Load the dataset from keras

```
import tensorflow as tf

# Load the dataset
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
```

Starting out

```
import numpy as np

# Initialize arrays for each digit
array_0 = []
array_1 = []
array_2 = []
array_3 = []
array_4 = []
array_5 = []
array_6 = []
array_7 = []
array_8 = []
array_9 = []

# Iterate over each image and label in the training set
for image, label in zip(train_images, train_labels):
    # Reshape the image to a column vector
    image_vector = image.reshape(784, 1)

    # Append the image to the corresponding array based on the label
    if label == 0:
        array_0.append(image_vector)
    elif label == 1:
        array_1.append(image_vector)
    elif label == 2:
        array_2.append(image_vector)
    elif label == 3:
        array_3.append(image_vector)
    elif label == 4:
        array_4.append(image_vector)
    elif label == 5:
        array_5.append(image_vector)
    elif label == 6:
        array_6.append(image_vector)
```

```python
    elif label == 7:
        array_7.append(image_vector)
    elif label == 8:
        array_8.append(image_vector)
    elif label == 9:
        array_9.append(image_vector)

# Convert the lists to NumPy arrays
array_0 = np.hstack(array_0)
array_1 = np.hstack(array_1)
array_2 = np.hstack(array_2)
array_3 = np.hstack(array_3)
array_4 = np.hstack(array_4)
array_5 = np.hstack(array_5)
array_6 = np.hstack(array_6)
array_7 = np.hstack(array_7)
array_8 = np.hstack(array_8)
array_9 = np.hstack(array_9)

# Verify the shapes of the arrays
print("Array shapes:")
print("array_0:", array_0.shape)
print("array_1:", array_1.shape)
print("array_2:", array_2.shape)
print("array_3:", array_3.shape)
print("array_4:", array_4.shape)
print("array_5:", array_5.shape)
print("array_6:", array_6.shape)
print("array_7:", array_7.shape)
print("array_8:", array_8.shape)
print("array_9:", array_9.shape)
```

```
Array shapes:
array_0: (784, 5923)
array_1: (784, 6742)
array_2: (784, 5958)
array_3: (784, 6131)
array_4: (784, 5842)
array_5: (784, 5421)
array_6: (784, 5918)
array_7: (784, 6265)
array_8: (784, 5851)
array_9: (784, 5949)
```

