ML Engineering Tutorial Part 2

Tutor: Ralf Mayet (mayet@campus.tu-berlin.de)
Adaptive Systems Group, Humboldt University Berlin

Content / Goals

- Build simple regression models with Tensorflow.
- Build a simple machine learning pipeline (preprocessing, learning, evaluation)
- Build a classification multi-layer perceptron with Tensorflow.
- Extend it to use convolutional layers and observe the difference.

Sources

- [1] Tensorflow and Keras Basic Regression
- [2] A line-by-line layman's guide to Linear Regression using TensorFlow (adapted to TF2)
- [3] Tensorflow Documentation: Loading MNIST
- [4] Basic classification: Classify images of clothing
- [5] Simple MNIST Convnet

In [1]:

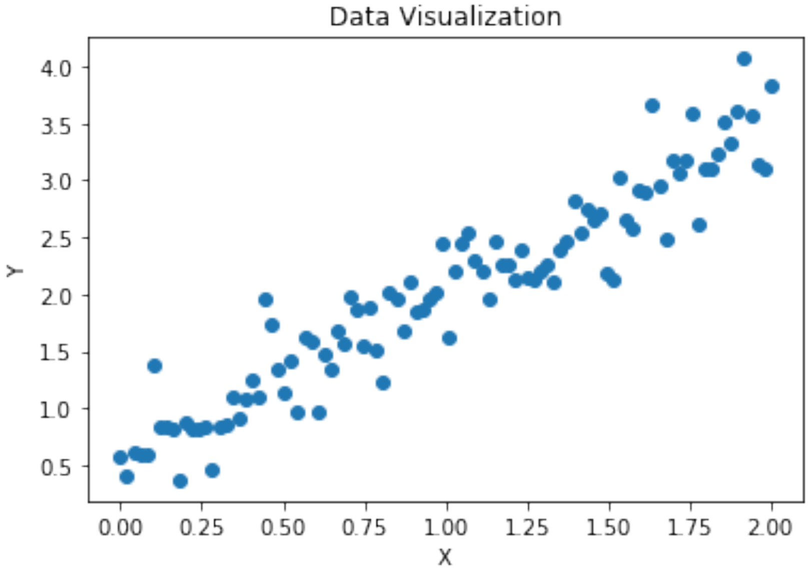
```
# Importing packages we'll be using
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
```

In [2]:

```
# Helper functions for plotting:
def plotData(X, Y, predictions=None, title="Data Visualization"):
  plt.scatter(X, Y)
  if predictions is not None:
    plt.scatter(X, predictions)
  plt.title(title)
  plt.xlabel("X")
  plt.ylabel("Y")
  plt.show()

def plotImage(image):
  plt.imshow(image)
  plt.colorbar()
  plt.show()
```

Part 2A: Re-visiting regression

In [3]:
```python
# Generating toy dataset
X = np.linspace(0, 2, 100)

# linear
y = 1.5 * X + np.random.randn(len(X)) * 0.3 + 0.5

# sinusoidal
# y = 1.5 * np.sin(X**2) + np.random.randn(len(X)) * 0.2 + 0.5

# Plot using our utility function
plotData(X, y)
```

