Capstone Project

May 12, 2020

1 Capstone Project

1.1 Image classifier for the SVHN dataset

1.1.1 Instructions

In this notebook, you will create a neural network that classifies real-world images digits. You will use concepts from throughout this course in building, training, testing, validating and saving your Tensorflow classifier model.

This project is peer-assessed. Within this notebook you will find instructions in each section for how to complete the project. Pay close attention to the instructions as the peer review will be carried out according to a grading rubric that checks key parts of the project instructions. Feel free to add extra cells into the notebook as required.

1.1.2 How to submit

When you have completed the Capstone project notebook, you will submit a pdf of the notebook for peer review. First ensure that the notebook has been fully executed from beginning to end, and all of the cell outputs are visible. This is important, as the grading rubric depends on the reviewer being able to view the outputs of your notebook. Save the notebook as a pdf (File -> Download as -> PDF via LaTeX). You should then submit this pdf for review.

1.1.3 Let's get started!

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

```
In [3]: import tensorflow as tf
        from scipy.io import loadmat
        from matplotlib import pyplot as plt
        import numpy as np
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D, BatchNormalizati
        from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

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 [4]: # Run this cell to load the dataset

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

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

1.2 1. Inspect and preprocess the dataset

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

```
In [5]: x_train = train['X']
        y_train = train['y']
        x_test = test['X']
        y_test = test['y']
        print(x_train.shape)
```

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

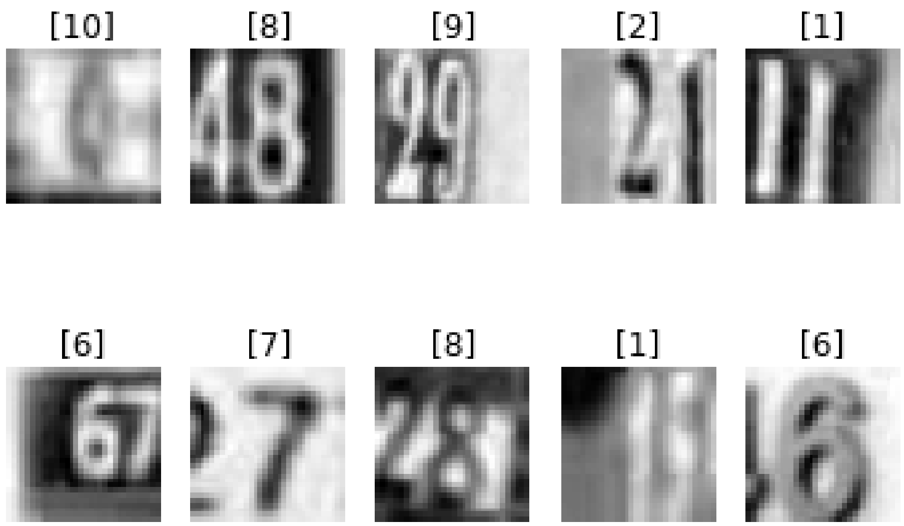
```
In [6]: def show_images(images, cols = 1, titles=[]):
            """Display a list of images in a single figure with matplotlib.

            Parameters
            ---------
            images: List of np.arrays compatible with plt.imshow.

            cols (Default = 1): Number of columns in figure (number of rows is
                                set to np.ceil(n_images/float(cols))).
            titles
            """
            n_images = len(images)
            fig = plt.figure()
            for n, image in enumerate(images):
                a = fig.add_subplot(cols, np.ceil(n_images/float(cols)), n + 1)
                if image.shape[-1] == 1:
                    plt.gray()
                    plt.imshow(image[...,-1])
                else:
                    plt.imshow(image)
                a.set_title(titles[n])
                plt.axis('off')
            plt.show()
```

```
In [7]: def show_train_set(n=10):
            samples_number = np.random.choice(x_train.shape[-1], n)
            show_images([x_train[...,sample] for sample in samples_number], cols=2, titles=[y_t
```

```
In [8]: n = 10
        show_train_set(n)
```

In [9]:
```
x_train = np.expand_dims(np.mean(x_train, axis=-2), axis=-2) / 255.
x_test = np.expand_dims(np.mean(x_test, axis=-2), axis=-2) / 255.
```

In [10]:
```
show_train_set(n)
```



In [11]:
```
x_train = np.moveaxis(x_train, -1, 0)
x_test = np.moveaxis(x_test, -1, 0)
print(x_train.shape)
print(x_test.shape)
```

```
(73257, 32, 32, 1)
(26032, 32, 32, 1)
```

```
In [12]: y_train = y_train - 1
         y_test = y_test - 1
```

1.3 2. MLP neural network classifier

- Build an MLP classifier model using the Sequential API. Your model should use only Flatten and Dense layers, with the final layer having a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different MLP architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 4 or 5 layers.*
- Print out the model summary (using the summary() method)
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- As a guide, you should aim to achieve a final categorical cross entropy training loss of less than 1.0 (the validation loss might be higher).
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

```
In [15]: def build_mlp(input_shape):
             mlp = Sequential([
                 InputLayer(input_shape=input_shape),
                 Flatten(),
                 Dense(64, activation='relu'),
                 Dense(128, activation='relu'),
                 Dense(128, activation='relu'),
                 Dense(10, activation='softmax')
             ])
             mlp.compile(optimizer=tf.keras.optimizers.Adam(0.0001), loss='sparse_categorical_c
             return mlp
```

```
In [20]: mlp = build_mlp(x_train[0].shape)
         print(mlp.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                65600
_________________________________________________________________
dense_9 (Dense)              (None, 128)               8320
_________________________________________________________________
```

```
dense_10 (Dense)             (None, 128)              16512
_________________________________________________________________
dense_11 (Dense)             (None, 10)               1290
=================================================================
Total params: 91,722
Trainable params: 91,722
Non-trainable params: 0
_________________________________________________________________
None
```

```
In [21]: callbacks = [ModelCheckpoint('mlp/checkpoint', save_best_only=True, save_weights_only=
                      EarlyStopping(monitor='val_loss', patience=3)]
```

```
In [23]: history_mlp = mlp.fit(x_train, y_train, epochs=30, batch_size=64, validation_split=0.
```

