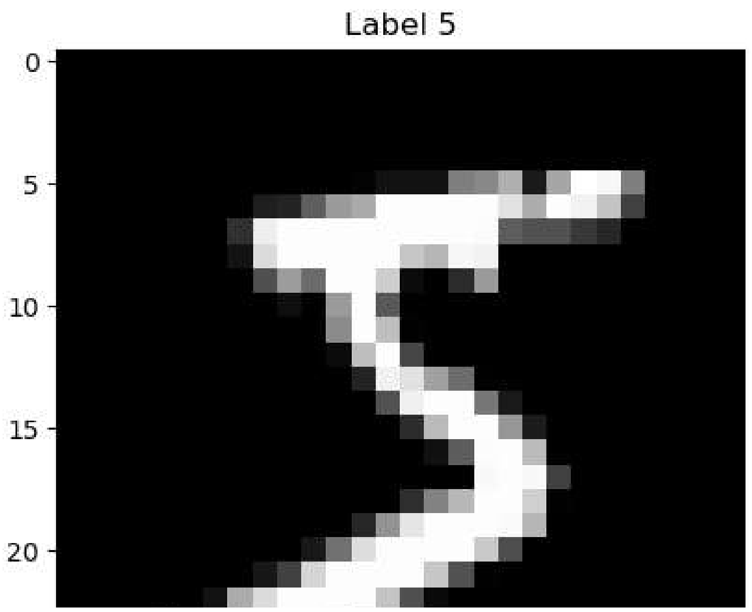
In [1]:
```python
import matplotlib.pyplot as plt
import  tensorflow as tf
from tensorflow.keras import layers, models
```

In [2]:
```python
# Load the MNIST dataset
(x_train,y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data()
```

In [3]:
```python
# Display the first 5 images and lable from training set
for i in range(5):
  plt.imshow(x_train[i],cmap='gray')
  plt.title("Label "+ str(y_train[i]))
  plt.show()
```



In [4]:
```python

# Normalize the images (from [0, 255] to [0, 1])
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

In [5]:
```python
#Reshape the images to add the channel dimension
x_train = x_train.reshape((x_train.shape[0],28,28,1))
x_test = x_test.reshape((x_test.shape[0],28,28,1))
```

In [6]:
```python
# Check the shapes of the data
print(f'Training data shape: {x_train.shape}, Labels shape: {y_train.shape}')
```

```
Training data shape: (60000, 28, 28, 1), Labels shape: (60000,)
```

In [7]:
```python
print(f'Test data shape: {x_test.shape}, Labels shape: {y_test.shape}')
```

```
Test data shape: (10000, 28, 28, 1), Labels shape: (10000,)
```

In [8]:
```
# One-hot encode the labels
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

In [9]:
```
# Build the CNN model
model = models.Sequential()
```

In [10]:
```

# First convolutional layer with 32 filters, 3x3 kernel size, and ReLU activation
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
```

```
C:\Users\FamiAmal\.anaconda\annaconda for me\Lib\site-packages\keras\src\layers\conv
olutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `Input(shape)` obje
ct as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [11]:
```
# Second convolutional layerwith 64 filters, 3x3 kernel size, and ReLU activation
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

In [12]:
```
# MaxPooling layer to downsample by 2x2
model.add(layers.MaxPooling2D((2, 2)))
```

In [13]:
```
# Dropout layer for regularization
model.add(layers.Dropout(0.25))
```

In [14]:
```
# Flatten the feature maps into a 1D feature vector
model.add(layers.Flatten())
```

In [15]:
```
# Fully connected Dense layer with 128 units and ReLU activation
model.add(layers.Dense(128, activation='relu'))
```

In [16]:
```
# Dropout layer to prevent overfitting
model.add(layers.Dropout(0.5))
```

In [17]:
```
# Output layer with 10 units (one for each class) and softmax activation
model.add(layers.Dense(10, activation='softmax'))
```

In [18]:
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']
```

In [19]:

```
# Display a summary of the model
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param
conv2d (Conv2D)	(None, 26, 26, 32)	32
conv2d_1 (Conv2D)	(None, 24, 24, 32)	9,24
conv2d_2 (Conv2D)	(None, 22, 22, 64)	18,49
max_pooling2d (MaxPooling2D)	(None, 11, 11, 64)	
dropout (Dropout)	(None, 11, 11, 64)	
flatten (Flatten)	(None, 7744)	
dense (Dense)	(None, 128)	991,36
dropout_1 (Dropout)	(None, 128)	
dense_1 (Dense)	(None, 10)	1,29

Total params: 1,020,714 (3.89 MB)

Trainable params: 1,020,714 (3.89 MB)

Non-trainable params: 0 (0.00 B)

In [20]:

```
# Train the model
history = model.fit(x_train, y_train, epochs=5, batch_size=64, validation_data=(x_tes
```

```
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 62s 61ms/step - accuracy: 0.8803 - loss: 0.3734 - val_a
ccuracy: 0.9865 - val_loss: 0.0432
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 56s 59ms/step - accuracy: 0.9766 - loss: 0.0797 - val_a
ccuracy: 0.9900 - val_loss: 0.0302
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 57s 60ms/step - accuracy: 0.9818 - loss: 0.0586 - val_a
ccuracy: 0.9883 - val_loss: 0.0347
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 56s 60ms/step - accuracy: 0.9847 - loss: 0.0491 - val_a
ccuracy: 0.9902 - val_loss: 0.0279
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 57s 60ms/step - accuracy: 0.9886 - loss: 0.0373 - val_a
ccuracy: 0.9922 - val_loss: 0.0237
```

In [21]:

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9893 - loss: 0.0309
Test accuracy: 0.9922
```

In [22]:

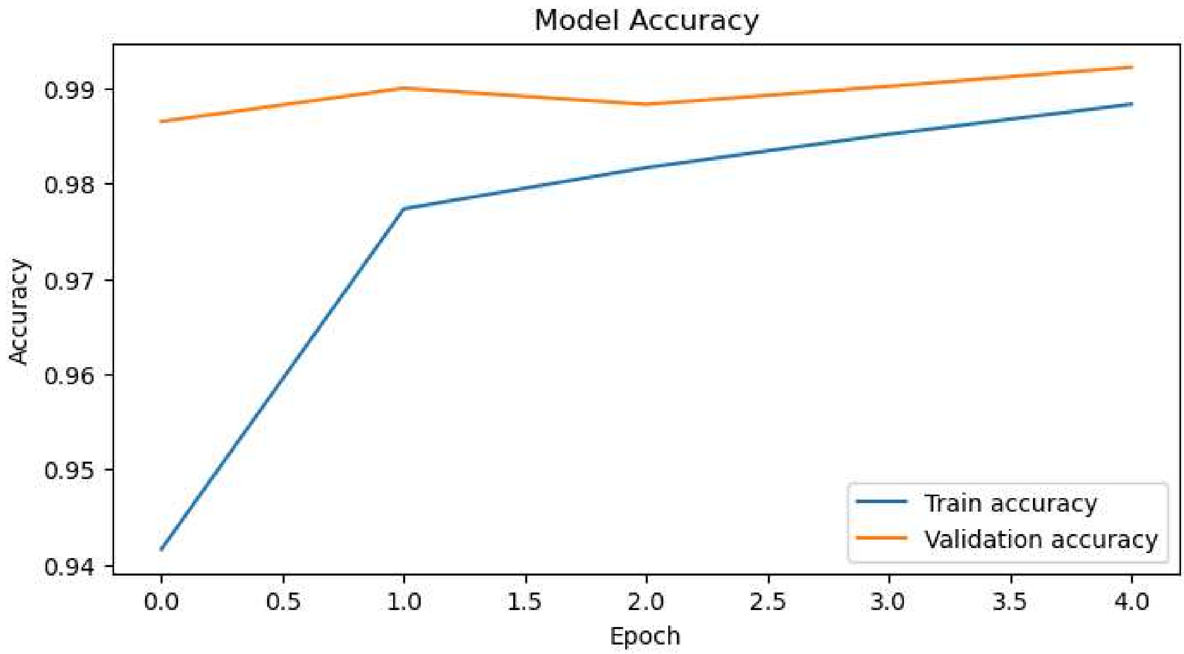
```
history.history['accuracy']
```

Out[22]:

```
[0.9416166543960571,
 0.9773499965667725,
 0.9816666841506958,
 0.9851666688919067,
 0.9883166551589966]
```

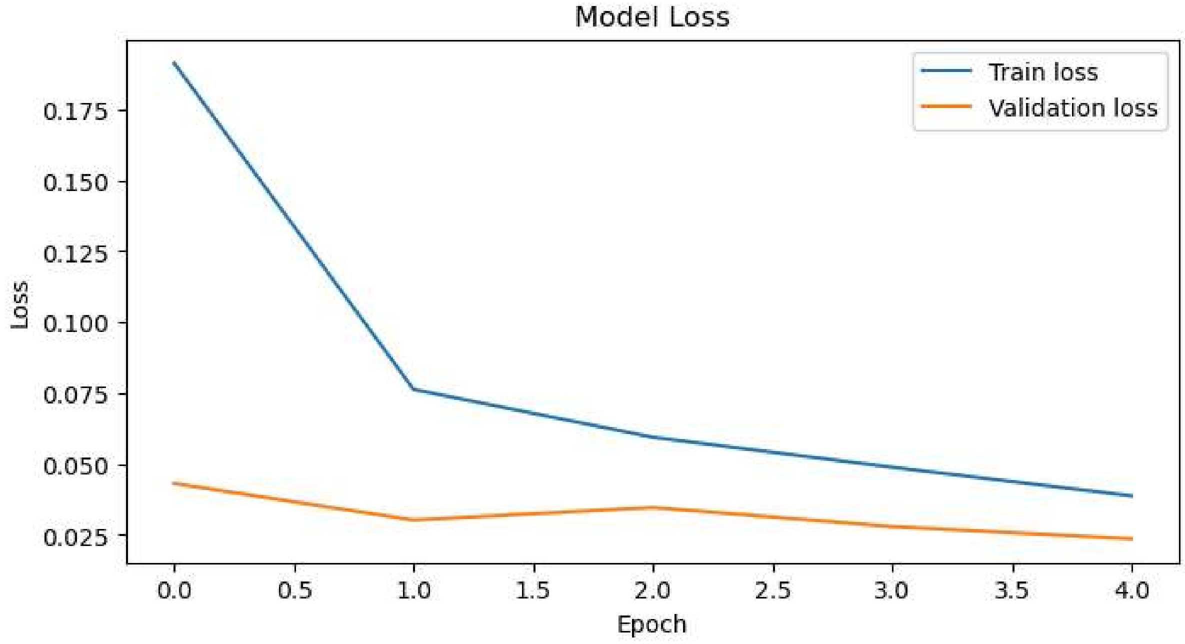
In [23]:

```
#Plot traning & validation  accuracy values
plt.figure(figsize=(8,4))
plt.plot(history.history['accuracy'], label="Train accuracy")
plt.plot(history.history['val_accuracy'], label="Validation accuracy")
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



In [24]:
```
#Plot traning & validation  accuracy values
plt.figure(figsize=(8,4))

plt.plot(history.history['loss'], label="Train loss")
plt.plot(history.history['val_loss'], label="Validation loss")
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



Summery

Dataset: The MNIST dataset is loaded, containing handwritten digits from 0 to 9. Each image is 28x28 pixels, grayscale. Preprocessing:

The pixel values of the images are normalized (scaled to values between 0 and 1).

The images are reshaped to include a channel dimension, necessary for TensorFlow's CNN input.

The labels are one-hot encoded, meaning they are converted into binary vectors, each representing a class (digit). Model Architecture:

The model is a sequential CNN built using TensorFlow's Keras API.

The model starts with two convolutional layers followed by a max-pooling layer for downsampling and dropout layers for regularization to prevent overfitting.

After flattening the output, the model adds a fully connected (dense) layer and another dropout layer.

The output layer uses softmax activation to predict the class of the input image.

Training: The model is compiled with the Adam optimizer and trained using categorical cross-entropy as the loss function. It trains for 5 epochs with a batch size of 64. Both training and validation accuracy and loss are tracked.

Evaluation: The model is evaluated on the test set, achieving a high accuracy (99.09%). Visualization:

Two plots are generated: one for training and validation accuracy and another for training and validation

In [ ]: