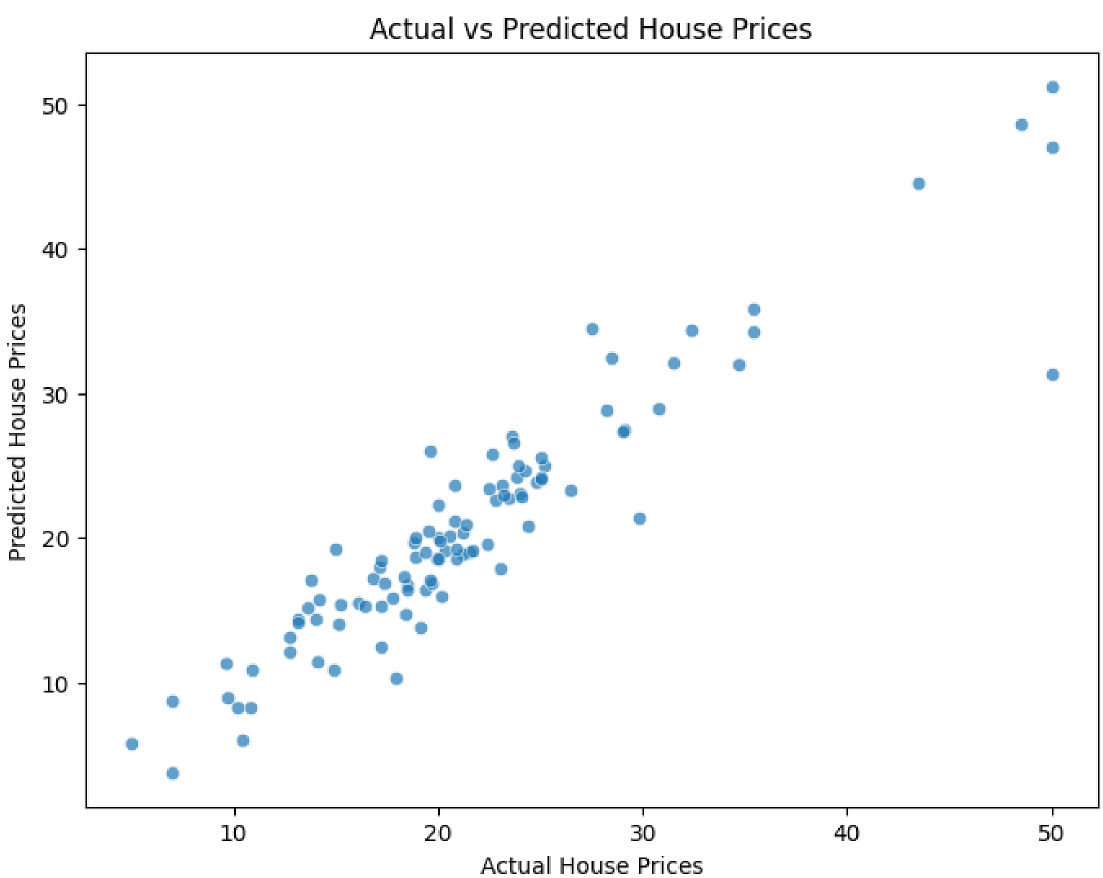
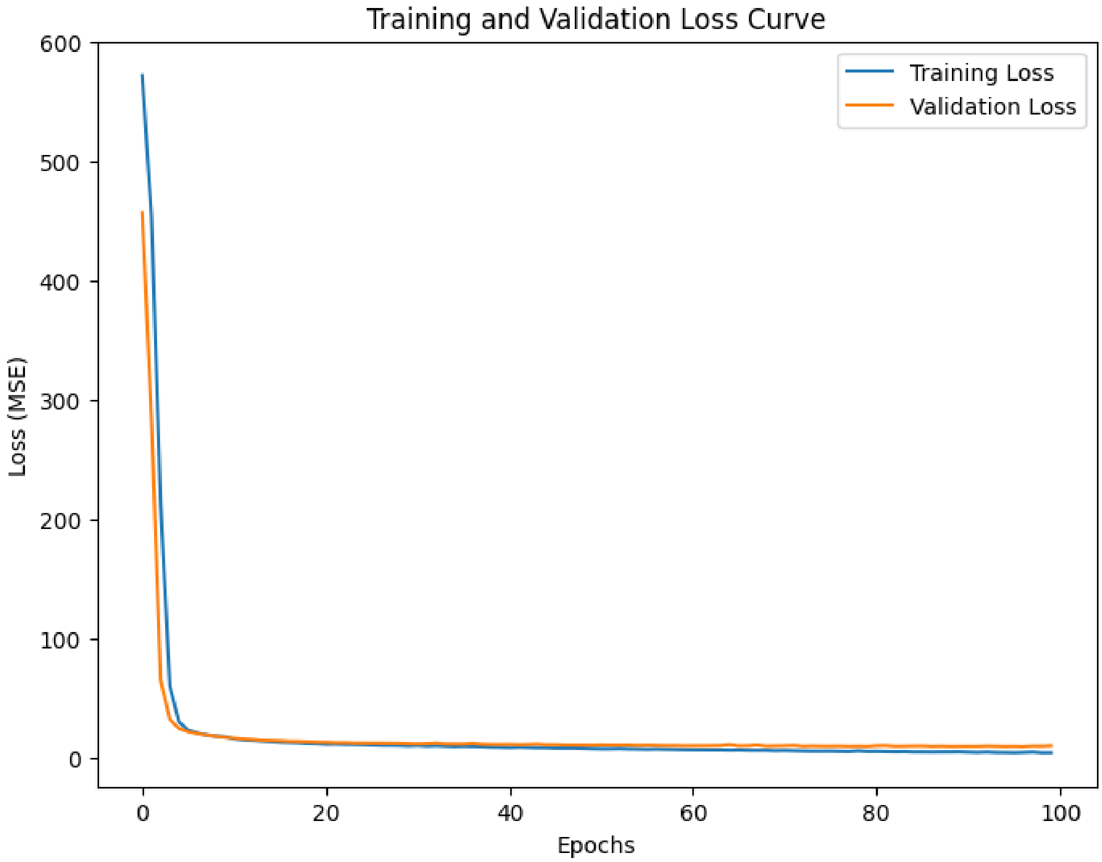
```python
# Step 1: Import Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Step 2: Load the Dataset
file_path = "BostonHousing.csv"  # Update the path if needed
df = pd.read_csv(file_path)

# Display the first few rows of the dataset
print(df.head())

# Step 3: Data Preprocessing
# Define features (X) and target variable (y)
X = df.drop(columns=['medv'])  # All columns except 'medv' are features
y = df['medv']  # 'medv' is the target variable (house prices)

# Normalize the features for better convergence
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Step 4: Build the Deep Neural Network Model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),  # Input layer
    Dense(32, activation='relu'),  # Hidden layer 1
    Dense(16, activation='relu'),  # Hidden layer 2
    Dense(1, activation='linear')  # Output layer (Regression -> Linear activation)
])

# Step 5: Compile the Model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Step 6: Train the Model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100,
batch_size=16, verbose=1)

# Step 7: Evaluate the Model
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```python
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R² Score: {r2}")

# Step 8: Visualization of Actual vs Predicted Values
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred.flatten(), alpha=0.7)
plt.xlabel("Actual House Prices")
plt.ylabel("Predicted House Prices")
plt.title("Actual vs Predicted House Prices")
plt.show()

# Step 9: Loss Curve
plt.figure(figsize=(8,6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss (MSE)")
plt.title("Training and Validation Loss Curve")
plt.legend()
plt.show()
```

```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2

C:\Users\ashuy\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\layers\core\dense.py:87: 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 1/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 574.9018 - mae: 22.1412 - val_loss:
456.9664 - val_mae: 19.5059

Epoch 92/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.6706 - mae: 1.5401 - val_loss: 9.2375 -
val_mae: 2.0314
Epoch 93/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.1273 - mae: 1.4691 - val_loss: 9.4506 -
val_mae: 2.0278
Epoch 94/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 3.8420 - mae: 1.4329 - val_loss: 9.3140 -
val_mae: 2.0600
Epoch 95/100
```