```
Train on 62268 samples, validate on 10989 samples
Epoch 1/30
WARNING:tensorflow:From /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/ops/resou
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 16s - loss: 2.1893 - accuracy: 0.2135 - val_loss: 2.0332 - val_accuracy: 0.2988
Epoch 2/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 1.7943 - accuracy: 0.3796 - val_loss: 1.5969 - val_accuracy: 0.4625
Epoch 3/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 1.4977 - accuracy: 0.5081 - val_loss: 1.4606 - val_accuracy: 0.5197
Epoch 4/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 1.3491 - accuracy: 0.5752 - val_loss: 1.2925 - val_accuracy: 0.5957
Epoch 5/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 15s - loss: 1.2451 - accuracy: 0.6133 - val_loss: 1.2162 - val_accuracy: 0.6221
Epoch 6/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 1.1747 - accuracy: 0.6365 - val_loss: 1.1639 - val_accuracy: 0.6339
Epoch 7/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 1.1221 - accuracy: 0.6563 - val_loss: 1.1230 - val_accuracy: 0.6578
Epoch 8/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 1.0821 - accuracy: 0.6690 - val_loss: 1.0721 - val_accuracy: 0.6781
Epoch 9/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 15s - loss: 1.0464 - accuracy: 0.6814 - val_loss: 1.0560 - val_accuracy: 0.6780
Epoch 10/30
```

```
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 1.0171 - accuracy: 0.6916 - val_loss: 1.0268 - val_accuracy: 0.6793
Epoch 11/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.9914 - accuracy: 0.6992 - val_loss: 0.9952 - val_accuracy: 0.6960
Epoch 12/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.9735 - accuracy: 0.7032 - val_loss: 0.9835 - val_accuracy: 0.6972
Epoch 13/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.9501 - accuracy: 0.7125 - val_loss: 0.9585 - val_accuracy: 0.7052
Epoch 14/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.9305 - accuracy: 0.7170 - val_loss: 0.9393 - val_accuracy: 0.7133
Epoch 15/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.9132 - accuracy: 0.7227 - val_loss: 0.9329 - val_accuracy: 0.7154
Epoch 16/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.8974 - accuracy: 0.7285 - val_loss: 0.9135 - val_accuracy: 0.7175
Epoch 17/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.8815 - accuracy: 0.7339 - val_loss: 0.8898 - val_accuracy: 0.7290
Epoch 18/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.8681 - accuracy: 0.7381 - val_loss: 0.8861 - val_accuracy: 0.7292
Epoch 19/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.8557 - accuracy: 0.7403 - val_loss: 0.8702 - val_accuracy: 0.7333
Epoch 20/30
62268/62268 - 13s - loss: 0.8409 - accuracy: 0.7452 - val_loss: 0.8742 - val_accuracy: 0.7305
Epoch 21/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.8297 - accuracy: 0.7496 - val_loss: 0.8519 - val_accuracy: 0.7431
Epoch 22/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.8163 - accuracy: 0.7521 - val_loss: 0.8414 - val_accuracy: 0.7453
Epoch 23/30
62268/62268 - 13s - loss: 0.8076 - accuracy: 0.7547 - val_loss: 0.8581 - val_accuracy: 0.7415
Epoch 24/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.7964 - accuracy: 0.7593 - val_loss: 0.8291 - val_accuracy: 0.7461
Epoch 25/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 15s - loss: 0.7850 - accuracy: 0.7617 - val_loss: 0.8197 - val_accuracy: 0.7505
Epoch 26/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.7747 - accuracy: 0.7660 - val_loss: 0.8179 - val_accuracy: 0.7521
```

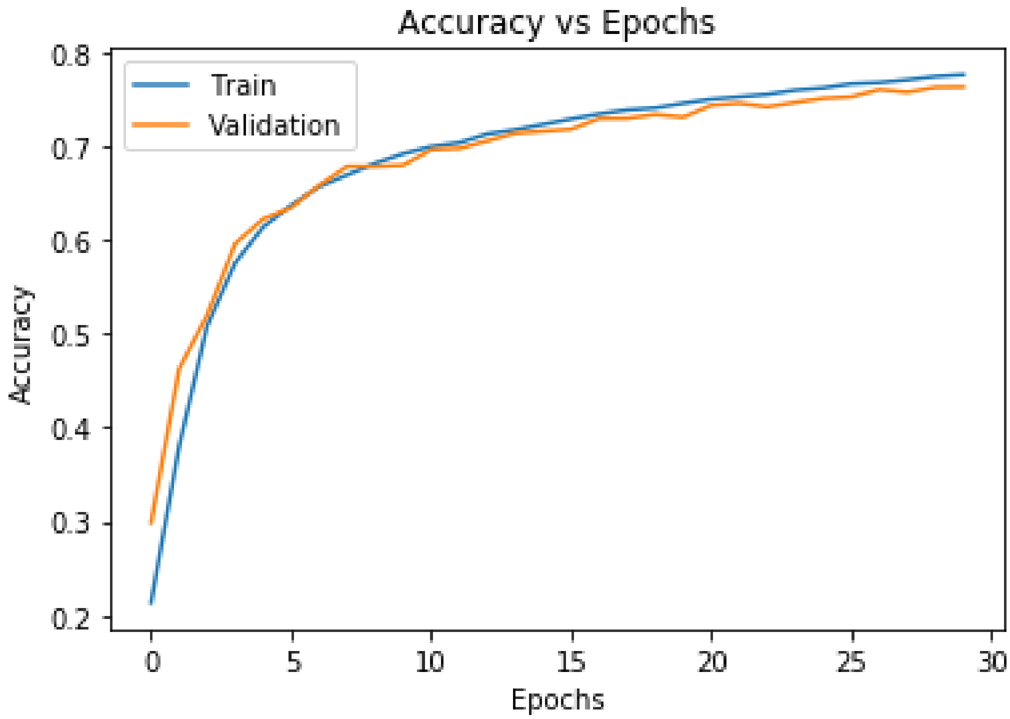
```
Epoch 27/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.7683 - accuracy: 0.7673 - val_loss: 0.7962 - val_accuracy: 0.7599
Epoch 28/30
62268/62268 - 13s - loss: 0.7603 - accuracy: 0.7703 - val_loss: 0.8092 - val_accuracy: 0.7571
Epoch 29/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 14s - loss: 0.7503 - accuracy: 0.7738 - val_loss: 0.7896 - val_accuracy: 0.7622
Epoch 30/30
INFO:tensorflow:Assets written to: mlp/checkpoint/assets
62268/62268 - 15s - loss: 0.7435 - accuracy: 0.7758 - val_loss: 0.7855 - val_accuracy: 0.7625
```

In [11]:
```
! ls -lh mlp
```

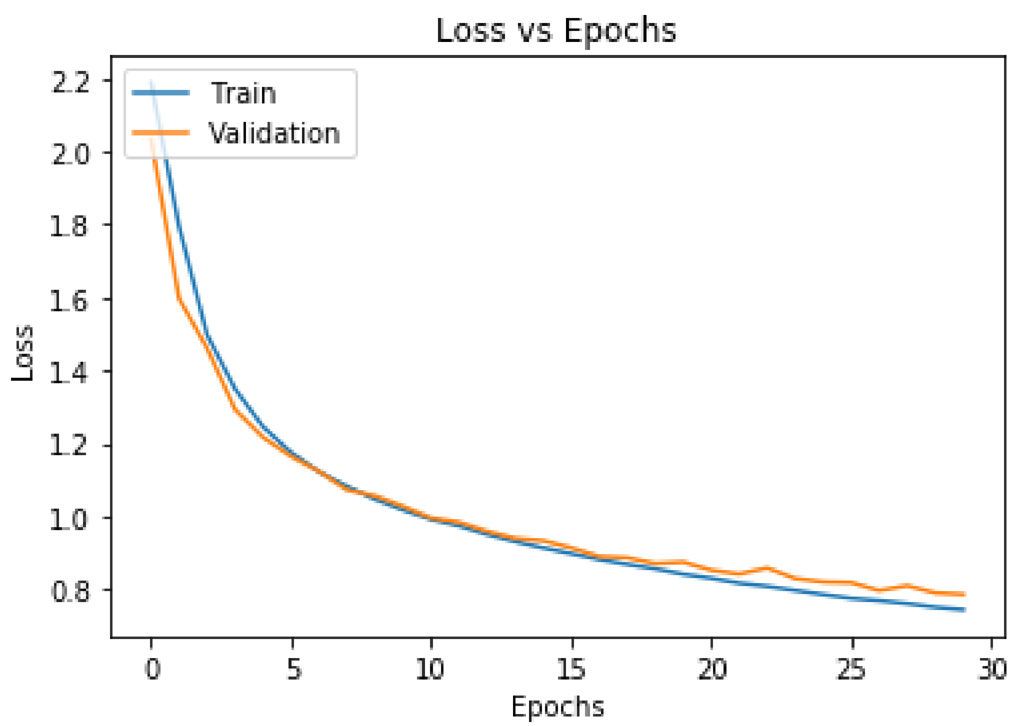
```
total 4.0K
drwxr-xr-x 4 jovyan users 6.0K May 12 06:49 checkpoint
```

In [17]:
```
def plot_metrics(train_metric, val_metric, title, x_label, y_label, titles=[], loc='u
    plt.plot(train_metric)
    plt.plot(val_metric)
    plt.title(title)
    plt.ylabel(y_label)
    plt.xlabel(x_label)
    plt.legend(titles, loc=loc)
    plt.show()
```

In [34]:
```
plot_metrics(history_mlp.history['accuracy'], history_mlp.history['val_accuracy'], 'A
             'Epochs', 'Accuracy', ['Train', 'Validation'])
plot_metrics(history_mlp.history['loss'], history_mlp.history['val_loss'], 'Loss vs E
             'Epochs', 'Loss', ['Train', 'Validation'])
```



Accuracy vs Epochs
Train
Validation
Accuracy
Epochs
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30



Loss vs Epochs
Train
Validation
Loss
Epochs
0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
0
5
10
15
20
25
30

