TensorFlow

Convolutional Neural Networks using Tensor Flow

Convolutional Neural Networks

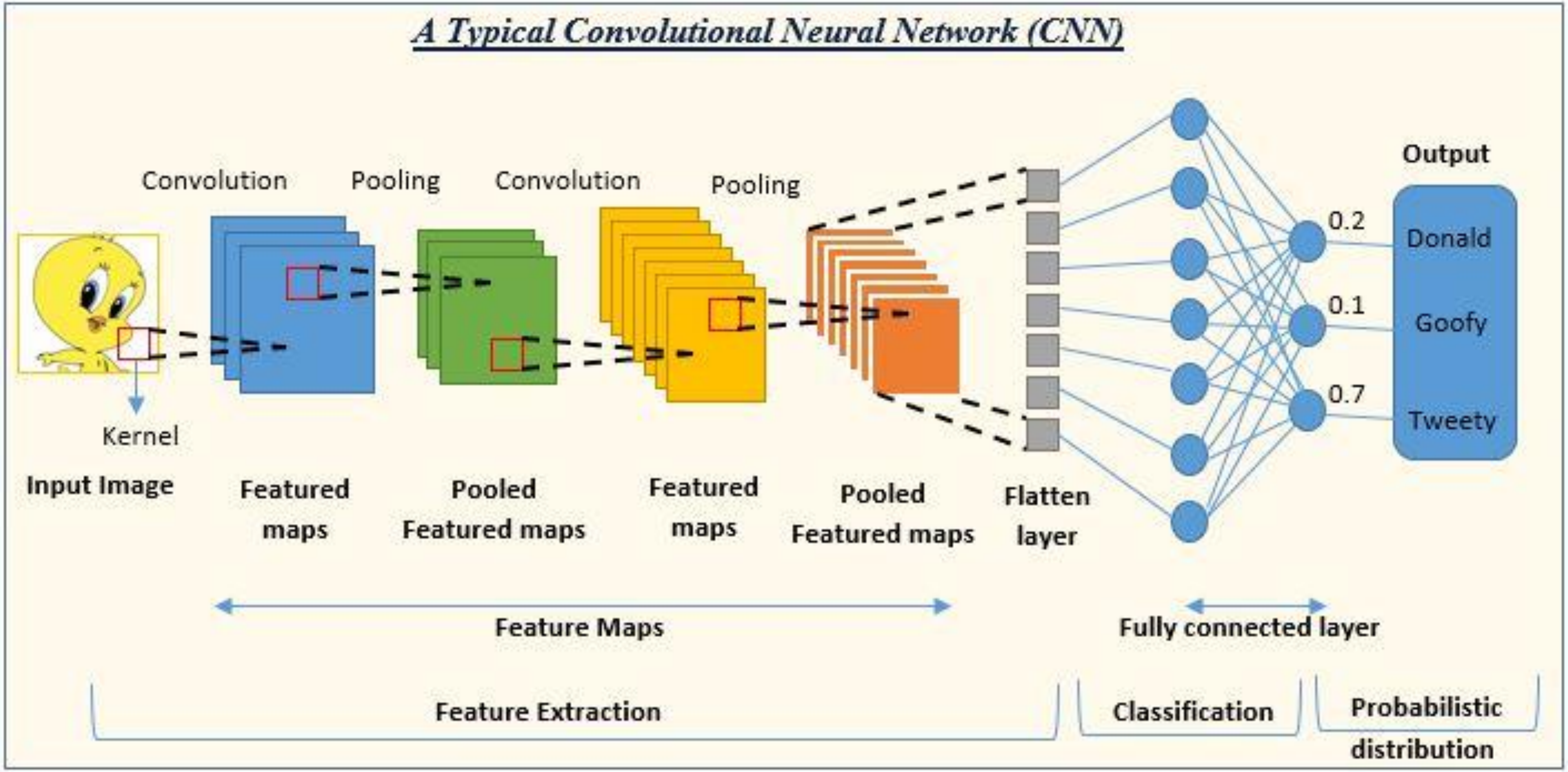
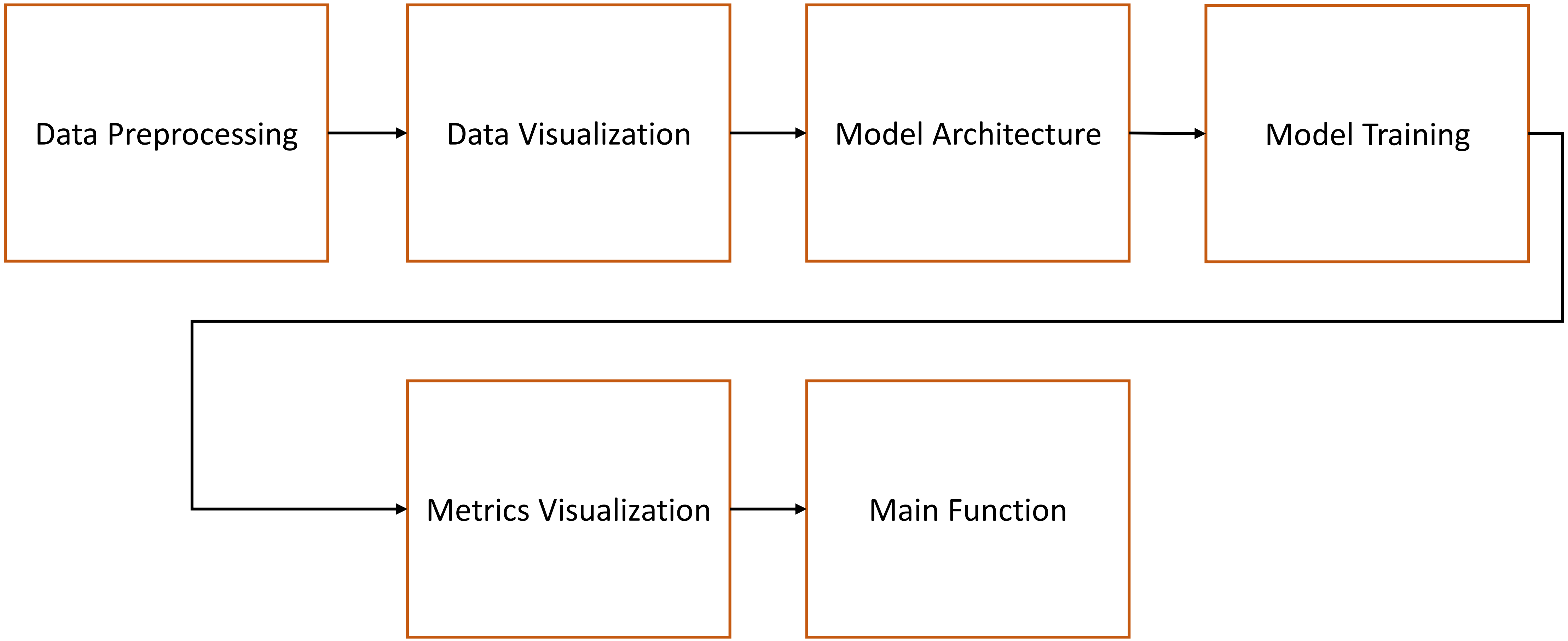


