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 1024)              0         
 dense (Dense)               (None, 128)               131200

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 128)               16512

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 165,514
Trainable params: 165,514
Non-trainable params: 0
_________________________________________________________________
```

In [9]:

```
# Set up early stopping
early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, mode='max'

# Create the directory for saving the best checkpoints if it doesn't exist
if not os.path.exists('checkpoints_best'):
    os.makedirs('checkpoints_best')

# Define the filepath for saving the best checkpoints
filepath = os.path.join('checkpoints_best', 'checkpoint.h5')

# Set up model checkpoint to save the best model based on validation accuracy
checkpoint_best = ModelCheckpoint(filepath=filepath, save_best_only=True, mon

# Compile the callbacks
callbacks = [checkpoint_best, early_stopping]

# Train the model with the defined callbacks
history = model.fit(train_images_gray, train_labels, epochs=30, batch_size=32
```

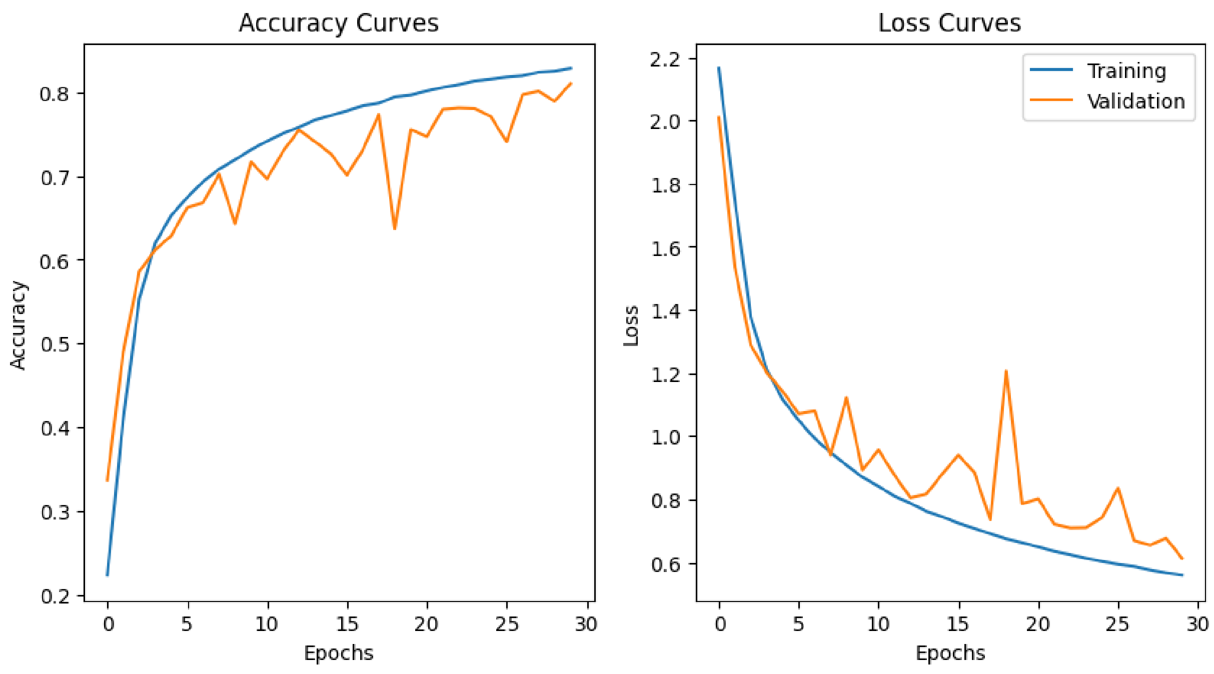
```
Epoch 1/30
2023-06-04 18:50:28.221265: W tensorflow/tsl/platform/profile_utils/cpu_utils.
cc:128] Failed to get CPU frequency: 0 Hz
  44/2061 [..............................] - ETA: 2s - loss: 2.2565 - accurac
y: 0.1982
2023-06-04 18:50:28.537969: I tensorflow/compiler/xla/service/service.cc:173]
XLA service 0x2b58dfb50 initialized for platform Host (this does not guarantee
that XLA will be used). Devices:
2023-06-04 18:50:28.537988: I tensorflow/compiler/xla/service/service.cc:181]
StreamExecutor device (0): Host, Default Version
2023-06-04 18:50:28.638962: I tensorflow/compiler/jit/xla_compilation_cache.c
c:477] Compiled cluster using XLA!  This line is logged at most once for the l
ifetime of the process.
2061/2061 [==============================] - 3s 1ms/step - loss: 2.1654 - accu
racy: 0.2230 - val_loss: 2.0085 - val_accuracy: 0.3365
Epoch 2/30
2061/2061 [==============================] - 2s 1ms/step - loss: 1.7500 - accu
racy: 0.4089 - val_loss: 1.5385 - val_accuracy: 0.4891
Epoch 3/30
2061/2061 [==============================] - 2s 1ms/step - loss: 1.3780 - accu
racy: 0.5527 - val_loss: 1.2890 - val_accuracy: 0.5859
Epoch 4/30
2061/2061 [==============================] - 2s 1ms/step - loss: 1.2114 - accu
racy: 0.6198 - val_loss: 1.2018 - val_accuracy: 0.6119
Epoch 5/30
2061/2061 [==============================] - 2s 1ms/step - loss: 1.1162 - accu
racy: 0.6529 - val_loss: 1.1419 - val_accuracy: 0.6286
Epoch 6/30
2061/2061 [==============================] - 2s 1ms/step - loss: 1.0503 - accu
```