```
In [36]: test_results = mlp.evaluate(x_test, y_test, verbose=2)
         print(test_results)
```

```
26032/1 - 4s - loss: 0.6813 - accuracy: 0.7434
[0.8715997453694264, 0.74339277]
```

1.4 3. CNN neural network classifier

- Build a CNN classifier model using the Sequential API. Your model should use the Conv2D, MaxPool2D, BatchNormalization, Flatten, Dense and Dropout layers. The final layer should again have a 10-way softmax output.
- You should design and build the model yourself. Feel free to experiment with different CNN architectures. *Hint: to achieve a reasonable accuracy you won't need to use more than 2 or 3 convolutional layers and 2 fully connected layers.)*
- The CNN model should use fewer trainable parameters than your MLP model.
- Compile and train the model (we recommend a maximum of 30 epochs), making use of both training and validation sets during the training run.
- Your model should track at least one appropriate metric, and use at least two callbacks during training, one of which should be a ModelCheckpoint callback.
- You should aim to beat the MLP model performance with fewer parameters!
- Plot the learning curves for loss vs epoch and accuracy vs epoch for both training and validation sets.
- Compute and display the loss and accuracy of the trained model on the test set.

```
In [12]: def get_cnn_model(input_shape):

             cnn = Sequential([
                 Conv2D(64, kernel_size=(3,3), padding='SAME', input_shape=input_shape, activa
                 MaxPool2D((4,4)),
                 Dropout(0.3),
                 Conv2D(32, kernel_size=(3,3), activation='relu'),
                 MaxPool2D((2,2)),
                 Flatten(),
                 BatchNormalization(),
                 Dense(64, activation='relu'),
                 Dense(10, activation='softmax')
             ])
             cnn.compile(optimizer=tf.keras.optimizers.Adam(0.001), loss='sparse_categorical_c
             return cnn
```

```
In [13]: cnn = get_cnn_model(x_train[0].shape)
         print(cnn.summary())
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        640
```

```
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 8, 64)          0
_________________________________________________________________
dropout (Dropout)            (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 6, 6, 32)          18464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 288)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 288)               1152
_________________________________________________________________
dense (Dense)                (None, 64)                18496
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 39,402
Trainable params: 38,826
Non-trainable params: 576
_________________________________________________________________
None
```