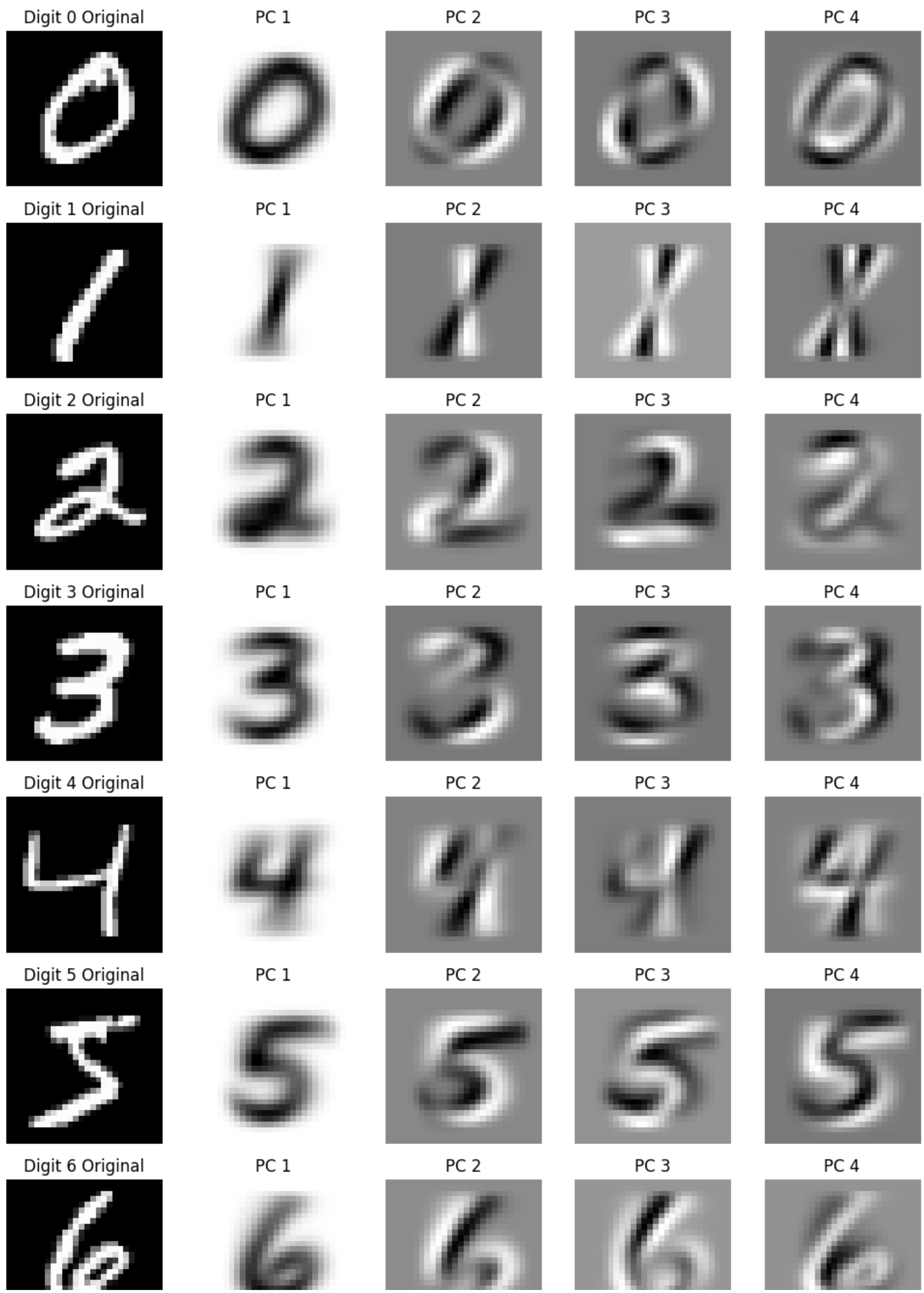
For each of this matrix D(n) write a routine to extract the first 4 left singular vectors

(i.e., the principal components). You may call function provided by numpy. Recall that

these are orthonormal basis vectors of the column space of D(n). Notice that we have used the name array_n in place of D(n).

```
import matplotlib.pyplot as plt

# Prepare to store the singular vectors for each digit
singular_vectors = {}

# Process each digit
for digit in range(10):
    # Get all images of the current digit using boolean filtering
    digit_images = train_images[train_labels == digit]

    # Reshape images to vectors (28*28 = 784)
    digit_vectors = digit_images.reshape(digit_images.shape[0], -1).T

    # Compute the Singular Value Decomposition
    U, S, Vt = np.linalg.svd(digit_vectors, full_matrices=False)

    # Store the first 4 left singular vectors
    singular_vectors[digit] = U[:, :4]

# Display the first image from the training data and the 4 principal
component images for each digit
fig, axs = plt.subplots(10, 5, figsize=(10, 20))

for digit in range(10):
    # Display the first image from the training set
    first_image = train_images[train_labels == digit][0]
    axs[digit, 0].imshow(first_image, cmap='gray')
    axs[digit, 0].title.set_text(f'Digit {digit} Original')
    axs[digit, 0].axis('off')

    # Display the first 4 principal components reshaped as images
```

```
    for i in range(4):
        axs[digit, i+1].imshow(singular_vectors[digit][:,
i].reshape(28, 28), cmap='gray')
        axs[digit, i+1].title.set_text(f'PC {i+1}')
        axs[digit, i+1].axis('off')

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


Digit 0 Original
PC 1
PC 2
PC 3
PC 4
Digit 1 Original
PC 1
PC 2
PC 3
PC 4
Digit 2 Original
PC 1
PC 2
PC 3
PC 4
Digit 3 Original
PC 1
PC 2
PC 3
PC 4
Digit 4 Original
PC 1
PC 2
PC 3
PC 4
Digit 5 Original
PC 1
PC 2
PC 3
PC 4
Digit 6 Original
PC 1
PC 2
PC 3
PC 4

