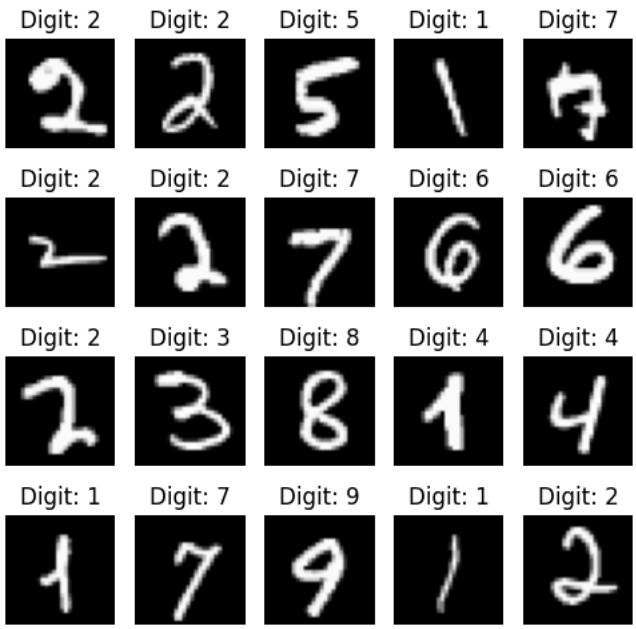
```
Email = "514682@bkbirlacollegekalyan.com"
student_id = 514682
name = "vishal kumar pal"


import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns


# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()


num_images = 20
random_indices = np.random.choice(x_train.shape[0], size=num_images, replace=False)
selected_images = x_train[random_indices]
selected_labels = y_train[random_indices]

# Create a figure with 2x4 subplots
plt.figure(figsize=(5, 5))

for i in range(num_images):
    plt.subplot(4, 5, i + 1)
    plt.imshow(selected_images[i], cmap='gray')
    plt.title(f'Digit: {selected_labels[i]}')
    plt.axis('off')  # Hide axes for clarity

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




```
# Reshape for ANN
x_train_ann = x_train.reshape(-1, 28 * 28)
x_test_ann = x_test.reshape(-1, 28 * 28)

# Reshape for CNN
x_train_cnn = x_train.reshape(-1, 28, 28, 1)
x_test_cnn = x_test.reshape(-1, 28, 28, 1)

y_train_cat = tf.keras.utils.to_categorical(y_train, 10)
y_test_cat = tf.keras.utils.to_categorical(y_test, 10)


print(f"Training data shape (ANN): {x_train_ann.shape}")
print(f"Training data shape (CNN): {x_train_cnn.shape}")
```

```
Training data shape (ANN): (60000, 784)
Training data shape (CNN): (60000, 28, 28, 1)
```

```
# Build ANN model
ann_model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(784,)),
    tf.keras.layers.Dense(128, activation='relu'),
```

```
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

# Train the model
ann_history = ann_model.fit(x_train_ann, y_train_cat, epochs=10, batch_size=32,
                            validation_data=(x_test_ann, y_test_cat), verbose=1)

