Image classifiers for the SVHN dataset

January 10, 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 (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.

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.

[1]:
```
import tensorflow as tf
from scipy.io import loadmat
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪BatchNormalization, MaxPool2D, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,␣
 ↪LearningRateScheduler
import matplotlib.pyplot as plt
```

```
%matplotlib inline
import numpy as np
import pandas as pd

gpu_options = tf.compat.v1.GPUOptions(allow_growth=True)
sess = tf.compat.v1.Session(config=tf.compat.v1.
 ↪ConfigProto(gpu_options=gpu_options))
```

[2]:
```
tf.__version__
```

[2]:
```
'2.2.0'
```

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.

[3]:
```
# Load the dataset from your drive folder

train = loadmat('/home/marcin/Pictures/capstone_project/train_32x32.mat')
test = loadmat('/home/marcin/Pictures/capstone_project/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.

[4]:
```
train.keys()
```

[4]:
```
dict_keys(['__header__', '__version__', '__globals__', 'X', 'y'])
```

[5]:
```
train_data, train_targets = train['X'], train['y']
test_data, test_targets = test['X'], test['y']
```

[6]:
```
print(train_data.shape)
print(train_targets.shape)
```

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

[7]:
```
print(test_data.shape)
print(test_targets.shape)
```

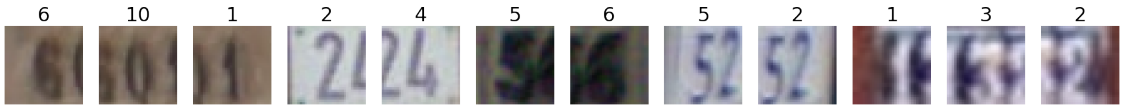
```
(32, 32, 3, 26032)
(26032, 1)
```

[8]:
```
i=0
for target in train_targets[:300]:
  if target == 10:
    print('index =', i)
  i += 1
```

```
index = 52
index = 84
index = 93
index = 96
index = 108
index = 144
index = 182
index = 206
index = 215
index = 218
index = 226
index = 236
index = 274
index = 294
```

Zeros are represented in targets as "10"

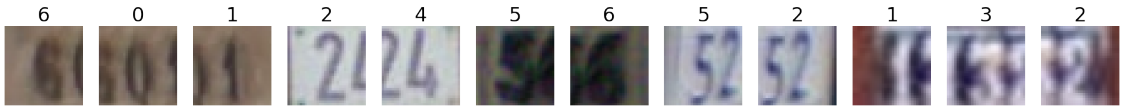
[9]:
```
num_of_img = 12
first_img_index = 51
fig, ax = plt.subplots(1, num_of_img, figsize=(32, 3))

for i in range(num_of_img):
    ax[i].set_axis_off()
    ax[i].imshow(train_data[..., i + first_img_index])
    ax[i].set_title(train_targets[i + first_img_index][0], size=32)
```



Instead 0's will be represented as 0

[10]:
```
i = 0
for target in train_targets[:]:
  if target == 10:
    train_targets[i] = 0
  i += 1

k=0
for target in test_targets[:]:
  if target == 10:
    test_targets[k] = 0
  k += 1
```

[11]:
```
num_of_img = 12
first_img_index = 51
fig, ax = plt.subplots(1, num_of_img, figsize=(32, 3))

for i in range(num_of_img):
    ax[i].set_axis_off()
    ax[i].imshow(train_data[..., i + first_img_index])
    ax[i].set_title(train_targets[i + first_img_index][0], size=32)
```



Converting images to grayscale

[12]:
```
def get_grayscale_image(data_images, samples=train_data.shape[3]):

  data_images_grayscale = np.zeros((samples, train_data.shape[0],
                                    train_data.shape[1]))

  for img_num in range(samples):
    data_images_grayscale[img_num, ...] = np.average(data_images[:, :, :,␣
↪img_num],
                                                     axis=2)
```

```
    return data_images_grayscale[..., np.newaxis]
```

[13]:
```
test_data.shape[3]
```

[13]: 26032

[14]:
```
train_data_grayscale = get_grayscale_image(train_data)
test_data_grayscale = get_grayscale_image(test_data, samples=test_data.shape[3])
```

Training images

[15]:
```
num_of_img = 12
first_img_index = 51
fig, ax = plt.subplots(1, num_of_img, figsize=(32, 1))

for i in range(num_of_img):
    ax[i].set_axis_off()
    ax[i].imshow(train_data_grayscale[i + first_img_index], cmap='gray')
    ax[i].set_title(train_targets[i + first_img_index][0], size=32)
```

Test images

[16]:
```
num_of_img = 12
first_img_index = 51
fig, ax = plt.subplots(1, num_of_img, figsize=(32, 1))

for i in range(num_of_img):
    ax[i].set_axis_off()
    ax[i].imshow(test_data_grayscale[i + first_img_index], cmap='gray')
    ax[i].set_title(test_targets[i + first_img_index][0], size=32)
```

Normalizing data

[17]:
```
train_data_grayscale = (train_data_grayscale - train_data_grayscale.mean()) /␣
↪train_data_grayscale.std()
test_data_grayscale = (test_data_grayscale - test_data_grayscale.mean()) /␣
↪test_data_grayscale.std()
```

[18]:
```
train_data_grayscale[4, 3, :5]
```

[18]:
```
array([[0.15943246],
       [0.22006684],
       [0.36828423],
       [0.46934153],
       [0.50976445]])
```

One-hot encoding test and train targets

[19]:
```
train_targets = tf.keras.utils.to_categorical(train_targets)
test_targets = tf.keras.utils.to_categorical(test_targets)
```

[20]:
```
train_targets.shape
```

[20]:
```
(73257, 10)
```

[21]:
```
train_targets[1:5,]
```

[21]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

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.

Model structure and compilation

[22]:
```
def get_mlp_model(input_shape=train_data_grayscale[1].shape):

  model = tf.keras.models.Sequential([
```

```
    Flatten(input_shape=input_shape),
    Dense(64, activation='relu', name='dense_1'),
    Dense(64, activation='relu', name='dense_2'),
    Dense(10, activation='softmax', name='dense_3')
  ])
  return model
```

[23]:
```
print('Shape passed to get_mlp_model function:',
      train_data_grayscale[1].shape)
```

```
Shape passed to get_mlp_model function: (32, 32, 1)
```

[24]:
```
model = get_mlp_model(train_data_grayscale[1].shape)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                65600
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 70,410
Trainable params: 70,410
Non-trainable params: 0
_________________________________________________________________
```

[25]:
```
lr = 0.001

def compile_model(model):
  model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

[26]:
```
compile_model(model)
```

Callbacks

[27]:
```
def get_best_epoch_callback():
  path='/home/marcin/Documents/Capstone_project/mlp_checkpoint_best'

  callback = ModelCheckpoint(path,
                             verbose=1,
```

```
                              save_best_only=True,
                              save_weights_only=False)
  return callback


def get_early_stopping_callback(patience=2):
  callback = EarlyStopping(monitor='val_loss',
                           patience=patience,
                           verbose=1)
  return callback
```

[28]:
```
best_epoch_callback = get_best_epoch_callback()
early_stopping_callback = get_early_stopping_callback()
```

