```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

# Set random seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# a. Perform Data Pre-processing
# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()

# Normalize pixel values to be between 0 and 1
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0

# Reshape images to include channel dimension (MNIST images are grayscale)
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1)
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1)

# One-hot encode the labels
train_labels = to_categorical(train_labels, 10)
test_labels_one_hot = to_categorical(test_labels, 10)

# Display sample images
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(train_images[i].reshape(28, 28), cmap='gray')
    plt.title(f"Label: {np.argmax(train_labels[i])}")
    plt.axis('off')
plt.tight_layout()
plt.savefig('mnist_samples.png')
plt.close()

# b. Define Model and perform training
def create_model():
    model = models.Sequential([
        # First Convolutional Layer
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        layers.MaxPooling2D((2, 2)),

        # Second Convolutional Layer
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),

        # Third Convolutional Layer
        layers.Conv2D(64, (3, 3), activation='relu'),

        # Flatten and Dense layers
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),  # Dropout for regularization
        layers.Dense(10, activation='softmax')  # 10 output classes for digits 0-9
    ])

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

    return model

# Create and train the model
model = create_model()
model.summary()

# Train the model with validation split
history = model.fit(train_images, train_labels, epochs=10,
                    batch_size=128, validation_split=0.2,
                    verbose=1)

# Plot training history
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.xlabel('Epoch')
```

```
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.tight_layout()
    plt.savefig('training_history.png')
    plt.close()

    # c. Evaluate Results using confusion matrix
    # Evaluate the model
    test_loss, test_acc = model.evaluate(test_images, test_labels_one_hot)
    print(f'Test accuracy: {test_acc:.4f}')

    # Make predictions
    predictions = model.predict(test_images)
    predicted_classes = np.argmax(predictions, axis=1)
    true_classes = np.argmax(test_labels_one_hot, axis=1)

    # Create confusion matrix
    cm = confusion_matrix(true_classes, predicted_classes)
    print("Confusion Matrix:")
    print(cm)

    # Plot confusion matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=list(range(10)),
                yticklabels=list(range(10)))
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.savefig('confusion_matrix.png')
    plt.close()

    # Print classification report
    print("\nClassification Report:")
    print(classification_report(true_classes, predicted_classes))

    # Visualize some misclassified examples
    misclassified_idx = np.where(predicted_classes != true_classes)[0]
    if len(misclassified_idx) > 0:
        plt.figure(figsize=(12, 4))
        for i, idx in enumerate(misclassified_idx[:10]):
            if i < 10:  # Display up to 10 misclassified examples
                plt.subplot(2, 5, i+1)
                plt.imshow(test_images[idx].reshape(28, 28), cmap='gray')
                plt.title(f"True: {true_classes[idx]}, Pred: {predicted_classes[idx]}")
                plt.axis('off')
        plt.tight_layout()
        plt.savefig('misclassified_examples.png')
        plt.close()

    print("Analysis completed.")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shap
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

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, 128)	73,856
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

```
 Total params: 130,890 (511.29 KB)
 Trainable params: 130,890 (511.29 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 43s 109ms/step - accuracy: 0.7614 - loss: 0.7353 - val_accuracy: 0.9766 - val_loss: 0.0753
Epoch 2/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 81s 108ms/step - accuracy: 0.9681 - loss: 0.1077 - val_accuracy: 0.9833 - val_loss: 0.0547
Epoch 3/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 39s 102ms/step - accuracy: 0.9792 - loss: 0.0694 - val_accuracy: 0.9878 - val_loss: 0.0423
Epoch 4/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 103ms/step - accuracy: 0.9843 - loss: 0.0552 - val_accuracy: 0.9887 - val_loss: 0.0414
Epoch 5/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 102ms/step - accuracy: 0.9858 - loss: 0.0442 - val_accuracy: 0.9902 - val_loss: 0.0407
Epoch 6/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 42s 105ms/step - accuracy: 0.9883 - loss: 0.0387 - val_accuracy: 0.9890 - val_loss: 0.0419
Epoch 7/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 45s 117ms/step - accuracy: 0.9892 - loss: 0.0338 - val_accuracy: 0.9902 - val_loss: 0.0374
Epoch 8/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 77s 105ms/step - accuracy: 0.9913 - loss: 0.0284 - val_accuracy: 0.9909 - val_loss: 0.0363
Epoch 9/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 106ms/step - accuracy: 0.9920 - loss: 0.0266 - val_accuracy: 0.9911 - val_loss: 0.0358
Epoch 10/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 41s 107ms/step - accuracy: 0.9940 - loss: 0.0186 - val_accuracy: 0.9918 - val_loss: 0.0359
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9886 - loss: 0.0380
Test accuracy: 0.9917
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
Confusion Matrix:
[[ 976    1    0    0    0    0    1    1    1    0]
 [   0 1132    0    1    0    2    0    0    0    0]
 [   2    2 1024    0    0    0    0    3    1    0]
 [   0    0    1 1005    0    3    0    1    0    0]
 [   0    0    0    0  973    0    3    0    0    6]
 [   1    0    0    3    0  886    1    1    0    0]
 [   2    2    0    0    1    5  947    0    1    0]
 [   0    4    3    4    0    0    0 1016    0    1]
 [   1    0    1    2    0    0    0    3  965    2]
 [   4    1    0    1    2    2    0    4    2  993]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       0.99      1.00      0.99      1135
           2       1.00      0.99      0.99      1032
           3       0.99      1.00      0.99      1010
           4       1.00      0.99      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```