Problem description

The MNIST dataset is a widely used and classic dataset in the field of machine learning and computer vision. It stands for "Modified National Institute of Standards and Technology" and is a collection of handwritten digits. This dataset is often considered the "Hello World" of computer vision because it serves as an entry point for many researchers and practitioners to explore image classification and pattern recognition algorithms.

The MNIST dataset consists of a training set and a testing set, each containing grayscale images of handwritten digits from 0 to 9. The images are of size 28x28 pixels, which results in a total of 784 pixels per image. Each pixel value represents the darkness of the pixel, with higher values indicating darker areas and lower values indicating lighter areas.

The training set usually contains 60,000 images, while the testing set contains 10,000 images. This dataset has been preprocessed and normalized, making it a convenient starting point for experimenting with various machine learning techniques. Researchers often use it to develop and test new image classification algorithms and to benchmark the performance of different models.

The MNIST dataset has played a crucial role in advancing the field of deep learning, particularly in the development and evaluation of convolutional neural networks (CNNs) for image recognition tasks. It has provided a standardized way to compare the performance of different models and has served as a foundational dataset for many research papers and educational materials.

Importing the libraries

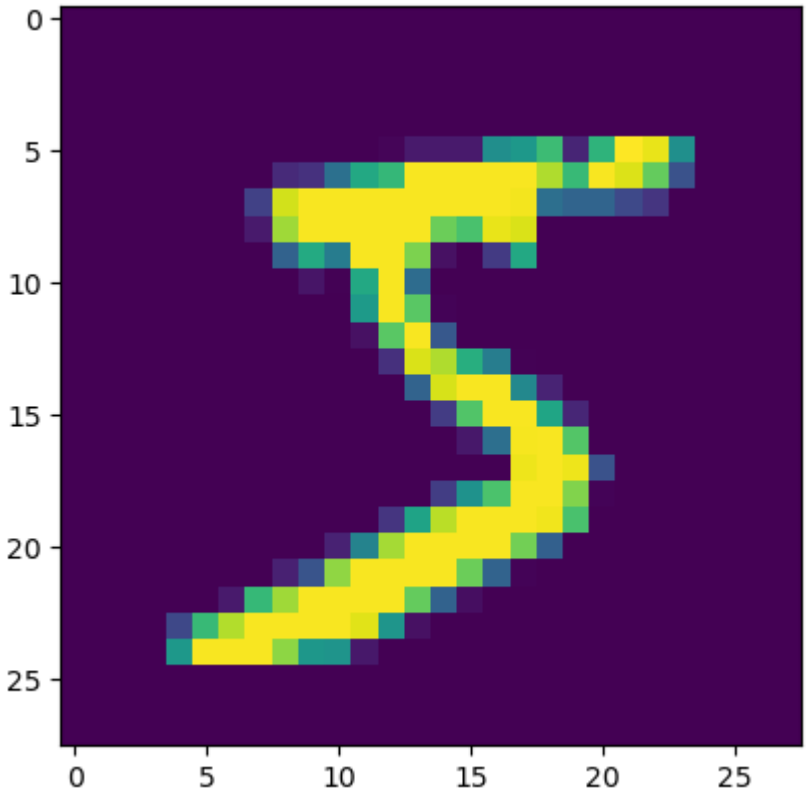
In [58]:
```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import models,layers


import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading the variables

In [47]:
```
(X_train,y_train_org),(X_test,y_test_org) = keras.datasets.mnist.load_data()
```

In [48]:
```
plt.imshow(X_train[0])
plt.show()
```



In [49]:
```
# to convert all value in between 0-1 we require to divide it 255
X_train = X_train/255
X_test = X_test/255