Fitting the model

[29]:
```
history = model.fit(train_data_grayscale,
                    train_targets,
                    epochs=30,
                    batch_size=128,
                    validation_split=0.15,
                    callbacks=[best_epoch_callback, early_stopping_callback])
```

```
Epoch 1/30
481/487 [============================>.] - ETA: 0s - loss: 1.3837 - accuracy:
0.5542
Epoch 00001: val_loss improved from inf to 1.02498, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
WARNING:tensorflow:From
/home/marcin/anaconda3/envs/tensorflow/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:1817: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 3ms/step - loss: 1.3793 -
accuracy: 0.5560 - val_loss: 1.0250 - val_accuracy: 0.6887
Epoch 2/30
462/487 [===========================>..] - ETA: 0s - loss: 0.9352 - accuracy:
0.7175
Epoch 00002: val_loss improved from 1.02498 to 0.86635, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 2ms/step - loss: 0.9303 -
accuracy: 0.7185 - val_loss: 0.8663 - val_accuracy: 0.7359
```

```
Epoch 3/30
455/487 [==========================>..] - ETA: 0s - loss: 0.8049 - accuracy:
0.7575
Epoch 00003: val_loss improved from 0.86635 to 0.79105, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 2ms/step - loss: 0.8039 -
accuracy: 0.7578 - val_loss: 0.7911 - val_accuracy: 0.7597
Epoch 4/30
452/487 [=========================>…] - ETA: 0s - loss: 0.7396 - accuracy:
0.7792
Epoch 00004: val_loss improved from 0.79105 to 0.74351, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 2ms/step - loss: 0.7372 -
accuracy: 0.7799 - val_loss: 0.7435 - val_accuracy: 0.7752
Epoch 5/30
480/487 [============================>.] - ETA: 0s - loss: 0.6865 - accuracy:
0.7954
Epoch 00005: val_loss improved from 0.74351 to 0.71619, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 3ms/step - loss: 0.6863 -
accuracy: 0.7953 - val_loss: 0.7162 - val_accuracy: 0.7869
Epoch 6/30
478/487 [============================>.] - ETA: 0s - loss: 0.6528 - accuracy:
0.8054
Epoch 00006: val_loss improved from 0.71619 to 0.70266, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 3ms/step - loss: 0.6532 -
accuracy: 0.8053 - val_loss: 0.7027 - val_accuracy: 0.7934
Epoch 7/30
456/487 [==========================>..] - ETA: 0s - loss: 0.6296 - accuracy:
0.8114
Epoch 00007: val_loss did not improve from 0.70266
487/487 [==============================] - 1s 2ms/step - loss: 0.6300 -
accuracy: 0.8117 - val_loss: 0.7103 - val_accuracy: 0.7902
Epoch 8/30
473/487 [============================>.] - ETA: 0s - loss: 0.6044 - accuracy:
0.8201
Epoch 00008: val_loss improved from 0.70266 to 0.67223, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
```

```
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 3ms/step - loss: 0.6043 -
accuracy: 0.8201 - val_loss: 0.6722 - val_accuracy: 0.8053
Epoch 9/30
462/487 [===========================>..] - ETA: 0s - loss: 0.5902 - accuracy:
0.8227
Epoch 00009: val_loss improved from 0.67223 to 0.66100, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 3ms/step - loss: 0.5898 -
accuracy: 0.8230 - val_loss: 0.6610 - val_accuracy: 0.8081
Epoch 10/30
464/487 [===========================>..] - ETA: 0s - loss: 0.5732 - accuracy:
0.8277
Epoch 00010: val_loss did not improve from 0.66100
487/487 [==============================] - 1s 2ms/step - loss: 0.5732 -
accuracy: 0.8278 - val_loss: 0.6640 - val_accuracy: 0.8047
Epoch 11/30
478/487 [============================>.] - ETA: 0s - loss: 0.5592 - accuracy:
0.8322
Epoch 00011: val_loss improved from 0.66100 to 0.64856, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 2ms/step - loss: 0.5596 -
accuracy: 0.8320 - val_loss: 0.6486 - val_accuracy: 0.8116
Epoch 12/30
479/487 [============================>.] - ETA: 0s - loss: 0.5432 - accuracy:
0.8372
Epoch 00012: val_loss did not improve from 0.64856
487/487 [==============================] - 1s 2ms/step - loss: 0.5434 -
accuracy: 0.8373 - val_loss: 0.6561 - val_accuracy: 0.8141
Epoch 13/30
480/487 [============================>.] - ETA: 0s - loss: 0.5348 - accuracy:
0.8385
Epoch 00013: val_loss improved from 0.64856 to 0.64586, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 2ms/step - loss: 0.5348 -
accuracy: 0.8384 - val_loss: 0.6459 - val_accuracy: 0.8137
Epoch 14/30
472/487 [============================>.] - ETA: 0s - loss: 0.5201 - accuracy:
0.8444
Epoch 00014: val_loss did not improve from 0.64586
487/487 [==============================] - 1s 2ms/step - loss: 0.5205 -
accuracy: 0.8443 - val_loss: 0.6500 - val_accuracy: 0.8176
```

```
Epoch 15/30
471/487 [============================>.] - ETA: 0s - loss: 0.5116 - accuracy:
0.8450
Epoch 00015: val_loss improved from 0.64586 to 0.63730, saving model to
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/mlp_checkpoint_best/assets
487/487 [==============================] - 1s 2ms/step - loss: 0.5136 -
accuracy: 0.8446 - val_loss: 0.6373 - val_accuracy: 0.8206
Epoch 16/30
458/487 [===========================>..] - ETA: 0s - loss: 0.5025 - accuracy:
0.8470
Epoch 00016: val_loss did not improve from 0.63730
487/487 [==============================] - 1s 2ms/step - loss: 0.5035 -
accuracy: 0.8464 - val_loss: 0.6545 - val_accuracy: 0.8156
Epoch 17/30
482/487 [============================>.] - ETA: 0s - loss: 0.5007 - accuracy:
0.8472
Epoch 00017: val_loss did not improve from 0.63730
487/487 [==============================] - 1s 2ms/step - loss: 0.5006 -
accuracy: 0.8472 - val_loss: 0.6438 - val_accuracy: 0.8174
Epoch 00017: early stopping
```

[30]:
```
test_loss, test_acc = model.evaluate(test_data_grayscale, test_targets)
```

```
814/814 [==============================] - 1s 1ms/step - loss: 0.7559 -
accuracy: 0.7956
```

[31]:
```
print('Test loss = {:.03f}\nTest accuracy = {:.03f}'.format(test_loss,␣
 ↪test_acc))
```

```
Test loss = 0.756
Test accuracy = 0.796
```

[32]:
```
history.history.keys()
```

[32]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

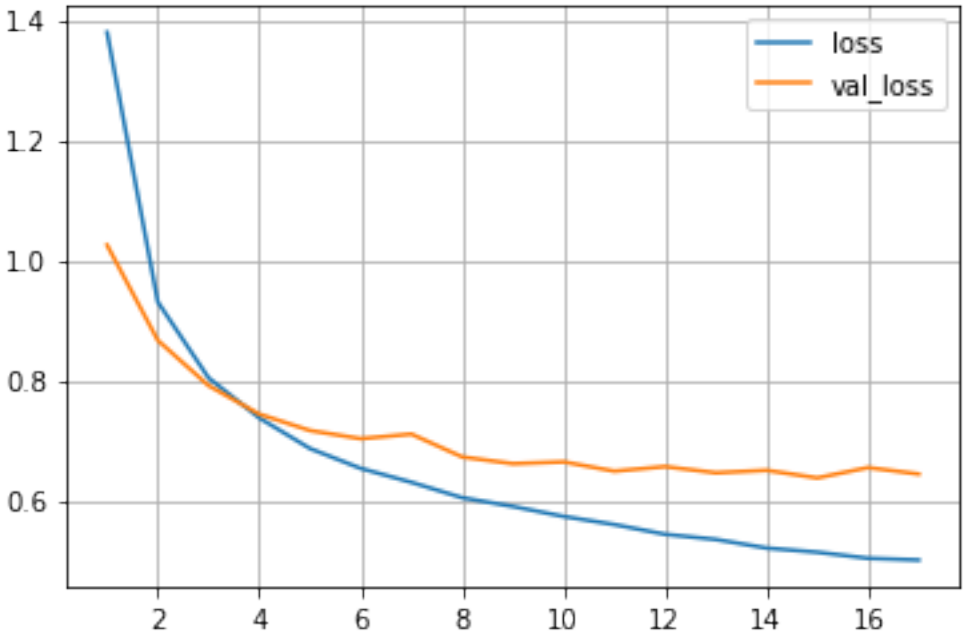
[34]:
```
df = pd.DataFrame(
    {'val_loss': history.history['val_loss'],
     'loss': history.history['loss'],
     'accuracy': history.history['accuracy'],
     'val_accuracy': history.history['val_accuracy']},
    index=range(1, 18),
    )

df.head()
```