# Evaluate the model
ann_test_loss, ann_test_acc = ann_model.evaluate(x_test_ann, y_test_cat)
print(f"ANN Test Accuracy: {ann_test_acc:.4f}")
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.8280 - loss: 0.5533 - val_accuracy: 0.9595 - val_loss: 0.1259
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9502 - loss: 0.1698 - val_accuracy: 0.9705 - val_loss: 0.1013
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9641 - loss: 0.1223 - val_accuracy: 0.9728 - val_loss: 0.0838
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9688 - loss: 0.1014 - val_accuracy: 0.9746 - val_loss: 0.0877
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9720 - loss: 0.0902 - val_accuracy: 0.9764 - val_loss: 0.0761
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9753 - loss: 0.0788 - val_accuracy: 0.9761 - val_loss: 0.0782
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9754 - loss: 0.0766 - val_accuracy: 0.9768 - val_loss: 0.0783
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9788 - loss: 0.0666 - val_accuracy: 0.9780 - val_loss: 0.0744
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 9ms/step - accuracy: 0.9804 - loss: 0.0654 - val_accuracy: 0.9786 - val_loss: 0.0738
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 6ms/step - accuracy: 0.9811 - loss: 0.0592 - val_accuracy: 0.9782 - val_loss: 0.0782
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9724 - loss: 0.0991
ANN Test Accuracy: 0.9782
```

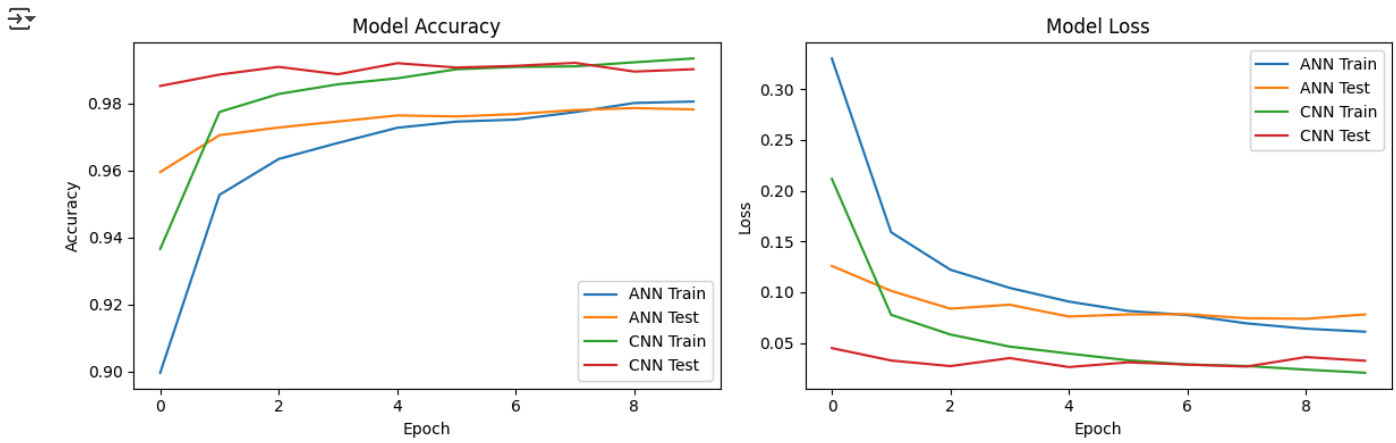
```
# Build CNN model
cnn_model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

# Train the model
cnn_history = cnn_model.fit(x_train_cnn, y_train_cat, epochs=10, batch_size=32,
                            validation_data=(x_test_cnn, y_test_cat), verbose=1)

# Evaluate the model
cnn_test_loss, cnn_test_acc = cnn_model.evaluate(x_test_cnn, y_test_cat)
print(f"CNN Test Accuracy: {cnn_test_acc:.4f}")
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 73s 38ms/step - accuracy: 0.8623 - loss: 0.4435 - val_accuracy: 0.9852 - val_loss: 0.0450
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 69s 37ms/step - accuracy: 0.9764 - loss: 0.0812 - val_accuracy: 0.9886 - val_loss: 0.0328
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 72s 32ms/step - accuracy: 0.9816 - loss: 0.0610 - val_accuracy: 0.9909 - val_loss: 0.0273
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 56s 30ms/step - accuracy: 0.9857 - loss: 0.0475 - val_accuracy: 0.9887 - val_loss: 0.0352
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9879 - loss: 0.0377 - val_accuracy: 0.9920 - val_loss: 0.0263
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 29ms/step - accuracy: 0.9902 - loss: 0.0335 - val_accuracy: 0.9907 - val_loss: 0.0310
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9911 - loss: 0.0278 - val_accuracy: 0.9912 - val_loss: 0.0288
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 85s 31ms/step - accuracy: 0.9914 - loss: 0.0264 - val_accuracy: 0.9921 - val_loss: 0.0268
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 79s 30ms/step - accuracy: 0.9924 - loss: 0.0234 - val_accuracy: 0.9895 - val_loss: 0.0362
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 55s 29ms/step - accuracy: 0.9933 - loss: 0.0195 - val_accuracy: 0.9902 - val_loss: 0.0325
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9878 - loss: 0.0401
```

```
CNN Test Accuracy: 0.9902
```

Start coding or generate with AI.

