Bài tập intsyst01

I. Mô hình DL với tập dữ liệu MNIST

1. DL

Bước 1: Import các thư viện cần thiết

```
[ ]: import tensorflow as tf
     from tensorflow.keras import layers, models
     import matplotlib.pyplot as plt
     import numpy as np
     from sklearn.metrics import mean_absolute_error, mean_squared_error
```

Bước 2: Thực hiện tải bộ dữ liệu Mnist

```
[ ]: # Tải dữ liệu MNIST
     mnist = tf.keras.datasets.mnist
     (x_train, y_train), (x_test, y_test) = mnist.load_data()

     # Chuẩn hóa dữ liệu
     x_train, x_test = x_train / 255.0, x_test / 255.0

     # Thêm kênh (1 kênh cho ảnh xám)
     x_train = x_train[..., tf.newaxis]
     x_test = x_test[..., tf.newaxis]
```

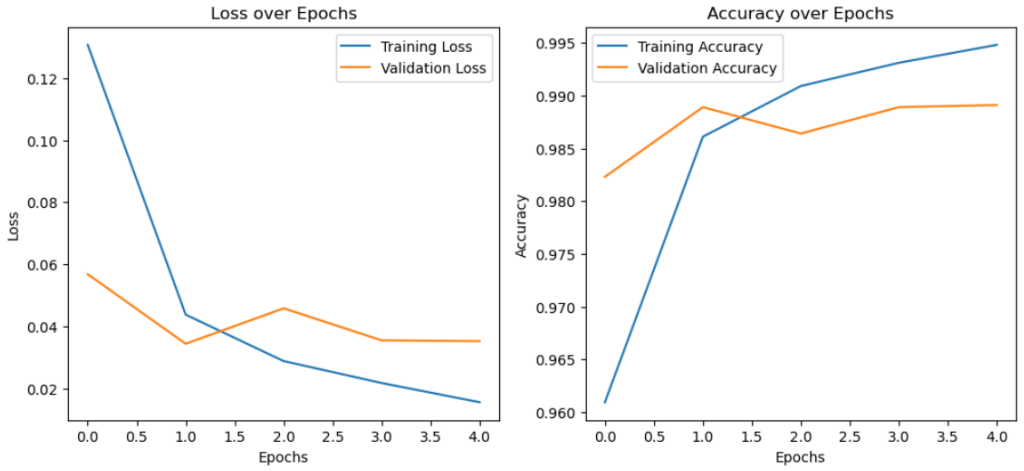
Bước 3:Xây dựng và huấn luyện mô hình

```
[11]: # Xây dựng mô hình CNN
      model_mnist = models.Sequential([
          layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),
          layers.MaxPooling2D(pool_size=(2, 2)),
          layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
          layers.MaxPooling2D(pool_size=(2, 2)),
          layers.Flatten(),
          layers.Dense(128, activation='relu'),
          layers.Dense(10, activation='softmax')  # 10 lớp cho các chữ số từ 0 đến 9
      ])

      # Compile mô hình
      model_mnist.compile(optimizer='adam',
                          loss='sparse_categorical_crossentropy',
                          metrics=['accuracy'])

      # Huấn luyện mô hình
      history = model_mnist.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

```
D:\App\Anaconda\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 33s 15ms/step - accuracy: 0.9141 - loss: 0.2908 - val_accuracy: 0.9823 - val_loss: 0.0568
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 26s 14ms/step - accuracy: 0.9855 - loss: 0.0454 - val_accuracy: 0.9889 - val_loss: 0.0344
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 27s 14ms/step - accuracy: 0.9908 - loss: 0.0290 - val_accuracy: 0.9864 - val_loss: 0.0458
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 27s 14ms/step - accuracy: 0.9929 - loss: 0.0224 - val_accuracy: 0.9889 - val_loss: 0.0355
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 26s 14ms/step - accuracy: 0.9953 - loss: 0.0138 - val_accuracy: 0.9891 - val_loss: 0.0353
```

Bước 4: Vẽ biểu đồ thể hiện quan hệ của accuracy với loss

```
[12]:
plt.figure(figsize=(12, 5))

# Biểu đồ Loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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

plt.show()
```

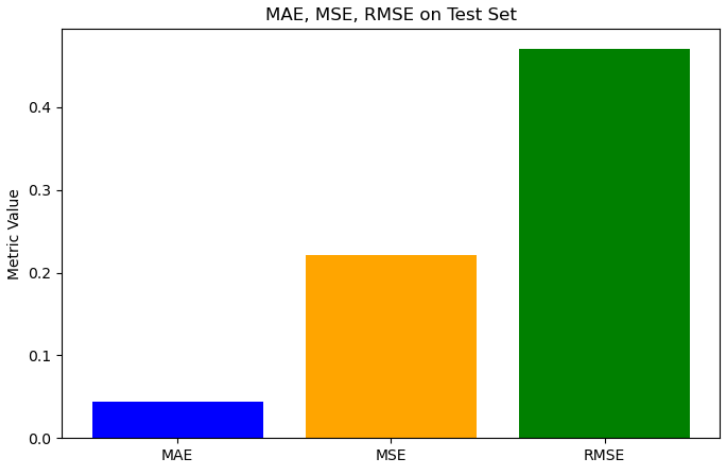


Bước 5: Dự đoán kết quả dựa trên model đã train và đưa ra độ đo mae, mse, rmse