[34]:

	val_loss	loss	accuracy	val_accuracy
1	1.024982	1.379323	0.555952	0.688689
2	0.866346	0.930328	0.718523	0.735918
3	0.791051	0.803931	0.757773	0.759669
4	0.743514	0.737203	0.779855	0.775230
5	0.716193	0.686276	0.795336	0.786878

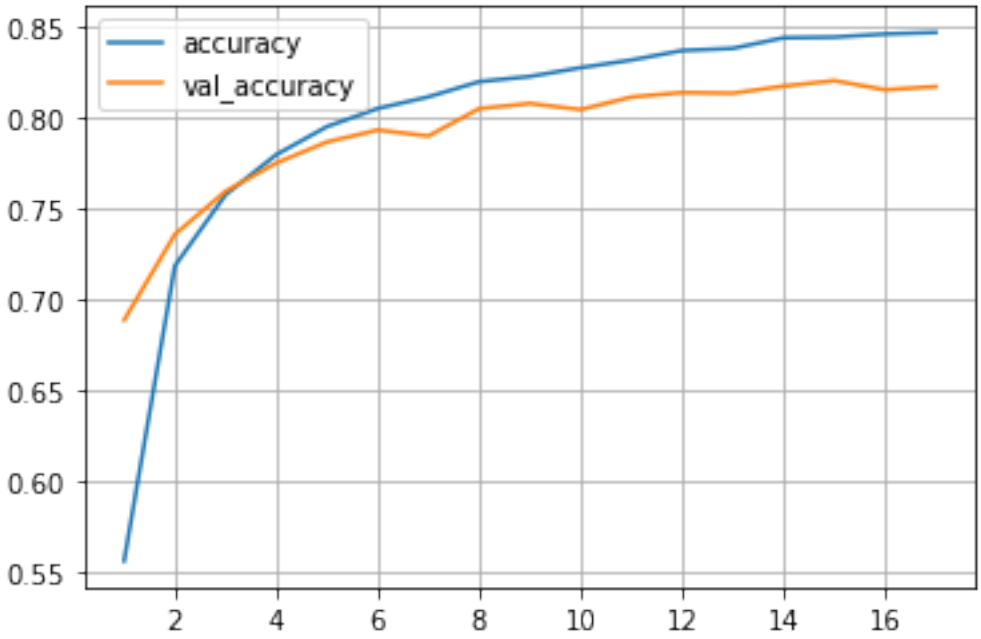
[35]:
```
df[['loss', 'val_loss']].plot(grid=True)
```

[35]: <AxesSubplot:>



[36]:
```
df[['accuracy', 'val_accuracy']].plot(grid=True)
```

[36]: <AxesSubplot:>



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.

Model structure and compilation

[37]:
```
print('input_shape =', train_data_grayscale[1].shape)
```

```
input_shape = (32, 32, 1)
```

[38]:
```
def get_cnn_model(input_shape=train_data_grayscale[1].shape):
  model = tf.keras.models.Sequential([

                      Conv2D(16, (3, 3), input_shape=(input_shape),
                             activation='relu',
                             bias_initializer='zeros'),

                      MaxPool2D((2, 2)),

                      BatchNormalization(),

                      Conv2D(32, (3,3), activation='relu'),

                      MaxPool2D((2, 2)),

                      BatchNormalization(),

                      Conv2D(64, (3,3), activation='relu'),

                      BatchNormalization(),

                      Flatten(),

                      Dropout(0.3),

                      Dense(32, activation='relu', kernel_regularizer='l2'),

                      Dense(10, activation='softmax'),
  ])

  return model
```

[39]:
```
def compile_model(model, lr=0.0001):
  model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=lr),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Callbacks

[40]:
```
def get_best_epoch_callback_cnn():
  path='/home/marcin/Documents/Capstone_project/cnn_checkpoint_best'

  callback = ModelCheckpoint(path,
                             verbose=1,
                             save_best_only=True,
                             save_weights_only=False)
  return callback
```

```
def get_early_stopping_callback_cnn(patience=2, monitor='val_loss'):
  callback = EarlyStopping(monitor=monitor,
                           patience=patience,
                           verbose=1)
  return callback
```

[41]:
```
cnn_best_epoch_callback = get_best_epoch_callback_cnn()
cnn_early_stopping_callback = get_early_stopping_callback_cnn(3)
```

Fitting the model

[42]:
```
model = get_cnn_model(train_data_grayscale[1].shape)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 16)        160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 16)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 15, 15, 16)        64
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 32)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 6, 6, 32)          128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 64)          256
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 32)                32800
_________________________________________________________________
dense_1 (Dense)              (None, 10)                330
=================================================================
Total params: 56,874
Trainable params: 56,650
Non-trainable params: 224
_________________________________________________________________
```

[43]:
```
compile_model(model)
```

[44]:
```
epochs=30

history = model.fit(train_data_grayscale,
                    train_targets,
                    epochs=epochs,
                    batch_size=128,
                    validation_split=0.15,
                    callbacks=[cnn_best_epoch_callback,␣
 ↪cnn_early_stopping_callback])
```

```
Epoch 1/30
487/487 [==============================] - ETA: 0s - loss: 2.5382 - accuracy:
0.3436
Epoch 00001: val_loss improved from inf to 1.97657, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 15s 31ms/step - loss: 2.5382 -
accuracy: 0.3436 - val_loss: 1.9766 - val_accuracy: 0.5415
Epoch 2/30
476/487 [============================>.] - ETA: 0s - loss: 1.5660 - accuracy:
0.6620
Epoch 00002: val_loss improved from 1.97657 to 1.22380, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 1.5597 -
accuracy: 0.6639 - val_loss: 1.2238 - val_accuracy: 0.7682
Epoch 3/30
486/487 [============================>.] - ETA: 0s - loss: 1.1457 - accuracy:
0.7765
Epoch 00003: val_loss improved from 1.22380 to 0.98271, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 1.1456 -
accuracy: 0.7765 - val_loss: 0.9827 - val_accuracy: 0.8208
Epoch 4/30
487/487 [==============================] - ETA: 0s - loss: 0.9487 - accuracy:
0.8170
Epoch 00004: val_loss improved from 0.98271 to 0.85136, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.9487 -
```

```
accuracy: 0.8170 - val_loss: 0.8514 - val_accuracy: 0.8406
Epoch 5/30
476/487 [============================>.] - ETA: 0s - loss: 0.8234 - accuracy:
0.8365
Epoch 00005: val_loss improved from 0.85136 to 0.74362, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.8230 -
accuracy: 0.8364 - val_loss: 0.7436 - val_accuracy: 0.8565
Epoch 6/30
473/487 [============================>.] - ETA: 0s - loss: 0.7290 - accuracy:
0.8515
Epoch 00006: val_loss improved from 0.74362 to 0.67021, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.7275 -
accuracy: 0.8516 - val_loss: 0.6702 - val_accuracy: 0.8651
Epoch 7/30
480/487 [============================>.] - ETA: 0s - loss: 0.6602 - accuracy:
0.8611
Epoch 00007: val_loss improved from 0.67021 to 0.61442, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.6591 -
accuracy: 0.8614 - val_loss: 0.6144 - val_accuracy: 0.8711
Epoch 8/30
485/487 [============================>.] - ETA: 0s - loss: 0.6034 - accuracy:
0.8692
Epoch 00008: val_loss improved from 0.61442 to 0.57789, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.6031 -
accuracy: 0.8693 - val_loss: 0.5779 - val_accuracy: 0.8726
Epoch 9/30
487/487 [==============================] - ETA: 0s - loss: 0.5593 - accuracy:
0.8758
Epoch 00009: val_loss improved from 0.57789 to 0.54271, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.5593 -
accuracy: 0.8758 - val_loss: 0.5427 - val_accuracy: 0.8770
Epoch 10/30
474/487 [============================>.] - ETA: 0s - loss: 0.5252 - accuracy:
```