y_train = to_categorical(y_train_org)
y_test = to_categorical(y_test_org)
```

Exploratory Data Analysis

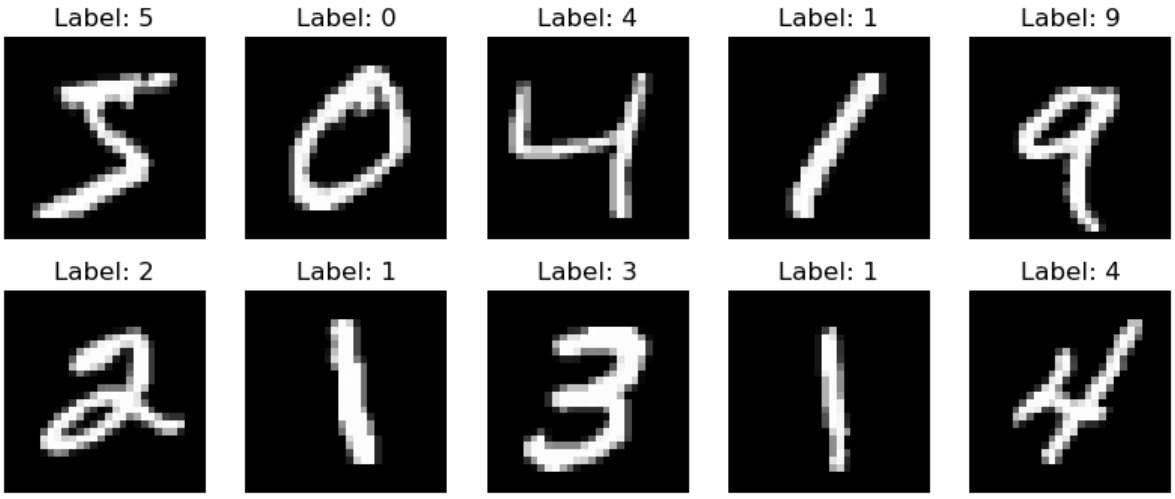
In [50]:
```
# Basic statistical summary on training dataset
print("Number of samples:", X_train.shape[0])
print("Number of features per sample:", X_train.shape[1])
print("Number of classes:", len(np.unique(y_train)))
```

```
Number of samples: 60000
Number of features per sample: 28
Number of classes: 2
```

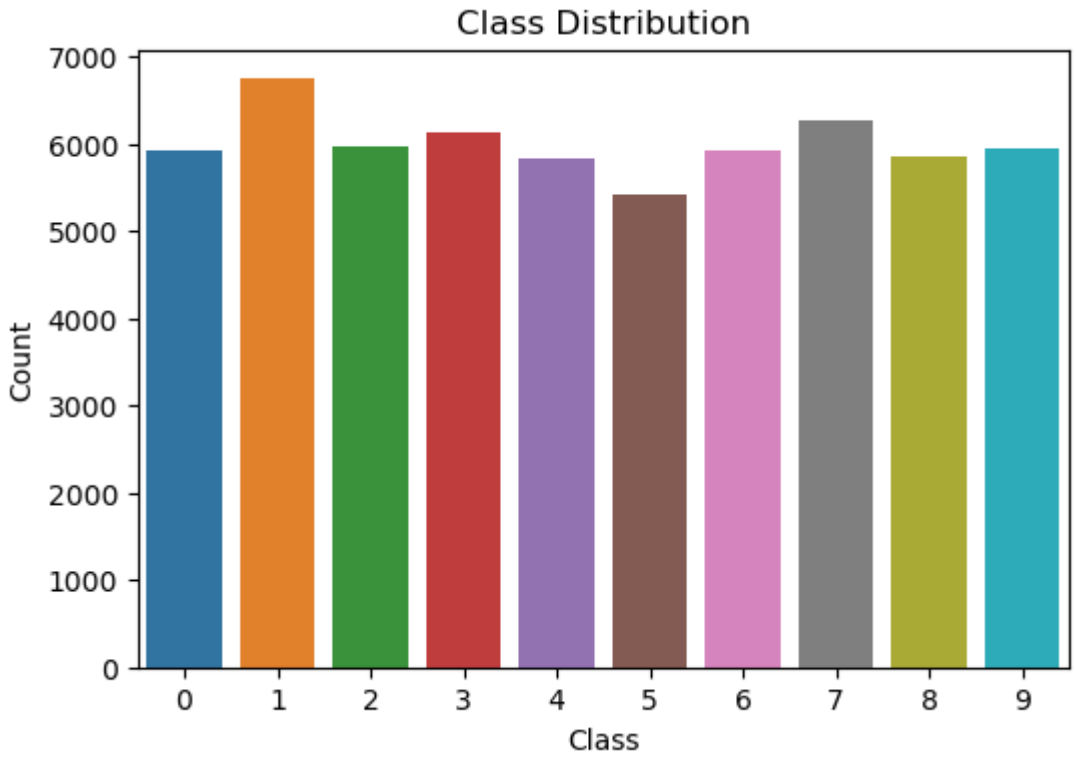
In [51]:
```
# Basic statistical summary on testing dataset
print("Number of samples:", X_test.shape[0])
print("Number of features per sample:", X_test.shape[1])
print("Number of classes:", len(np.unique(y_test)))
```

