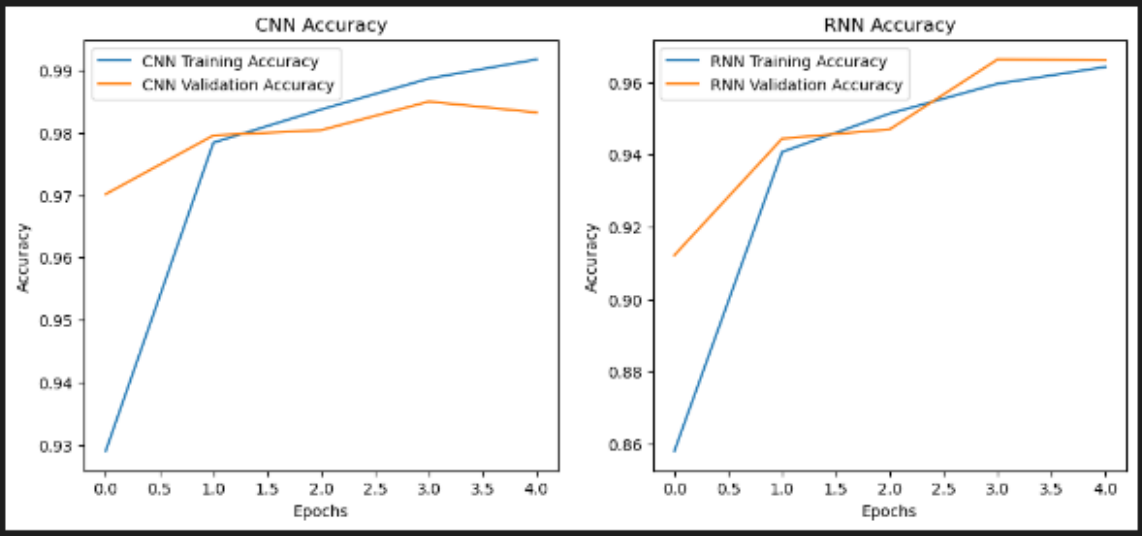
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
SimpleRNN
from tensorflow.keras.datasets import mnist
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape data for CNN (28x28x1) and RNN (28 timesteps, 28 features)
x_train_cnn = x_train.reshape(-1, 28, 28, 1)
x_test_cnn = x_test.reshape(-1, 28, 28, 1)
x_train_rnn = x_train.reshape(-1, 28, 28)
x_test_rnn = x_test.reshape(-1, 28, 28)

# Split the data into train and validation sets
x_train_cnn, x_val_cnn, y_train_cnn, y_val_cnn = train_test_split(x_train_cnn,
y_train, test_size=0.2, random_state=42)
x_train_rnn, x_val_rnn, y_train_rnn, y_val_rnn = train_test_split(x_train_rnn,
y_train, test_size=0.2, random_state=42)

# CNN model
cnn_model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28,
1)),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
cnn_history = cnn_model.fit(x_train_cnn, y_train_cnn,
validation_data=(x_val_cnn, y_val_cnn), epochs=5, batch_size=128)

# RNN model
rnn_model = Sequential([
    SimpleRNN(128, input_shape=(28, 28)),
    Dense(10, activation='softmax')
])
```

```
rnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
rnn_history = rnn_model.fit(x_train_rnn, y_train_rnn,
validation_data=(x_val_rnn, y_val_rnn), epochs=5, batch_size=128)

# Evaluate both models on the test set
cnn_test_loss, cnn_test_acc = cnn_model.evaluate(x_test_cnn, y_test)
rnn_test_loss, rnn_test_acc = rnn_model.evaluate(x_test_rnn, y_test)