```
0.8807
Epoch 00010: val_loss improved from 0.54271 to 0.50865, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 7ms/step - loss: 0.5260 -
accuracy: 0.8806 - val_loss: 0.5087 - val_accuracy: 0.8843
Epoch 11/30
487/487 [==============================] - ETA: 0s - loss: 0.4966 - accuracy:
0.8850
Epoch 00011: val_loss improved from 0.50865 to 0.49117, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.4966 -
accuracy: 0.8850 - val_loss: 0.4912 - val_accuracy: 0.8873
Epoch 12/30
472/487 [============================>.] - ETA: 0s - loss: 0.4727 - accuracy:
0.8887
Epoch 00012: val_loss improved from 0.49117 to 0.46931, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.4723 -
accuracy: 0.8887 - val_loss: 0.4693 - val_accuracy: 0.8906
Epoch 13/30
476/487 [============================>.] - ETA: 0s - loss: 0.4549 - accuracy:
0.8915
Epoch 00013: val_loss improved from 0.46931 to 0.45431, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.4544 -
accuracy: 0.8915 - val_loss: 0.4543 - val_accuracy: 0.8929
Epoch 14/30
474/487 [============================>.] - ETA: 0s - loss: 0.4380 - accuracy:
0.8949
Epoch 00014: val_loss improved from 0.45431 to 0.43920, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.4377 -
accuracy: 0.8947 - val_loss: 0.4392 - val_accuracy: 0.8943
Epoch 15/30
475/487 [============================>.] - ETA: 0s - loss: 0.4222 - accuracy:
0.8974
Epoch 00015: val_loss improved from 0.43920 to 0.42864, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
```

```
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.4225 -
accuracy: 0.8974 - val_loss: 0.4286 - val_accuracy: 0.8965
Epoch 16/30
475/487 [============================>.] - ETA: 0s - loss: 0.4097 - accuracy:
0.9006
Epoch 00016: val_loss improved from 0.42864 to 0.41901, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.4105 -
accuracy: 0.9004 - val_loss: 0.4190 - val_accuracy: 0.8977
Epoch 17/30
486/487 [============================>.] - ETA: 0s - loss: 0.3995 - accuracy:
0.9014
Epoch 00017: val_loss improved from 0.41901 to 0.40966, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.3995 -
accuracy: 0.9014 - val_loss: 0.4097 - val_accuracy: 0.9007
Epoch 18/30
476/487 [============================>.] - ETA: 0s - loss: 0.3894 - accuracy:
0.9039
Epoch 00018: val_loss improved from 0.40966 to 0.40391, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.3898 -
accuracy: 0.9038 - val_loss: 0.4039 - val_accuracy: 0.9004
Epoch 19/30
472/487 [============================>.] - ETA: 0s - loss: 0.3797 - accuracy:
0.9059
Epoch 00019: val_loss improved from 0.40391 to 0.39456, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.3804 -
accuracy: 0.9057 - val_loss: 0.3946 - val_accuracy: 0.9034
Epoch 20/30
482/487 [============================>.] - ETA: 0s - loss: 0.3729 - accuracy:
0.9068
Epoch 00020: val_loss improved from 0.39456 to 0.39107, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.3725 -
```

```
accuracy: 0.9069 - val_loss: 0.3911 - val_accuracy: 0.9039
Epoch 21/30
478/487 [============================>.] - ETA: 0s - loss: 0.3647 - accuracy:
0.9092
Epoch 00021: val_loss improved from 0.39107 to 0.38811, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.3653 -
accuracy: 0.9091 - val_loss: 0.3881 - val_accuracy: 0.9031
Epoch 22/30
477/487 [============================>.] - ETA: 0s - loss: 0.3581 - accuracy:
0.9109
Epoch 00022: val_loss improved from 0.38811 to 0.37752, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.3574 -
accuracy: 0.9111 - val_loss: 0.3775 - val_accuracy: 0.9048
Epoch 23/30
484/487 [============================>.] - ETA: 0s - loss: 0.3501 - accuracy:
0.9129
Epoch 00023: val_loss improved from 0.37752 to 0.37297, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 7ms/step - loss: 0.3504 -
accuracy: 0.9127 - val_loss: 0.3730 - val_accuracy: 0.9073
Epoch 24/30
473/487 [============================>.] - ETA: 0s - loss: 0.3455 - accuracy:
0.9133
Epoch 00024: val_loss improved from 0.37297 to 0.37189, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 7ms/step - loss: 0.3458 -
accuracy: 0.9131 - val_loss: 0.3719 - val_accuracy: 0.9054
Epoch 25/30
483/487 [============================>.] - ETA: 0s - loss: 0.3395 - accuracy:
0.9146
Epoch 00025: val_loss improved from 0.37189 to 0.36655, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 7ms/step - loss: 0.3389 -
accuracy: 0.9148 - val_loss: 0.3666 - val_accuracy: 0.9086
Epoch 26/30
487/487 [==============================] - ETA: 0s - loss: 0.3357 - accuracy:
```

```
0.9151
Epoch 00026: val_loss improved from 0.36655 to 0.36263, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 7ms/step - loss: 0.3357 -
accuracy: 0.9151 - val_loss: 0.3626 - val_accuracy: 0.9092
Epoch 27/30
474/487 [============================>.] - ETA: 0s - loss: 0.3296 - accuracy:
0.9176
Epoch 00027: val_loss improved from 0.36263 to 0.36020, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 7ms/step - loss: 0.3296 -
accuracy: 0.9176 - val_loss: 0.3602 - val_accuracy: 0.9076
Epoch 28/30
481/487 [============================>.] - ETA: 0s - loss: 0.3246 - accuracy:
0.9186
Epoch 00028: val_loss improved from 0.36020 to 0.35781, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 7ms/step - loss: 0.3247 -
accuracy: 0.9185 - val_loss: 0.3578 - val_accuracy: 0.9098
Epoch 29/30
485/487 [============================>.] - ETA: 0s - loss: 0.3194 - accuracy:
0.9200
Epoch 00029: val_loss improved from 0.35781 to 0.35492, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.3191 -
accuracy: 0.9201 - val_loss: 0.3549 - val_accuracy: 0.9117
Epoch 30/30
474/487 [============================>.] - ETA: 0s - loss: 0.3152 - accuracy:
0.9214
Epoch 00030: val_loss improved from 0.35492 to 0.35248, saving model to
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best
INFO:tensorflow:Assets written to:
/home/marcin/Documents/Capstone_project/cnn_checkpoint_best/assets
487/487 [==============================] - 3s 6ms/step - loss: 0.3154 -
accuracy: 0.9214 - val_loss: 0.3525 - val_accuracy: 0.9124
```

Graphs

[45]:
```
df = pd.DataFrame(
    {'val_loss': history.history['val_loss'],
```

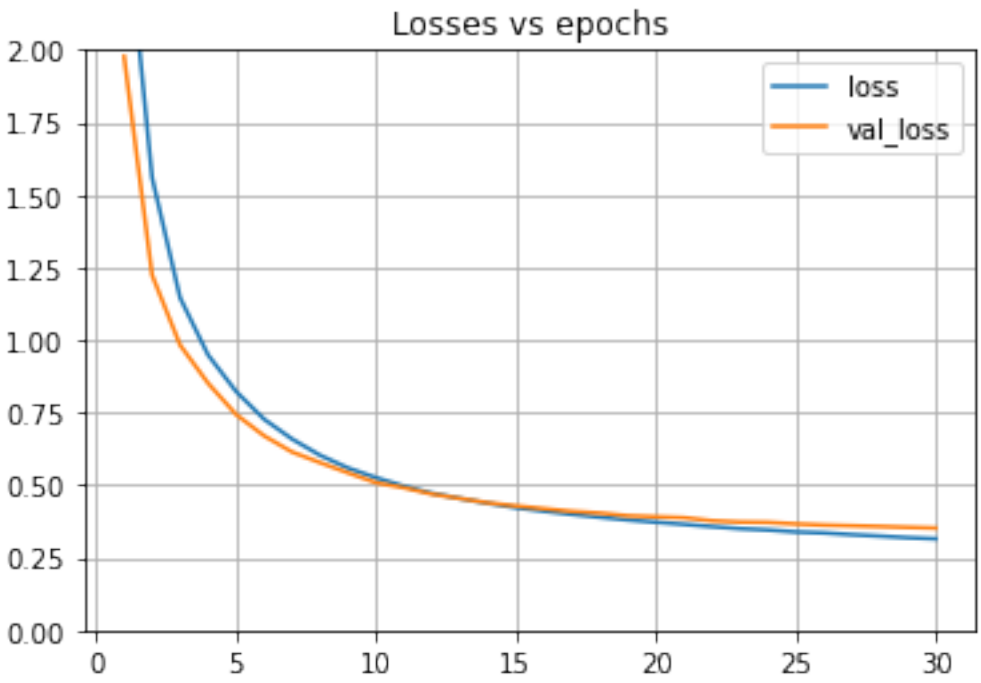
```
     'loss': history.history['loss'],
     'accuracy': history.history['accuracy'],
     'val_accuracy': history.history['val_accuracy']},
    index=range(1, 30 + 1),
    )