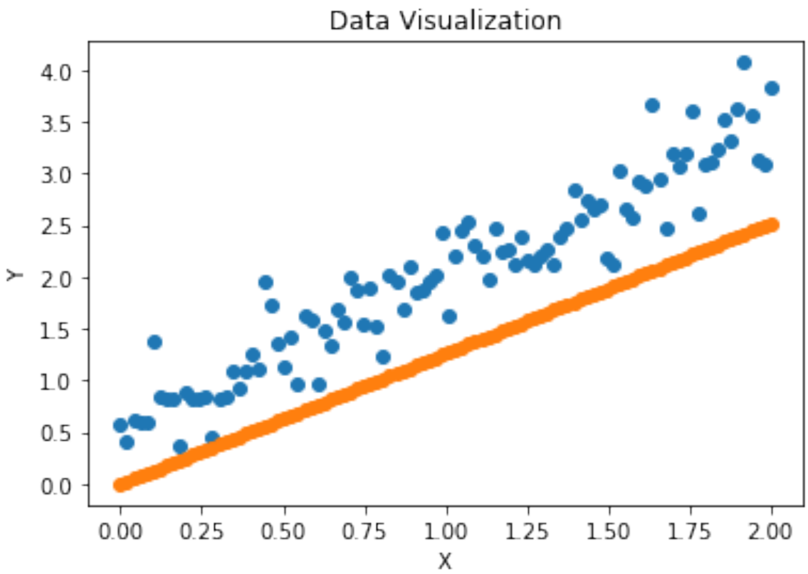


In [4]:
```python
# Making a model
# Ref activation functions: https://www.researchgate.net/profile/Junxi_Feng

model = tf.keras.Sequential([
    layers.Dense(input_shape=[1,], units=1)
])

# model = tf.keras.Sequential([
#     layers.Dense(input_shape=[1,], units=1, activation="tanh"),
#     layers.Dense(units=4, activation="tanh"),
#     layers.Dense(units=1)
# ])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 1)                 2         
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

In [5]:

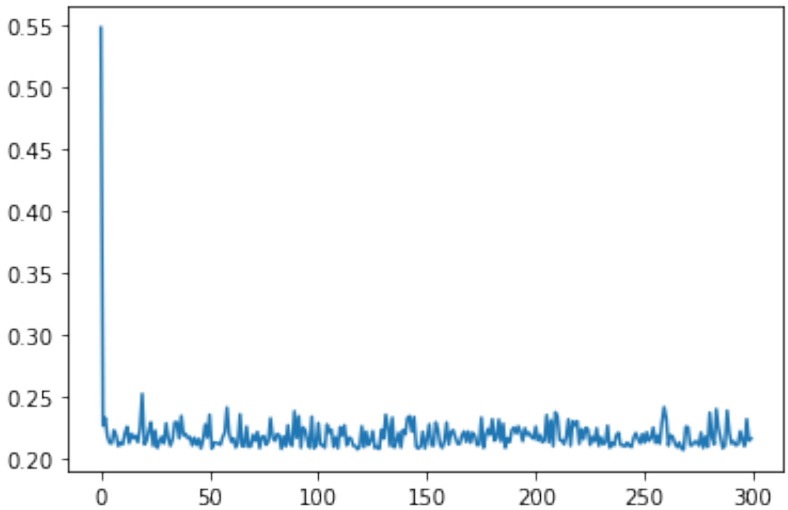
```
# Get some example predictions
predictions = model.predict(X)
plotData(X,y,predictions)

meanSquareError = ((y-predictions)**2).mean()
print("Mean Square Error: %.2f" % meanSquareError)
```




```
Mean Square Error: 1.91
```

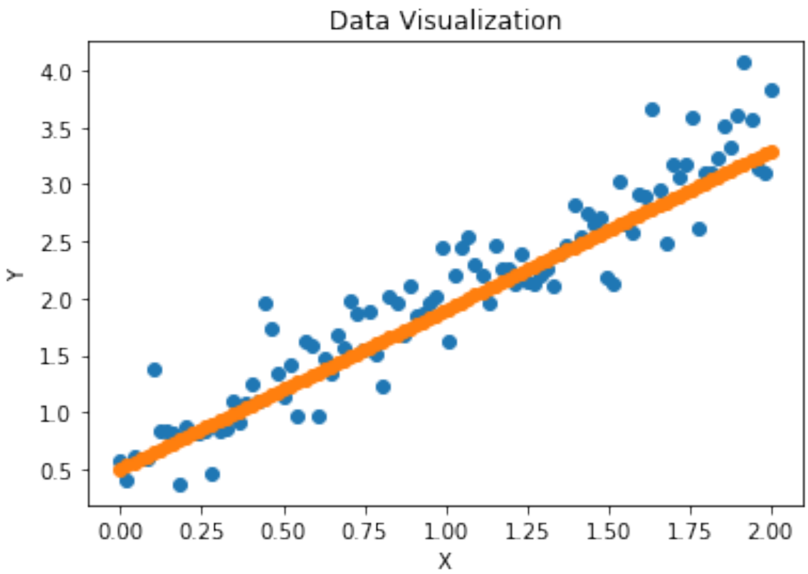
In [6]:

```
# Train the model
model.compile(optimizer=tf.optimizers.SGD(learning_rate=0.1), loss='mean_ab
history = model.fit(X,y, epochs=300, verbose=0)

# Plot the loss
plt.plot(history.history['loss'])
plt.show()
```



In [7]:

```
# Evaluate the final predictions
predictions = model.predict(X)
plotData(X,y,predictions)

meanSquareError = ((y-predictions)**2).mean()
print("Mean Square Error: %.2f" % meanSquareError)
```




```
Mean Square Error: 1.48
```

In [8]:

```
# We can inspect the layer weights after training and observe they
# match our toy data
model.layers[0].weights
```

Out[8]:

```
[<tf.Variable 'dense/kernel:0' shape=(1, 1) dtype=float32, numpy=array([[1.
3991199]], dtype=float32)>,
 <tf.Variable 'dense/bias:0' shape=(1,) dtype=float32, numpy=array([0.50000
083], dtype=float32)>]
```

Part 2B: Loading Images

In [9]:

```
# Loading MNIST using built-in TF function
# Ref https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/lc
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
```

In [10]:

```
# Inspect dataset
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28, 1)
(60000,)
(10000, 28, 28, 1)
(10000,)
```

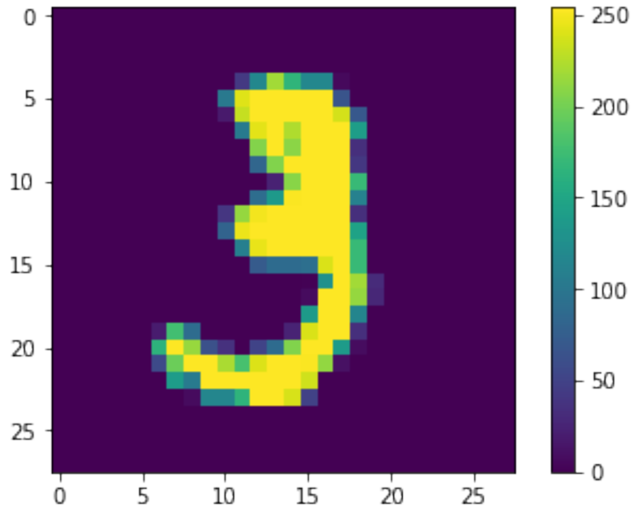
In [11]:

```
# Printing individual image and label
print(x_train[0,:,:,0])
print(y_train[0])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
5
```

In [12]:

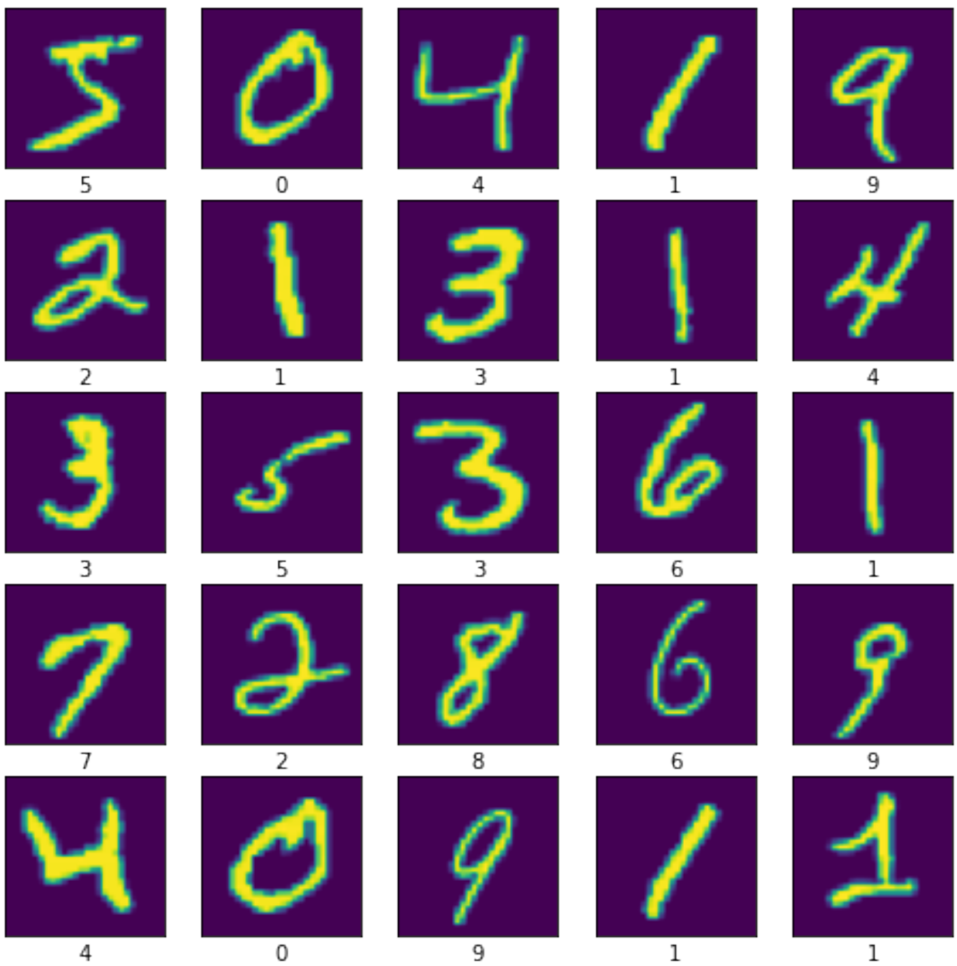
```
# Plotting individual image and label
plotImage(x_train[10,:,:,0])
print(y_train[10])
```



3

In [13]:

```
# Plot 25 examples
plt.figure(figsize=(8,8))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i,:,:,0])
    plt.xlabel(y_train[i])
plt.show()
```



Part 2C: Classification of Handwritten Digits

In [14]:

```
# We have our data in x_train, y_train (see above)

# Let's build a model for classification:
model_mlp = tf.keras.Sequential([
    layers.Flatten(input_shape=(28, 28, 1)),
    layers.Dense(128, activation='relu'),
    layers.Dense(10)
], name="mnist_mlp_model")

# Convolutional version
model_cnn = tf.keras.Sequential([
    keras.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(10)
], name="mnist_cnn_model")

model_mlp.summary()
model_cnn.summary()
```

```
Model: "mnist_mlp_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 784)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480    
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290      
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
Model: "mnist_cnn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0         
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                16010     
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

In [15]:

```python
# Compile the models
model_mlp.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logit
              metrics=['accuracy'])

model_cnn.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logit
              metrics=['accuracy'])
```

In [16]:

```python
# Evaluate Accuracy on test data
_, test_acc_mlp = model_mlp.evaluate(x_test,  y_test, verbose=2)
_, test_acc_cnn = model_cnn.evaluate(x_test,  y_test, verbose=2)
print('Test accuracy MLP:', test_acc_mlp)
print('Test accuracy CNN:', test_acc_cnn)
```