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.2360 - mae: 1.5458 - val_loss: 9.1549 -
val_mae: 2.0140
Epoch 96/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.0647 - mae: 1.5233 - val_loss: 9.2595 -
val_mae: 2.0125
Epoch 97/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.3073 - mae: 1.4862 - val_loss: 9.0666 -
val_mae: 2.0384
Epoch 98/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.1789 - mae: 1.5145 - val_loss: 9.4771 -
val_mae: 2.1142
Epoch 99/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.2960 - mae: 1.5308 - val_loss: 9.4006 -
val_mae: 2.0284
Epoch 100/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.0481 - mae: 1.4436 - val_loss: 9.7360 -
val_mae: 2.0568
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Mean Squared Error (MSE): 9.735979803475566
Mean Absolute Error (MAE): 2.0567669602001417
R² Score: 0.8672374534095605
```



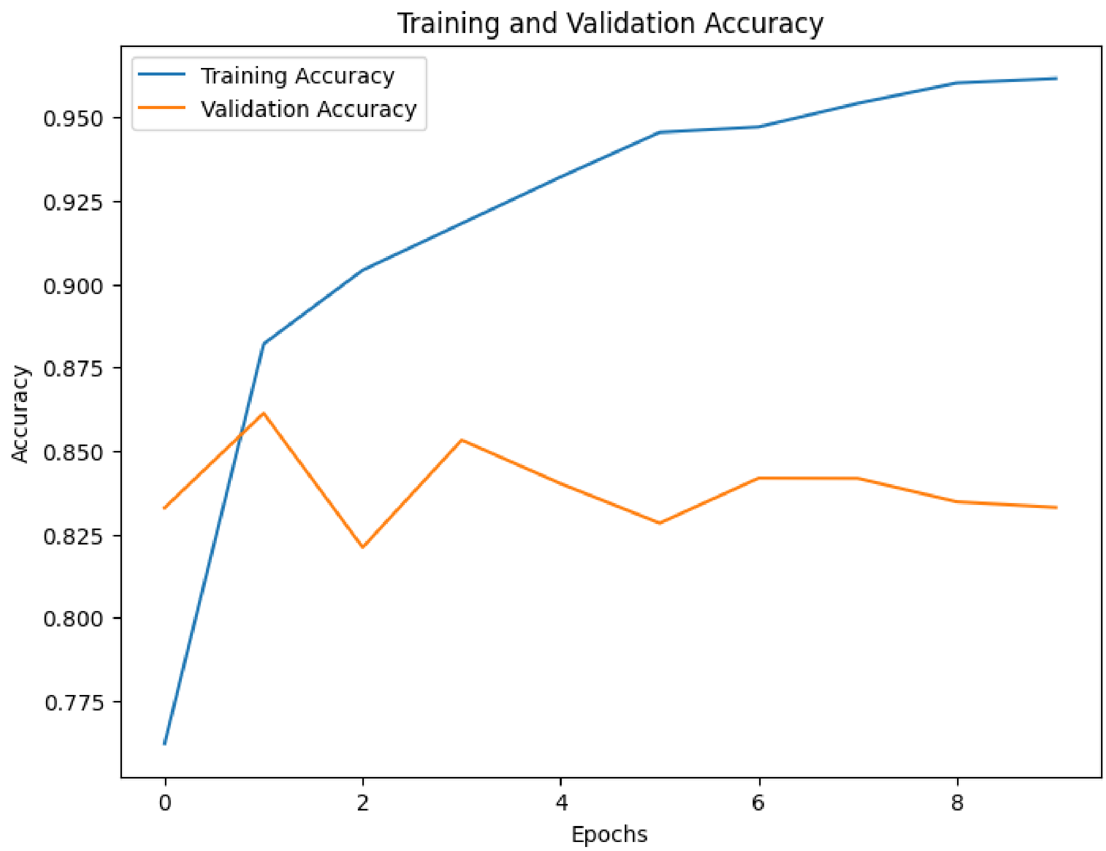


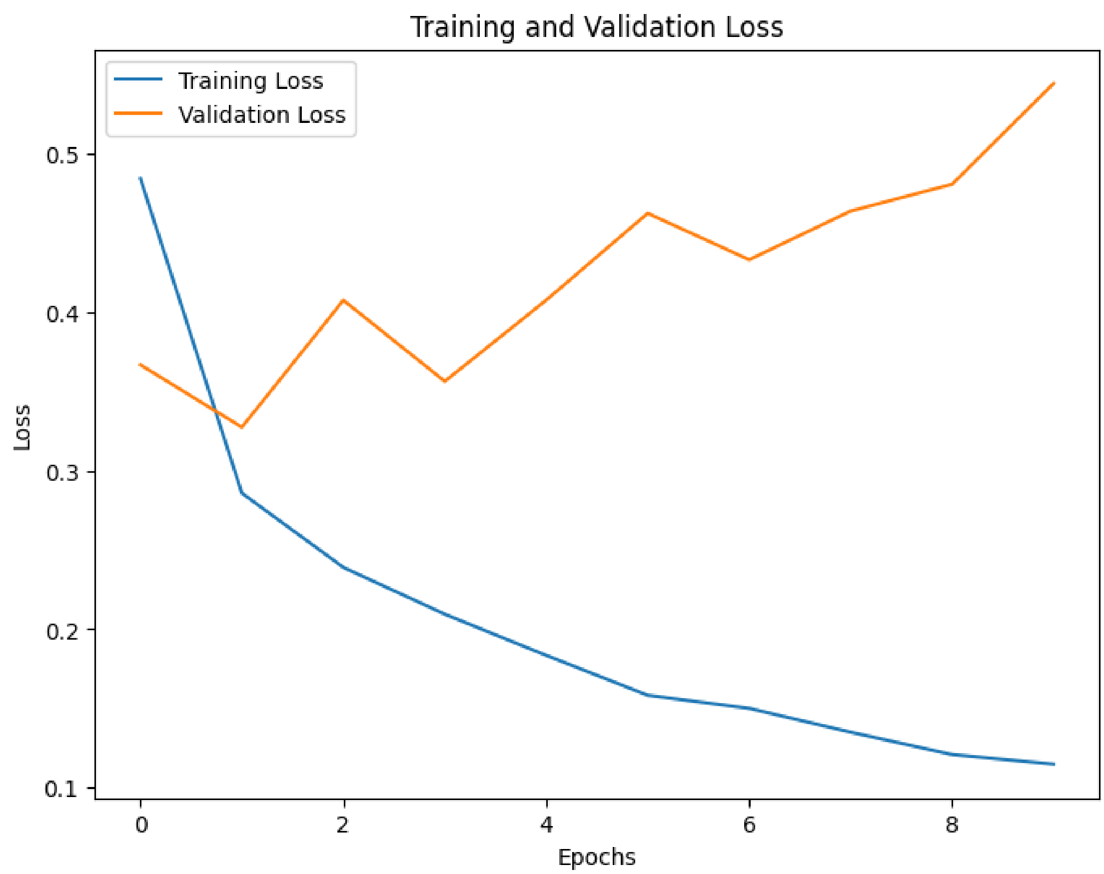
Training and Validation Loss Curve
Training Loss
Validation Loss
Loss (MSE)
Epochs
0
100
200
300
400
500
600
0
20
40
60
80
100