```
[13]:
# Đánh giá mô hình trên tập test
test_loss, test_accuracy = model_mnist.evaluate(x_test, y_test)

# Dự đoán nhãn trên tập test
y_pred = model_mnist.predict(x_test)
y_pred_labels = np.argmax(y_pred, axis=1)

# Tính MAE, MSE, RMSE
mae = mean_absolute_error(y_test, y_pred_labels)
mse = mean_squared_error(y_test, y_pred_labels)
rmse = np.sqrt(mse)

# In kết quả
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")

313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9866 - loss: 0.0407
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
Test Loss: 0.035267919301986694
Test Accuracy: 0.9890999794006348
Mean Absolute Error (MAE): 0.0436
Mean Squared Error (MSE): 0.2218
Root Mean Squared Error (RMSE): 0.4709564735726647
```

Bước 6. Vẽ sơ đồ thể hiện sự tương quan giữa các thông số mae, rmse, mse

```
[14]: # Tính toán MAE, MSE, RMSE đã có ở phần trước
metrics = [mae, mse, rmse]
metric_names = ['MAE', 'MSE', 'RMSE']

# Vẽ biểu đồ
plt.figure(figsize=(8, 5))
plt.bar(metric_names, metrics, color=['blue', 'orange', 'green'])
plt.title('MAE, MSE, RMSE on Test Set')
plt.ylabel('Metric Value')
plt.show()
```



2. CNN

Bước 1: Import các thư viện cần thiết

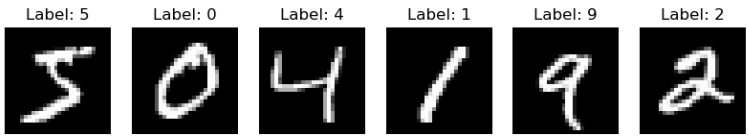
```
[ ]: import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
```

Bước 2: Load dữ liệu

```
*[37]: # Load MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

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

# One-hot encode labels
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Bước 3: Hiển thị một số tập dữ liệu mẫu

[38]:
```
import matplotlib.pyplot as plt

# Display the first 5 images and their labels
num_images = 6
plt.figure(figsize=(10, 5))
for i in range(num_images):
    plt.subplot(1, num_images, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {np.argmax(y_train[i])}")
    plt.axis('off')
plt.show()
```



Bước 4: Xây dựng và huấn luyện mô hình CNN

[39]:
```
def build_cnn():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Reshape the data for CNN input
cnn_x_train = x_train.reshape(-1, 28, 28, 1)
cnn_x_test = x_test.reshape(-1, 28, 28, 1)

# Build and train CNN model
cnn_model = build_cnn()
cnn_model.fit(cnn_x_train, y_train, epochs=5, validation_data=(cnn_x_test, y_test))
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 40s 17ms/step - accuracy: 0.8861 - loss: 0.3531 - val_accuracy: 0.9836 - val_loss: 0.0522
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 39s 16ms/step - accuracy: 0.9842 - loss: 0.0514 - val_accuracy: 0.9888 - val_loss: 0.0348
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 26s 14ms/step - accuracy: 0.9885 - loss: 0.0347 - val_accuracy: 0.9912 - val_loss: 0.0251
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 27s 14ms/step - accuracy: 0.9925 - loss: 0.0234 - val_accuracy: 0.9884 - val_loss: 0.0391
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 33s 18ms/step - accuracy: 0.9939 - loss: 0.0198 - val_accuracy: 0.9925 - val_loss: 0.0252
```