```
In [14]: callbacks = [ModelCheckpoint('cnn/checkpoint', save_best_only=True, save_weights_only=
                      EarlyStopping(monitor='val_loss', patience=3)]
         history_cnn = cnn.fit(x_train, y_train, epochs=7, batch_size=128, validation_split=0.
```

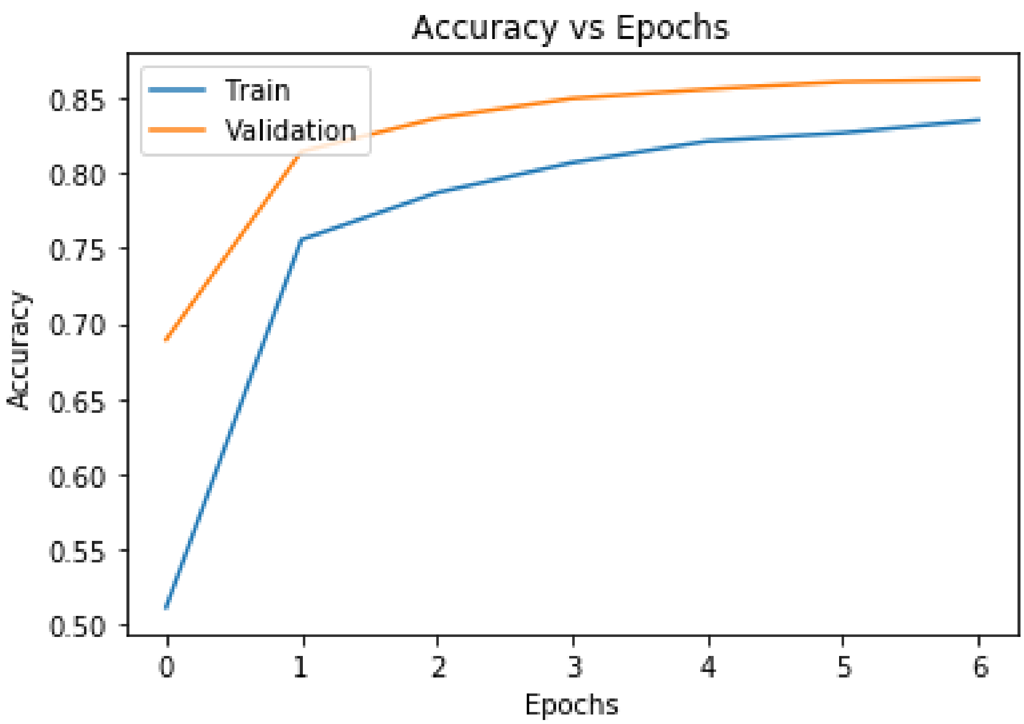
```
Train on 62268 samples, validate on 10989 samples
Epoch 1/7
WARNING:tensorflow:From /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/ops/resou
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: cnn/checkpoint/assets
62268/62268 - 242s - loss: 1.4306 - accuracy: 0.5109 - val_loss: 1.1046 - val_accuracy: 0.6893
Epoch 2/7
INFO:tensorflow:Assets written to: cnn/checkpoint/assets
62268/62268 - 238s - loss: 0.7668 - accuracy: 0.7559 - val_loss: 0.6005 - val_accuracy: 0.8146
Epoch 3/7
INFO:tensorflow:Assets written to: cnn/checkpoint/assets
62268/62268 - 237s - loss: 0.6733 - accuracy: 0.7871 - val_loss: 0.5397 - val_accuracy: 0.8369
Epoch 4/7
INFO:tensorflow:Assets written to: cnn/checkpoint/assets
62268/62268 - 237s - loss: 0.6101 - accuracy: 0.8072 - val_loss: 0.5015 - val_accuracy: 0.8500
Epoch 5/7
INFO:tensorflow:Assets written to: cnn/checkpoint/assets
62268/62268 - 227s - loss: 0.5676 - accuracy: 0.8215 - val_loss: 0.4837 - val_accuracy: 0.8561
Epoch 6/7
```

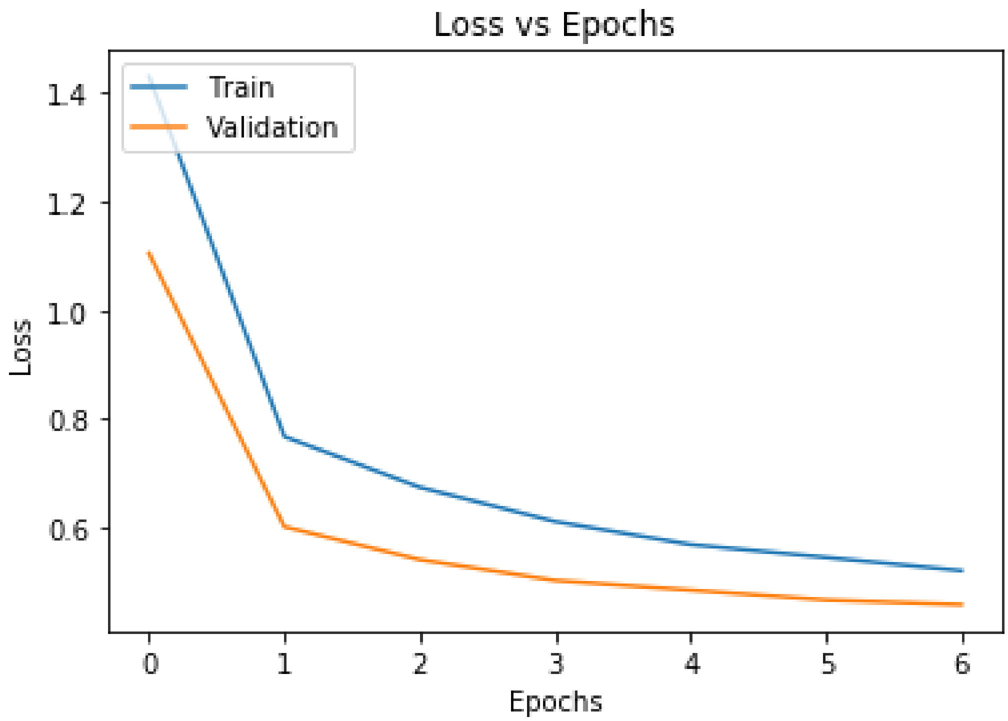
```
INFO:tensorflow:Assets written to: cnn/checkpoint/assets
62268/62268 - 223s - loss: 0.5439 - accuracy: 0.8271 - val_loss: 0.4656 - val_accuracy: 0.8610
Epoch 7/7
INFO:tensorflow:Assets written to: cnn/checkpoint/assets
62268/62268 - 223s - loss: 0.5194 - accuracy: 0.8356 - val_loss: 0.4570 - val_accuracy: 0.8625
```

In [15]:
```
!ls -lh cnn
```