```python
test_images.shape
```

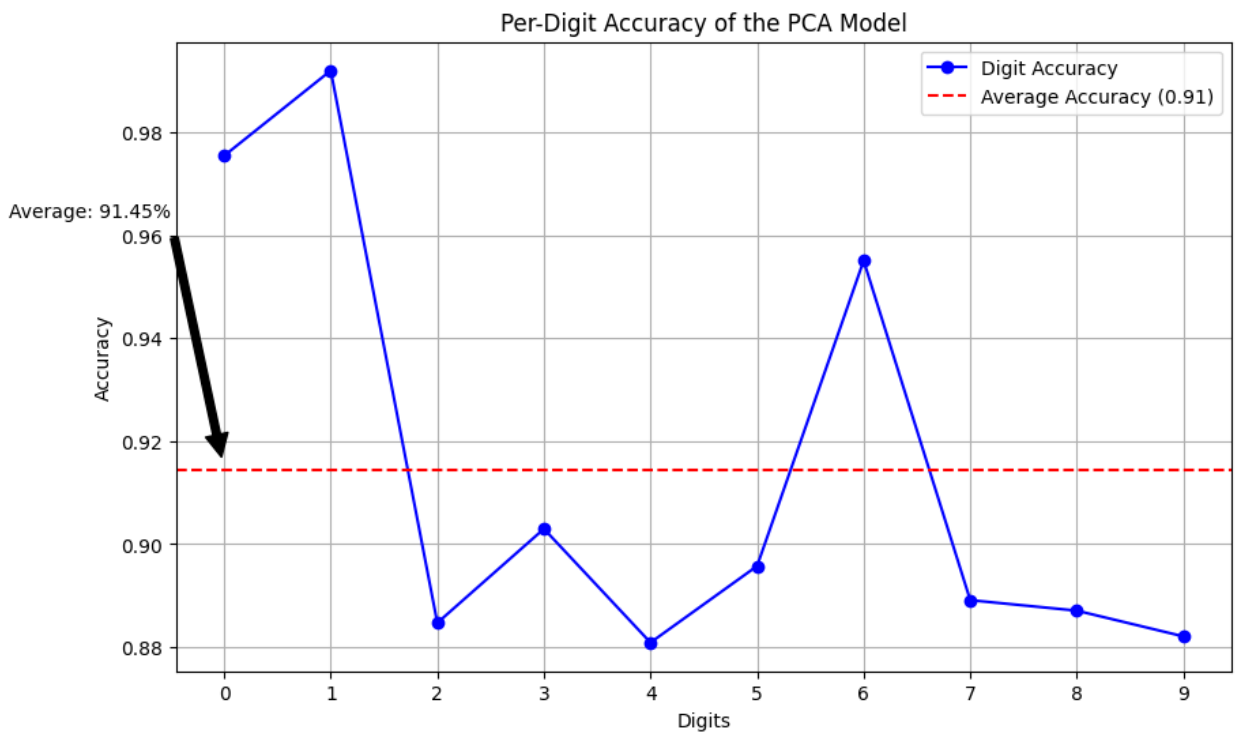
```
(10000, 28, 28)
```

```python
def predict_digit(z, U_matrices):
    residuals = []
    for U in U_matrices:
        # Project z onto the space spanned by the columns of U
        x = np.dot(U.T, z)
        # Calculate the reconstruction from the low-rank approximation
        z_approx = np.dot(U, x)
        # Calculate the norm of the residual
        residual = np.linalg.norm(z_approx - z)
        residuals.append(residual)

    # The predicted digit is the one with the minimum residual
    predicted_digit = np.argmin(residuals)
    return predicted_digit



for digit in range(10):
    digit_images = train_images[train_labels == digit]

    # Reshape images to vectors (28*28 = 784)
    digit_vectors = digit_images.reshape(digit_images.shape[0], -1).T
    # Compute the Singular Value Decomposition
    U, S, Vt = np.linalg.svd(digit_vectors, full_matrices=False)

    if digit==0:
        u_0=U[:,:4]
    elif digit==1:
        u_1=U[:,:4]
    elif digit==2:
        u_2=U[:,:4]
    elif digit==3:
        u_3=U[:,:4]
    elif digit==4:
        u_4=U[:,:4]
    elif digit==5:
        u_5=U[:,:4]
    elif digit==6:
        u_6=U[:,:4]
    elif digit==7:
        u_7=U[:,:4]
    elif digit==8:
        u_8=U[:,:4]
    elif digit==9:
        u_9=U[:,:4]
```

```python
# Collect all U matrices (the first 4 singular vectors for each digit)
into a list
U_matrices = [u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9]
# Reshape and predict each test image
predicted_labels = []
for test_image in test_images:
    # Flatten the test image into a vector
    test_vector = test_image.flatten()

    # Use the predict_digit function to determine the digit
    predicted_digit = predict_digit(test_vector, U_matrices)
    predicted_labels.append(predicted_digit)


# Calculate accuracy
correct_predictions = np.sum(predicted_labels == test_labels)
total_predictions = len(test_labels)
accuracy = correct_predictions / total_predictions
print(f'Accuracy of the PCA model: {accuracy:.2%}')

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix and per-digit accuracy
cm = confusion_matrix(test_labels, predicted_labels)
per_digit_accuracy = cm.diagonal() / cm.sum(axis=1)

# Calculate the overall average accuracy
average_accuracy = np.mean(per_digit_accuracy)

# Plotting per-digit accuracy using a line plot
plt.figure(figsize=(10, 6))
plt.plot(range(10), per_digit_accuracy, marker='o', linestyle='-',
color='b', label='Digit Accuracy')
plt.xlabel('Digits')
plt.ylabel('Accuracy')
plt.title('Per-Digit Accuracy of the PCA Model')
plt.xticks(range(10), [str(digit) for digit in range(10)])
plt.grid(True)

# Draw a horizontal line for the average accuracy
plt.axhline(y=average_accuracy, color='r', linestyle='--',
label=f'Average Accuracy ({average_accuracy:.2f})')
plt.legend()

# Show where the average line cuts the y-axis
plt.annotate(f'Average: {average_accuracy:.2%}',
             xy=(0, average_accuracy),
```

```
             xytext=(-0.5, average_accuracy + 0.05),
             arrowprops=dict(facecolor='black', shrink=0.05),
             horizontalalignment='right',
             verticalalignment='center')