```
racy: 0.6745 - val_loss: 1.0711 - val_accuracy: 0.6624
Epoch 7/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.9930 - accu
racy: 0.6935 - val_loss: 1.0799 - val_accuracy: 0.6683
Epoch 8/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.9483 - accu
racy: 0.7084 - val_loss: 0.9394 - val_accuracy: 0.7024
Epoch 9/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.9079 - accu
racy: 0.7196 - val_loss: 1.1219 - val_accuracy: 0.6433
Epoch 10/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.8698 - accu
racy: 0.7316 - val_loss: 0.8930 - val_accuracy: 0.7170
Epoch 11/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.8411 - accu
racy: 0.7418 - val_loss: 0.9560 - val_accuracy: 0.6967
Epoch 12/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.8096 - accu
racy: 0.7510 - val_loss: 0.8766 - val_accuracy: 0.7296
Epoch 13/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.7876 - accu
racy: 0.7586 - val_loss: 0.8048 - val_accuracy: 0.7553
Epoch 14/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.7617 - accu
racy: 0.7671 - val_loss: 0.8165 - val_accuracy: 0.7413
Epoch 15/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.7445 - accu
racy: 0.7725 - val_loss: 0.8804 - val_accuracy: 0.7267
Epoch 16/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.7242 - accu
racy: 0.7781 - val_loss: 0.9400 - val_accuracy: 0.7011
Epoch 17/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.7077 - accu
racy: 0.7842 - val_loss: 0.8850 - val_accuracy: 0.7314
Epoch 18/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.6908 - accu
racy: 0.7872 - val_loss: 0.7351 - val_accuracy: 0.7738
Epoch 19/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.6747 - accu
racy: 0.7949 - val_loss: 1.2060 - val_accuracy: 0.6370
Epoch 20/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.6623 - accu
racy: 0.7971 - val_loss: 0.7861 - val_accuracy: 0.7558
Epoch 21/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.6490 - accu
racy: 0.8019 - val_loss: 0.8015 - val_accuracy: 0.7473
Epoch 22/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.6354 - accu
racy: 0.8061 - val_loss: 0.7211 - val_accuracy: 0.7800
Epoch 23/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.6245 - accu
racy: 0.8094 - val_loss: 0.7094 - val_accuracy: 0.7816
Epoch 24/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.6132 - accu
racy: 0.8137 - val_loss: 0.7103 - val_accuracy: 0.7809
Epoch 25/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.6032 - accu
racy: 0.8159 - val_loss: 0.7421 - val_accuracy: 0.7715
Epoch 26/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.5941 - accu
racy: 0.8184 - val_loss: 0.8353 - val_accuracy: 0.7412
Epoch 27/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.5873 - accu
racy: 0.8200 - val_loss: 0.6692 - val_accuracy: 0.7974
```