Image Processing Pipeline



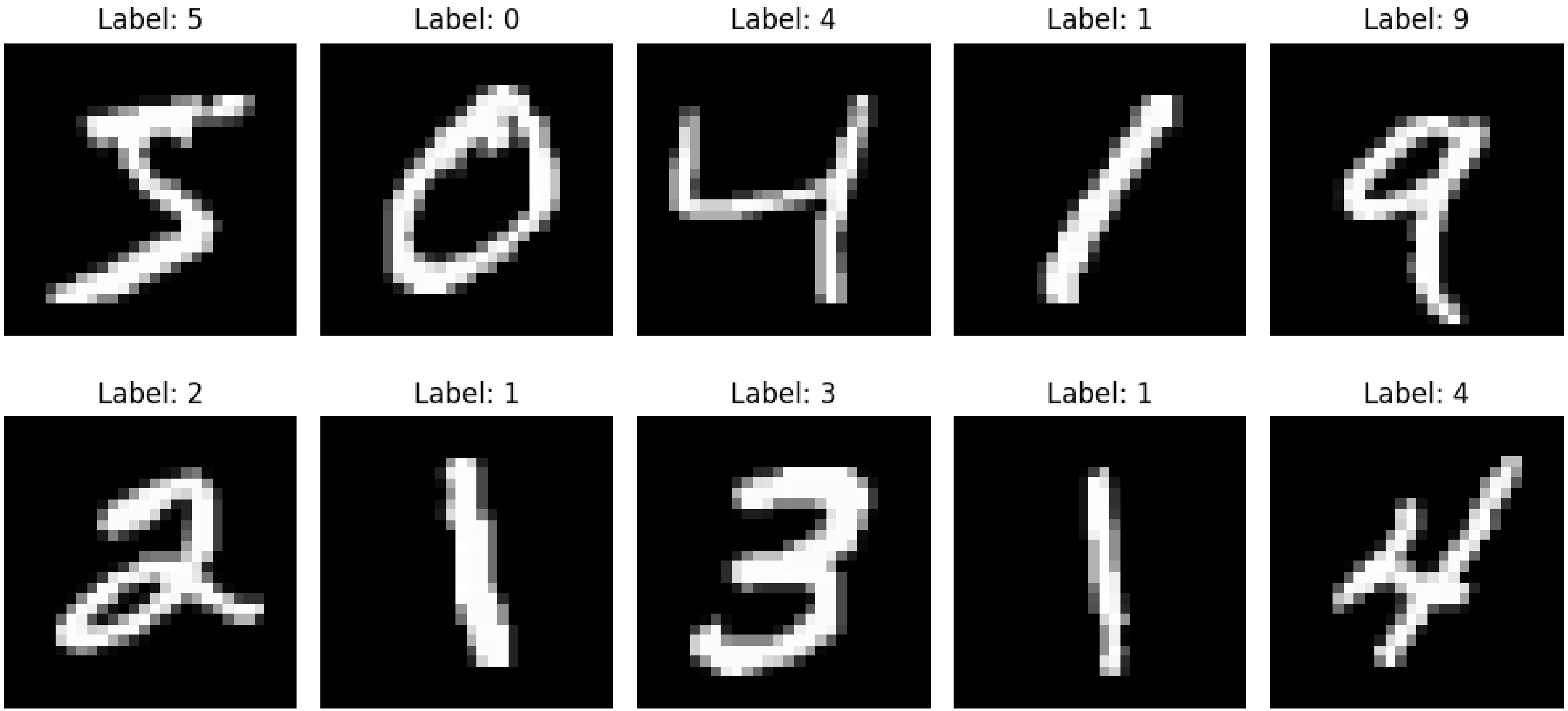
Data Preprocessing

- For loading the dataset, normalizing the pixel values, and reshaping the images to have a single channel

```
# Load and preprocess the MNIST dataset
def load_and_preprocess_data():
    # Load MNIST dataset
    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    # Normalize pixel values to be between 0 and 1
    x_train, x_test = x_train / 255.0, x_test / 255.0

    # Reshape images to have a single channel
    x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
    x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

    return (x_train, y_train), (x_test, y_test)
```

Data Visualization

- For displaying a sample of some images from the training set along with their labels



Data Visualization

```python
# Visualize sample images from the dataset
def visualize_sample_images(x_train, y_train):
    plt.figure(figsize=(10, 5))
    for i in range(10):
        plt.subplot(2, 5, i+1)
        plt.imshow(x_train[i].reshape(28, 28), cmap='gray')
        plt.title(f"Label: {y_train[i]}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

Model Architecture

- For constructing the CNN with specified number of convolutional, max-pooling, and fully connected layers

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)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36,928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dense_1 (Dense)	(None, 10)	650

Model Architecture

```
# Build the CNN model
def build_model():
    model = tf.keras.models.Sequential([
        # First Convolutional Layer
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        tf.keras.layers.MaxPooling2D((2, 2)),
        # Second Convolutional Layer
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        # Third Convolutional Layer
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        # Flatten the output for the dense layers
        tf.keras.layers.Flatten(),
        # Fully connected layers
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Model Training