```
total 4.0K
drwxr-xr-x 4 jovyan users 6.0K May 12 07:34 checkpoint
```

In [19]:
```
plot_metrics(history_cnn.history['accuracy'], history_cnn.history['val_accuracy'], 'Ac
             'Epochs', 'Accuracy', ['Train', 'Validation'])
plot_metrics(history_cnn.history['loss'], history_cnn.history['val_loss'], 'Loss vs Ep
             'Epochs', 'Loss', ['Train', 'Validation'])
```





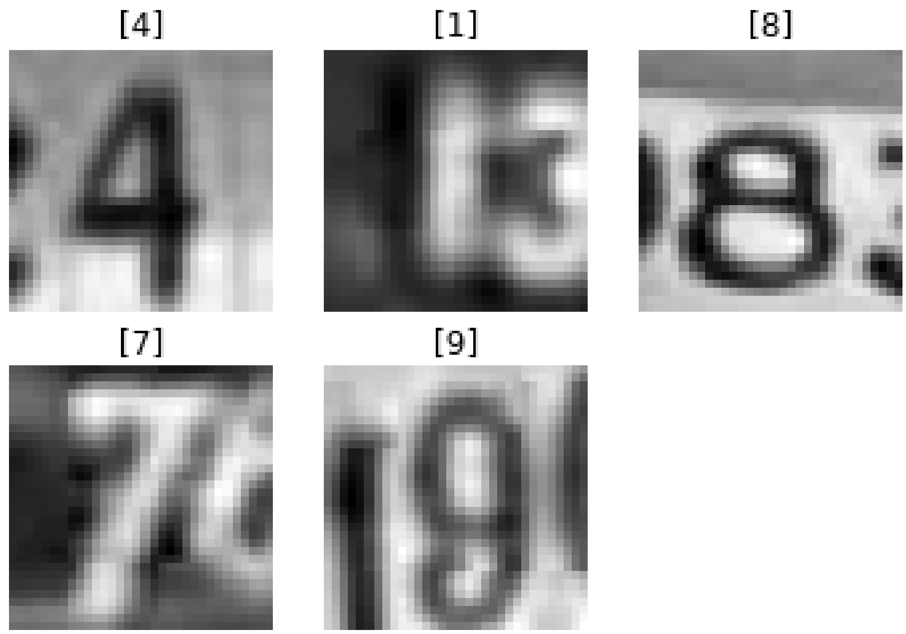

```
In [20]: test_results_cnn = cnn.evaluate(x_test, y_test, verbose=2)
         print(test_results_cnn)
```

```
26032/1 - 28s - loss: 0.5268 - accuracy: 0.8490
[0.4959425484668864, 0.849032]
```

1.5 4. Get model predictions

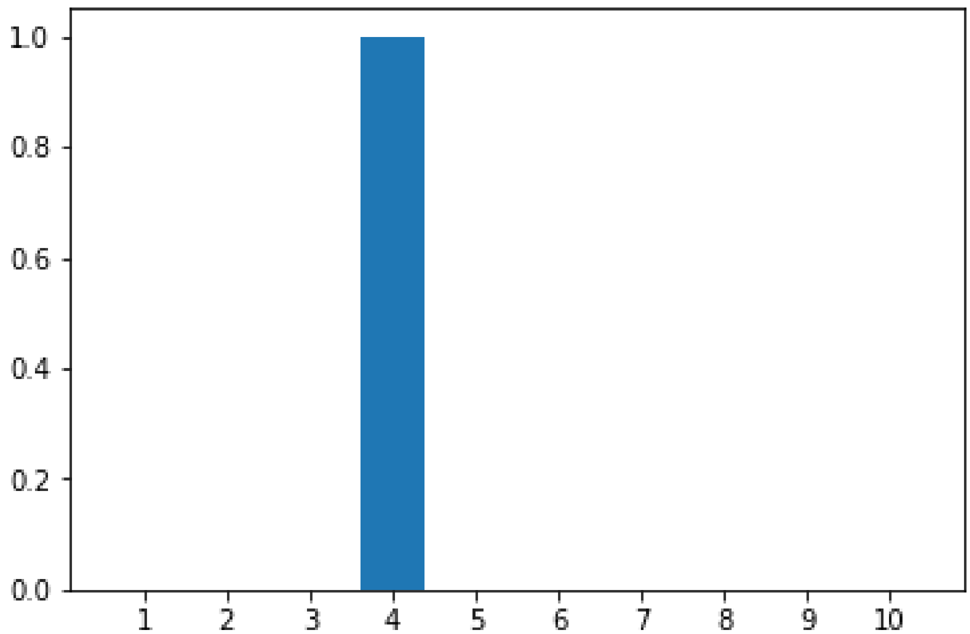
- Load the best weights for the MLP and CNN models that you saved during the training run.
- Randomly select 5 images and corresponding labels from the test set and display the images with their labels.
- Alongside the image and label, show each model's predictive distribution as a bar chart, and the final model prediction given by the label with maximum probability.