```python
# Step 1: Import Required Libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, LSTM, Flatten,
GlobalAveragePooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt
import numpy as np

# Step 2: Load the IMDB Dataset
from tensorflow.keras.datasets import imdb

# Load IMDB dataset with the top 10,000 most common words
vocab_size = 10000
max_length = 200  # Maximum words per review
(tr_x, tr_y), (te_x, te_y) = imdb.load_data(num_words=vocab_size)

# Step 3: Preprocess the Data
# Pad sequences to ensure all reviews have the same length
tr_x = pad_sequences(tr_x, maxlen=max_length, padding='post', truncating='post')
te_x = pad_sequences(te_x, maxlen=max_length, padding='post', truncating='post')

# Step 4: Build the Deep Neural Network Model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=32, input_length=max_length),  # Embedding
Layer
    GlobalAveragePooling1D(),  # Pooling Layer to reduce dimensionality
    Dense(64, activation='relu'),  # Fully connected layer
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Output layer (Binary Classification)
])

# Step 5: Compile the Model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 6: Train the Model
history = model.fit(tr_x, tr_y, validation_data=(te_x, te_y), epochs=10, batch_size=64,
verbose=1)

# Step 7: Evaluate the Model
loss, accuracy = model.evaluate(te_x, te_y)
print(f"Test Accuracy: {accuracy*100:.2f}%")

# Step 8: Visualization of Training and Validation Accuracy/Loss
# Plot Accuracy
plt.figure(figsize=(8,6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```

```python
# Plot Loss
plt.figure(figsize=(8,6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
Epoch 1/10