```
# Plot accuracy and loss curves
plt.figure(figsize=(12, 4))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(ann_history.history['accuracy'], label='ANN Train')
plt.plot(ann_history.history['val_accuracy'], label='ANN Test')
plt.plot(cnn_history.history['accuracy'], label='CNN Train')
plt.plot(cnn_history.history['val_accuracy'], label='CNN Test')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(ann_history.history['loss'], label='ANN Train')
plt.plot(ann_history.history['val_loss'], label='ANN Test')
plt.plot(cnn_history.history['loss'], label='CNN Train')
plt.plot(cnn_history.history['val_loss'], label='CNN Test')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

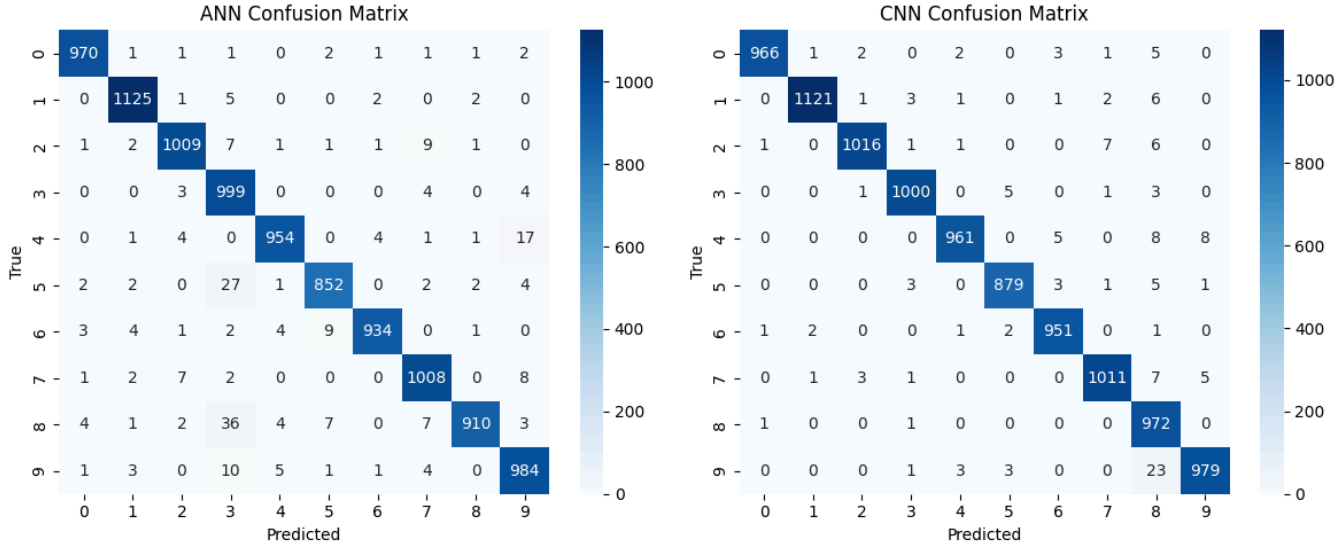


Start coding or generate with AI.

```
# Predict classes
ann_pred = np.argmax(ann_model.predict(x_test_ann), axis=1)
cnn_pred = np.argmax(cnn_model.predict(x_test_cnn), axis=1)

# Confusion matrices
ann_cm = confusion_matrix(y_test, ann_pred)
cnn_cm = confusion_matrix(y_test, cnn_pred)

# Plot confusion matrices
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.heatmap(ann_cm, annot=True, fmt='d', cmap='Blues')
plt.title('ANN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')

plt.subplot(1, 2, 2)
sns.heatmap(cnn_cm, annot=True, fmt='d', cmap='Blues')
plt.title('CNN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
```

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

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 19ms/step
```




```
# Classification reports
print("ANN Classification Report:")
print(classification_report(y_test, ann_pred))

print("CNN Classification Report:")
print(classification_report(y_test, cnn_pred))
```

```
ANN Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.92      0.99      0.95      1010
           4       0.98      0.97      0.98       982
           5       0.98      0.96      0.97       892
           6       0.99      0.97      0.98       958
           7       0.97      0.98      0.98      1028
           8       0.99      0.93      0.96       974
           9       0.96      0.98      0.97      1009

    accuracy                           0.97     10000
   macro avg       0.98      0.97      0.97     10000
weighted avg       0.98      0.97      0.97     10000

CNN Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       980
           1       1.00      0.99      0.99      1135
           2       0.99      0.98      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.98      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.98      0.99      1028
           8       0.94      1.00      0.97       974
           9       0.99      0.97      0.98      1009

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

Start coding or generate with AI.