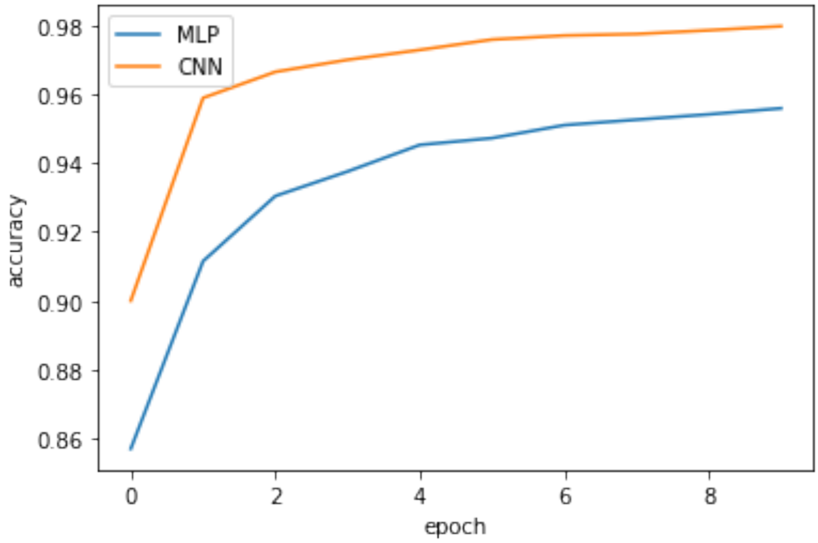
```
313/313 - 0s - loss: 155.0851 - accuracy: 0.1694
313/313 - 1s - loss: 45.1181 - accuracy: 0.0951
Test accuracy MLP: 0.16940000653266907
Test accuracy CNN: 0.09510000050067902
```

In [17]:

```python
# Fit to data
history_mlp = model_mlp.fit(x_train, y_train, epochs=10)
history_cnn = model_cnn.fit(x_train, y_train, epochs=10)

# Plot the accuracy
plt.plot(history_mlp.history['accuracy'], label="MLP")
plt.plot(history_cnn.history['accuracy'], label="CNN")
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```

```
Epoch 1/10
1875/1875 [==============================] - 2s 1ms/step - loss: 2.4870 - a
ccuracy: 0.8571
Epoch 2/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3628 - a
ccuracy: 0.9115
Epoch 3/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2786 - a
ccuracy: 0.9304
Epoch 4/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2497 - a
ccuracy: 0.9376
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2276 - a
ccuracy: 0.9453
Epoch 6/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2147 - a
ccuracy: 0.9472
Epoch 7/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2019 - a
ccuracy: 0.9510
Epoch 8/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2010 - a
ccuracy: 0.9526
Epoch 9/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1909 - a
ccuracy: 0.9541
Epoch 10/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1870 - a
ccuracy: 0.9559
Epoch 1/10
1875/1875 [==============================] - 37s 20ms/step - loss: 0.5589 -
accuracy: 0.9000
Epoch 2/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.1368 -
accuracy: 0.9589
Epoch 3/10
1875/1875 [==============================] - 30s 16ms/step - loss: 0.1128 -
accuracy: 0.9665
Epoch 4/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0996 -
accuracy: 0.9700
Epoch 5/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0907 -
accuracy: 0.9728
Epoch 6/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.0816 -
accuracy: 0.9758
Epoch 7/10
1875/1875 [==============================] - 34s 18ms/step - loss: 0.0772 -
```

```
accuracy: 0.9770
Epoch 8/10
1875/1875 [==============================] - 33s 18ms/step - loss: 0.0752 -
accuracy: 0.9775
Epoch 9/10
1875/1875 [==============================] - 31s 17ms/step - loss: 0.0710 -
accuracy: 0.9785
Epoch 10/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.0667 -
accuracy: 0.9797
```



In [18]:

```
# Evaluate Accuracy on test data
_, test_acc_mlp = model_mlp.evaluate(x_test,  y_test, verbose=2)
_, test_acc_cnn = model_cnn.evaluate(x_test,  y_test, verbose=2)
print('Test accuracy MLP:', test_acc_mlp)
print('Test accuracy CNN:', test_acc_cnn)
```