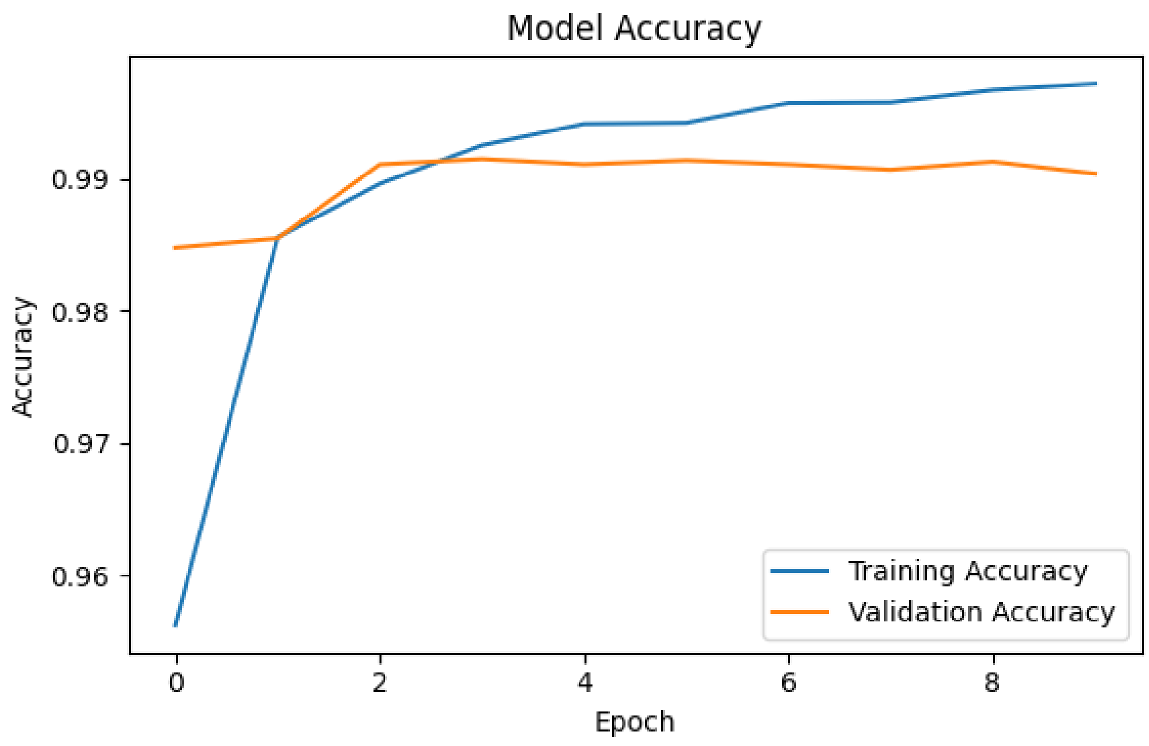
- For training the model for the specified number of epochs, using the specified optimizer and loss function