# Print the results
print(f"CNN Test Accuracy: {cnn_test_acc}")
print(f"RNN Test Accuracy: {rnn_test_acc}")

# Plot the training history
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(cnn_history.history['accuracy'], label='CNN Training Accuracy')
plt.plot(cnn_history.history['val_accuracy'], label='CNN Validation Accuracy')
plt.title('CNN Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(rnn_history.history['accuracy'], label='RNN Training Accuracy')
plt.plot(rnn_history.history['val_accuracy'], label='RNN Validation Accuracy')
plt.title('RNN Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

```
Epoch 1/5
C:\Users\u21629545\Anaconda3\lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`
  super().__init__(
375/375 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.8620 - loss: 0.4812 - val_accuracy: 0.9702 - val_loss: 0.1038
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step - accuracy: 0.9766 - loss: 0.0869 - val_accuracy: 0.9796 - val_loss: 0.0688
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - accuracy: 0.9840 - loss: 0.0542 - val_accuracy: 0.9804 - val_loss: 0.0649
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - accuracy: 0.9887 - loss: 0.0371 - val_accuracy: 0.9850 - val_loss: 0.0492
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - accuracy: 0.9922 - loss: 0.0272 - val_accuracy: 0.9833 - val_loss: 0.0548
Epoch 1/5
C:\Users\u21629545\Anaconda3\lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argu
  super().__init__(**kwargs)
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.7498 - loss: 0.8053 - val_accuracy: 0.9122 - val_loss: 0.2881
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9365 - loss: 0.2196 - val_accuracy: 0.9444 - val_loss: 0.1884
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9508 - loss: 0.1680 - val_accuracy: 0.9470 - val_loss: 0.1832
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9591 - loss: 0.1411 - val_accuracy: 0.9663 - val_loss: 0.1200
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9638 - loss: 0.1214 - val_accuracy: 0.9661 - val_loss: 0.1223
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9764 - loss: 0.0700
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9578 - loss: 0.1454
CNN Test Accuracy: 0.982699990272522
RNN Test Accuracy: 0.9650999903678894
```