df.head()
```

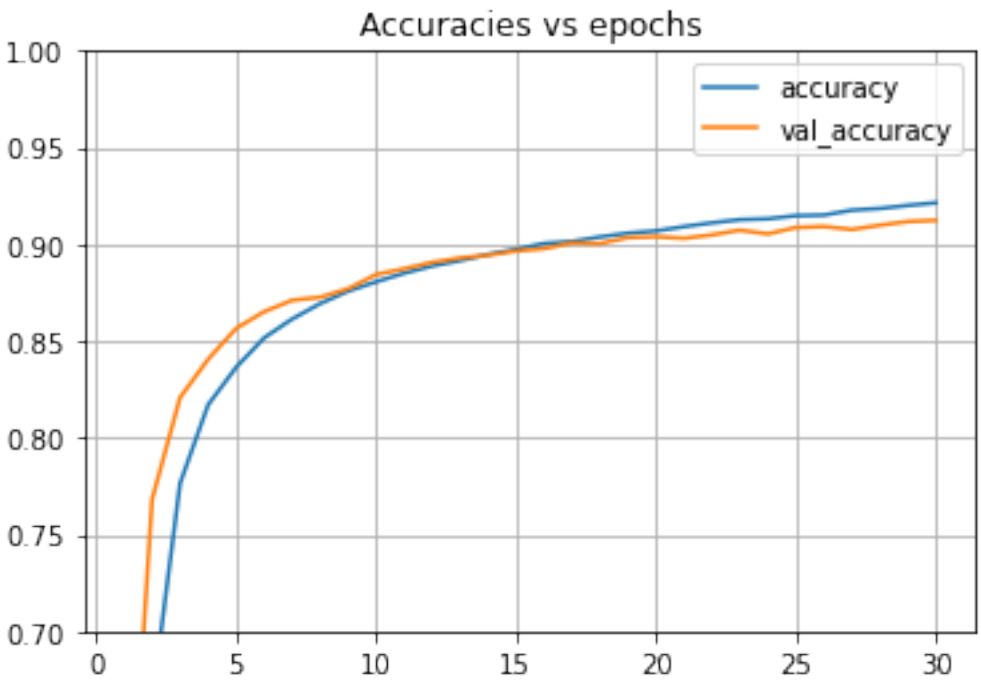
[45]:

	val_loss	loss	accuracy	val_accuracy
1	1.976572	2.538208	0.343579	0.541542
2	1.223801	1.559700	0.663888	0.768223
3	0.982709	1.145626	0.776482	0.820821
4	0.851361	0.948722	0.817017	0.840568
5	0.743621	0.823024	0.836385	0.856493

[46]:
```
df[['loss', 'val_loss']].plot(grid=True, ylim=(0, 2), title='Losses vs epochs')
df[['accuracy', 'val_accuracy']].plot(grid=True, ylim=(0.7, 1),␣
↪title='Accuracies vs epochs')
```

[46]: <AxesSubplot:title={'center':'Accuracies vs epochs'}>





Model evaluation on test set

[47]:
```
test_loss, test_accuracy = model.evaluate(test_data_grayscale, test_targets)
print('Test loss = {:.03f}'.format(test_loss))
print('Test accuracy = {:.03f}'.format(test_accuracy))
```

```
814/814 [==============================] - 4s 5ms/step - loss: 0.3840 -
accuracy: 0.9009
Test loss = 0.384
Test accuracy = 0.901
```

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.

[48]:
```
! ls -lh /home/marcin/Documents/Capstone_project
```

```
total 8,0K
drwxr-xr-x 4 marcin marcin 4,0K sty 10 11:23 cnn_checkpoint_best
drwxr-xr-x 4 marcin marcin 4,0K sty 10 11:20 mlp_checkpoint_best
```

[49]:
```
test_data_raw, test_targets_raw = test['X'], test['y']
```

Predictions for MLP model

[50]:
```
from tensorflow.keras.models import load_model
```

[51]:
```
best_mlp_model = load_model('/home/marcin/Documents/Capstone_project/
↪mlp_checkpoint_best')
```

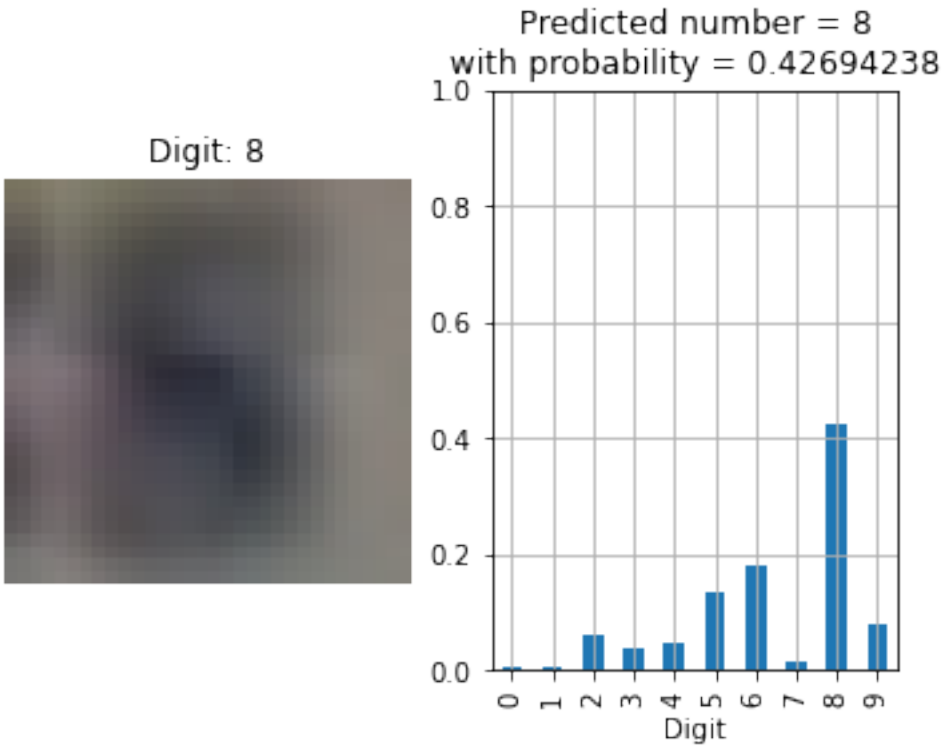
[52]:
```
image_indexes = [32, 435, 4533, 7567, 25543]

df_probabilities = pd.DataFrame({}, index=range(0, 10))

def get_prediction_mlp(id):
  test_img = test_data_grayscale[id, ...]
  preds = best_mlp_model.predict(test_img[np.newaxis, ...])
  return preds

for i in range(len(image_indexes)):
  df_probabilities.insert(i, str(image_indexes[i]),
                          get_prediction_mlp(image_indexes[i])[0])