C:\Users\ashuy\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(

391/391 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.6677 - loss: 0.5976 - val_accuracy: 0.8329 - val_loss: 0.3667
Epoch 2/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8753 - loss: 0.3011 - val_accuracy: 0.8613 - val_loss: 0.3273
Epoch 3/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9069 - loss: 0.2373 - val_accuracy: 0.8210 - val_loss: 0.4074
Epoch 4/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9258 - loss: 0.1968 - val_accuracy: 0.8532 - val_loss: 0.3563
Epoch 5/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9358 - loss: 0.1789 - val_accuracy: 0.8402 - val_loss: 0.4075
Epoch 6/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9461 - loss: 0.1570 - val_accuracy: 0.8283 - val_loss: 0.4623
Epoch 7/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9518 - loss: 0.1399 - val_accuracy: 0.8418 - val_loss: 0.4329
Epoch 8/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9576 - loss: 0.1294 - val_accuracy: 0.8418 - val_loss: 0.4637
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9652 - loss: 0.1121 - val_accuracy: 0.8347 - val_loss: 0.4805
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9663 - loss: 0.1044 - val_accuracy: 0.8330 - val_loss: 0.5441
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8341 - loss: 0.5385
Test Accuracy: 83.30%
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0
2
4
6
8
Epochs

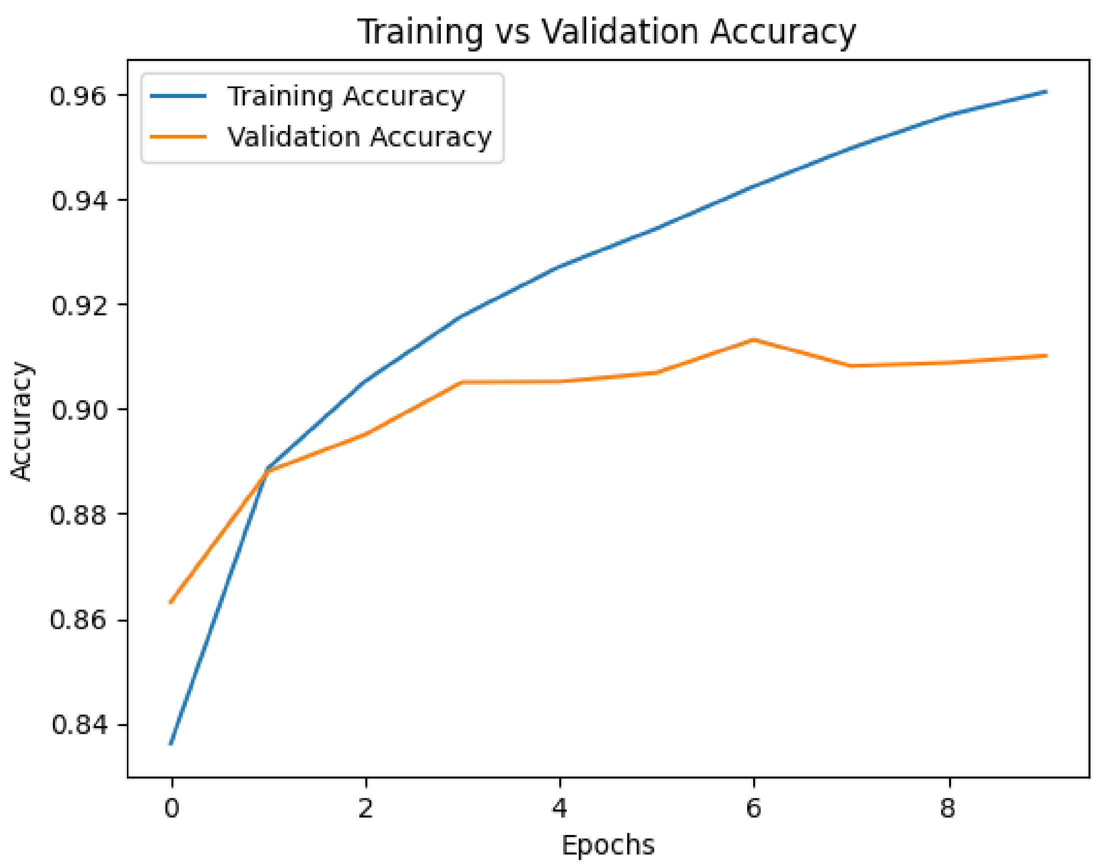

Training and Validation Loss
Training Loss
Validation Loss
Loss
0.5
0.4
0.3
0.2
0.1
0
2
4
6
8
Epochs

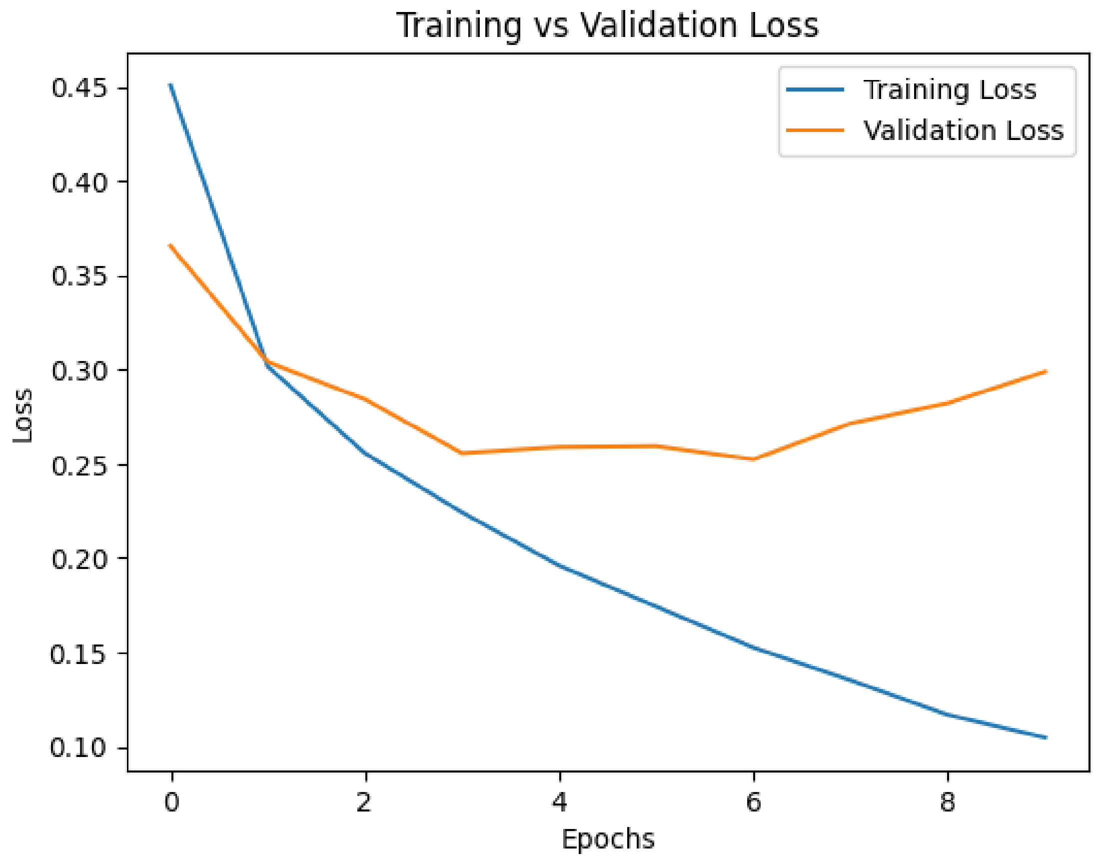
```python
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