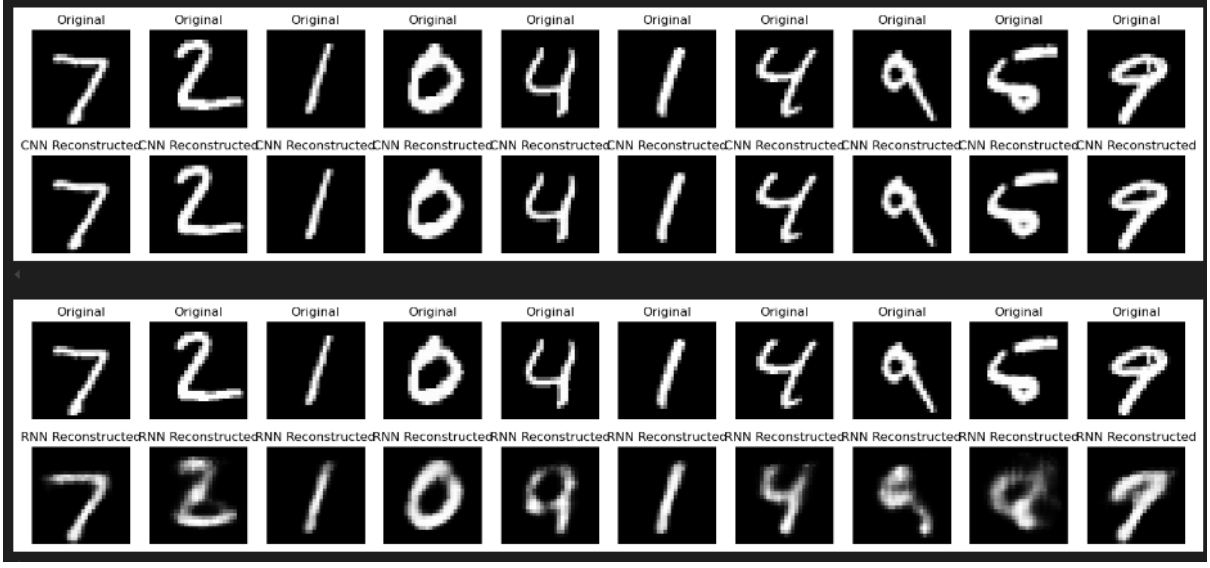



```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
UpSampling2D, Reshape, SimpleRNN, TimeDistributed, RepeatVector
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape data for CNN (28x28x1) and RNN (28 timesteps, 28 features)
x_train_cnn = x_train.reshape(-1, 28, 28, 1)
x_test_cnn = x_test.reshape(-1, 28, 28, 1)
```

```
x_train_rnn = x_train.reshape(-1, 28, 28)
x_test_rnn = x_test.reshape(-1, 28, 28)

# CNN autoencoder model
cnn_autoencoder = Sequential([
    # Encoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2), padding='same'),
    # Decoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    UpSampling2D(size=(2, 2)),
    Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same')
])

cnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
cnn_autoencoder.fit(x_train_cnn, x_train_cnn, epochs=5, batch_size=128,
validation_split=0.2)

# RNN autoencoder model
rnn_autoencoder = Sequential([
    # Encoder
    SimpleRNN(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    SimpleRNN(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(28, activation='sigmoid'))
])

rnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
rnn_autoencoder.fit(x_train_rnn, x_train_rnn, epochs=5, batch_size=128,
validation_split=0.2)

# Reconstruct images using both models
cnn_reconstructed = cnn_autoencoder.predict(x_test_cnn)
rnn_reconstructed = rnn_autoencoder.predict(x_test_rnn)

# Plot original and reconstructed images
n = 10  # number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title("Original")
    plt.axis('off')
```

```
    # Display CNN reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(cnn_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("CNN Reconstructed")
    plt.axis('off')

plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title("Original")
    plt.axis('off')

    # Display RNN reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(rnn_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("RNN Reconstructed")
    plt.axis('off')

plt.show()
```

```
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 11s 26ms/step - loss: 0.2188 - val_loss: 0.0678
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 10s 27ms/step - loss: 0.0664 - val_loss: 0.0653
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - loss: 0.0645 - val_loss: 0.0643
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 25ms/step - loss: 0.0637 - val_loss: 0.0636
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - loss: 0.0630 - val_loss: 0.0631
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 15s 34ms/step - loss: 0.3249 - val_loss: 0.2086
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 31ms/step - loss: 0.1961 - val_loss: 0.1693
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 31ms/step - loss: 0.1645 - val_loss: 0.1519
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 31ms/step - loss: 0.1492 - val_loss: 0.1421
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 31ms/step - loss: 0.1407 - val_loss: 0.1351
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```



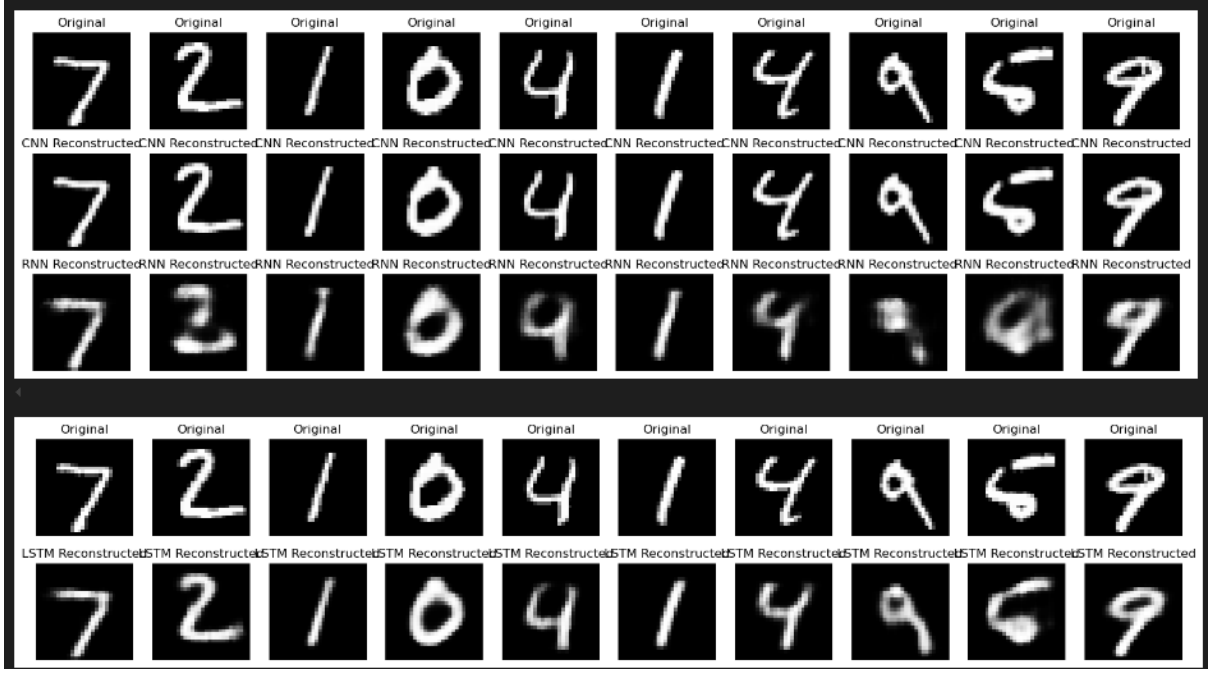
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
UpSampling2D, Reshape, SimpleRNN, LSTM, TimeDistributed, RepeatVector
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape data for CNN (28x28x1), RNN and LSTM (28 timesteps, 28 features)
x_train_cnn = x_train.reshape(-1, 28, 28, 1)
x_test_cnn = x_test.reshape(-1, 28, 28, 1)
x_train_rnn = x_train.reshape(-1, 28, 28)
x_test_rnn = x_test.reshape(-1, 28, 28)

# CNN autoencoder model
cnn_autoencoder = Sequential([
    # Encoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2), padding='same'),
    # Decoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    UpSampling2D(size=(2, 2)),
    Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same')
])

cnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
cnn_autoencoder.fit(x_train_cnn, x_train_cnn, epochs=5, batch_size=128,
validation_split=0.2)

# RNN autoencoder model
rnn_autoencoder = Sequential([
    # Encoder
    SimpleRNN(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    SimpleRNN(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(28, activation='sigmoid'))
])
```

```
rnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
rnn_autoencoder.fit(x_train_rnn, x_train_rnn, epochs=5, batch_size=128,
validation_split=0.2)

# LSTM autoencoder model
lstm_autoencoder = Sequential([
    # Encoder
    LSTM(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    LSTM(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(28, activation='sigmoid'))
])

lstm_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
lstm_autoencoder.fit(x_train_rnn, x_train_rnn, epochs=5, batch_size=128,
validation_split=0.2)

# Reconstruct images using all models
cnn_reconstructed = cnn_autoencoder.predict(x_test_cnn)
rnn_reconstructed = rnn_autoencoder.predict(x_test_rnn)
lstm_reconstructed = lstm_autoencoder.predict(x_test_rnn)

# Plot original and reconstructed images
n = 10  # number of images to display
plt.figure(figsize=(20, 6))

for i in range(n):
    # Display original
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title("Original")
    plt.axis('off')

    # Display CNN reconstructed
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(cnn_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("CNN Reconstructed")
    plt.axis('off')

    # Display RNN reconstructed
    ax = plt.subplot(3, n, i + 1 + 2 * n)
    plt.imshow(rnn_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("RNN Reconstructed")
    plt.axis('off')

plt.figure(figsize=(20, 4))
```

```python
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title("Original")
    plt.axis('off')

    # Display LSTM reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(lstm_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("LSTM Reconstructed")
    plt.axis('off')

plt.show()
```

```
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 11s 26ms/step - loss: 0.2301 - val_loss: 0.0676
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - loss: 0.0663 - val_loss: 0.0654
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - loss: 0.0646 - val_loss: 0.0642
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - loss: 0.0635 - val_loss: 0.0635
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - loss: 0.0630 - val_loss: 0.0631
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 15s 33ms/step - loss: 0.3157 - val_loss: 0.1926
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 32ms/step - loss: 0.1831 - val_loss: 0.1646
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 32ms/step - loss: 0.1597 - val_loss: 0.1518
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 32ms/step - loss: 0.1473 - val_loss: 0.1413
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 32ms/step - loss: 0.1401 - val_loss: 0.1401
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 42s 105ms/step - loss: 0.3565 - val_loss: 0.1988
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 39s 104ms/step - loss: 0.1837 - val_loss: 0.1457
Epoch 3/5
...
375/375 ━━━━━━━━━━━━━━━━━━━━ 39s 104ms/step - loss: 0.1101 - val_loss: 0.1042
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...