plt.show()
```

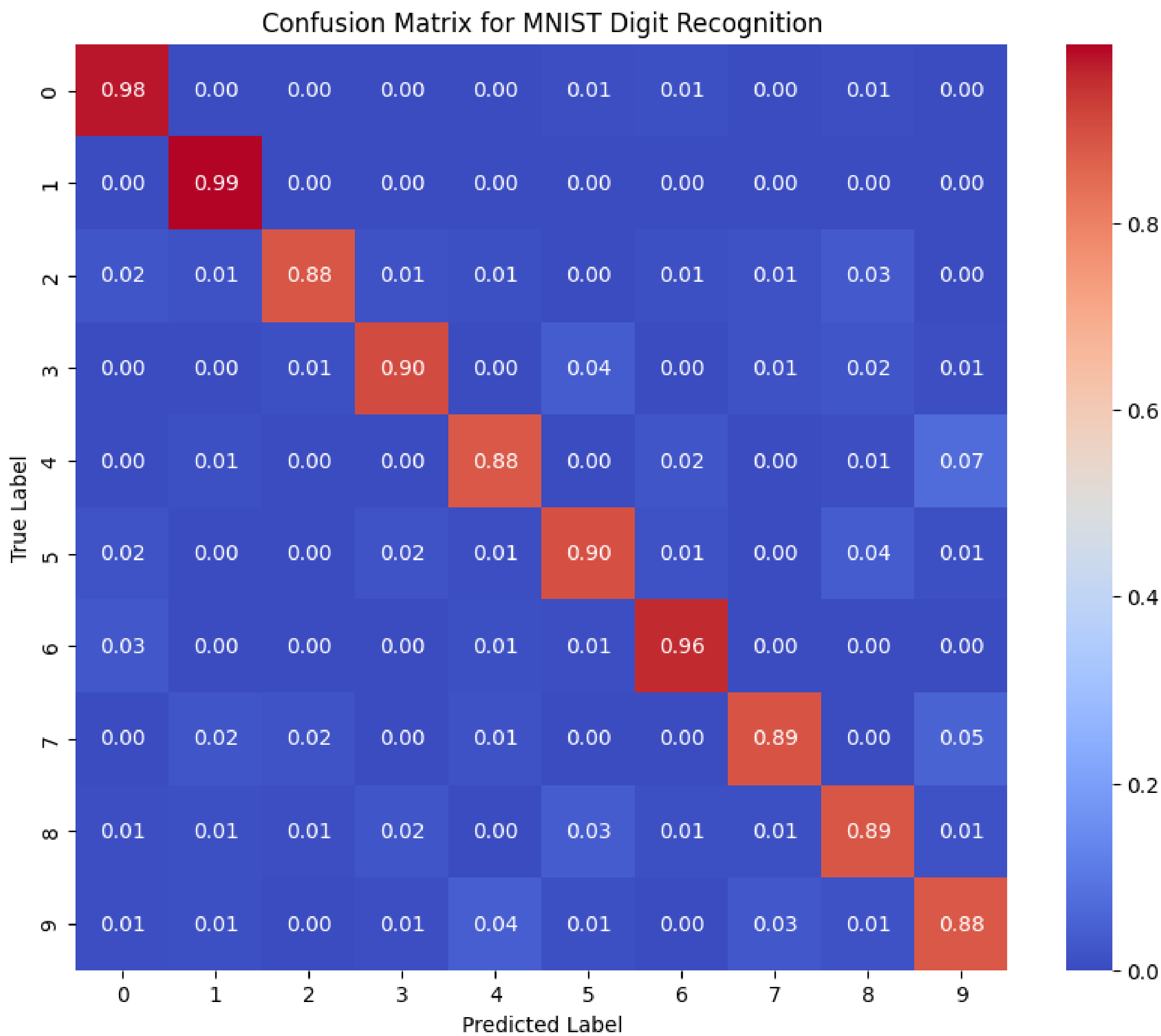
```
Accuracy of the PCA model: 91.54%
```



Confusion matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the confusion matrix
cm = confusion_matrix(test_labels, predicted_labels)

# Normalize the confusion matrix by the number of samples in each
class
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
plt.figure(figsize=(10, 8))
```