```
Epoch 28/30
2061/2061 [==============================] - 3s 1ms/step - loss: 0.5762 - accu
racy: 0.8242 - val_loss: 0.6544 - val_accuracy: 0.8017
Epoch 29/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.5671 - accu
racy: 0.8256 - val_loss: 0.6769 - val_accuracy: 0.7895
Epoch 30/30
2061/2061 [==============================] - 2s 1ms/step - loss: 0.5600 - accu
racy: 0.8290 - val_loss: 0.6132 - val_accuracy: 0.8105
```

In [10]:

```
# plot accuracy and loss over training epochs
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot the accuracy and loss curves side by side
plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1) # Create the left subplot
plt.plot(accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.title('Accuracy Curves')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.subplot(1, 2, 2) # Create the right subplot
plt.plot(loss, label='Training')
plt.plot(val_loss, label='Validation')
plt.legend()
plt.title('Loss Curves')
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.show() # Show the plots
```



In [11]:

```
test_loss, test_accuracy = model.evaluate(test_images_gray, test_labels)
```

```
814/814 [==============================] - 1s 619us/step - loss: 0.7190 - accu
racy: 0.7936
```

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 [12]:

```
#Build the model
model = Sequential([
    Conv2D(8, (3, 3), padding='same', activation = 'relu', input_shape=(32, 3
    BatchNormalization(),
    Conv2D(8, (3, 3), padding='same', activation = 'relu', kernel_initializer
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.2),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

model.compile(optimizer = 'sgd', loss='sparse_categorical_crossentropy', metr
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 8)         80        
                                                                 
 batch_normalization (BatchN  (None, 32, 32, 8)        32        
 ormalization)                                                   
                                                                 
 conv2d_1 (Conv2D)           (None, 32, 32, 8)         584       
                                                                 
 batch_normalization_1 (Batc  (None, 32, 32, 8)        32        
 hNormalization)                                                 
                                                                 
 max_pooling2d (MaxPooling2D  (None, 16, 16, 8)        0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 16, 16, 8)         0         
                                                                 
 flatten_1 (Flatten)         (None, 2048)              0         
 dense_4 (Dense)             (None, 64)                131136