```python
import numpy as np
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
UpSampling2D, Reshape, SimpleRNN, LSTM, TimeDistributed, RepeatVector
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train_cnn = x_train.reshape(-1, 28, 28, 1)
x_test_cnn = x_test.reshape(-1, 28, 28, 1)
x_train_rnn = x_train.reshape(-1, 28, 28)
x_test_rnn = x_test.reshape(-1, 28, 28)

# Define CNN autoencoder model
cnn_autoencoder = Sequential([
    # Encoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2), padding='same'),
    # Decoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    UpSampling2D(size=(2, 2)),
    Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same')
])

# Define RNN autoencoder model
rnn_autoencoder = Sequential([
    # Encoder
    SimpleRNN(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    SimpleRNN(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(28, activation='sigmoid'))
])

# Define LSTM autoencoder model
lstm_autoencoder = Sequential([
    # Encoder
    LSTM(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    LSTM(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(28, activation='sigmoid'))
```

```
])

# Compile models (no need to fit to display summaries)
cnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
rnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
lstm_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Print summaries of the models
print("CNN Autoencoder Summary:")
cnn_autoencoder.summary()

print("\nRNN Autoencoder Summary:")
rnn_autoencoder.summary()

print("\nLSTM Autoencoder Summary:")
lstm_autoencoder.summary()
```

CNN Autoencoder Summary:

Model: "sequential_7"

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_8 (Conv2D)	(None, 14, 14, 32)	9,248
up_sampling2d_2 (UpSampling2D)	(None, 28, 28, 32)	0
conv2d_9 (Conv2D)	(None, 28, 28, 1)	289

Total params: 9,857 (38.50 KB)

Trainable params: 9,857 (38.50 KB)

Non-trainable params: 0 (0.00 B)

RNN Autoencoder Summary:

Model: "sequential_8"

Layer (type)	Output Shape	Param #
simple_rnn_5 (SimpleRNN)	(None, 128)	20,096
repeat_vector_3 (RepeatVector)	(None, 28, 128)	0
simple_rnn_6 (SimpleRNN)	(None, 28, 128)	32,896
time_distributed_3 (TimeDistributed)	(None, 28, 28)	3,612

Total params: 56,604 (221.11 KB)

Trainable params: 56,604 (221.11 KB)

Non-trainable params: 0 (0.00 B)

LSTM Autoencoder Summary:

Model: "sequential_9"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 128)	80,384
repeat_vector_4 (RepeatVector)	(None, 28, 128)	0
lstm_3 (LSTM)	(None, 28, 128)	131,584
time_distributed_4 (TimeDistributed)	(None, 28, 28)	3,612

Total params: 215,580 (842.11 KB)

Trainable params: 215,580 (842.11 KB)

Non-trainable params: 0 (0.00 B)