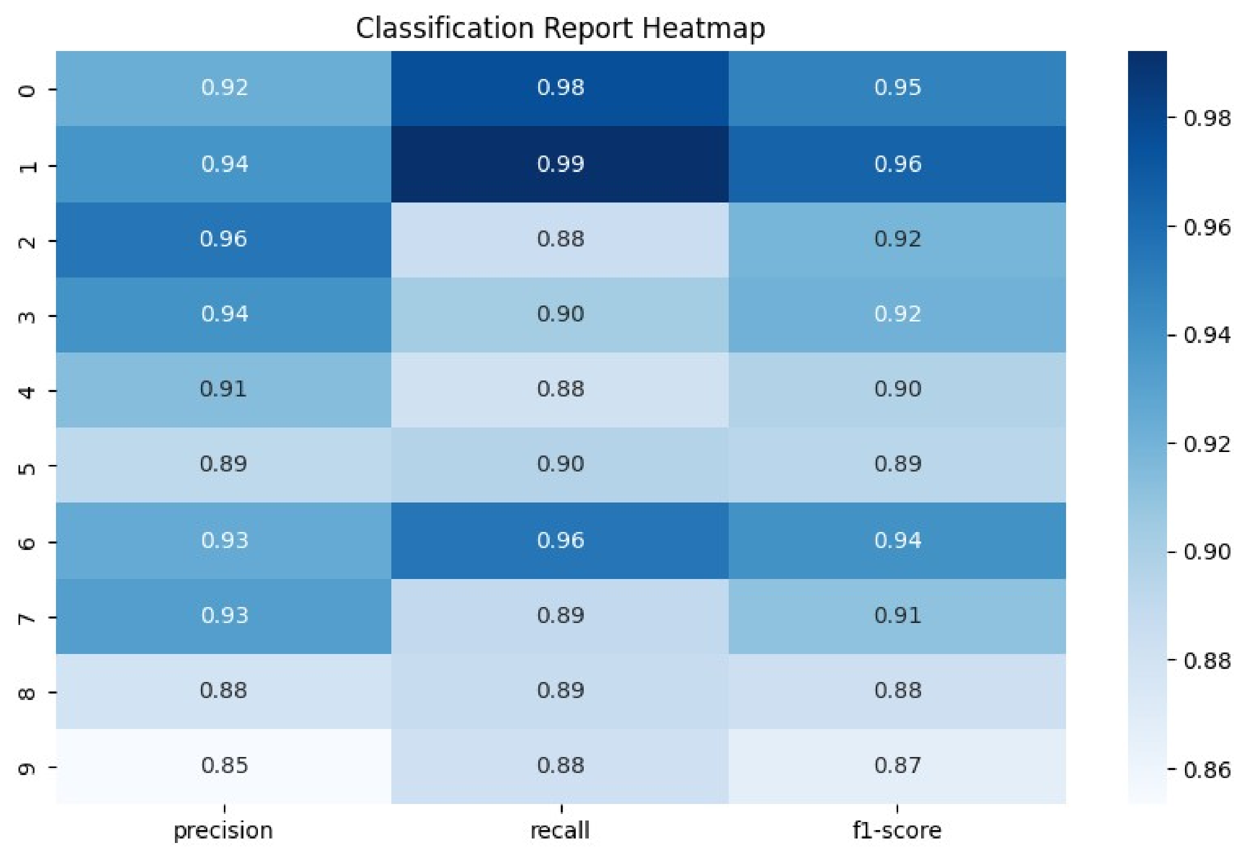
```
sns.heatmap(cm_normalized, annot=True, fmt=".2f", cmap="coolwarm",
cbar=True, xticklabels=range(10), yticklabels=range(10))
plt.title('Confusion Matrix for MNIST Digit Recognition')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Classification report Heatmap

```
from sklearn.metrics import classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