[39]:
```
<keras.src.callbacks.history.History at 0x17b46defce0>
```

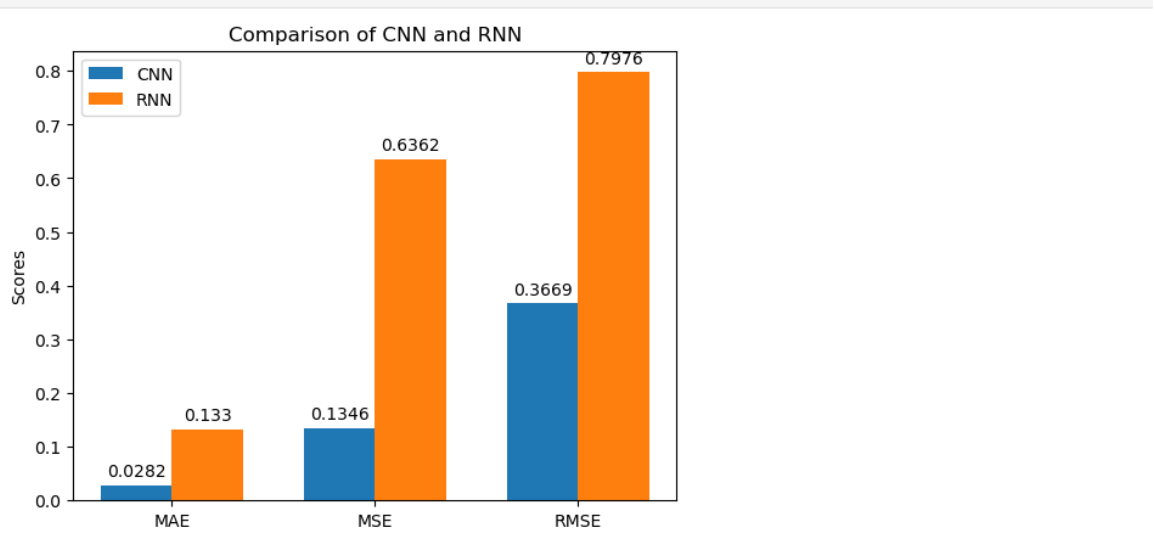
Bước 5: Hiển thị độ đo mae, rmse, mse

[41]:
```
# Predict using both models
cnn_predictions = cnn_model.predict(cnn_x_test)
rnn_predictions = rnn_model.predict(x_test)

# Convert predictions and true labels back to categorical labels
cnn_pred_labels = np.argmax(cnn_predictions, axis=1)
rnn_pred_labels = np.argmax(rnn_predictions, axis=1)
true_labels = np.argmax(y_test, axis=1)

# Calculate MAE, MSE, RMSE for CNN
cnn_mae = mean_absolute_error(true_labels, cnn_pred_labels)
cnn_mse = mean_squared_error(true_labels, cnn_pred_labels)
cnn_rmse = np.sqrt(cnn_mse)

# Calculate MAE, MSE, RMSE for RNN
rnn_mae = mean_absolute_error(true_labels, rnn_pred_labels)
rnn_mse = mean_squared_error(true_labels, rnn_pred_labels)
rnn_rmse = np.sqrt(rnn_mse)

# Print the results
print("CNN - MAE: {:.4f}, MSE: {:.4f}, RMSE: {:.4f}".format(cnn_mae, cnn_mse, cnn_rmse))
print("RNN - MAE: {:.4f}, MSE: {:.4f}, RMSE: {:.4f}".format(rnn_mae, rnn_mse, rnn_rmse))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step
CNN - MAE: 0.0282, MSE: 0.1346, RMSE: 0.3669
RNN - MAE: 0.1330, MSE: 0.6362, RMSE: 0.7976
```

Bước 6 : Vẽ biểu đồ thể hiện

[42]:
```
# Bar chart to compare MAE, MSE, and RMSE
metrics = ['MAE', 'MSE', 'RMSE']
cnn_scores = [cnn_mae, cnn_mse, cnn_rmse]
rnn_scores = [rnn_mae, rnn_mse, rnn_rmse]

x = np.arange(len(metrics))
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, cnn_scores, width, label='CNN')
rects2 = ax.bar(x + width/2, rnn_scores, width, label='RNN')

ax.set_ylabel('Scores')
ax.set_title('Comparison of CNN and RNN')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()

# Add value labels on bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height, 4)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

plt.show()
```



3. RNN

Bước 1: Import các thư viện cần thiết

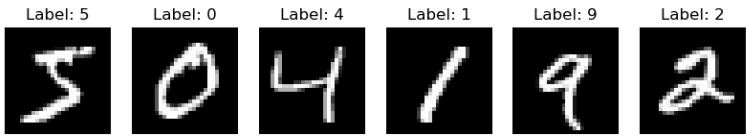
```
[ ]: import tensorflow as tf
     from tensorflow.keras import layers, models
     import numpy as np
     from sklearn.metrics import mean_absolute_error, mean_squared_error
     import matplotlib.pyplot as plt
```

Bước 2: Load dữ liệu

```
[37]: # Load MNIST dataset
      (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

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

      # One-hot encode labels
      y_train = tf.keras.utils.to_categorical(y_train, 10)
      y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Bước 3: Hiển thị một số tập dữ liệu mẫu

[38]:
```
import matplotlib.pyplot as plt

# Display the first 5 images and their labels
num_images = 6
plt.figure(figsize=(10, 5))
for i in range(num_images):
    plt.subplot(1, num_images, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {np.argmax(y_train[i])}")
    plt.axis('off')
plt.show()
```



Bước 4: Xây dựng và huấn luyện mô hình RNN

[40]:
```
def build_rnn():
    model = models.Sequential()
    model.add(layers.SimpleRNN(128, input_shape=(28, 28), return_sequences=True))
    model.add(layers.SimpleRNN(128))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Build and train RNN model
rnn_model = build_rnn()
rnn_model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

```
D:\App\Anaconda\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 55s 23ms/step - accuracy: 0.8369 - loss: 0.5212 - val_accuracy: 0.9484 - val_loss: 0.1756
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 38s 20ms/step - accuracy: 0.9411 - loss: 0.1967 - val_accuracy: 0.9426 - val_loss: 0.2024
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 39s 21ms/step - accuracy: 0.9462 - loss: 0.1844 - val_accuracy: 0.9585 - val_loss: 0.1411
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 37s 20ms/step - accuracy: 0.9484 - loss: 0.1778 - val_accuracy: 0.9468 - val_loss: 0.1854
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 37s 20ms/step - accuracy: 0.9482 - loss: 0.1834 - val_accuracy: 0.9638 - val_loss: 0.1283
```