 dense_5 (Dense)             (None, 64)                4160

 dropout_1 (Dropout)         (None, 64)                0

 dense_6 (Dense)             (None, 10)                650

=================================================================
Total params: 136,674
Trainable params: 136,642
Non-trainable params: 32
_________________________________________________________________
```

In [13]:

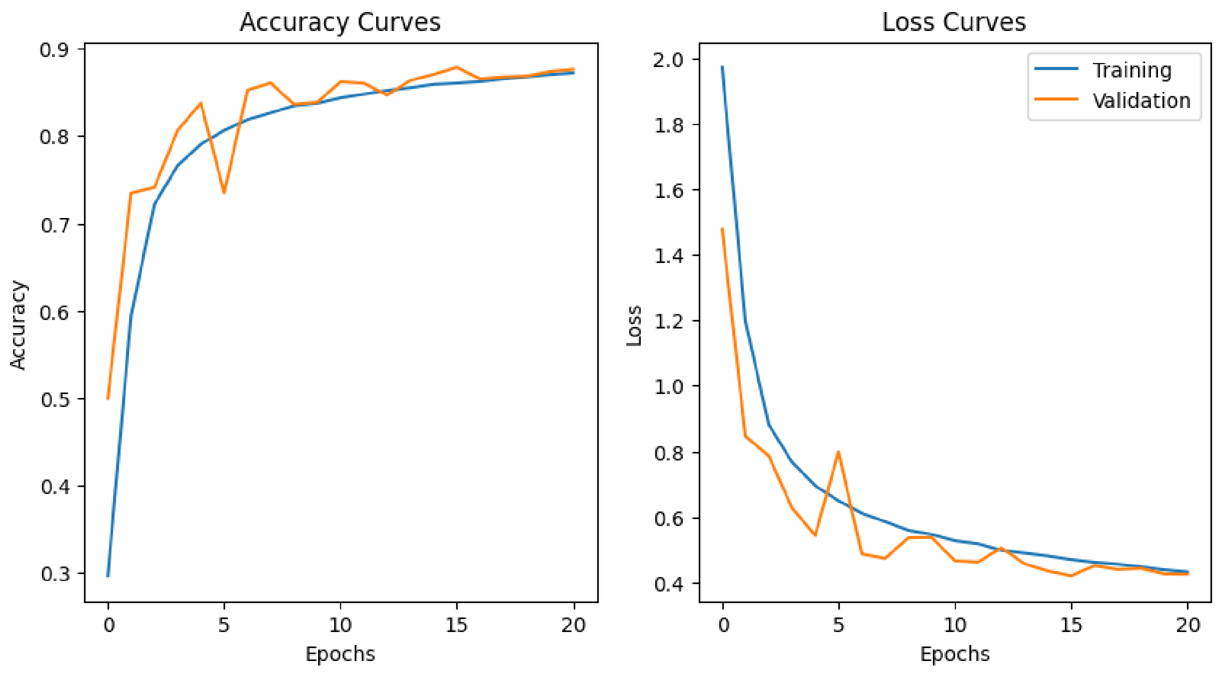
```
#Loss
early_stopping = EarlyStopping(monitor='val_loss', patience=5, mode = 'min')

if not os.path.exists('checkpoints_best_CNN'):
    os.makedirs('checkpoints_best_CNN')

filepath = os.path.join('checkpoints_best_CNN', 'checkpoint.h5')
checkpoint_best = ModelCheckpoint(filepath=filepath,
                                  save_best_only=True, monitor='val_loss', sa

callbacks = [checkpoint_best, early_stopping]

history = model.fit(train_images_gray, train_labels, epochs = 30, batch_size 
```

```
Epoch 1/30
2061/2061 [==============================] - 23s 11ms/step - loss: 1.9714 - ac
curacy: 0.2968 - val_loss: 1.4781 - val_accuracy: 0.4996
Epoch 2/30
2061/2061 [==============================] - 23s 11ms/step - loss: 1.1964 - ac
curacy: 0.5939 - val_loss: 0.8459 - val_accuracy: 0.7345
Epoch 3/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.8815 - ac
curacy: 0.7215 - val_loss: 0.7860 - val_accuracy: 0.7409
Epoch 4/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.7670 - ac
curacy: 0.7660 - val_loss: 0.6271 - val_accuracy: 0.8063
Epoch 5/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.6948 - ac
curacy: 0.7905 - val_loss: 0.5429 - val_accuracy: 0.8376
Epoch 6/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.6487 - ac
curacy: 0.8063 - val_loss: 0.7980 - val_accuracy: 0.7348
Epoch 7/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.6107 - ac
curacy: 0.8185 - val_loss: 0.4866 - val_accuracy: 0.8523
Epoch 8/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.5852 - ac
curacy: 0.8263 - val_loss: 0.4728 - val_accuracy: 0.8605
Epoch 9/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.5585 - ac
curacy: 0.8345 - val_loss: 0.5359 - val_accuracy: 0.8361
Epoch 10/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.5461 - ac
curacy: 0.8373 - val_loss: 0.5371 - val_accuracy: 0.8384
Epoch 11/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.5269 - ac
curacy: 0.8435 - val_loss: 0.4655 - val_accuracy: 0.8619
Epoch 12/30
```