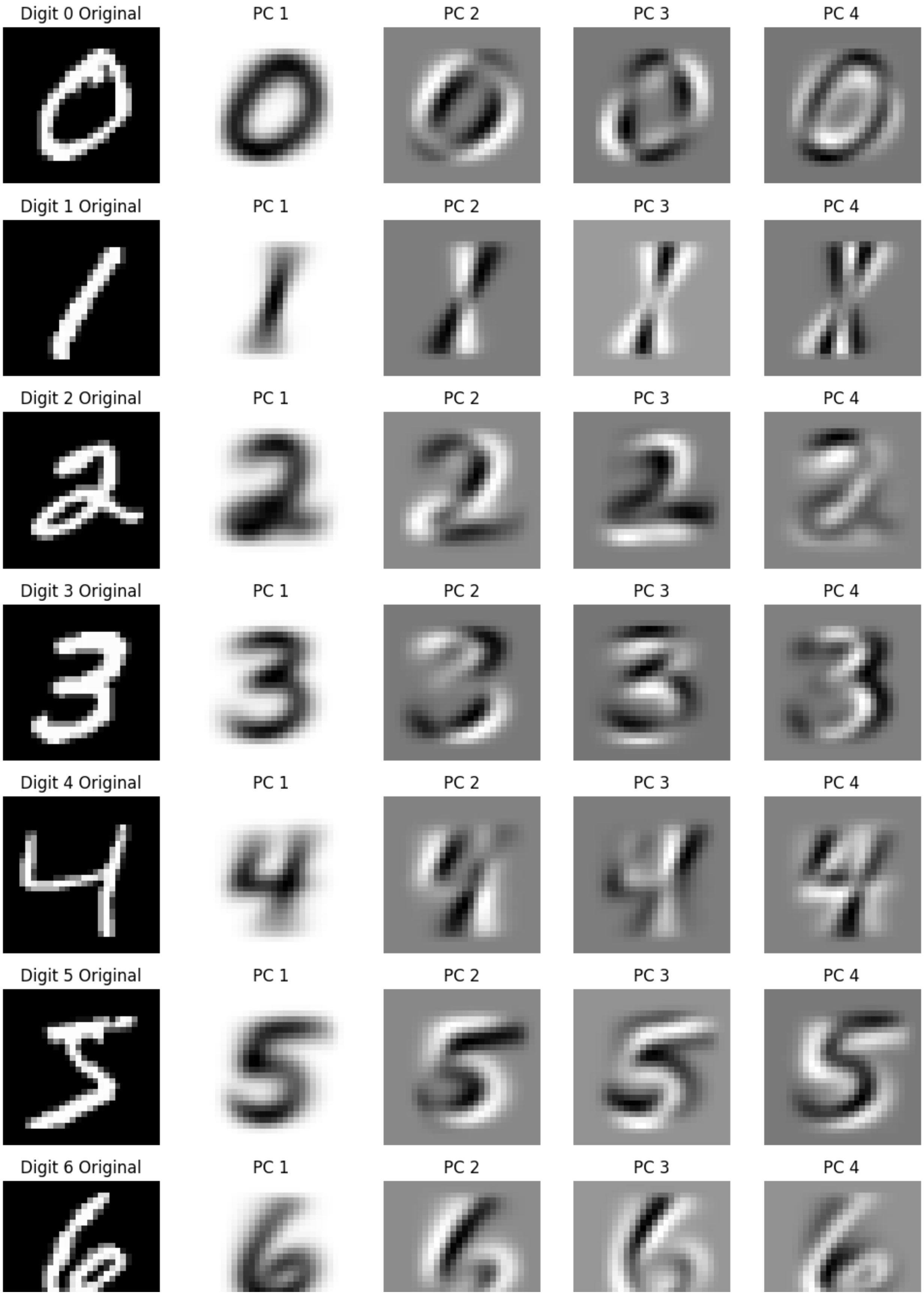
```
# Generate a classification report
report = classification_report(test_labels, predicted_labels,
output_dict=True)

# Convert the report to a DataFrame
df_report = pd.DataFrame(report).transpose()

# Heatmap of the classification report
plt.figure(figsize=(10, 6))
sns.heatmap(df_report.iloc[:-3, :-1], annot=True, cmap="Blues",
fmt=".2f")  # Exclude the last 3 rows and the last column
plt.title('Classification Report Heatmap')
plt.show()
```



Displaying the required 50 images.

```
# Display the first image from the training data and the 4 principal
component images for each digit
fig, axs = plt.subplots(10, 5, figsize=(10, 20))

for digit in range(10):
    # Display the first image from the training set
```

```
    first_image = train_images[train_labels == digit][0]
    axs[digit, 0].imshow(first_image, cmap='gray')
    axs[digit, 0].title.set_text(f'Digit {digit} Original')
    axs[digit, 0].axis('off')

    # Display the first 4 principal components reshaped as images
    for i in range(4):
        axs[digit, i+1].imshow(singular_vectors[digit][:,
i].reshape(28, 28), cmap='gray')
        axs[digit, i+1].title.set_text(f'PC {i+1}')
        axs[digit, i+1].axis('off')

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


Digit 0 Original
PC 1
PC 2
PC 3
PC 4
Digit 1 Original
PC 1
PC 2
PC 3
PC 4
Digit 2 Original
PC 1
PC 2
PC 3
PC 4
Digit 3 Original
PC 1
PC 2
PC 3
PC 4
Digit 4 Original
PC 1
PC 2
PC 3
PC 4
Digit 5 Original
PC 1
PC 2
PC 3
PC 4
Digit 6 Original
PC 1
PC 2
PC 3
PC 4