[40]:
```
<keras.src.callbacks.history.History at 0x17b4a9cb890>
```

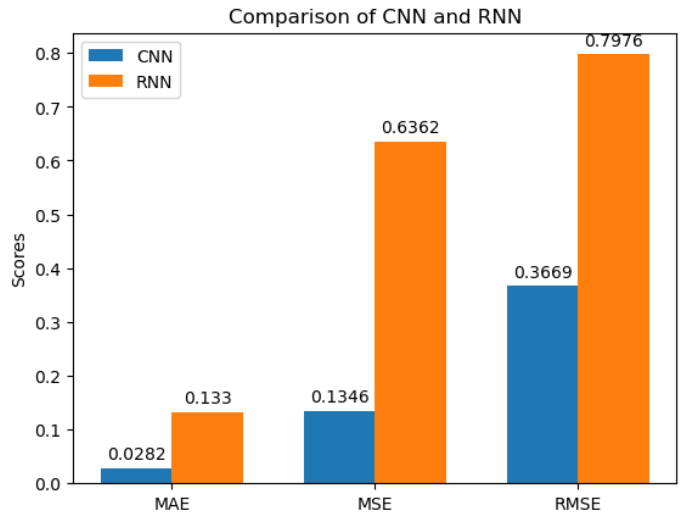
Bước 5: Hiển thị độ đo mae, rmse, mse

[41]:
```
# Predict using both models
cnn_predictions = cnn_model.predict(cnn_x_test)
rnn_predictions = rnn_model.predict(x_test)

# Convert predictions and true labels back to categorical labels
cnn_pred_labels = np.argmax(cnn_predictions, axis=1)
rnn_pred_labels = np.argmax(rnn_predictions, axis=1)
true_labels = np.argmax(y_test, axis=1)

# Calculate MAE, MSE, RMSE for CNN
cnn_mae = mean_absolute_error(true_labels, cnn_pred_labels)
cnn_mse = mean_squared_error(true_labels, cnn_pred_labels)
cnn_rmse = np.sqrt(cnn_mse)

# Calculate MAE, MSE, RMSE for RNN
rnn_mae = mean_absolute_error(true_labels, rnn_pred_labels)
rnn_mse = mean_squared_error(true_labels, rnn_pred_labels)
rnn_rmse = np.sqrt(rnn_mse)

# Print the results
print("CNN - MAE: {:.4f}, MSE: {:.4f}, RMSE: {:.4f}".format(cnn_mae, cnn_mse, cnn_rmse))
print("RNN - MAE: {:.4f}, MSE: {:.4f}, RMSE: {:.4f}".format(rnn_mae, rnn_mse, rnn_rmse))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step
CNN - MAE: 0.0282, MSE: 0.1346, RMSE: 0.3669
RNN - MAE: 0.1330, MSE: 0.6362, RMSE: 0.7976
```

Bước 6 : Vẽ biểu đồ thể hiện

```
[42]: # Bar chart to compare MAE, MSE, and RMSE
metrics = ['MAE', 'MSE', 'RMSE']
cnn_scores = [cnn_mae, cnn_mse, cnn_rmse]
rnn_scores = [rnn_mae, rnn_mse, rnn_rmse]

x = np.arange(len(metrics))
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, cnn_scores, width, label='CNN')
rects2 = ax.bar(x + width/2, rnn_scores, width, label='RNN')

ax.set_ylabel('Scores')
ax.set_title('Comparison of CNN and RNN')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()

# Add value labels on bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height, 4)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

plt.show()
```



II. Mô hình DL với tập dữ liệu Imdb

Import các thư viện cần thiết

```
[1]: import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
```

Load bộ dữ liệu Imdb

```python
# Load IMDB dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.imdb.load_data(num_words=10000)

# Padding sequences to make them all the same length (use 256 for this example)
max_len = 256
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)

# One-hot encode labels (binary classification)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

Hiển thị một số bộ dữ liệu cơ bản

```python
# Load the word index dictionary
word_index = tf.keras.datasets.imdb.get_word_index()

# Create a reverse dictionary to map indices back to words
reverse_word_index = {value: key for (key, value) in word_index.items()}

# The IMDB dataset encodes words starting from index 3, so we need to adjust for it
def decode_review(encoded_review):
    return ' '.join([reverse_word_index.get(i - 3, '?') for i in encoded_review])
```

```python
# Display the first 5 reviews and their labels
num_reviews = 5
for i in range(num_reviews):
    print(f"Review {i + 1}: {decode_review(x_train[i])}")
    print(f"Label: {'Positive' if y_train[i] == 1 else 'Negative'}")
    print('-' * 100)
```