```
In [13]: samples_number = np.random.choice(x_test.shape[0], 5)
         show_images([x_test[sample] for sample in samples_number], cols=2, titles=[y_test[samp
```

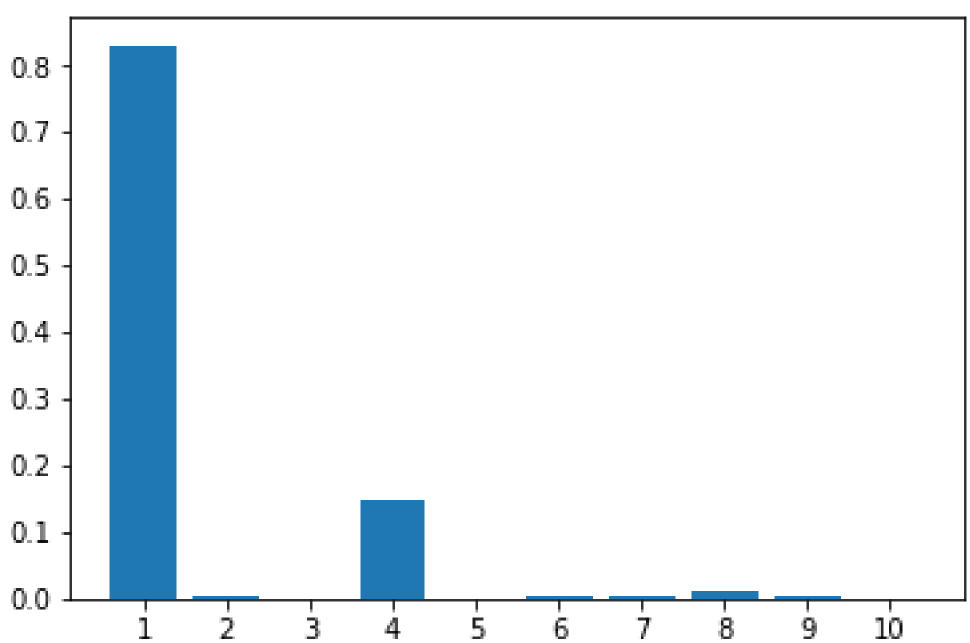



```
In [14]: mlp_model = tf.keras.models.load_model('mlp/checkpoint/')
         cnn_model = tf.keras.models.load_model('cnn/checkpoint/')

In [15]: def show_predictions(predictions):
             for idx, prediction in enumerate(predictions):
                 print("Prediction of the model: {0}".format(np.argmax(prediction) + 1))
                 tick_labels = list(range(1, len(prediction) + 1))
                 plt.bar(tick_labels, prediction, tick_label=tick_labels)
                 plt.show()
         predictions_mlp = mlp_model.predict(np.take(x_test, samples_number, axis=0))
         show_predictions(predictions_mlp)
```

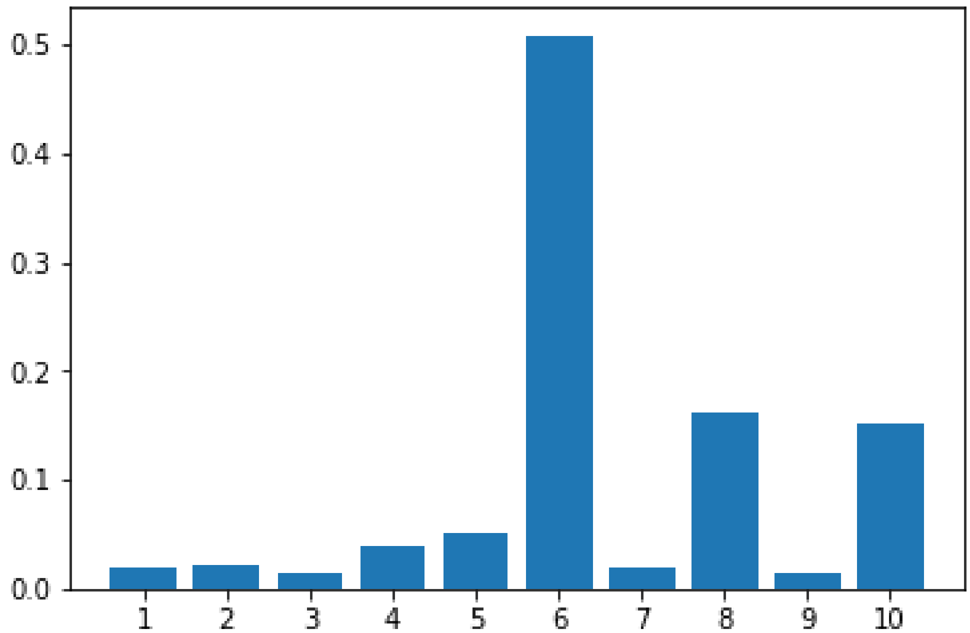
```
Prediction of the model: 4
```



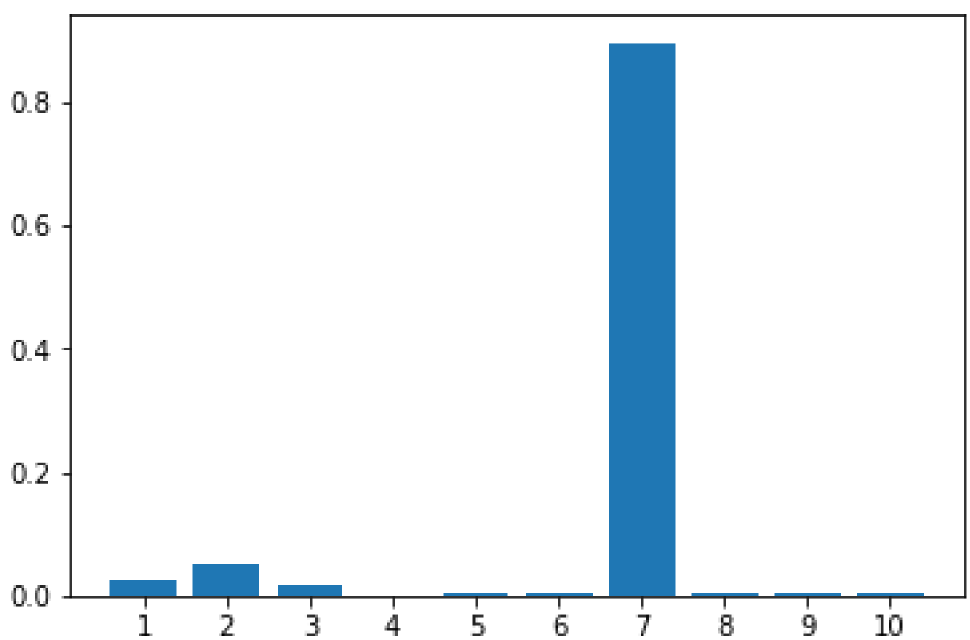