```
Number of samples: 10000
Number of features per sample: 28
Number of classes: 2
```

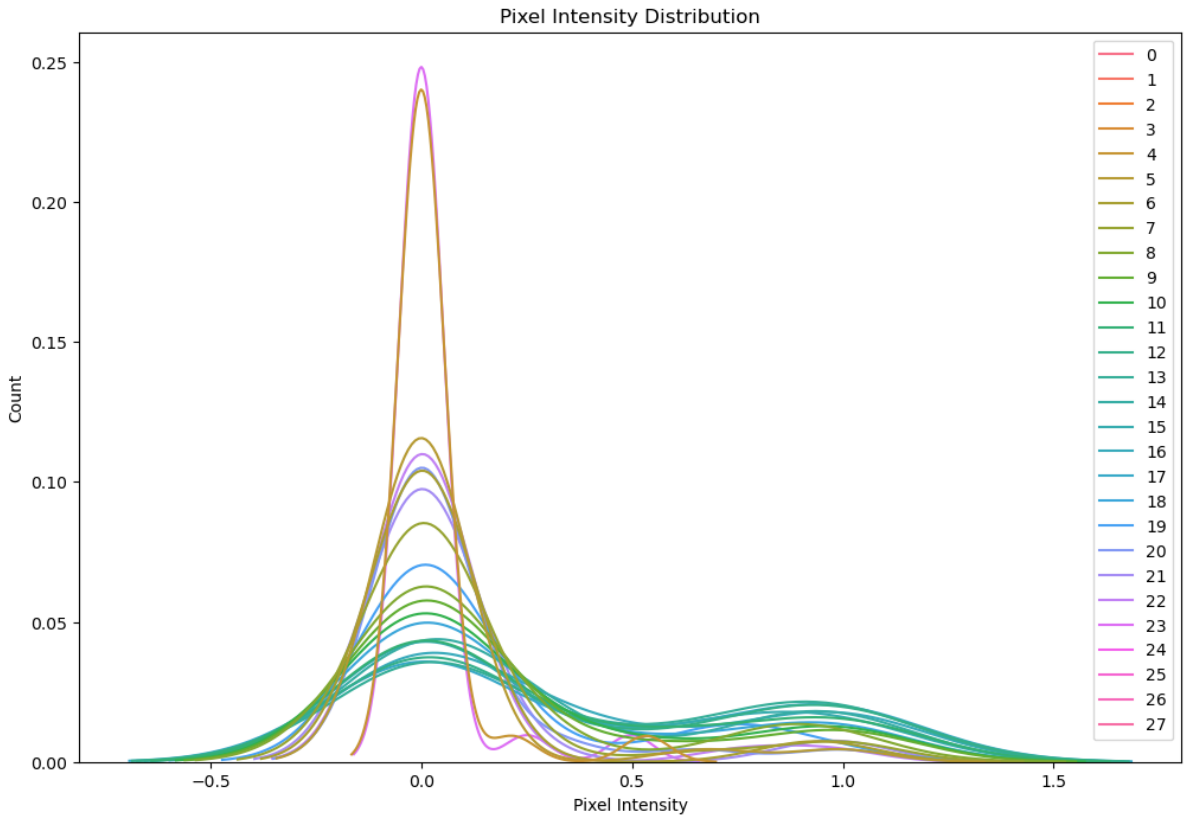
In [52]:
```python
# Visualize some sample images
plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_train[i], cmap="gray")
    plt.title(f"Label: {y_train_org[i]}")
    plt.axis("off")
plt.show()
```



In [53]:
```python
# Class distribution
plt.figure(figsize=(6, 4))
sns.countplot(x=y_train_org.astype(int))
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```



In [54]:
```
# Pixel intensity distribution
plt.figure(figsize=(12, 8))
sns.kdeplot(X_train[0])
plt.title("Pixel Intensity Distribution")
plt.xlabel("Pixel Intensity")
plt.ylabel("Count")
plt.show()
```