```
Review 1: ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? this film was just brilliant casting location scenery story dire
ction everyone's really suited the part they played and you could just imagine being there robert ? is an amazing actor and now the same being director
? father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the fi
lm were great it was just brilliant so much that i bought the film as soon as it was released for ? and would recommend it to everyone to watch and the
fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely
was also ? to the two little boy's that played the ? of norman and paul they were just brilliant children are often left out of the ? list i think becau
se the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they ha
ve done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all
Label: Positive
----------------------------------------------------------------------------------------------------
Review 2: ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? big ha
ir big boobs bad music and a giant safety pin these are the words to best describe this terrible movie i love cheesy horror movies and i've seen hundred
s but this had got to be on of the worst ever made the plot is paper thin and ridiculous the acting is an abomination the script is completely laughable
the best is the end showdown with the cop and how he worked out who the killer is it's just so damn terribly written the clothes are sickening and funny
in equal ? the hair is big lots of boobs ? men wear those cut ? shirts that show off their ? sickening that men actually wore them and the music is just
? trash that plays over and over again in almost every scene there is trashy music boobs and ? taking away bodies and the gym still doesn't close for ?
all joking aside this is a truly bad film whose only charm is to look back on the disaster that was the 80's and have a good old laugh at how bad everyt
hing was back then
Label: Negative
----------------------------------------------------------------------------------------------------
Review 3: ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? this has to be one of the worst films of the 1990s when my fri
ends i were watching this film being the target audience it was aimed at we just sat watched the first half an hour with our jaws touching the floor at
how bad it really was the rest of the time everyone else in the theatre just started talking to each other leaving or generally crying into their popcor
n that they actually paid money they had ? working to watch this feeble excuse for a film it must have looked like a great idea on paper but on film it
```

Xây dựng mô hình CNN và train dữ liệu

[4]:
```
def build_cnn():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Embedding(10000, 128, input_length=max_len))
    model.add(tf.keras.layers.Conv1D(128, 5, activation='relu'))
    model.add(tf.keras.layers.GlobalMaxPooling1D())
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(1, activation='sigmoid'))  # Binary classification
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Build and train CNN model
cnn_model = build_cnn()
cnn_model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

```
Epoch 1/5
D:\App\Anaconda\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
782/782 ━━━━━━━━━━━━━━━━━━━━ 55s 63ms/step - accuracy: 0.7328 - loss: 0.4955 - val_accuracy: 0.8892 - val_loss: 0.2644
Epoch 2/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 51s 65ms/step - accuracy: 0.9483 - loss: 0.1458 - val_accuracy: 0.8895 - val_loss: 0.2685
Epoch 3/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 47s 61ms/step - accuracy: 0.9920 - loss: 0.0346 - val_accuracy: 0.8876 - val_loss: 0.3625
Epoch 4/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 47s 60ms/step - accuracy: 0.9997 - loss: 0.0037 - val_accuracy: 0.8867 - val_loss: 0.4357
Epoch 5/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 45s 58ms/step - accuracy: 1.0000 - loss: 7.4350e-04 - val_accuracy: 0.8933 - val_loss: 0.4344
```

[4]:
```
<keras.src.callbacks.history.History at 0x23e1f07a600>
```

Xây dựng mô hình RNN và training dữ liệu

[5]:
```
def build_rnn():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Embedding(10000, 128, input_length=max_len))
    model.add(tf.keras.layers.LSTM(128, return_sequences=False))
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(1, activation='sigmoid'))  # Binary classification
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Build and train RNN model
rnn_model = build_rnn()
rnn_model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
```

```
Epoch 1/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 311s 392ms/step - accuracy: 0.6960 - loss: 0.5600 - val_accuracy: 0.6956 - val_loss: 0.6061
Epoch 2/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 301s 384ms/step - accuracy: 0.8032 - loss: 0.4191 - val_accuracy: 0.8386 - val_loss: 0.3893
Epoch 3/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 317s 405ms/step - accuracy: 0.9072 - loss: 0.2363 - val_accuracy: 0.8661 - val_loss: 0.3638
Epoch 4/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 310s 397ms/step - accuracy: 0.9365 - loss: 0.1718 - val_accuracy: 0.8440 - val_loss: 0.3706
Epoch 5/5
782/782 ━━━━━━━━━━━━━━━━━━━━ 297s 379ms/step - accuracy: 0.9553 - loss: 0.1238 - val_accuracy: 0.8660 - val_loss: 0.3800
```