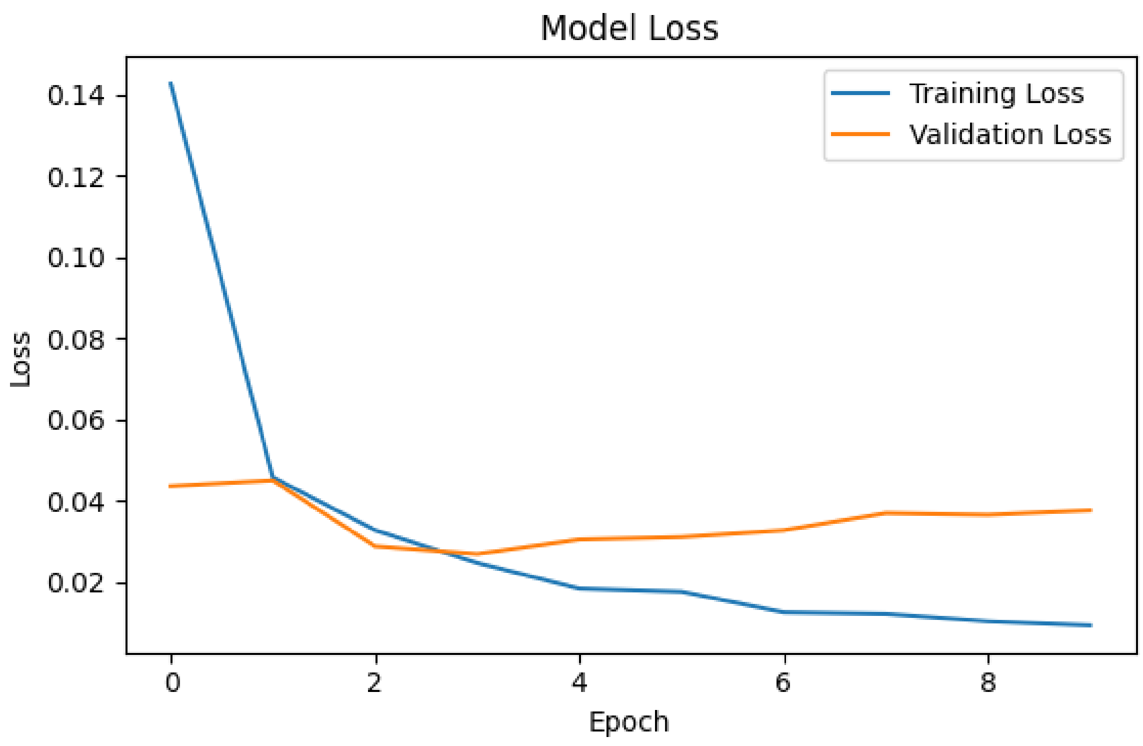
```
# Train the model
def train_model(model, x_train, y_train, x_test, y_test):
    history = model.fit(x_train, y_train, epochs=10,
                        validation_data=(x_test, y_test))
    return history
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.8978 - loss: 0.3357 - val_accuracy: 0.9848 - val_loss: 0.0437
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9856 - loss: 0.0460 - val_accuracy: 0.9855 - val_loss: 0.0451
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9898 - loss: 0.0329 - val_accuracy: 0.9911 - val_loss: 0.0289
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9931 - loss: 0.0230 - val_accuracy: 0.9915 - val_loss: 0.0269
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9942 - loss: 0.0180 - val_accuracy: 0.9911 - val_loss: 0.0305
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9952 - loss: 0.0152 - val_accuracy: 0.9914 - val_loss: 0.0311
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9963 - loss: 0.0108 - val_accuracy: 0.9911 - val_loss: 0.0328
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9964 - loss: 0.0107 - val_accuracy: 0.9907 - val_loss: 0.0370
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9977 - loss: 0.0082 - val_accuracy: 0.9913 - val_loss: 0.0366
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9976 - loss: 0.0079 - val_accuracy: 0.9904 - val_loss: 0.0377
```

Metrics Visualization

- For creating plots for training and validation accuracy and loss over the epochs



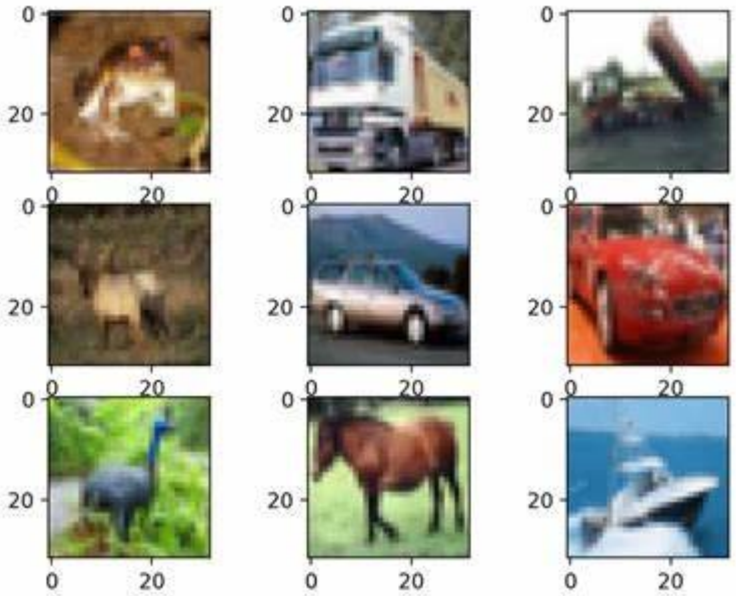


Metrics Visualization

```python
# Plot training and validation metrics
def plot_metrics(history):
    # Plot accuracy
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

Exercise

- You are tasked with building a convolutional neural network (CNN) to classify images from the CIFAR-10 dataset into 10 different classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). You will preprocess input images and visualize some of them, build a CNN architecture, and evaluate its performance through metrics such accuracy and loss.
- To-do list
  - Data Preprocessing
  - Build the CNN Model
  - Train the Model
  - Evaluate and Analyze the Model



Model Training

- For training the model for the specified number of epochs, using the specified optimizer and loss function