```
Prediction of the model: 1
```



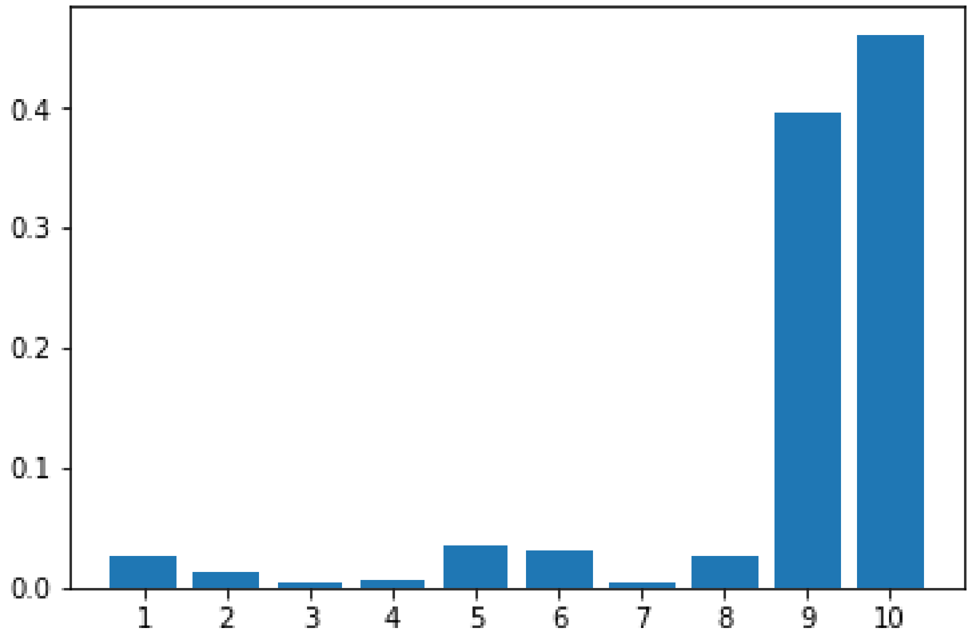
```
Prediction of the model: 6
```