# Load dataset from Keras datasets
fashion_mnist = keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Define class names for reference
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

# Display dataset shape
print("Training data shape:", x_train.shape, y_train.shape)
print("Testing data shape:", x_test.shape, y_test.shape)


# Normalize pixel values to [0,1] range
x_train, x_test = x_train / 255.0, x_test / 255.0

# Reshape data to add channel dimension (needed for CNN)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

print("New Training data shape:", x_train.shape)
print("New Testing data shape:", x_test.shape)


# Define CNN Model
model = keras.Sequential([
    # Convolutional Layer 1
    keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    keras.layers.MaxPooling2D(2,2),  # Max Pooling Layer

    # Convolutional Layer 2
    keras.layers.Conv2D(64, (3,3), activation='relu'),
    keras.layers.MaxPooling2D(2,2),

    # Flatten the output from convolutional layers
    keras.layers.Flatten(),

    # Fully Connected Layer (Dense)
    keras.layers.Dense(128, activation='relu'),

    # Output Layer with Softmax Activation for 10 classes
    keras.layers.Dense(10, activation='softmax')
])

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

```python
# Display model architecture
model.summary()


# Train the model
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))


# Evaluate on test data
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print("\nTest Accuracy:", test_acc)

# Plot Training and Validation Accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training vs Validation Accuracy')
plt.show()

# Plot Training and Validation Loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training vs Validation Loss')
plt.show()


# Make predictions
predictions = model.predict(x_test)

# Function to display image and prediction
def display_image(index):
    plt.imshow(x_test[index].reshape(28,28), cmap=plt.cm.binary)
    plt.title(f"Predicted: {class_names[np.argmax(predictions[index])]}, Actual: 
{class_names[y_test[index]]}")
    plt.show()

# Display a random test image and its predicted label
import random
index = random.randint(0, len(x_test))
display_image(index)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━━0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━━5s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-
labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━━0s 0us/step
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-
images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━1s 0us/step
Training data shape: (60000, 28, 28) (60000,)
Testing data shape: (10000, 28, 28) (10000,)
New Training data shape: (60000, 28, 28, 1)
New Testing data shape: (10000, 28, 28, 1)

C:\Users\ashuy\AppData\Local\Programs\Python\Python311\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)

Non-trainable params: 0 (0.00 B)

Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━11s 5ms/step - accuracy: 0.7766 - loss: 0.6153 -
val_accuracy: 0.8632 - val_loss: 0.3654
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━10s 5ms/step - accuracy: 0.8835 - loss: 0.3140 - Epoch
10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━10s 5ms/step - accuracy: 0.9619 - loss: 0.1016 -
val_accuracy: 0.9101 - val_loss: 0.2988
313/313 - 1s - 3ms/step - accuracy: 0.9101 - loss: 0.2988

Test Accuracy: 0.910099983215332
```






```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```