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()



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()

A screenshot of a computer

Description automatically generated

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))

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()

A screenshot of a computer

Description automatically generated

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()

A screenshot of a computer

Description automatically generated