```
Prediction of the model: 7
```

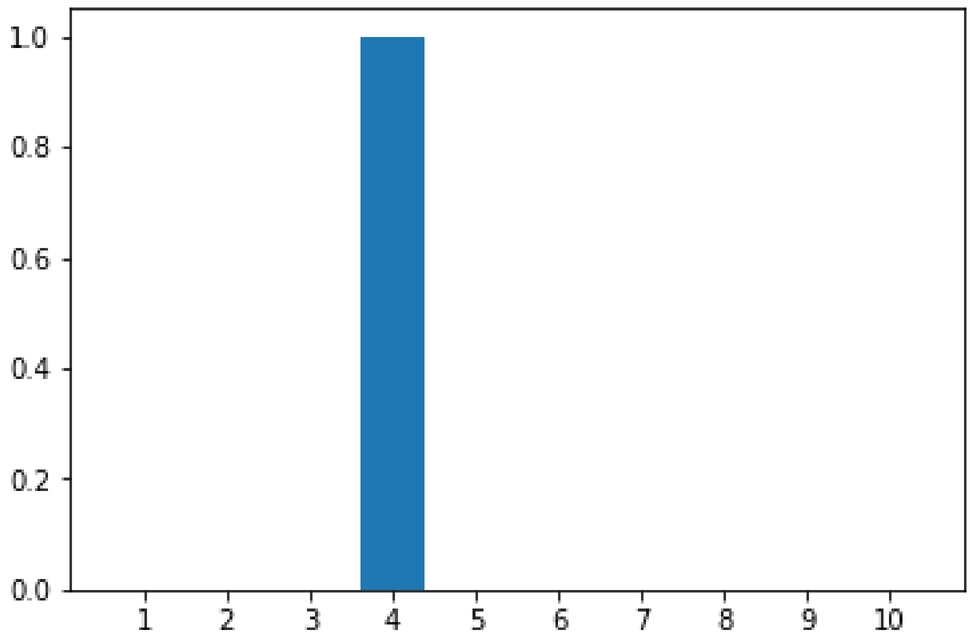


```
Prediction of the model: 10
```

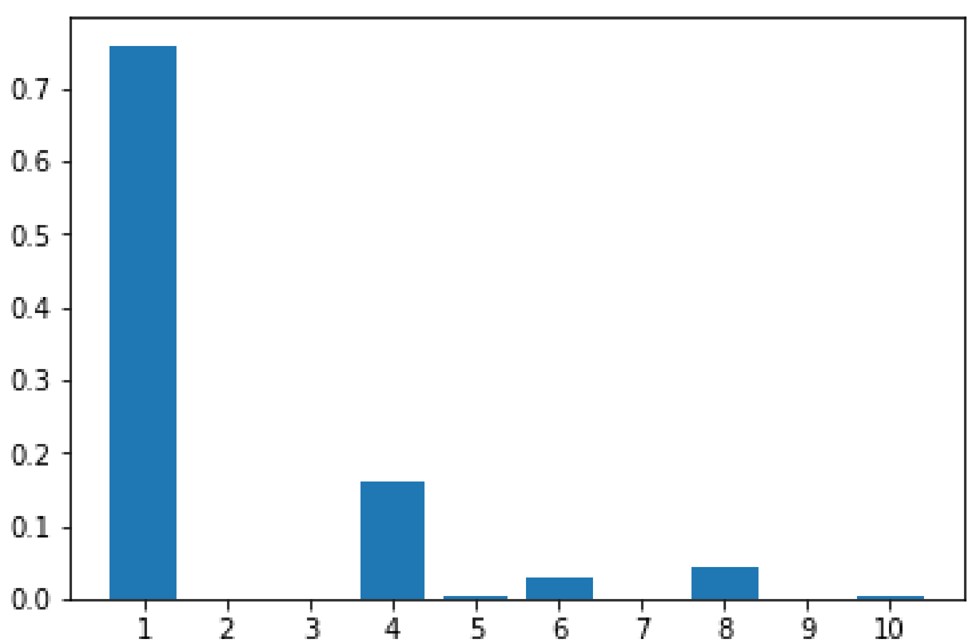



```
In [16]: predictions_cnn = cnn_model.predict(np.take(x_test, samples_number, axis=0))
         show_predictions(predictions_cnn)
```

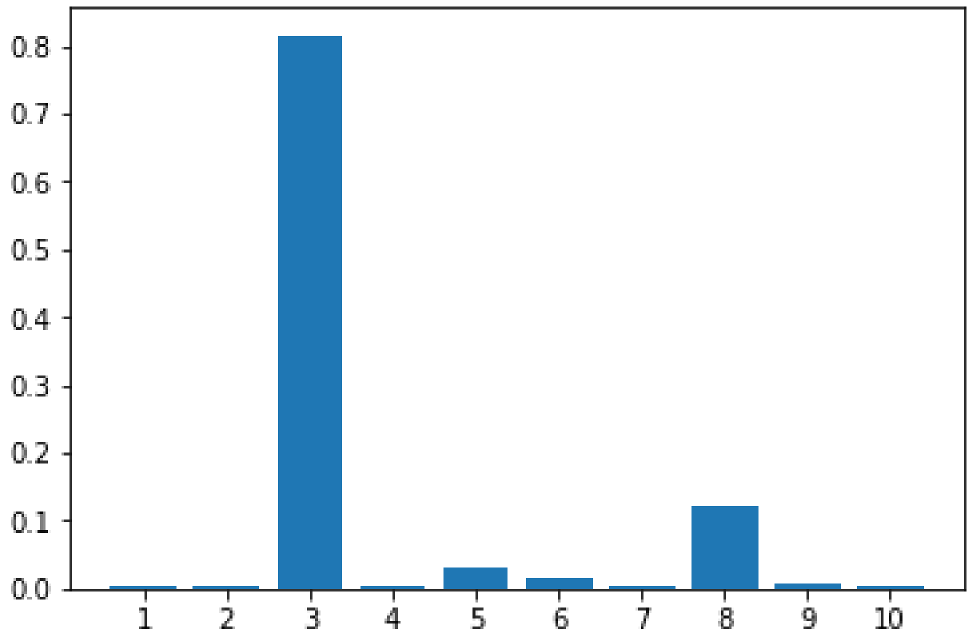
```
Prediction of the model: 4
```



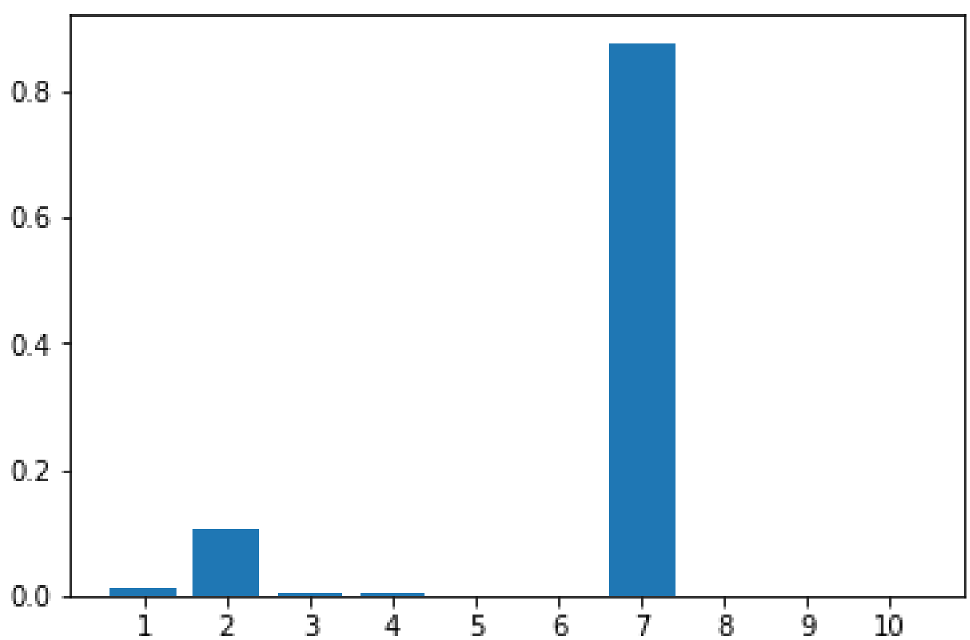

```
Prediction of the model: 1
```



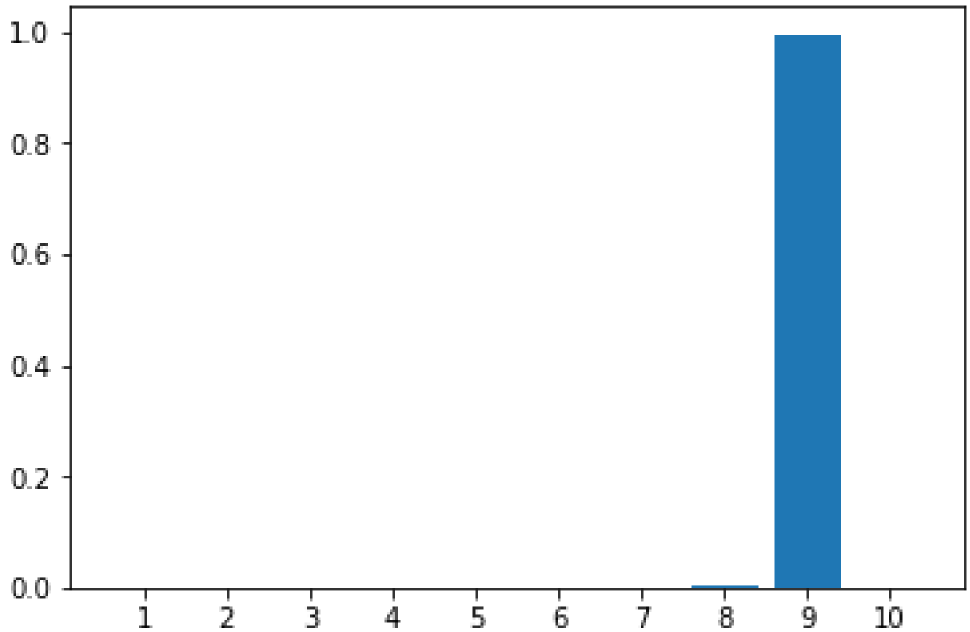
```
Prediction of the model: 3
```




```
Prediction of the model: 7
```



```
Prediction of the model: 9
```




```
In [ ]:
```

```
In [ ]:
```