```
2061/2061 [==============================] - 23s 11ms/step - loss: 0.5175 - ac
curacy: 0.8477 - val_loss: 0.4619 - val_accuracy: 0.8602
Epoch 13/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.4977 - ac
curacy: 0.8518 - val_loss: 0.5052 - val_accuracy: 0.8466
Epoch 14/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.4891 - ac
curacy: 0.8550 - val_loss: 0.4562 - val_accuracy: 0.8634
Epoch 15/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.4803 - ac
curacy: 0.8589 - val_loss: 0.4344 - val_accuracy: 0.8703
Epoch 16/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.4693 - ac
curacy: 0.8603 - val_loss: 0.4200 - val_accuracy: 0.8782
Epoch 17/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.4609 - ac
curacy: 0.8621 - val_loss: 0.4509 - val_accuracy: 0.8651
Epoch 18/30
2061/2061 [==============================] - 25s 12ms/step - loss: 0.4552 - ac
curacy: 0.8651 - val_loss: 0.4395 - val_accuracy: 0.8675
Epoch 19/30
2061/2061 [==============================] - 24s 12ms/step - loss: 0.4472 - ac
curacy: 0.8676 - val_loss: 0.4426 - val_accuracy: 0.8683
Epoch 20/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.4382 - ac
curacy: 0.8701 - val_loss: 0.4258 - val_accuracy: 0.8736
Epoch 21/30
2061/2061 [==============================] - 23s 11ms/step - loss: 0.4318 - ac
curacy: 0.8721 - val_loss: 0.4250 - val_accuracy: 0.8761
```

In [14]:

```python
#Plotting
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1)
plt.plot(accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.title('Accuracy Curves')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.subplot(1, 2, 2)
plt.plot(loss, label='Training')
plt.plot(val_loss, label='Validation')
plt.legend()
plt.title('Loss Curves')
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.show()
```



In [15]:
```
test_loss, test_accuracy = model.evaluate(test_images_gray, test_labels)
```