[5]:
```
<keras.src.callbacks.history.History at 0x23e10230d40>
```

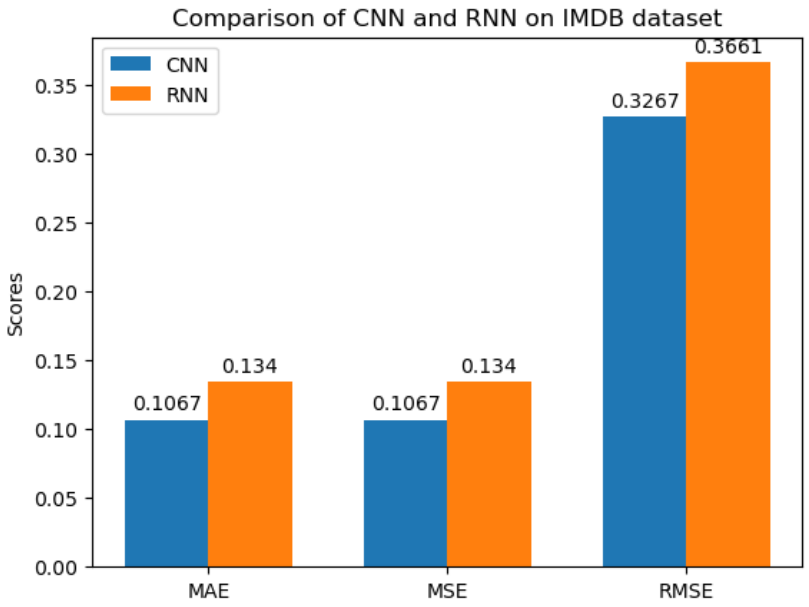
Sử dụng mae, mse, rmse để dự đoán

[15]:
```
# Predict using both models
cnn_predictions = cnn_model.predict(x_test).flatten()
rnn_predictions = rnn_model.predict(x_test).flatten()

# Convert predictions to binary labels (0 or 1)
cnn_pred_labels = (cnn_predictions >= 0.5).astype(int)
rnn_pred_labels = (rnn_predictions >= 0.5).astype(int)

# Calculate MAE, MSE, RMSE for CNN
cnn_mae = mean_absolute_error(y_test, cnn_pred_labels)
cnn_mse = mean_squared_error(y_test, cnn_pred_labels)
cnn_rmse = np.sqrt(cnn_mse)

# Calculate MAE, MSE, RMSE for RNN
rnn_mae = mean_absolute_error(y_test, rnn_pred_labels)
rnn_mse = mean_squared_error(y_test, rnn_pred_labels)
rnn_rmse = np.sqrt(rnn_mse)

# Print the results
print("CNN - MAE: {:.4f}, MSE: {:.4f}, RMSE: {:.4f}".format(cnn_mae, cnn_mse, cnn_rmse))
print("RNN - MAE: {:.4f}, MSE: {:.4f}, RMSE: {:.4f}".format(rnn_mae, rnn_mse, rnn_rmse))
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 16ms/step
782/782 ━━━━━━━━━━━━━━━━━━━━ 128s 164ms/step
CNN - MAE: 0.1067, MSE: 0.1067, RMSE: 0.3267
RNN - MAE: 0.1340, MSE: 0.1340, RMSE: 0.3661
```

Vẽ biểu đồ thể hiện sự tương quan

```
[16]: # Bar chart to compare MAE, MSE, and RMSE
metrics = ['MAE', 'MSE', 'RMSE']
cnn_scores = [cnn_mae, cnn_mse, cnn_rmse]
rnn_scores = [rnn_mae, rnn_mse, rnn_rmse]

x = np.arange(len(metrics))
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, cnn_scores, width, label='CNN')
rects2 = ax.bar(x + width/2, rnn_scores, width, label='RNN')

ax.set_ylabel('Scores')
ax.set_title('Comparison of CNN and RNN on IMDB dataset')
ax.set_xticks(x)
ax.set_xticklabels(metrics)
ax.legend()

# Add value labels on bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(round(height, 4)),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

plt.show()
```



III. Mô hình DL với tập dữ liệu house pricing bostom

Import các thư viện cần thiết và tải dữ liệu

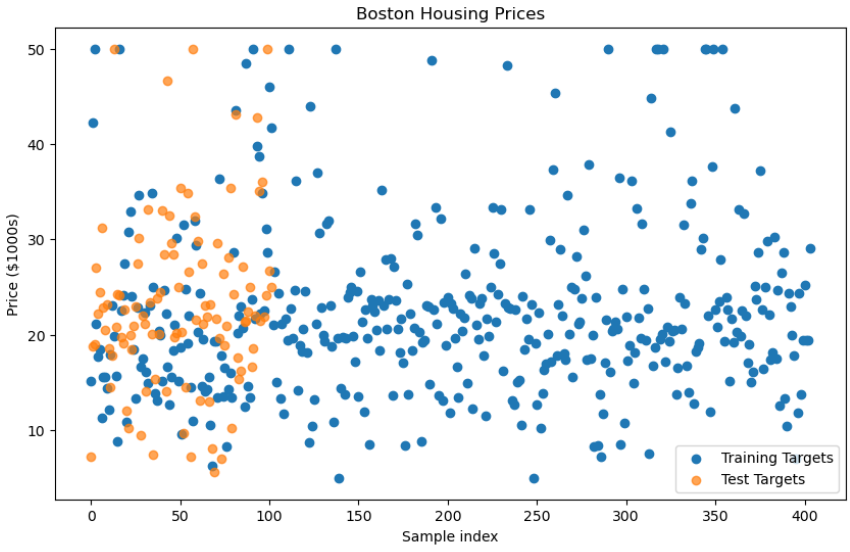
```
[7]: from tensorflow.keras.datasets import boston_housing

# Load the Boston housing dataset
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()

print("Train data shape:", train_data.shape)
print("Test data shape:", test_data.shape)
print("Train targets shape:", train_targets.shape)
print("Test targets shape:", test_targets.shape)
```

```
Train data shape: (404, 13)
Test data shape: (102, 13)
Train targets shape: (404,)
Test targets shape: (102,)
```

```
[9]: # Normalize the data: subtract mean and divide by stddev
mean = train_data.mean(axis=0)
std = train_data.std(axis=0)

train_data = (train_data - mean) / std
test_data = (test_data - mean) / std
```

Hiển thị data mẫu

```
[8]: import matplotlib.pyplot as plt

# Plot some samples from the dataset
plt.figure(figsize=(10, 6))
plt.scatter(range(len(train_targets)), train_targets, label='Training Targets')
plt.scatter(range(len(test_targets)), test_targets, label='Test Targets', alpha=0.7)
plt.title("Boston Housing Prices")
plt.xlabel("Sample index")
plt.ylabel("Price ($1000s)")
plt.legend()
plt.show()
```