```
313/313 - 0s - loss: 0.2574 - accuracy: 0.9535
313/313 - 1s - loss: 0.0451 - accuracy: 0.9859
Test accuracy MLP: 0.953499972820282
Test accuracy CNN: 0.9858999848365784
```

Part 2D: Evaluating the classification output

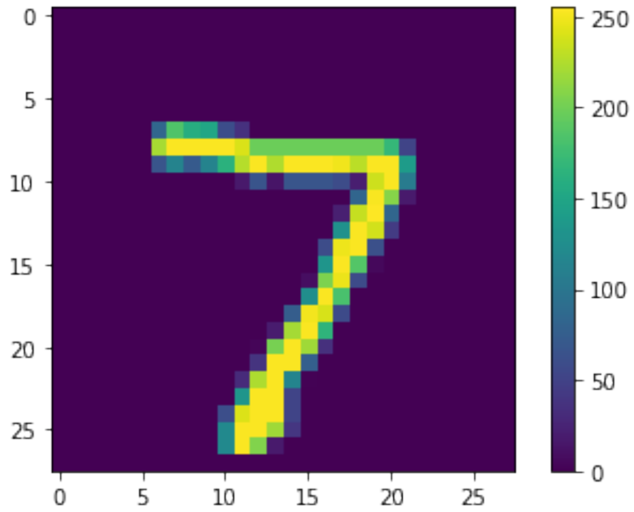
In [19]:

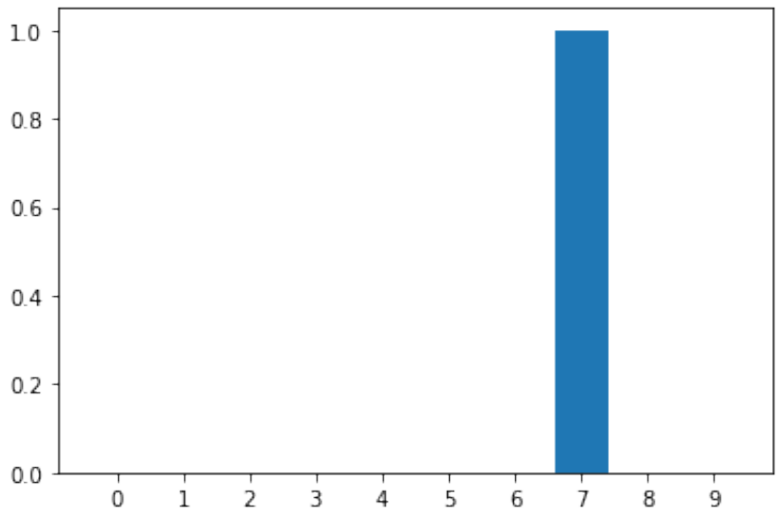
```
# Get some predictions
# Attach a softmax layer to convert the logits to probabilities, which are
probability_model = tf.keras.Sequential([model_cnn, tf.keras.layers.Softmax
predictions = probability_model.predict(x_test)

print(predictions.shape)
print(predictions[0])
```

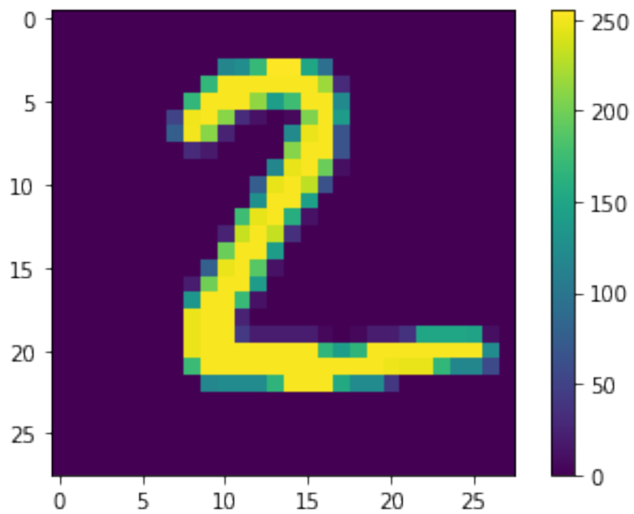
```
(10000, 10)
[1.11056151e-08 1.24207344e-14 4.61679619e-08 3.08107388e-08
 1.85735148e-16 1.80415318e-12 3.65252618e-20 1.00000000e+00
 9.83751830e-11 1.84875182e-09]
```

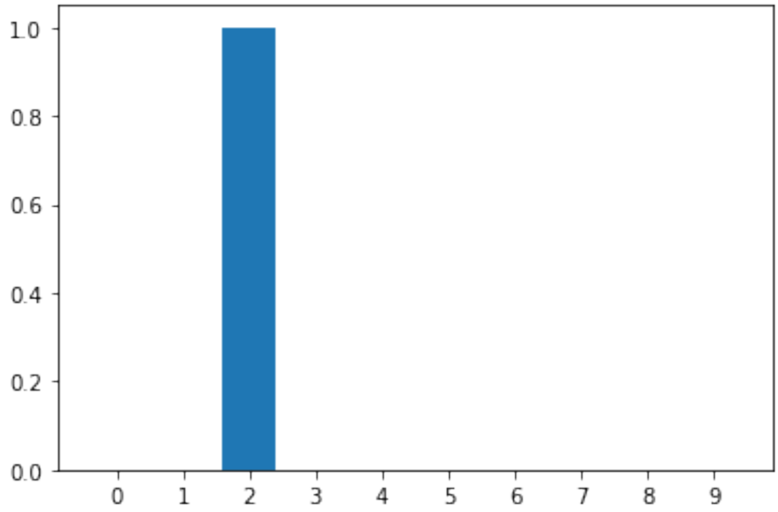
In [20]:

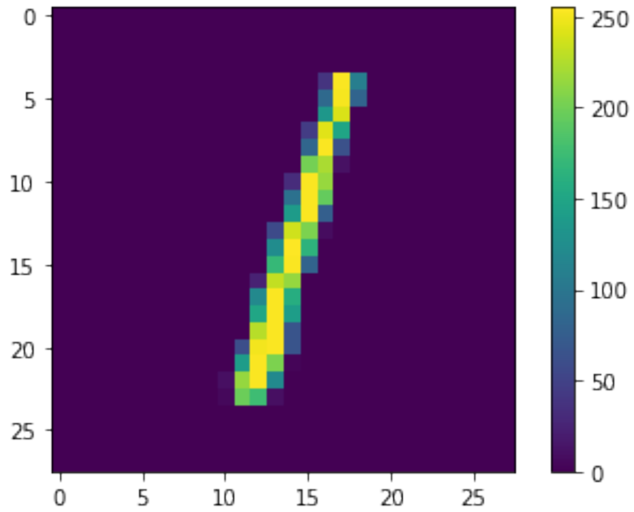
```
# Plot classification results
for i in range(3):
  plotImage(x_test[i,:,:,0])
  plt.xticks(range(10))
  plt.bar(range(10), predictions[i])
  plt.show()
```

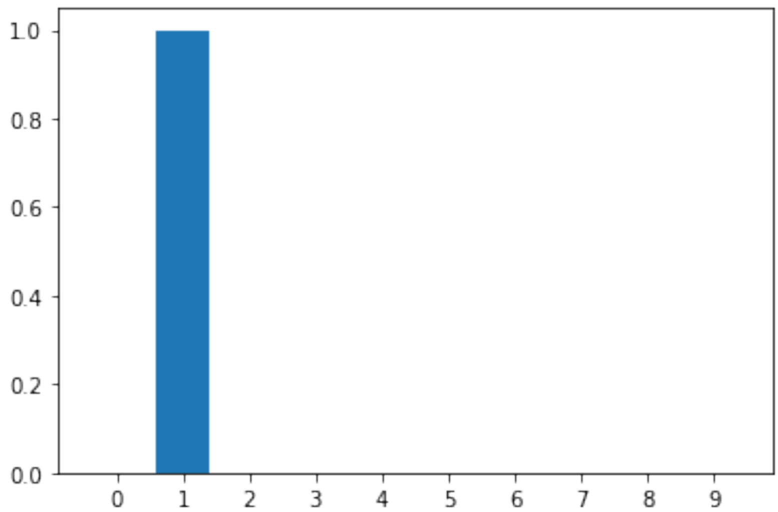












Part 2E: Loading images from disk

Demo on Desktop