```
814/814 [==============================] - 2s 3ms/step - loss: 0.4986 - accura
cy: 0.8542
```

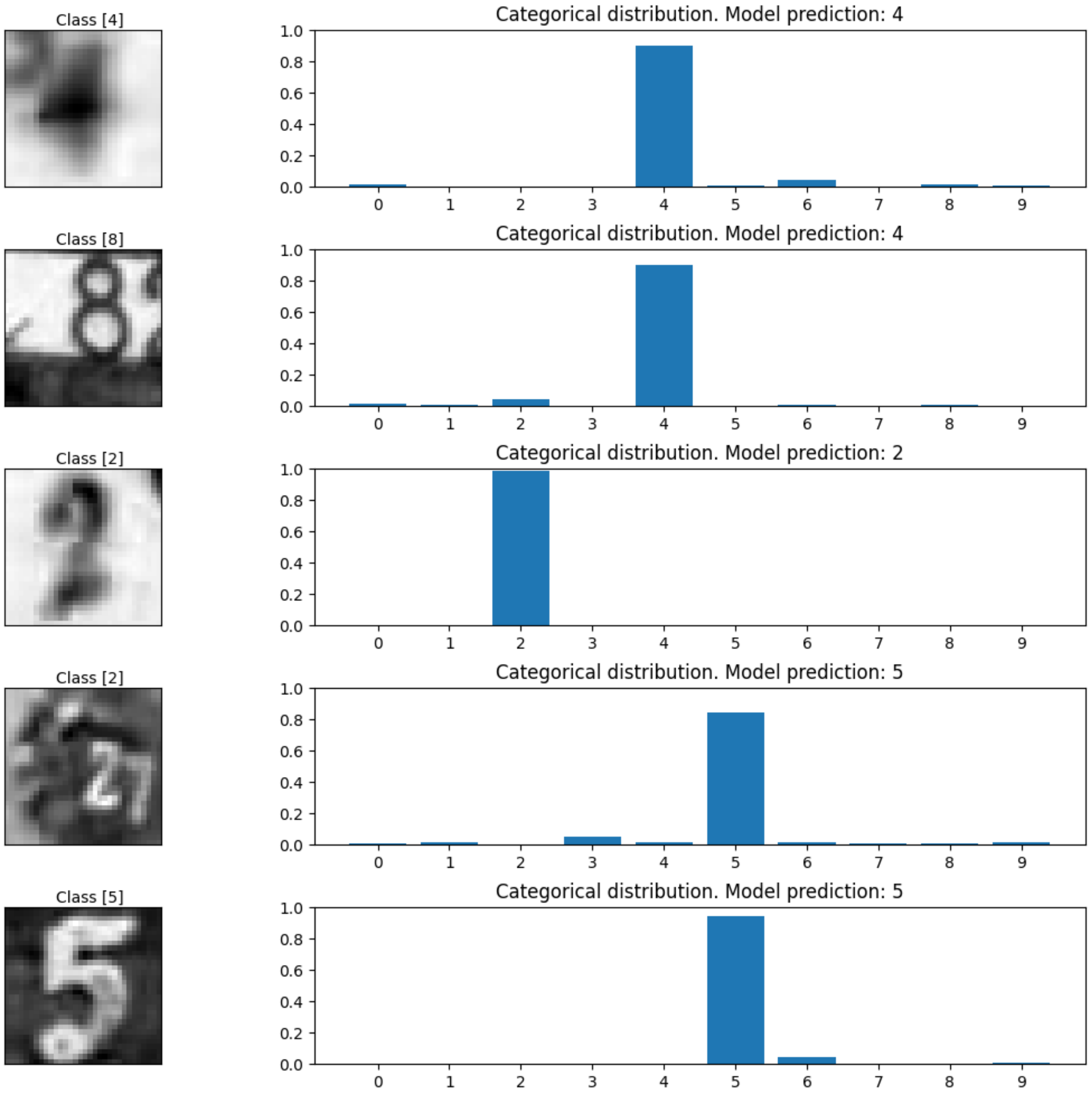
4. Get model predictions

- 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 [20]:
```
from tensorflow.keras.models import load_model
MLP='checkpoints_best/checkpoint.h5'
CNN='checkpoints_best_CNN/checkpoint.h5'
```

In [25]:
```
#Select random 5 images MLP
from sklearn.neural_network import MLPClassifier
MLP = load_model(MLP)
num_test_images = test_images_gray.shape[0]

random_inx = np.random.choice(num_test_images, 5)
random_test_images = test_images_gray[random_inx, ...]
random_test_labels = test_labels[random_inx, ...]

predictions = MLP.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_i
    axes[i, 0].imshow(np.squeeze(image), cmap = 'gray')
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Class {label}')
    axes[i, 1].bar(np.arange(len(prediction)), prediction)
    axes[i, 1].set_xticks(np.arange(len(prediction)))
    axes[i, 1].set_title(f"Categorical distribution. Model prediction: {np.ar

for ax in axes[:,1]:
    ax.set_ylim([0,1])