Định nghĩa mô hình cnn

[4]:
```python
from tensorflow.keras import models, layers

# CNN Model
cnn_model = models.Sequential()

# Reshape input to match Conv1D format
cnn_model.add(layers.Reshape((13, 1), input_shape=(train_data.shape[1],)))

# Add Conv1D layers
cnn_model.add(layers.Conv1D(64, 3, activation='relu'))
cnn_model.add(layers.MaxPooling1D(2))
cnn_model.add(layers.Conv1D(32, 3, activation='relu'))
cnn_model.add(layers.Flatten())
cnn_model.add(layers.Dense(64, activation='relu'))
cnn_model.add(layers.Dense(1))

# Compile the model
cnn_model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

Định nghĩa mô hình rnn

[5]:
```python
# RNN Model
rnn_model = models.Sequential()

# Reshape input to match LSTM format
rnn_model.add(layers.Reshape((13, 1), input_shape=(train_data.shape[1],)))

# Add LSTM layer
rnn_model.add(layers.LSTM(64, return_sequences=False))
rnn_model.add(layers.Dense(64, activation='relu'))
rnn_model.add(layers.Dense(1))

# Compile the model
rnn_model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

Train mô hình

[6]:
```python
# Train CNN Model
cnn_history = cnn_model.fit(train_data, train_targets, epochs=100, batch_size=32, validation_split=0.2, verbose=1)

# Train RNN Model
rnn_history = rnn_model.fit(train_data, train_targets, epochs=100, batch_size=32, validation_split=0.2, verbose=1)
```

```
Epoch 1/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 7s 77ms/step - loss: 548.4911 - mae: 21.7602 - val_loss: 582.1924 - val_mae: 22.3482
Epoch 2/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 499.7446 - mae: 20.3022 - val_loss: 428.5109 - val_mae: 18.6490
Epoch 3/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 305.9208 - mae: 15.1592 - val_loss: 178.6423 - val_mae: 10.9764
Epoch 4/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 108.7055 - mae: 8.4561 - val_loss: 128.0366 - val_mae: 8.9006
Epoch 5/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 121.0082 - mae: 8.8847 - val_loss: 108.9003 - val_mae: 7.8800
Epoch 6/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 92.7146 - mae: 7.6333 - val_loss: 102.2351 - val_mae: 7.5168
Epoch 7/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 90.5328 - mae: 7.5044 - val_loss: 88.3071 - val_mae: 7.1019
Epoch 8/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 75.0370 - mae: 6.7190 - val_loss: 88.6684 - val_mae: 6.8500
Epoch 9/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 68.3553 - mae: 6.2615 - val_loss: 78.2135 - val_mae: 6.5549
```

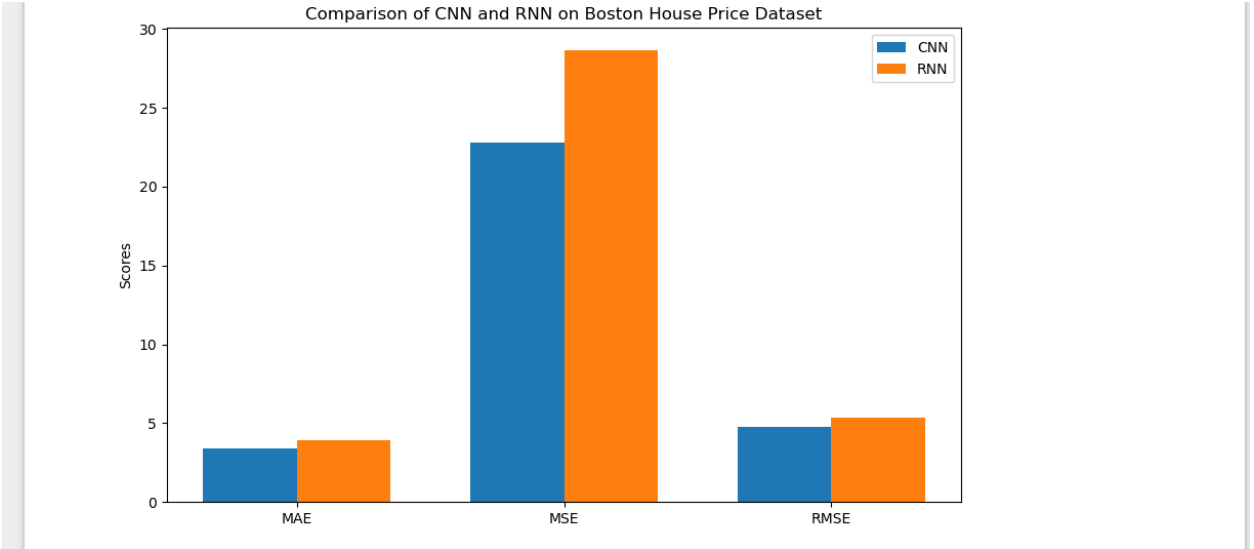
Tính toán các tham số mae, mse

[7]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

# CNN Predictions
cnn_predictions = cnn_model.predict(test_data)
cnn_mae = mean_absolute_error(test_targets, cnn_predictions)
cnn_mse = mean_squared_error(test_targets, cnn_predictions)
cnn_rmse = np.sqrt(cnn_mse)

print(f"CNN - MAE: {cnn_mae}, MSE: {cnn_mse}, RMSE: {cnn_rmse}")