df_probabilities
```

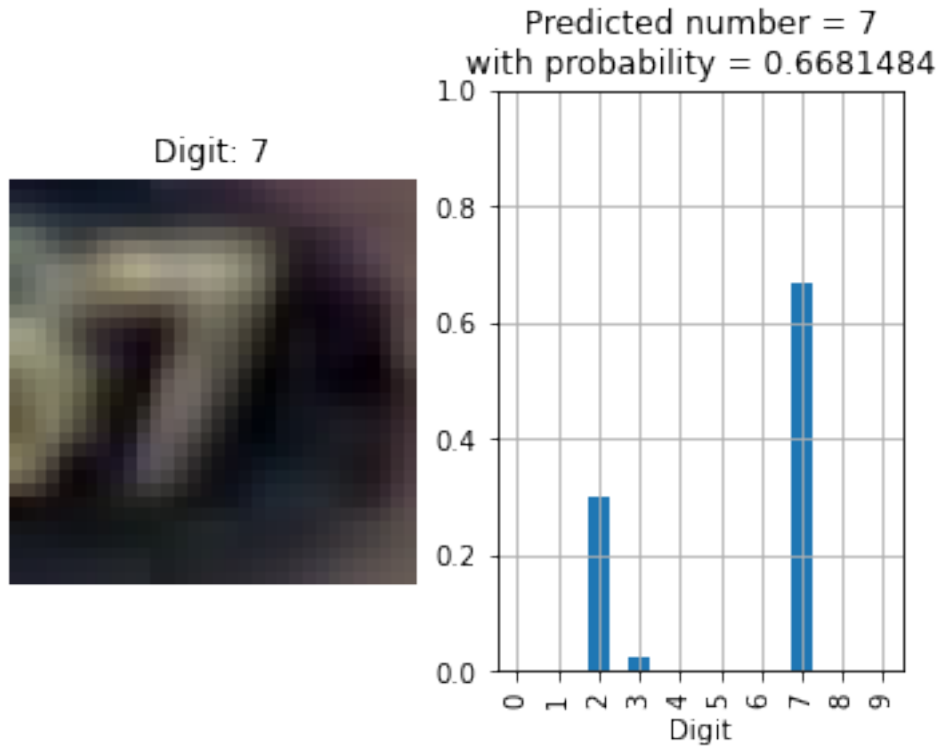
[52]:

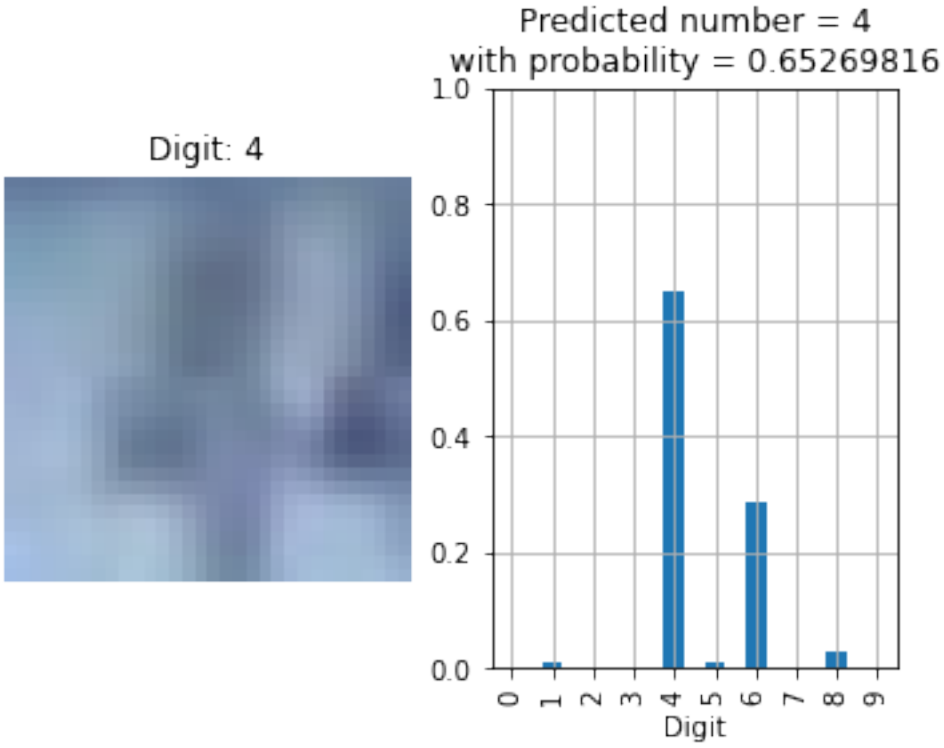
	32	435	4533	7567	25543
0	0.006686	0.000012	0.000352	1.538472e-10	1.138494e-16
1	0.006602	0.002809	0.011463	3.752055e-09	2.440249e-13
2	0.063725	0.300409	0.001956	2.887760e-11	1.397905e-09
3	0.038065	0.026662	0.002078	7.451017e-05	9.999996e-01
4	0.046457	0.000019	0.652698	4.119255e-09	1.047568e-09
5	0.133722	0.000016	0.013237	9.992267e-01	1.965999e-07
6	0.182703	0.000002	0.285654	6.831470e-04	2.959821e-14
7	0.014488	0.668148	0.000160	1.313782e-11	3.986903e-14
8	0.426942	0.000348	0.029940	3.176923e-06	1.319038e-07
9	0.080610	0.001574	0.002463	1.243213e-05	2.300961e-09

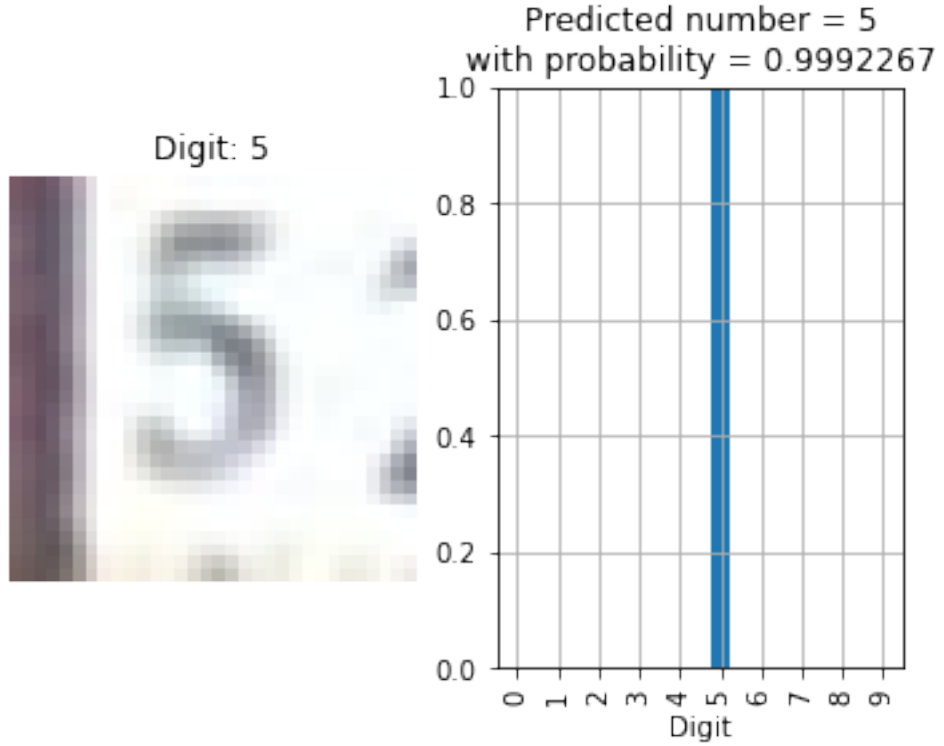
[53]:
```
for i in range(5):
  fig, (ax, ax2) = plt.subplots(ncols=2)

  ax.imshow(test_data_raw[..., image_indexes[i]])
  ax.set_title('Digit: ' + str(test_targets_raw[image_indexes[i]][0]))
  ax.set_axis_off()