plt.show()
```

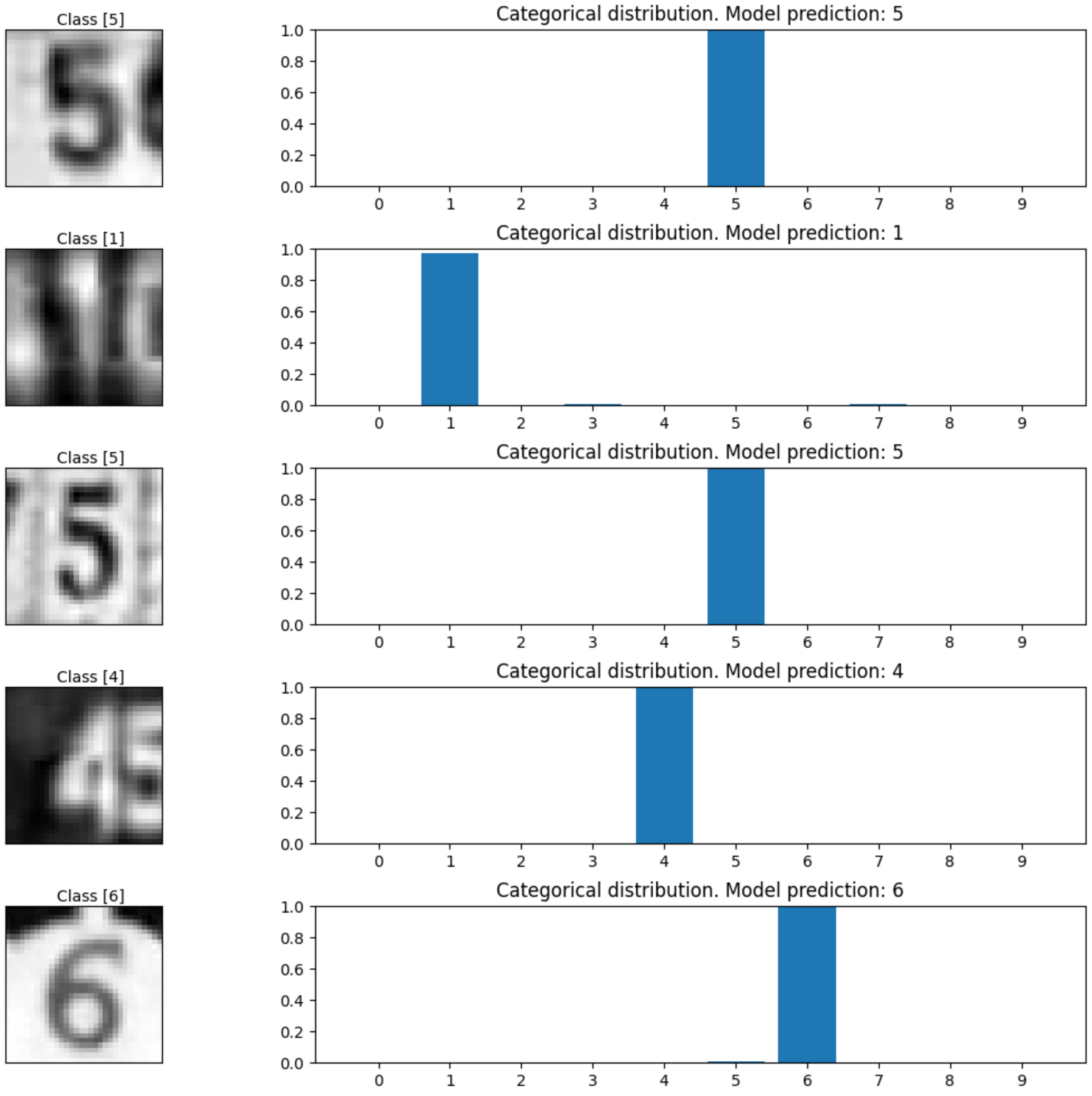
```
1/1 [==============================] - 0s 135ms/step
```



In [26]:

```
#Select random 5 images CNN
CNN = load_model(CNN)
num_test_images = test_images_gray.shape[0]

random_inx = np.random.choice(num_test_images, 5)
random_test_images = test_images_gray[random_inx, ...]
random_test_labels = test_labels[random_inx, ...]

predictions = 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_i
    axes[i, 0].imshow(np.squeeze(image), cmap = 'gray')
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Class {label}')
    axes[i, 1].bar(np.arange(len(prediction)), prediction)
    axes[i, 1].set_xticks(np.arange(len(prediction)))
    axes[i, 1].set_title(f"Categorical distribution. Model prediction: {np.ar

for ax in axes[:,1]:
    ax.set_ylim([0,1])

plt.show()
```

```
1/1 [==============================] - 0s 54ms/step
```



```
In [ ]:
```