# RNN Predictions
rnn_predictions = rnn_model.predict(test_data)
rnn_mae = mean_absolute_error(test_targets, rnn_predictions)
rnn_mse = mean_squared_error(test_targets, rnn_predictions)
rnn_rmse = np.sqrt(rnn_mse)

print(f"RNN - MAE: {rnn_mae}, MSE: {rnn_mse}, RMSE: {rnn_rmse}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 134ms/step
CNN - MAE: 3.375703497493969, MSE: 22.800314310701435, RMSE: 4.774967466978328
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 260ms/step
RNN - MAE: 3.925712807973226, MSE: 28.621072435189433, RMSE: 5.349866581064376
```

Vẽ biểu đồ thể hiện sự tương quan

[9]:
```
# Plotting the metrics
metrics = ['MAE', 'MSE', 'RMSE']
cnn_values = [cnn_mae, cnn_mse, cnn_rmse]
rnn_values = [rnn_mae, rnn_mse, rnn_rmse]

plt.figure(figsize=(10, 6))
x = np.arange(len(metrics))
width = 0.35

plt.bar(x - width/2, cnn_values, width, label='CNN')
plt.bar(x + width/2, rnn_values, width, label='RNN')

plt.ylabel('Scores')
plt.title('Comparison of CNN and RNN on Boston House Price Dataset')

plt.xticks(x, metrics)
plt.legend()

plt.show()
```



IV. Giải thích một số khái niệm

1. Epoch:

Khái niệm: Epoch là một lần huấn luyện toàn bộ dữ liệu qua mô hình. Mỗi epoch có nghĩa là mô hình được xem qua toàn bộ tập dữ liệu một lần.
Trong câu lệnh: epochs=10 nghĩa là mô hình sẽ được huấn luyện trên toàn bộ tập dữ liệu 10 lần, mỗi lần được gọi là một "epoch". Nếu tập dữ liệu của bạn có 10,000 mẫu, mô hình sẽ nhìn thấy mỗi mẫu 10 lần khi quá trình huấn luyện hoàn tất.
2. Batch:

Khái niệm: Batch là một nhóm dữ liệu nhỏ mà mô hình sẽ sử dụng để cập nhật trọng số (weights). Thay vì huấn luyện trên toàn bộ dữ liệu cùng lúc (gọi là "batch gradient descent"), mô hình chia nhỏ dữ liệu thành các phần và thực hiện việc cập nhật trọng số sau khi chạy qua mỗi batch (gọi là "mini-batch gradient descent").

Trong câu lệnh: batch_size=64 nghĩa là thay vì huấn luyện trên toàn bộ tập dữ liệu cùng lúc, mô hình sẽ chia tập dữ liệu thành các nhóm có 64 mẫu để huấn luyện từng nhóm một. Sau mỗi batch, mô hình sẽ cập nhật trọng số dựa trên lỗi tính toán từ batch đó.
3. Training Data (Dữ liệu huấn luyện):

Khái niệm: Đây là tập dữ liệu được sử dụng để huấn luyện mô hình, nghĩa là mô hình sử dụng tập dữ liệu này để học cách dự đoán chính xác dựa trên các đặc trưng đầu vào và nhãn kết quả.

Trong câu lệnh: x_train và y_train là các biến lưu trữ dữ liệu huấn luyện. Mô hình sẽ học từ tập dữ liệu này trong mỗi epoch.

4. Validation Data (Dữ liệu kiểm định):
Khái niệm: Validation data được sử dụng để đánh giá hiệu suất của mô hình trong quá trình huấn luyện. Đây là tập dữ liệu mà mô hình chưa được huấn luyện trực tiếp, nhằm kiểm tra xem mô hình có "overfitting" hay không (mô hình học quá kỹ trên dữ liệu huấn luyện mà không tổng quát hóa tốt cho dữ liệu mới).
Trong câu lệnh: validation_split=0.2 nghĩa là 20% của tập dữ liệu huấn luyện sẽ được tách ra làm dữ liệu kiểm định (validation data). Mô hình sẽ không sử dụng phần này để học, mà chỉ dùng để đánh giá hiệu suất sau mỗi epoch.
5. Test Data (Dữ liệu kiểm tra):

Khái niệm: Test data là tập dữ liệu độc lập với dữ liệu huấn luyện và kiểm định, được sử dụng sau khi quá trình huấn luyện hoàn tất để đánh giá xem mô hình hoạt động tốt như thế nào trên dữ liệu chưa từng gặp.

Lưu ý: Test data không xuất hiện trong đoạn lệnh trên, nhưng bạn có thể dùng nó sau khi huấn luyện để đánh giá mô hình trên dữ liệu thực tế.
6. Validation:

Khái niệm: Validation là quá trình kiểm tra hiệu suất của mô hình trên tập dữ liệu validation. Kết quả từ validation giúp bạn điều chỉnh các siêu tham số (hyperparameters) của mô hình và kiểm tra xem mô hình có học quá sâu vào dữ liệu huấn luyện (overfitting) hay không.

Trong câu lệnh: Với validation_split=0.2, validation sẽ được thực hiện sau mỗi epoch bằng cách sử dụng 20% dữ liệu huấn luyện đã được tách riêng làm validation set. Mô hình sẽ báo cáo hiệu suất (ví dụ như độ chính xác và hàm mất mát) trên cả tập huấn luyện và tập kiểm định.