  max_probability = 'Predicted number = '\
  + str(df_probabilities[str(image_indexes[i])].argmax()) \
  +'\nwith probability = ' + str(df_probabilities[str(image_indexes[i])].max())
  ax2 = df_probabilities[str(image_indexes[i])].plot.bar(
```

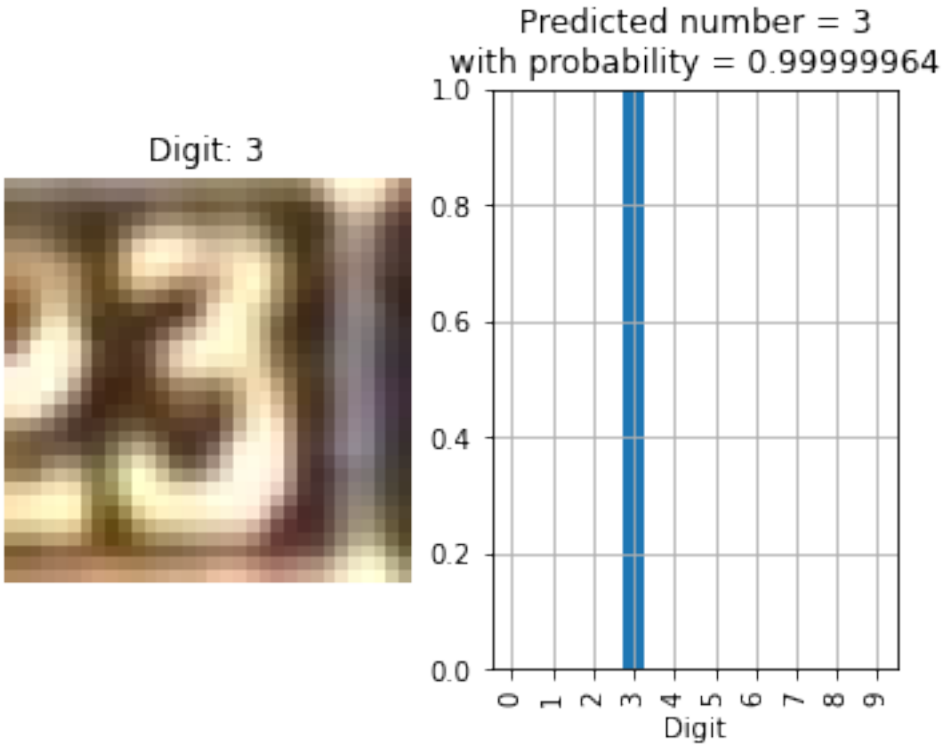
```
    xlabel='Digit', title=max_probability , ylim=(0,1),
    grid=True, legend=False)
```




Digit: 7
Predicted number = 7
with probability = 0.6681484
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
Digit


Digit: 4
Predicted number = 4
with probability = 0.65269816
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
Digit


Digit: 5
Predicted number = 5
with probability = 0.9992267
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
Digit


Digit: 3
Predicted number = 3
with probability = 0.99999964
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
Digit

Predictions for CNN model

[54]:
```
best_cnn_model = load_model('/home/marcin/Documents/Capstone_project/
↪cnn_checkpoint_best')
```

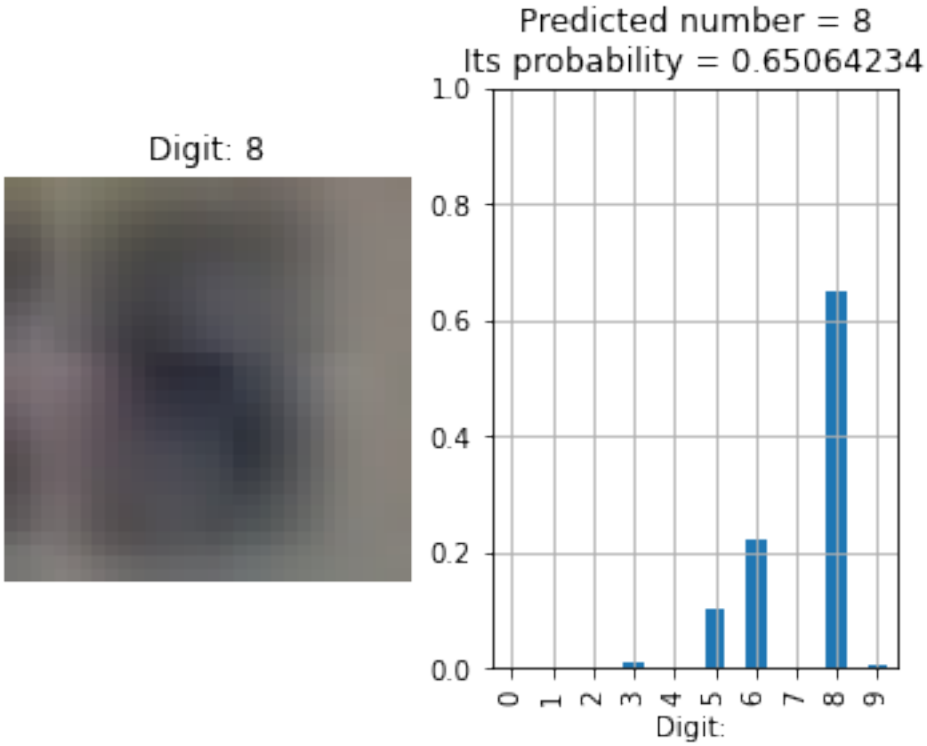
[55]:
```
image_indexes = [32, 435, 4533, 7567, 25543]

df_probabilities = pd.DataFrame({}, index=range(0, 10))

def get_prediction_cnn(id):
  test_img = test_data_grayscale[id, ...]
  preds = best_cnn_model.predict(test_img[np.newaxis, ...])
  return preds

for i in range(len(image_indexes)):
  df_probabilities.insert(i, str(image_indexes[i]),
                          get_prediction_cnn(image_indexes[i])[0])