Model building and training

In [59]:
```python
# Build the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 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 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 64)                36928

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

In [60]:
```
# Train the model
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.1)
```

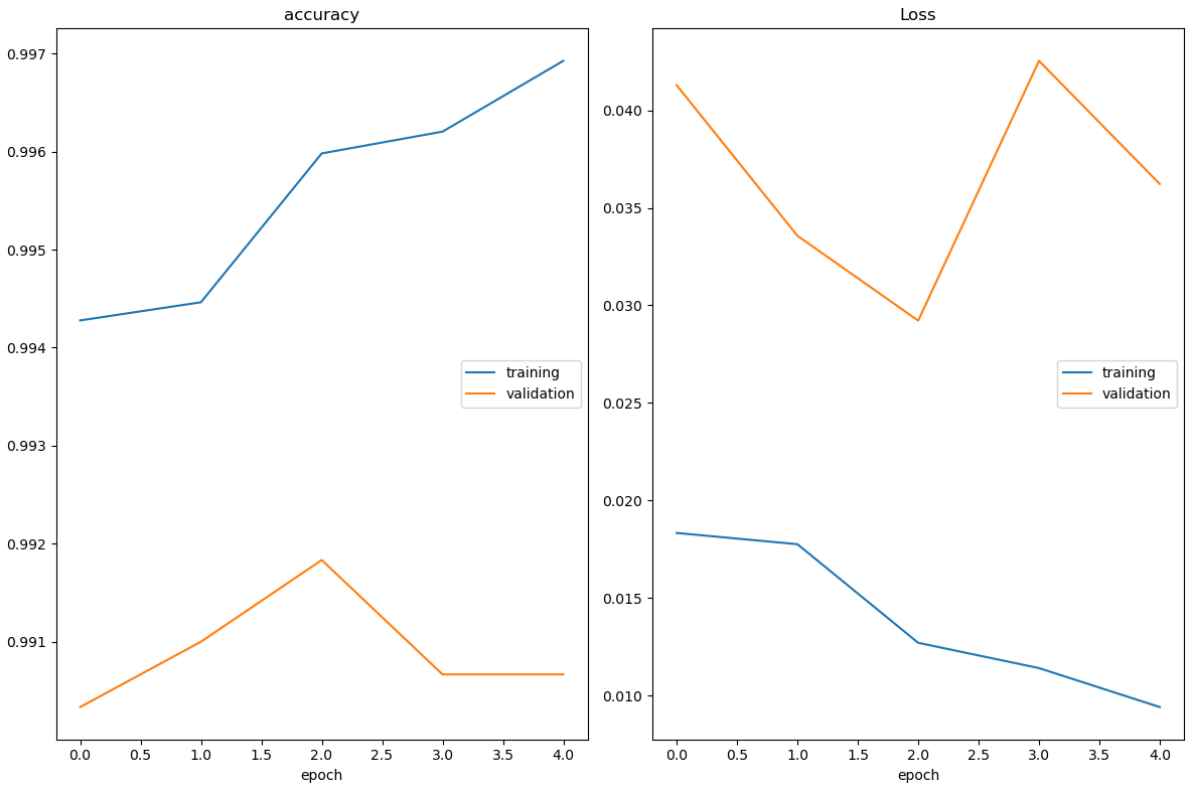
```
Epoch 1/5
844/844 [==============================] - 19s 11ms/step - loss: 0.1991 - accurac
y: 0.9391 - val_loss: 0.0505 - val_accuracy: 0.9850
Epoch 2/5
844/844 [==============================] - 8s 9ms/step - loss: 0.0531 - accuracy:
0.9838 - val_loss: 0.0460 - val_accuracy: 0.9853
Epoch 3/5
844/844 [==============================] - 8s 9ms/step - loss: 0.0368 - accuracy:
0.9888 - val_loss: 0.0405 - val_accuracy: 0.9897
Epoch 4/5
844/844 [==============================] - 8s 9ms/step - loss: 0.0284 - accuracy:
0.9908 - val_loss: 0.0391 - val_accuracy: 0.9890
Epoch 5/5
844/844 [==============================] - 8s 9ms/step - loss: 0.0240 - accuracy:
0.9923 - val_loss: 0.0363 - val_accuracy: 0.9907
```

Out[60]:
```
<keras.callbacks.History at 0x27ef62f7ac0>
```

Result

In [62]:
```
from livelossplot import PlotLossesKerasTF
```

```
# Train the model
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.1,callbacks
```




```
accuracy
        training                (min:    0.994, max:    0.997, cur:    0.997)
        validation              (min:    0.990, max:    0.992, cur:    0.991)
Loss
        training                (min:    0.009, max:    0.018, cur:    0.009)
        validation              (min:    0.029, max:    0.043, cur:    0.036)
844/844 [==============================] - 8s 10ms/step - loss: 0.0094 - accuracy:
0.9969 - val_loss: 0.0362 - val_accuracy: 0.9907
```

Out[62]: `<keras.callbacks.History at 0x27ef2ab6b30>`

In [66]:
```
# Evaluate the model on the test data
train_loss, train_acc = model.evaluate(X_train,y_train)
test_loss, test_acc = model.evaluate(X_test,y_test)
print('Train accuracy:', train_acc)
print('Test accuracy:', test_acc)
```

```
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0095 - accurac
y: 0.9972
313/313 [==============================] - 2s 7ms/step - loss: 0.0306 - accuracy:
0.9917
Train accuracy: 0.9971666932106018
Test accuracy: 0.9916999936103821
```

Conclusion

The MNIST dataset, a collection of handwritten digit images, has been a cornerstone in the field of machine learning and computer vision for many years. It serves as an ideal starting

point for researchers, practitioners, and students to explore image classification algorithms and evaluate model performance. The dataset's simplicity, size, and standardized format make it a valuable resource for developing and benchmarking various machine learning techniques.

In conclusion, the MNIST dataset remains a valuable resource for learning and experimentation in the field of machine learning and computer vision. It's a great starting point for understanding the basics of model training, evaluation, and hyperparameter tuning. However, it's important to recognize its limitations and to apply the knowledge gained to more complex and diverse datasets as you progress in your machine learning journey.