df_probabilities
```

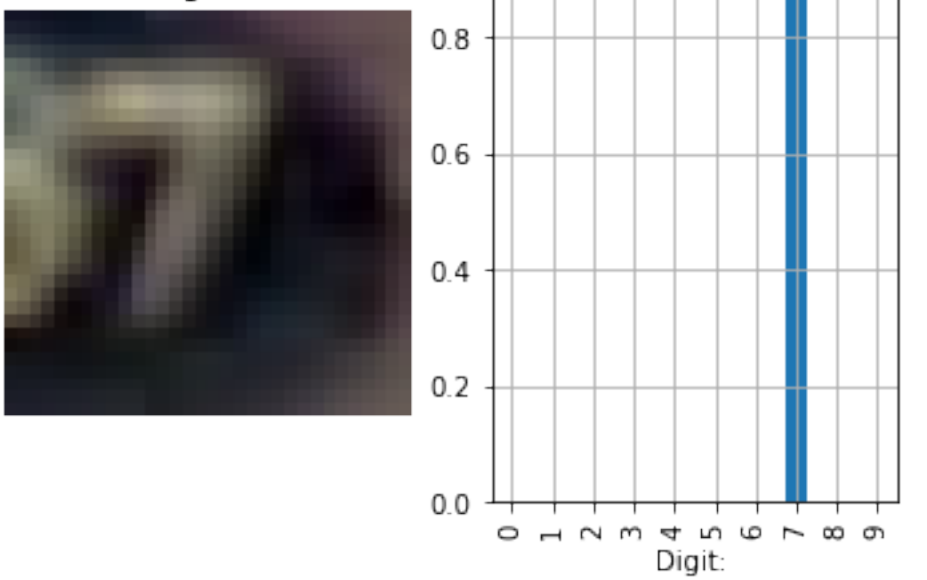
[55]:

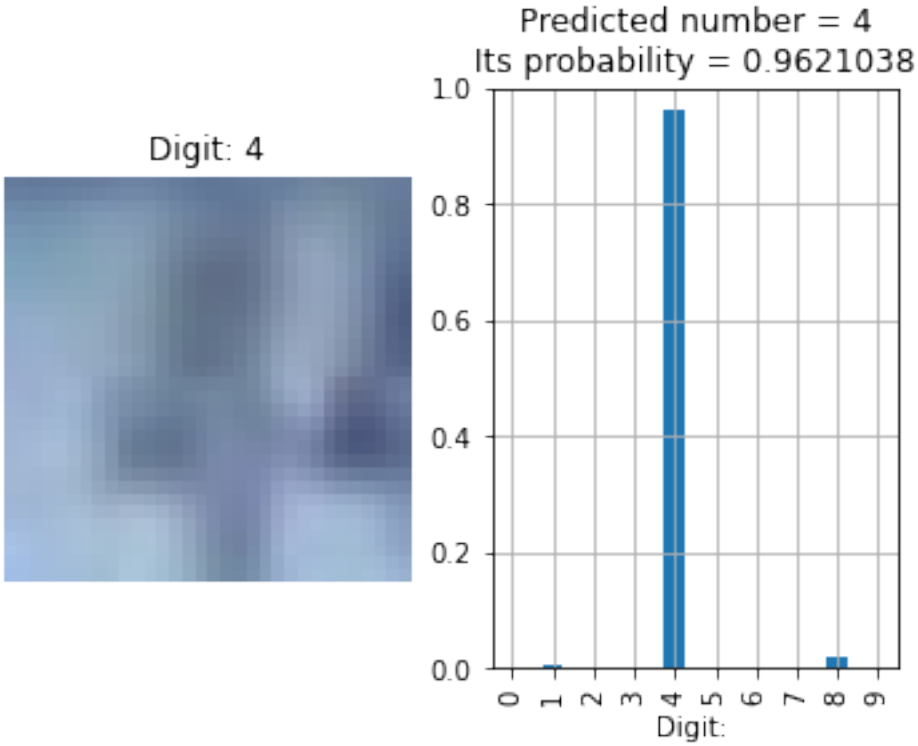
	32	435	4533	7567	25543
0	0.000368	3.915983e-06	0.000208	6.201173e-07	6.771975e-08
1	0.000573	1.220845e-04	0.006696	1.535265e-06	3.202008e-06
2	0.002299	1.633808e-05	0.002963	3.694105e-06	8.601296e-05
3	0.012669	2.897376e-06	0.002379	4.570804e-04	9.998178e-01
4	0.004001	8.337068e-07	0.962104	7.016377e-07	1.898987e-06
5	0.103324	6.797343e-07	0.002269	9.994210e-01	3.545312e-05
6	0.220922	1.617369e-05	0.001846	7.265915e-05	1.003812e-07
7	0.000276	9.998360e-01	0.000261	8.960946e-07	1.051866e-06
8	0.650642	2.127282e-07	0.020448	2.868987e-05	2.724594e-05
9	0.004924	7.807609e-07	0.000826	1.310900e-05	2.725213e-05

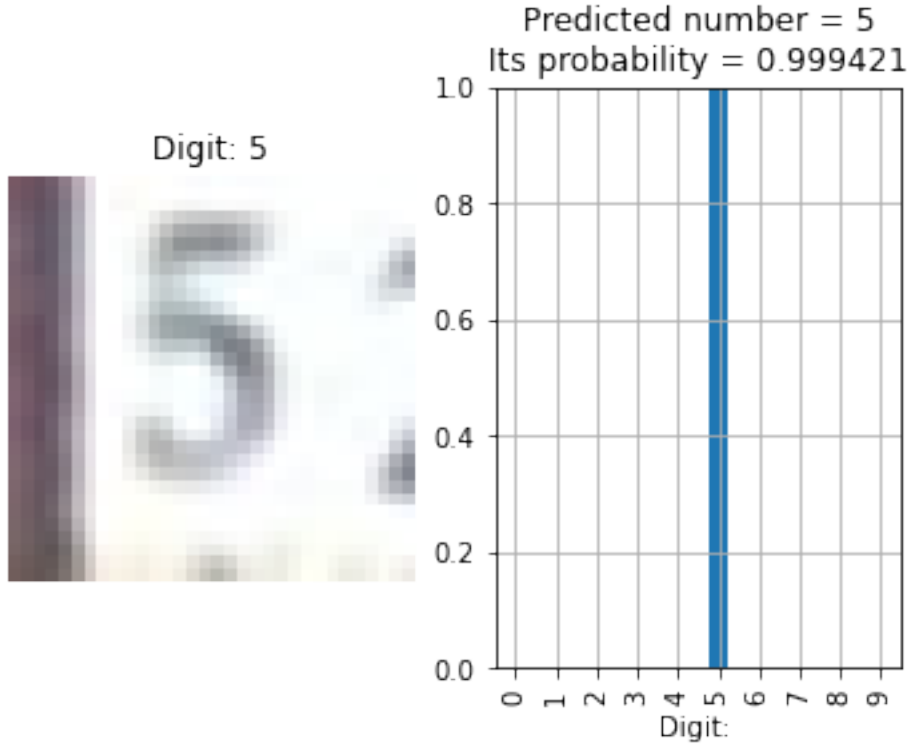
[56]:
```
for i in range(5):
  fig, (ax, ax2) = plt.subplots(ncols=2)

  ax.imshow(test_data_raw[..., image_indexes[i]])
  ax.set_title('Digit: ' + str(test_targets_raw[image_indexes[i]][0]))
  ax.set_axis_off()

  max_probability = 'Predicted number = '\
  + str(df_probabilities[str(image_indexes[i])].argmax()) \
  +'\nIts probability = ' + str(df_probabilities[str(image_indexes[i])].max())

  ax2 = df_probabilities[str(image_indexes[i])].plot.bar(
      xlabel='Digit: ', title=max_probability , ylim=(0,1),
```

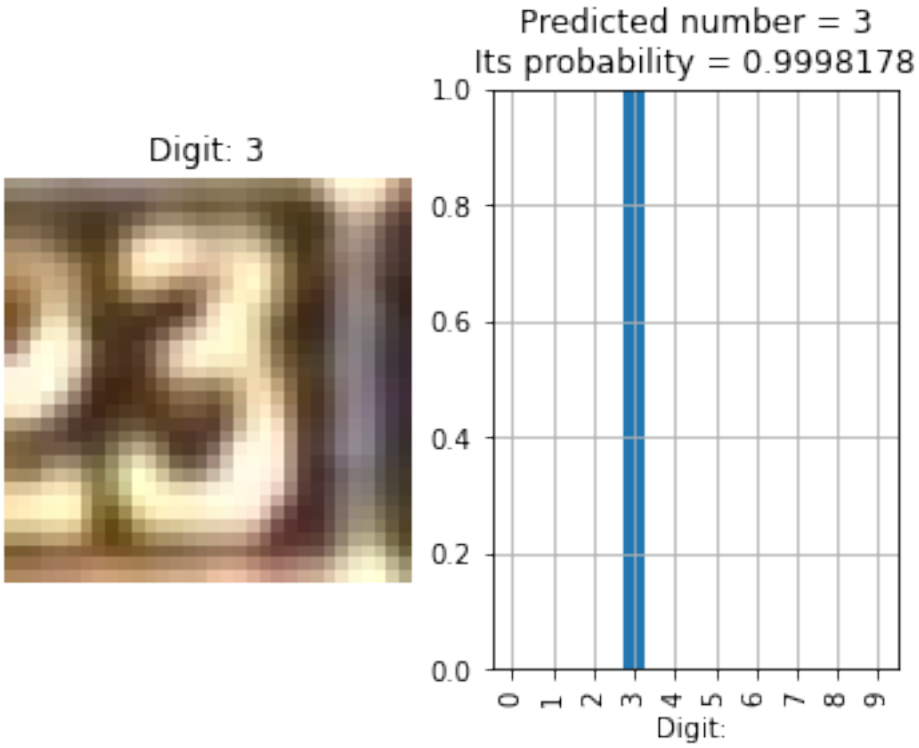
```
    grid=True, legend=False)
```




Digit: 7
Predicted number = 7
Its probability = 0.99983597
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
Digit:


Digit: 4
Predicted number = 4
Its probability = 0.9621038
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
Digit:


Digit: 5
Predicted number = 5
Its probability = 0.999421
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
Digit:


Digit: 3
Predicted number = 3
Its probability = 0.9998178
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
Digit: