لطفا توجه کنید ، توضیحات هر سوال در پایان پیاده سازی های مربوطه به زبان انگلیسی نوشته شده است. در مورد انگلیسی نوشتن توضیحات نیز به دلیل مشکلات کولب با خانم غلامی هماهنگ شده است. با تشکر. همچنین استاد فرمودند که بجای توضیحات در داکیومنت جداگانه توضیحات در کولب قبول است.

#Ehsan Espandar - 99442011 - Neural Networks - CS - AI - DR.Abdi

#Part 1

#Question 1

Here I have tried to first define my three functions as below:

- linear function: 2*x+1
- nonlinear function: 3 * x^2 + 2*x
- complex function: sinx + cos2x

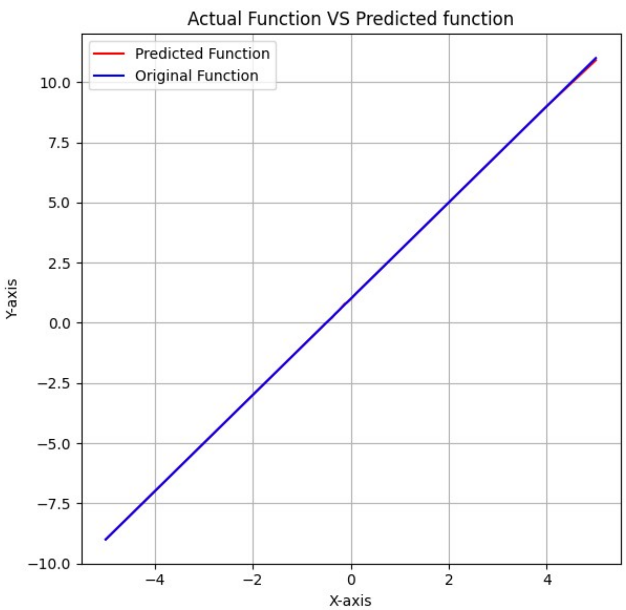
Then I tried to generate 200 points using those functions and save them inside X_train in a specific X_range, after that I computed the Y value for those X_train points.

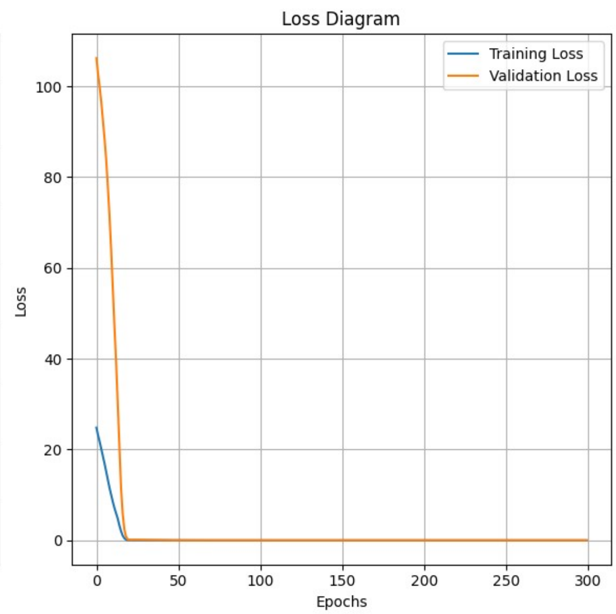
In the next step, tried to fit my MLP using X_train points. I have used Tensorflow library for my MLP. Then generated another 200 points of X_test for testing my MLP. Finally I presented the results using matplotlib and also mentioned the MSE for each one of my functions and the respective prediction for them.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

def linear(X):
    return 2*X+1

def nonlinear(X):
    return 3 * X**2 + 2*X

def Complex(X):
    return np.sin(X) + np.cos(2*X)


import tensorflow as tf
functions = [linear,nonlinear,Complex]

for function in functions:
  X_train = np.linspace(-5, 5, 200).reshape(-1, 1)
  y_train = function(X_train)

  model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1)
  ])

  model.compile(optimizer='adam', loss='mean_squared_error',)

  history = model.fit(X_train, y_train,batch_size=16,
```

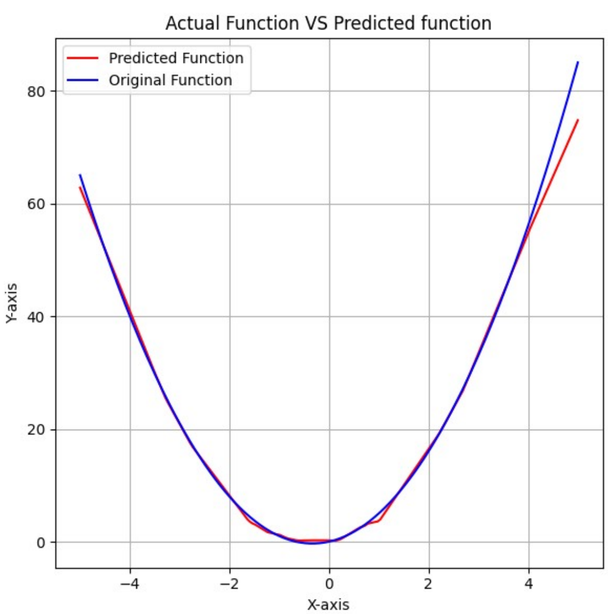
```
epochs=300,validation_split=0.1,verbose=0)

  X_test = np.linspace(-5,5,200).reshape(-1,1)

  y_pred_function = model.predict(X_test)

  mse = np.mean(np.square(y_pred_function - function(X_train)))


  plt.figure(figsize=(12, 6))

  plt.subplot(1, 2, 1)
  plt.plot(X_test, y_pred_function, color='red', label='Predicted
Function')
  plt.plot(X_train, function(X_train), color='blue', label='Original
Function')
  plt.xlabel('X-axis')
  plt.ylabel('Y-axis')
  plt.title('Actual Function VS Predicted function')
  plt.legend()
  plt.grid(True)

  plt.subplot(1, 2, 2)
  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('Loss Diagram')
  plt.legend()
  plt.grid(True)

  plt.tight_layout()
  plt.show()

  print(f"Mean Squared Error: {mse:.4f}")
```

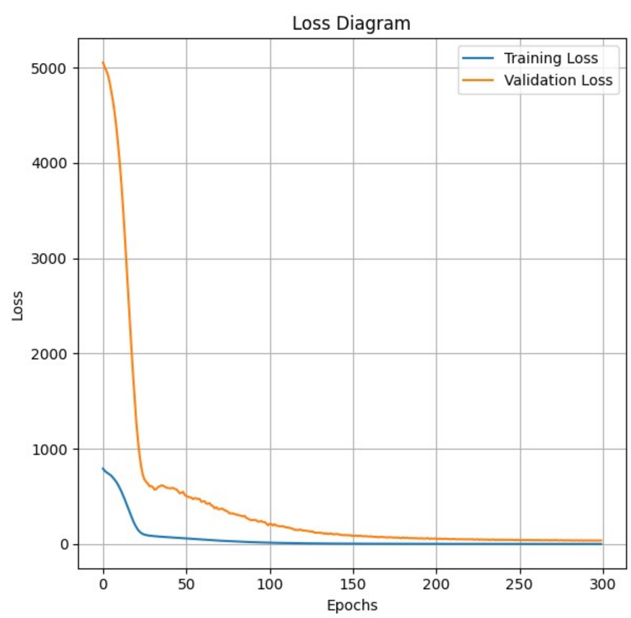
```
7/7 [==============================] - 0s 2ms/step
```



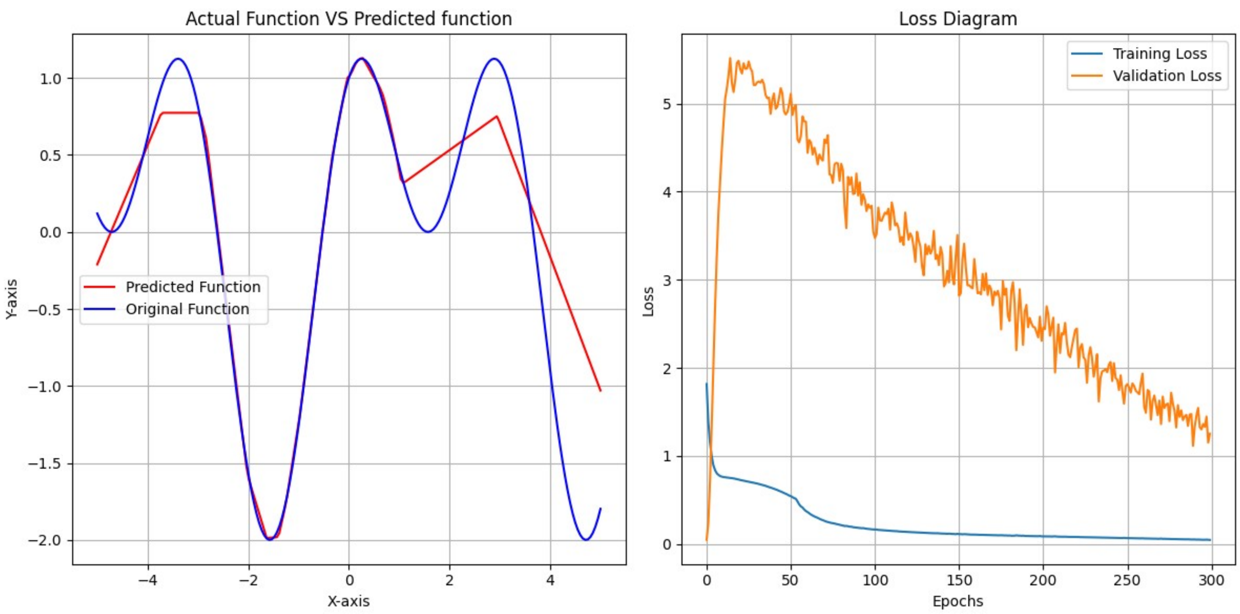



```
Mean Squared Error: 0.0003
7/7 [==============================] - 0s 2ms/step
```





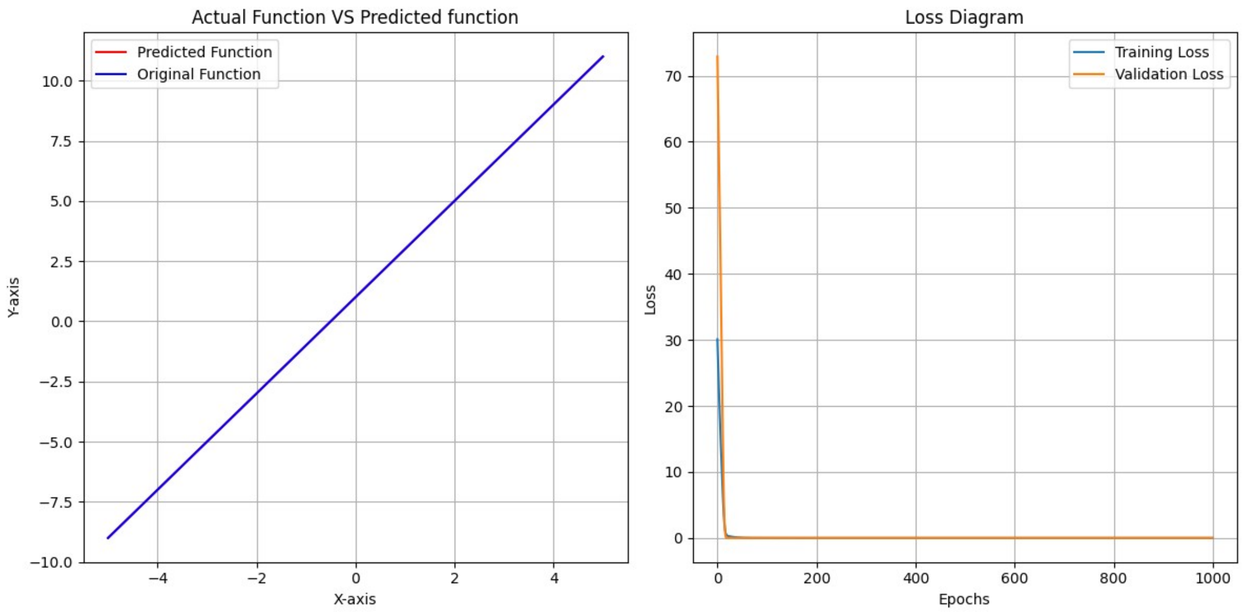

```
Mean Squared Error: 3.9406
7/7 [==============================] - 0s 2ms/step
```



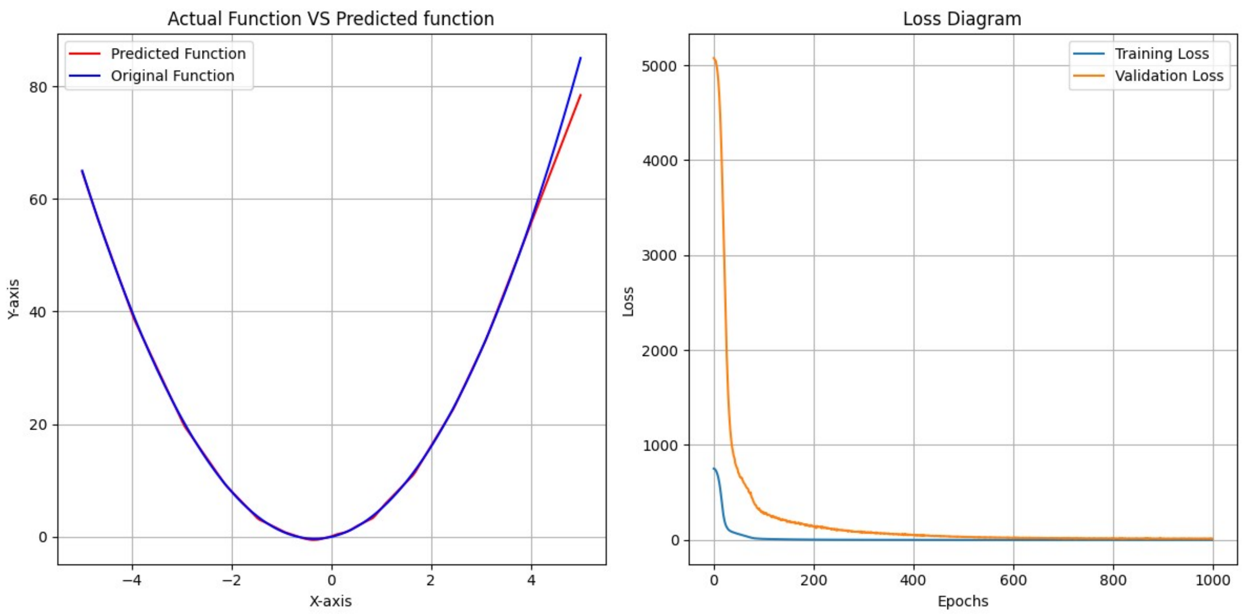

```
Mean Squared Error: 0.1662

functions = [linear,nonlinear,Complex]

for function in functions:
  X_train = np.linspace(-5, 5, 200).reshape(-1, 1)
  y_train = function(X_train)

  model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1)
  ])

  model.compile(optimizer='adam', loss='mean_squared_error')

  history = model.fit(X_train, y_train,batch_size=16,
epochs=1000,validation_split=0.1,verbose=0)

  X_test = np.linspace(-5,5,200).reshape(-1,1)

  y_pred_function = model.predict(X_test)


  mse = np.mean(np.square(y_pred_function - function(X_train)))

  plt.figure(figsize=(12, 6))

  plt.subplot(1, 2, 1)
  plt.plot(X_test, y_pred_function, color='red', label='Predicted
Function')
```

```
  plt.plot(X_train, function(X_train), color='blue', label='Original
Function')
  plt.xlabel('X-axis')
  plt.ylabel('Y-axis')
  plt.title('Actual Function VS Predicted function')
  plt.legend()
  plt.grid(True)

  plt.subplot(1, 2, 2)
  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('Loss Diagram')
  plt.legend()
  plt.grid(True)

  plt.tight_layout()
  plt.show()

  print(f"Mean Squared Error: {mse:.4f}")
```

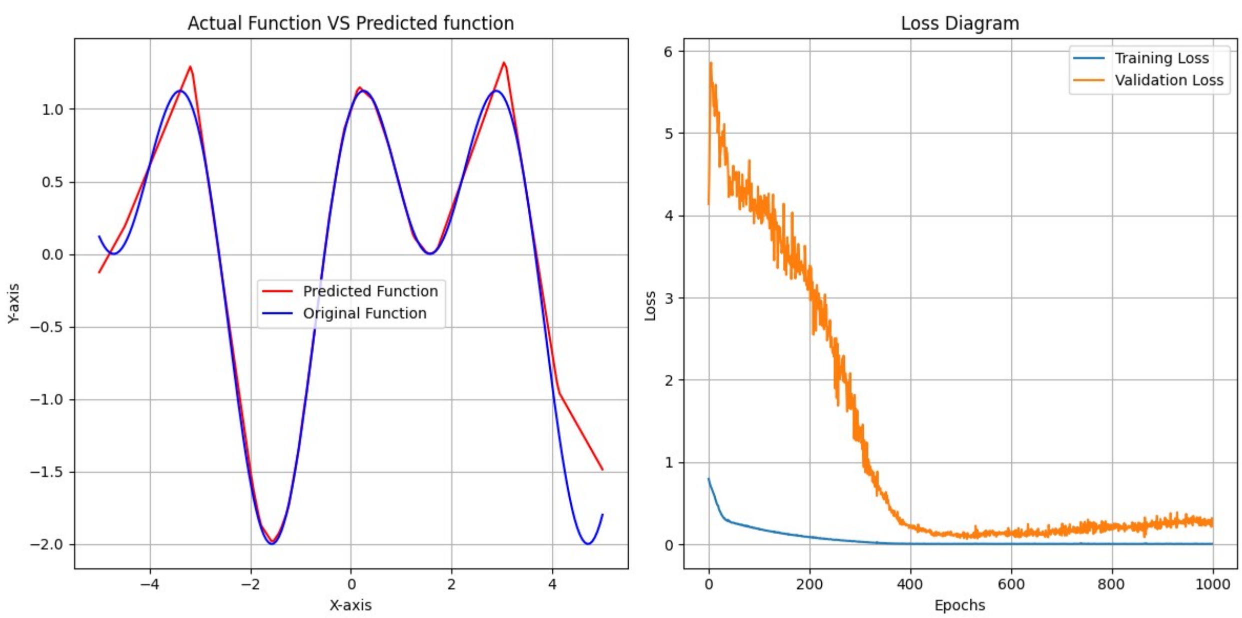
```
7/7 [==============================] - 0s 2ms/step
```




```
Mean Squared Error: 0.0000
7/7 [==============================] - 0s 2ms/step
```




```
Mean Squared Error: 1.3675
7/7 [==============================] - 0s 2ms/step
```

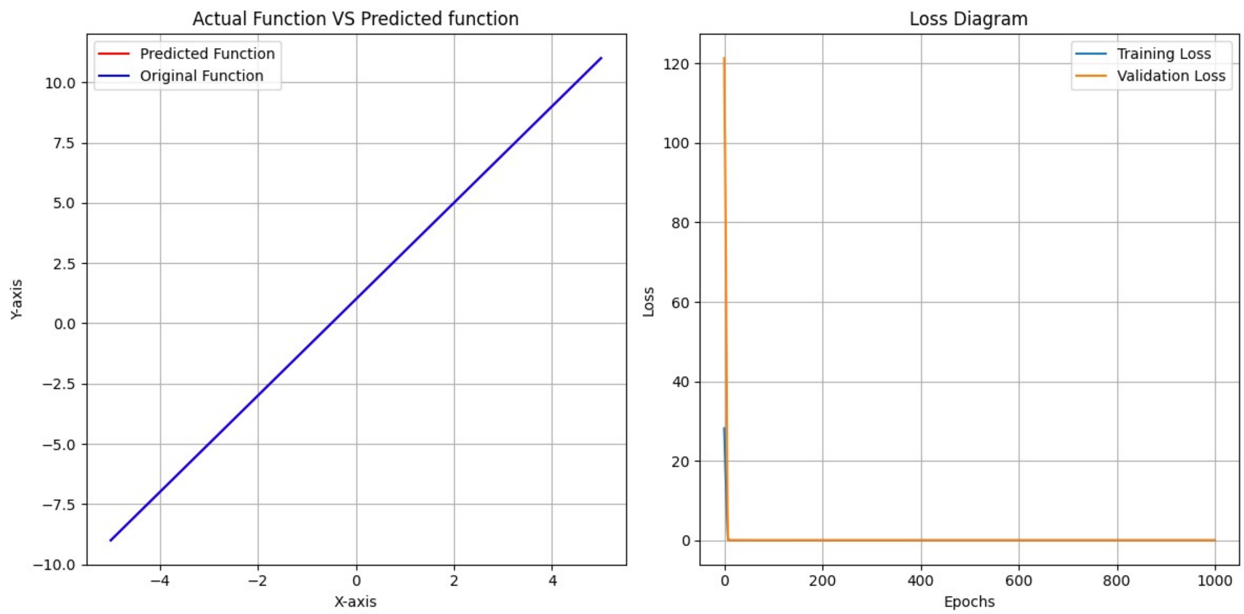



```
Mean Squared Error: 0.0371
```

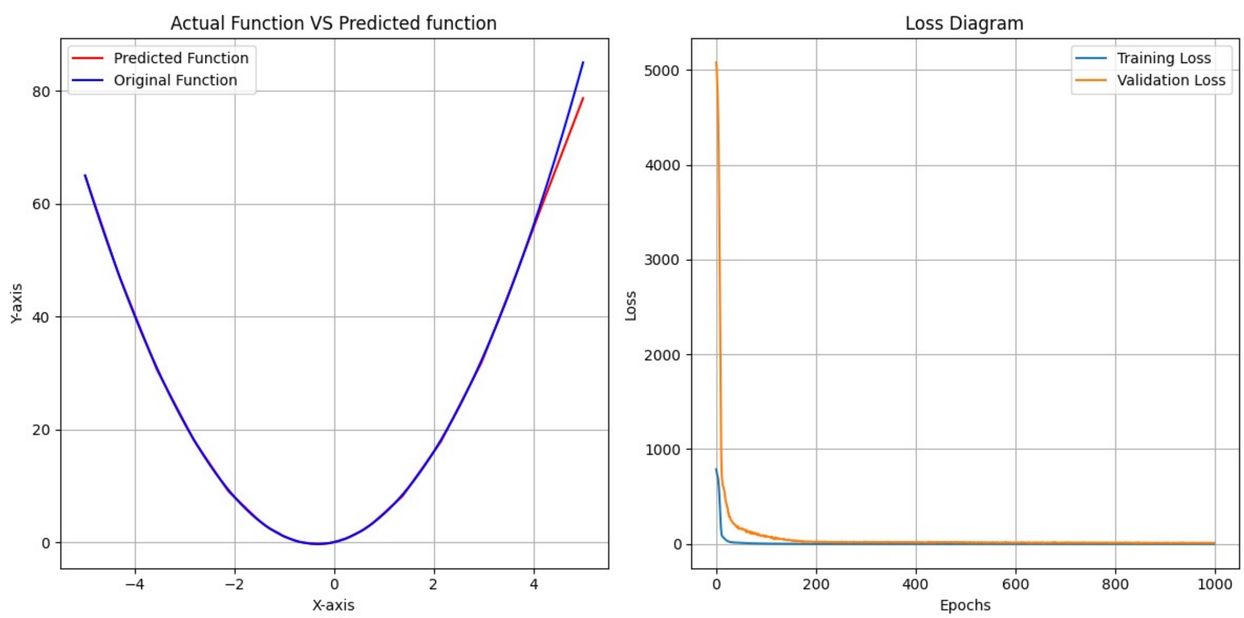
```
functions = [linear,nonlinear,Complex]

for function in functions:
  X_train = np.linspace(-5, 5, 500).reshape(-1, 1)
  y_train = function(X_train)
```

```
  model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1)
  ])

  model.compile(optimizer='adam', loss='mean_squared_error')

  history = model.fit(X_train, y_train,batch_size=16,
epochs=1000,validation_split=0.1,verbose=0)

  X_test = np.linspace(-5,5,500).reshape(-1,1)

  y_pred_function = model.predict(X_test)


  mse = np.mean(np.square(y_pred_function - function(X_train)))

  plt.figure(figsize=(12, 6))

  plt.subplot(1, 2, 1)
  plt.plot(X_test, y_pred_function, color='red', label='Predicted
Function')
  plt.plot(X_train, function(X_train), color='blue', label='Original
Function')
  plt.xlabel('X-axis')
  plt.ylabel('Y-axis')
  plt.title('Actual Function VS Predicted function')
  plt.legend()
  plt.grid(True)

  plt.subplot(1, 2, 2)
  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('Loss Diagram')
  plt.legend()
  plt.grid(True)

  plt.tight_layout()
  plt.show()

  print(f"Mean Squared Error: {mse:.4f}")
```

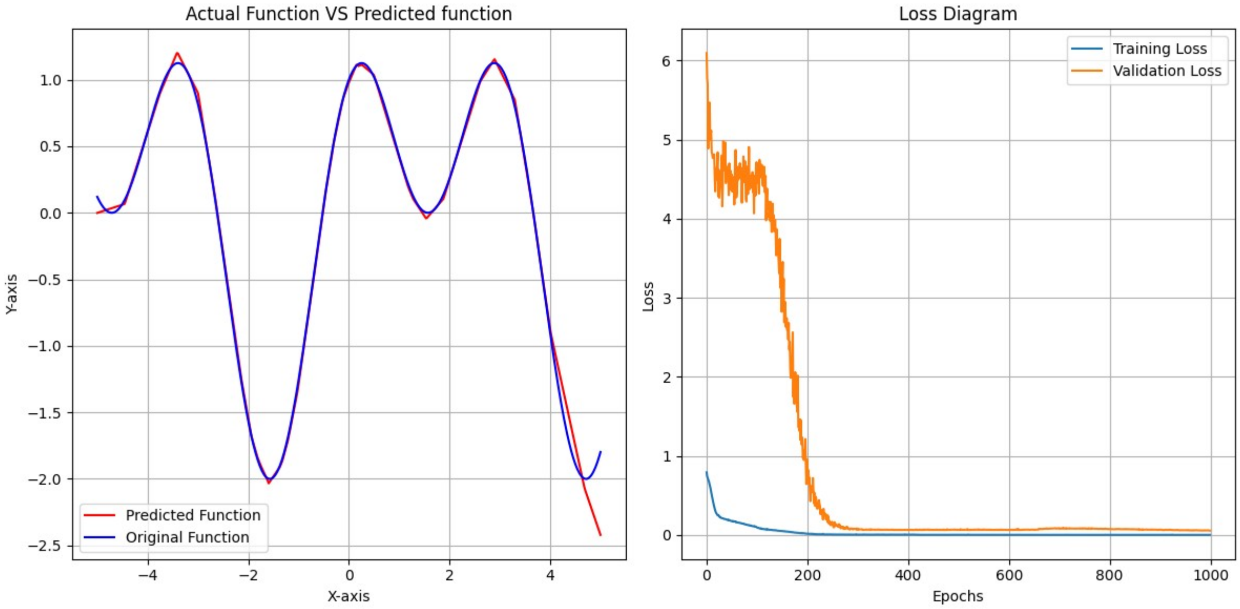
```
16/16 [==============================] - 0s 2ms/step
```




```
Mean Squared Error: 0.0000
16/16 [==============================] - 0s 2ms/step
```



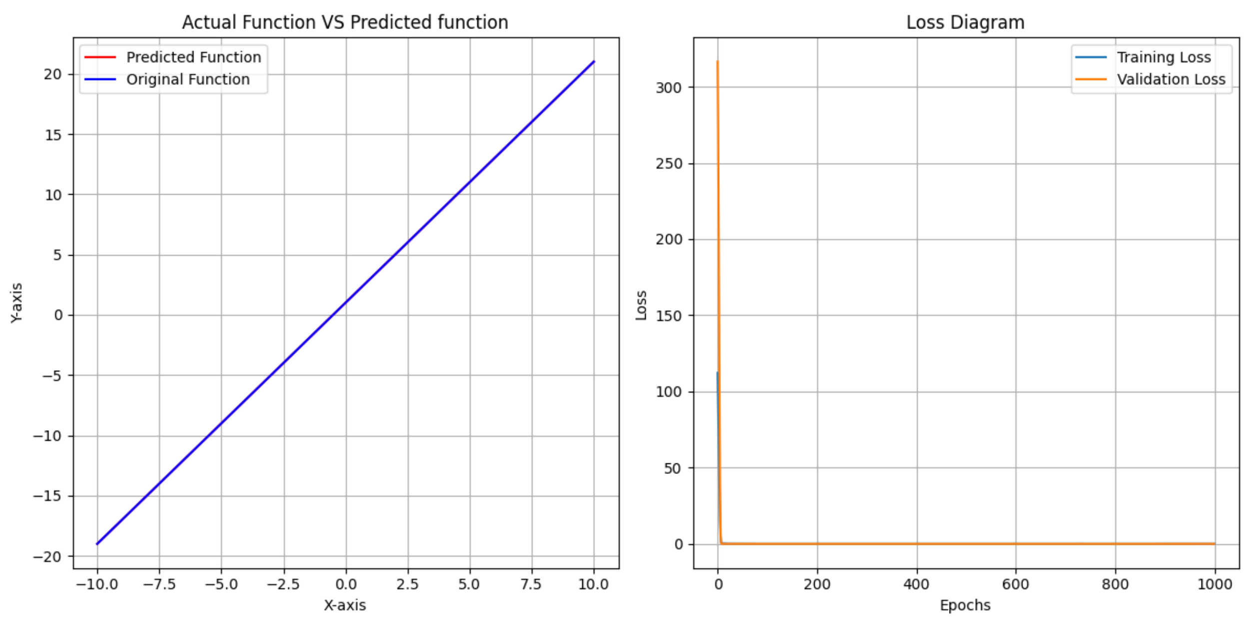

```
Mean Squared Error: 1.1664
16/16 [==============================] - 0s 2ms/step
```



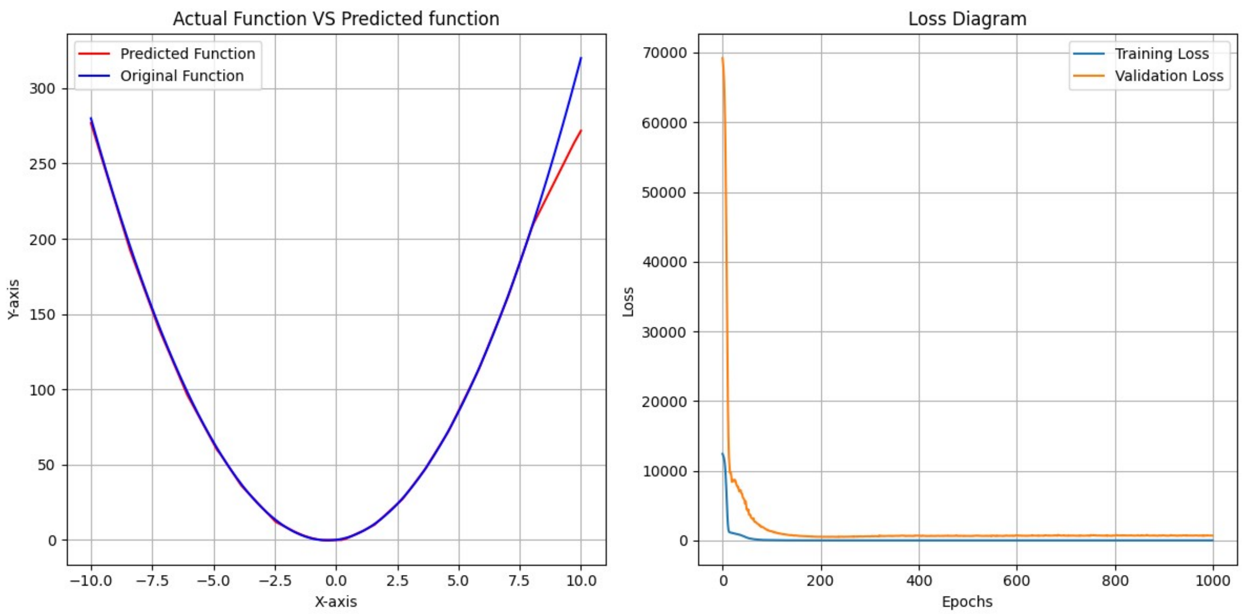

```
Mean Squared Error: 0.0060

functions = [linear,nonlinear,Complex]

for function in functions:
  X_train = np.linspace(-10, 10, 500).reshape(-1, 1)
  y_train = function(X_train)

  model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1)
  ])

  model.compile(optimizer='adam', loss='mean_squared_error')

  history = model.fit(X_train, y_train,batch_size=16,
epochs=1000,validation_split=0.1,verbose=0)

  X_test = np.linspace(-10,10,500).reshape(-1,1)

  y_pred_function = model.predict(X_test)


  mse = np.mean(np.square(y_pred_function - function(X_train)))

  plt.figure(figsize=(12, 6))

  plt.subplot(1, 2, 1)
  plt.plot(X_test, y_pred_function, color='red', label='Predicted
Function')
```

```
  plt.plot(X_train, function(X_train), color='blue', label='Original
Function')
  plt.xlabel('X-axis')
  plt.ylabel('Y-axis')
  plt.title('Actual Function VS Predicted function')
  plt.legend()
  plt.grid(True)

  plt.subplot(1, 2, 2)
  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('Loss Diagram')
  plt.legend()
  plt.grid(True)

  plt.tight_layout()
  plt.show()

  print(f"Mean Squared Error: {mse:.4f}")
```

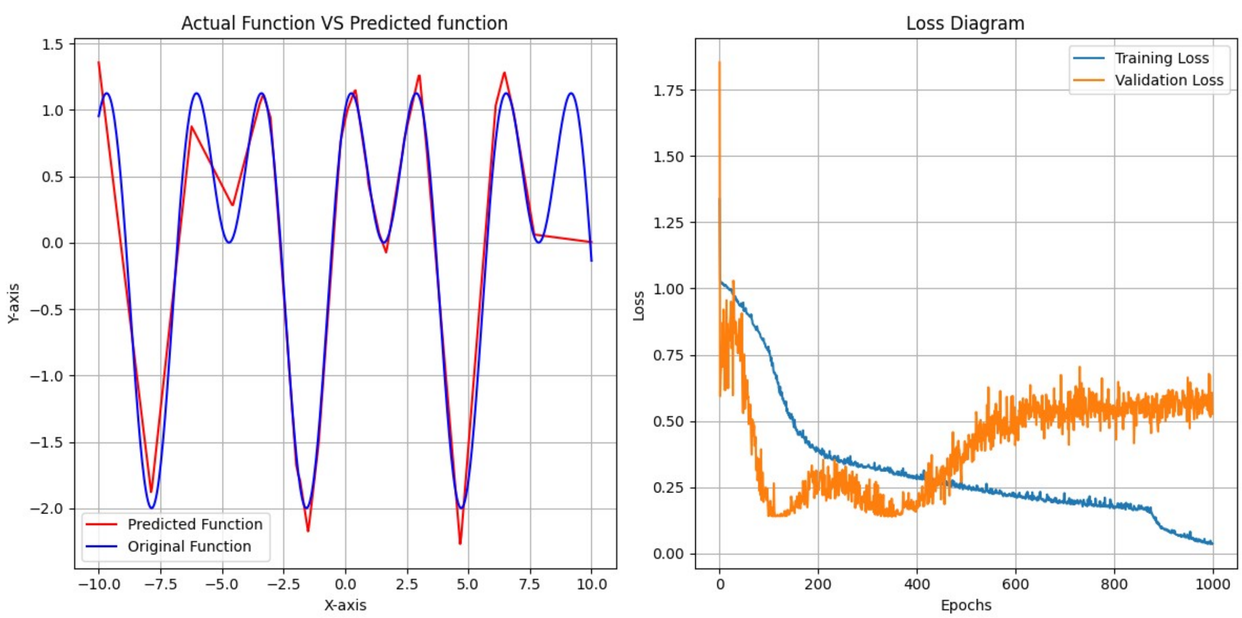
```
16/16 [==============================] - 0s 2ms/step
```




```
Mean Squared Error: 0.0000
16/16 [==============================] - 0s 2ms/step
```




```
Mean Squared Error: 68.7150
16/16 [==============================] - 0s 2ms/step
```

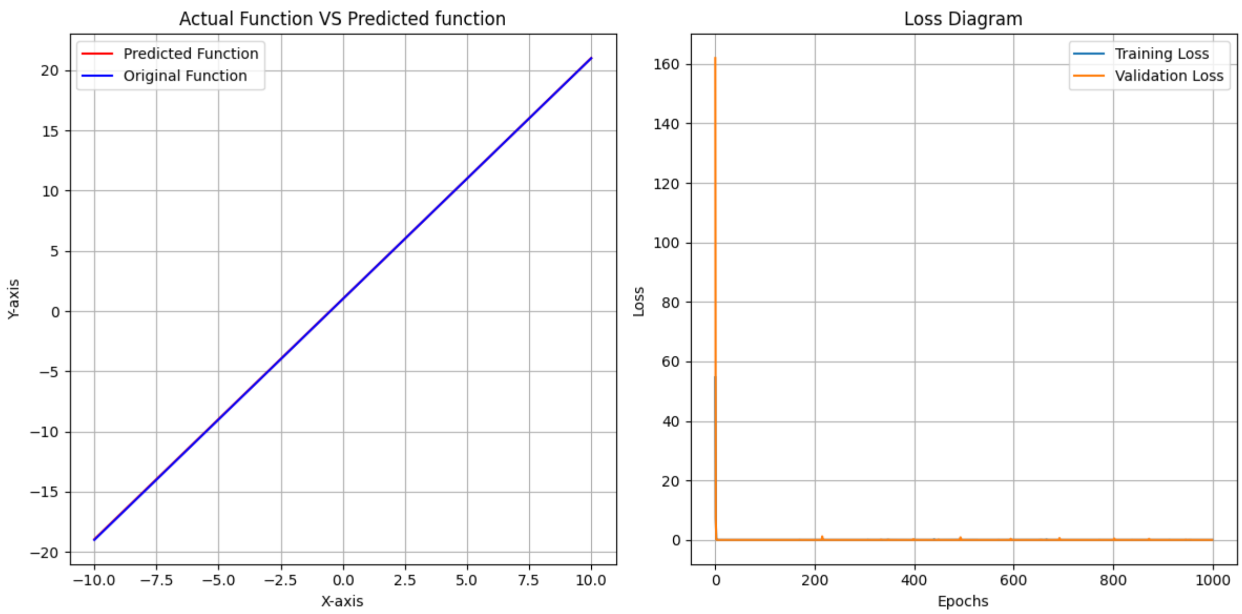



```
Mean Squared Error: 0.0859
```

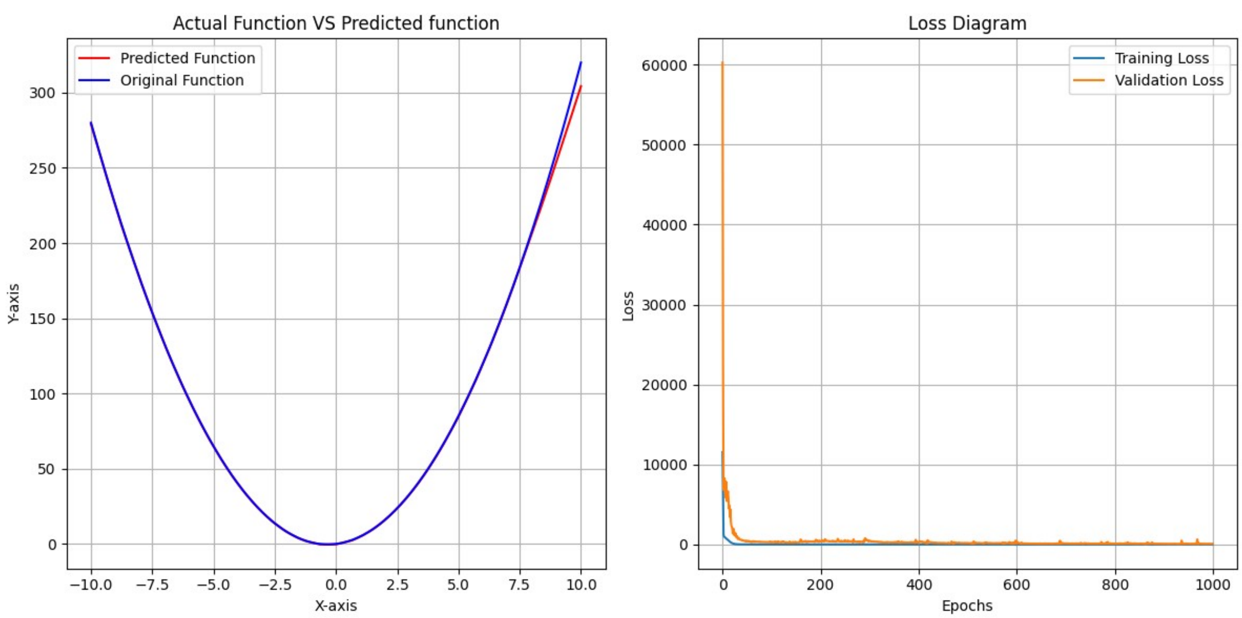
```
functions = [linear,nonlinear,Complex]

for function in functions:
  X_train = np.linspace(-10, 10, 500).reshape(-1, 1)
  y_train = function(X_train)
```

```
  model = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(1)
  ])

  model.compile(optimizer='adam', loss='mean_squared_error')

  history = model.fit(X_train, y_train,batch_size=16,
epochs=1000,validation_split=0.1,verbose=0)

  X_test = np.linspace(-10,10,500).reshape(-1,1)

  y_pred_function = model.predict(X_test)

  mse = np.mean(np.square(y_pred_function - function(X_train)))

  plt.figure(figsize=(12, 6))

  plt.subplot(1, 2, 1)
  plt.plot(X_test, y_pred_function, color='red', label='Predicted
Function')
  plt.plot(X_train, function(X_train), color='blue', label='Original
Function')
  plt.xlabel('X-axis')
  plt.ylabel('Y-axis')
  plt.title('Actual Function VS Predicted function')
  plt.legend()
  plt.grid(True)

  plt.subplot(1, 2, 2)
  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('Loss Diagram')
  plt.legend()
  plt.grid(True)

  plt.tight_layout()
  plt.show()

  print(f"Mean Squared Error: {mse:.4f}")
```

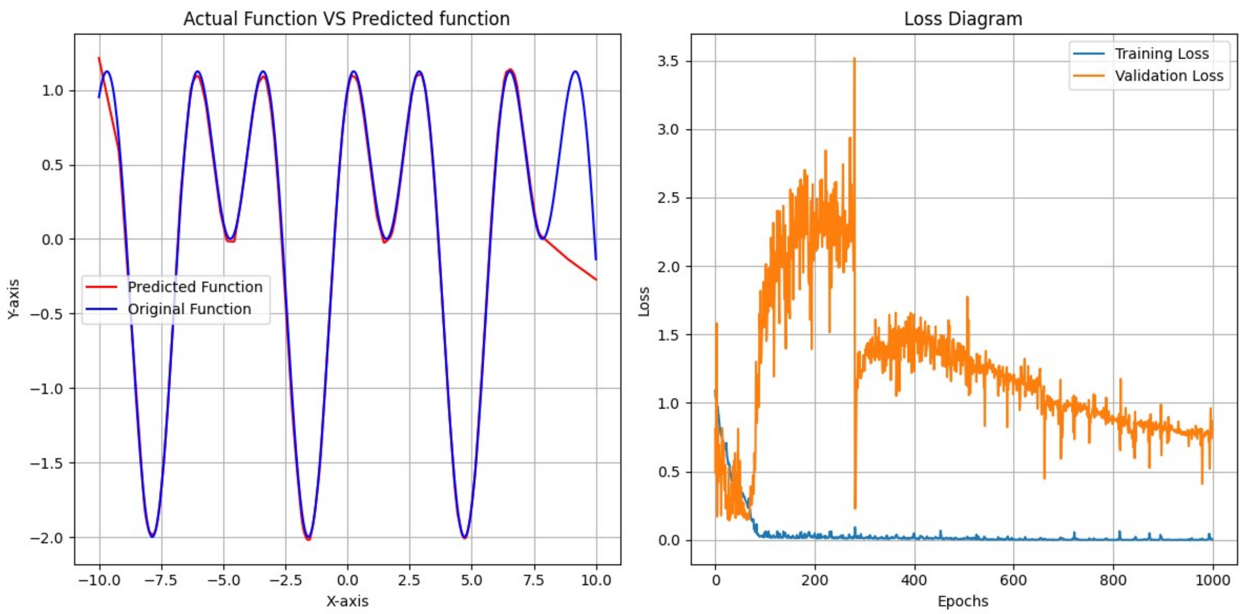
```
16/16 [==============================] - 0s 2ms/step
```




```
Mean Squared Error: 0.0004
16/16 [==============================] - 0s 2ms/step
```




```
Mean Squared Error: 7.3657
16/16 [==============================] - 0s 2ms/step
```




```
Mean Squared Error: 0.0813
```

- In the first run I used 200 data points, 3 layers of neurons with 10 in each layer, X_range between -5 and 5, 300 epochs. the first linear function was almost fully accurate even without changing any factor, now we argue other functions:

  In the second run, I changed the epochs to 1000 which decreased the Error for both nonlinear and complex functions.

  In the third run, I changed the number of data points to 500 which again made the predictions even more accurate.

  In the forth run, I chaged the X_range to -10 and 10 which led to less accuracy in predicted function for both functions, although complex function was more accurate in the 400 epochs of fitting than 1000 epochs when points number was 500.

  Finally I increased the number of neurons to 100 in each layer which slightly improved the results of my predictions, but still the wideness of X_range has its negetave effect.

#Question 2

Here I try to add noise in three levels to my data points, first 5% of noise, then 10% of noise, and finally 25% of noise.

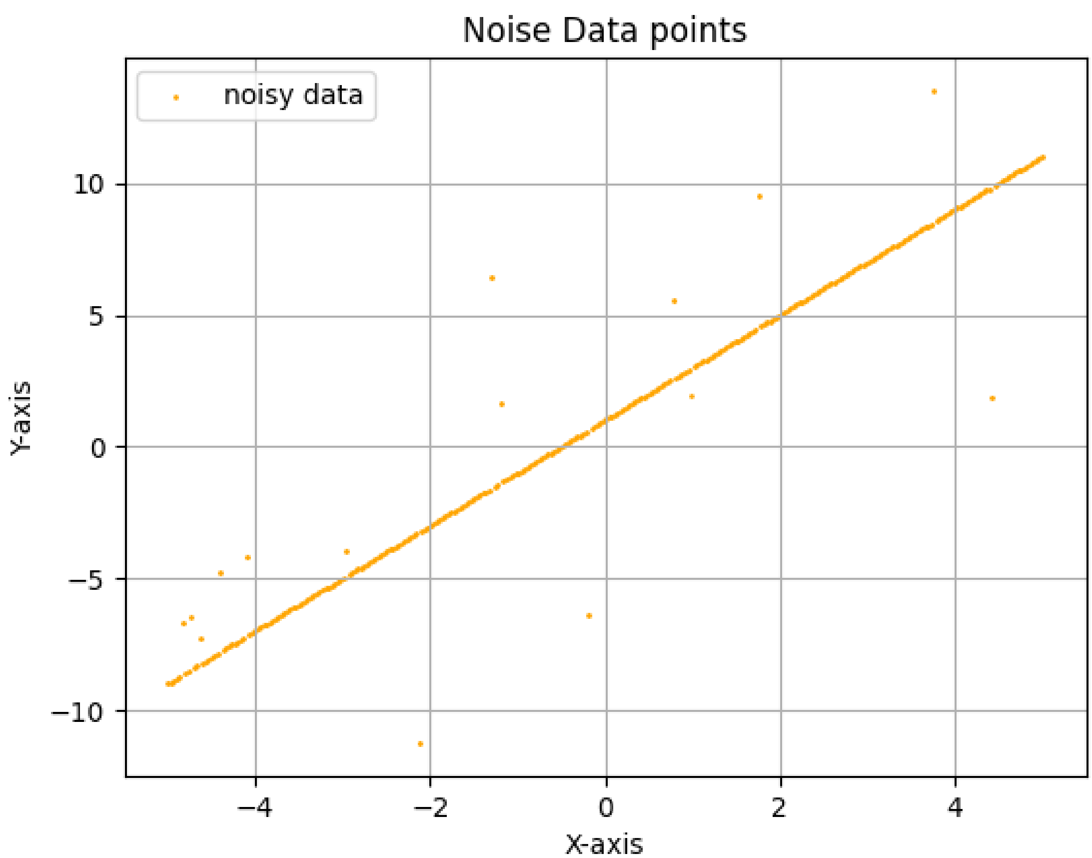
The final goal is to see how the results of my prediction will change.

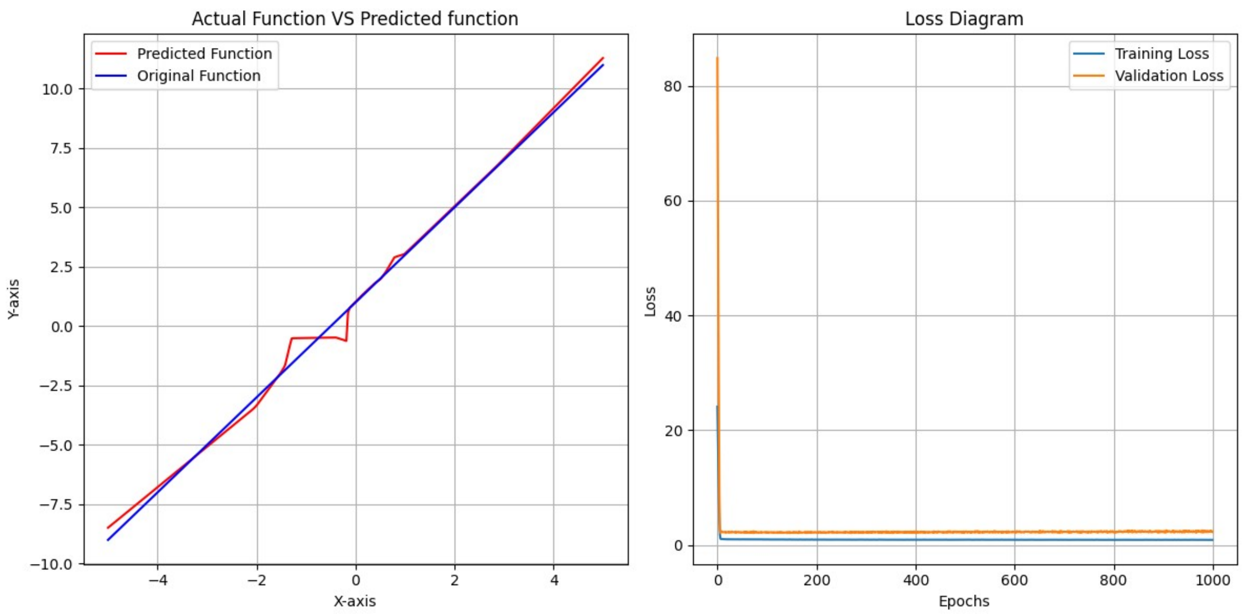
```
functions = [linear, nonlinear, Complex]
noise_level = 0.05

for function in functions:
    X_train = np.linspace(-5, 5, 300).reshape(-1, 1)
    y_train = function(X_train)
```

```
     # Select a portion of points randomly and add noise to them
    num_noisy_points = int(len(y_train) * noise_level)
    noisy_indices = np.random.choice(len(y_train), num_noisy_points,
replace=False)
    y_train_noisy = y_train.copy()

    # Generate noise for the selected points
    noise =
np.random.randint(y_train.min(),y_train.max(),size=num_noisy_points)
    y_train_noisy[noisy_indices] += noise.reshape(-1, 1)

    model = tf.keras.Sequential([
    tf.keras.layers.Dense(20, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dense(1)
    ])

    model.compile(optimizer='adam', loss='mean_squared_error')

    history = model.fit(X_train, y_train_noisy,batch_size=16,
epochs=1000, verbose=0,validation_split=0.1)

    X_test = np.linspace(-5, 5, 300).reshape(-1, 1)
    y_pred_function = model.predict(X_test)

    plt.scatter(X_train,y_train_noisy,color='orange',label='noisy
data',s=1)

    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Noise Data points')
    plt.legend()
    plt.grid(True)
    plt.show()

    mse = np.mean(np.square(y_pred_function - function(X_test)))

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(X_test, y_pred_function, color='red', label='Predicted
Function')
    plt.plot(X_train, function(X_train), color='blue', label='Original
Function')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Actual Function VS Predicted function')
    plt.legend()
    plt.grid(True)
```

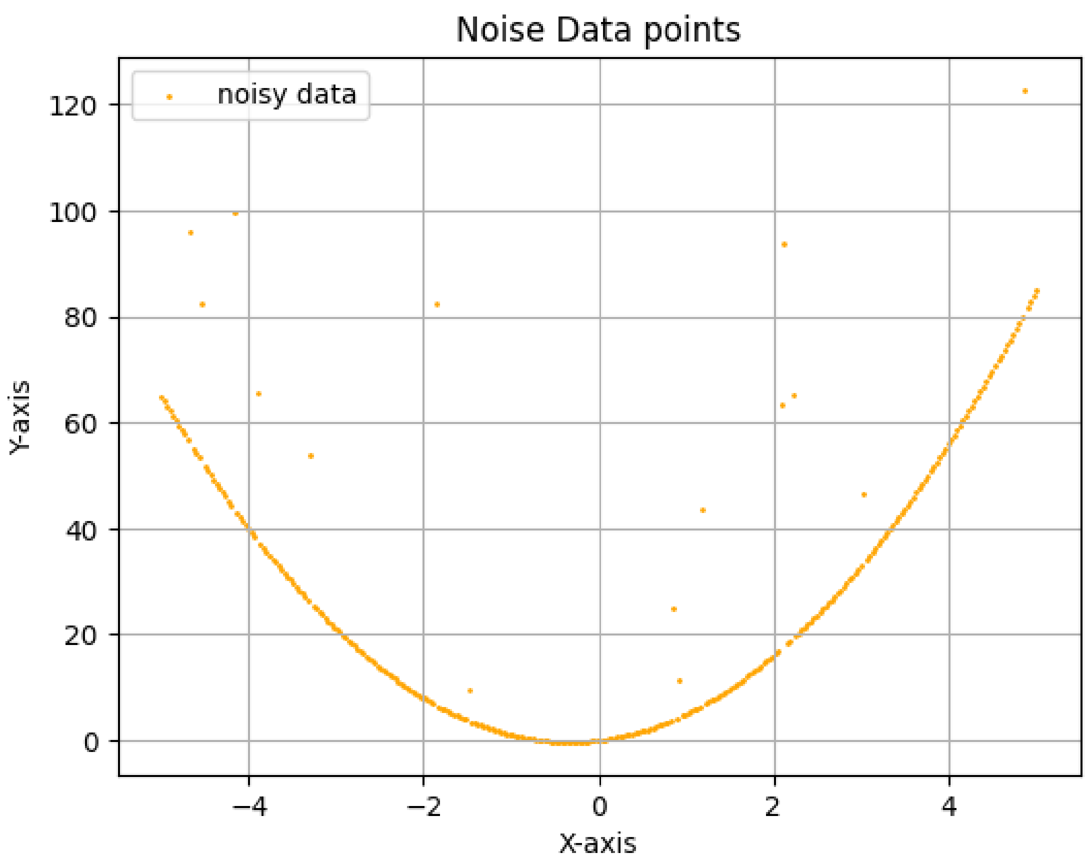
```
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Loss Diagram')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

    print(f"Noise Level: {noise_level*100}%, Mean Squared Error:
{mse:.4f}")
```

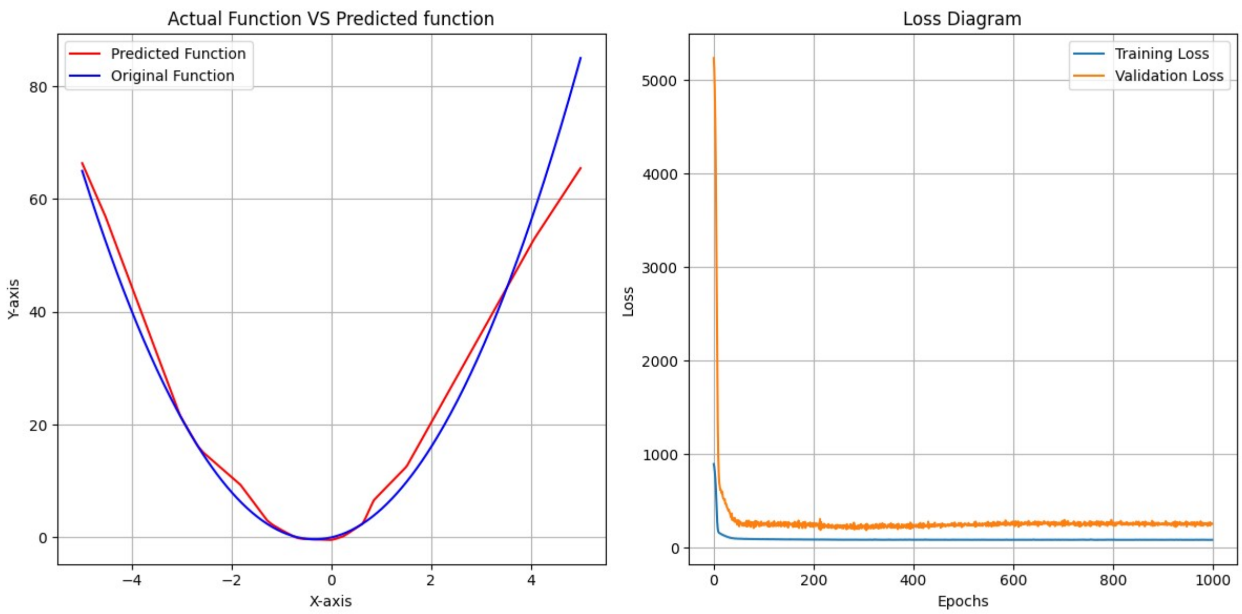
```
10/10 [==============================] - 0s 2ms/step
```



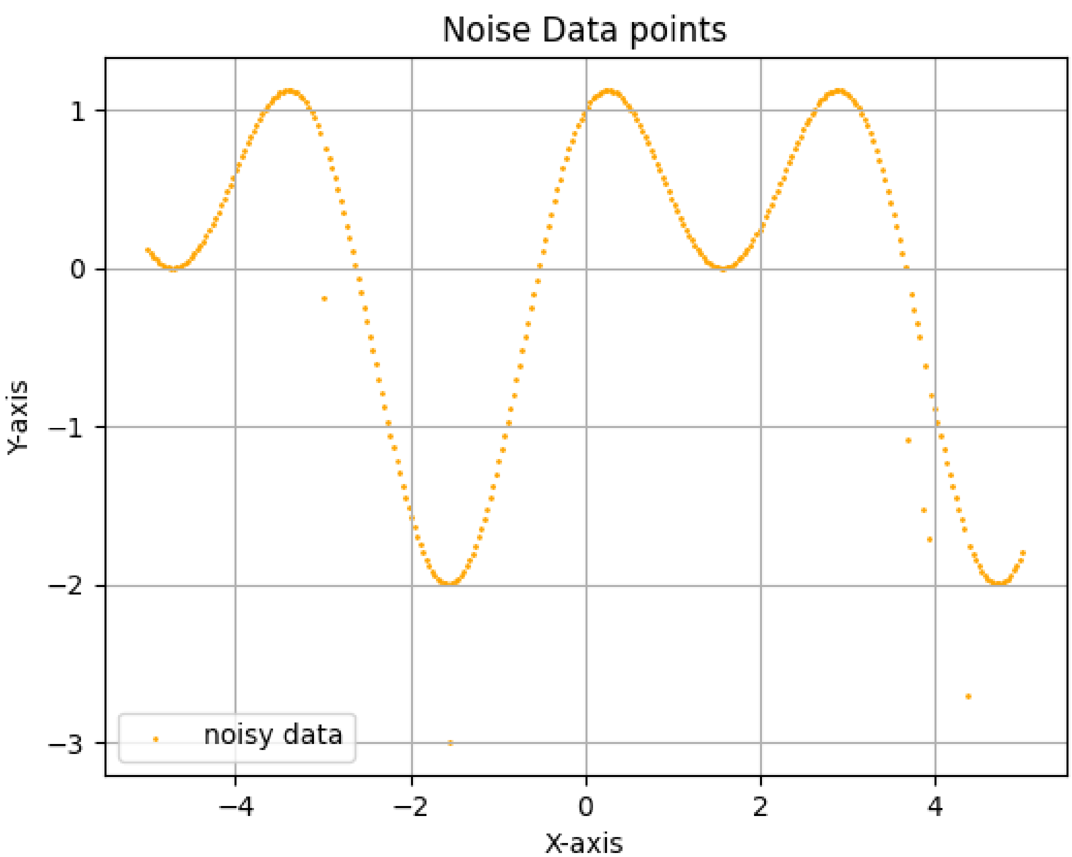


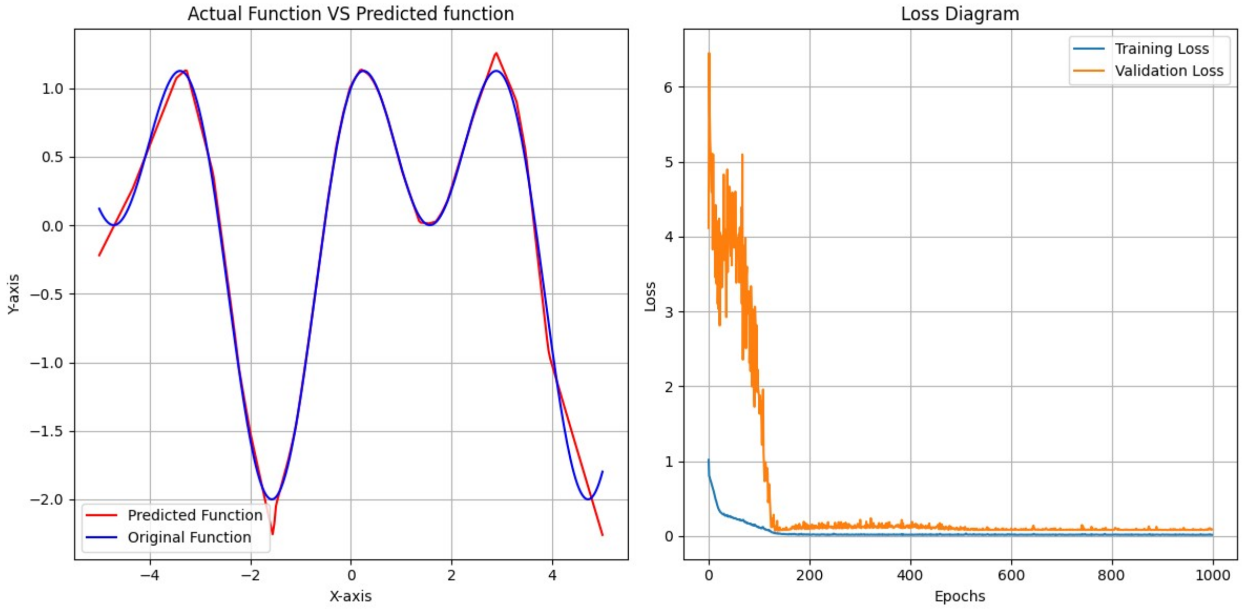

```
Noise Level: 5.0%, Mean Squared Error: 0.0880
10/10 [==============================] - 0s 2ms/step
```






```
Noise Level: 5.0%, Mean Squared Error: 21.6792
10/10 [==============================] - 0s 2ms/step
```

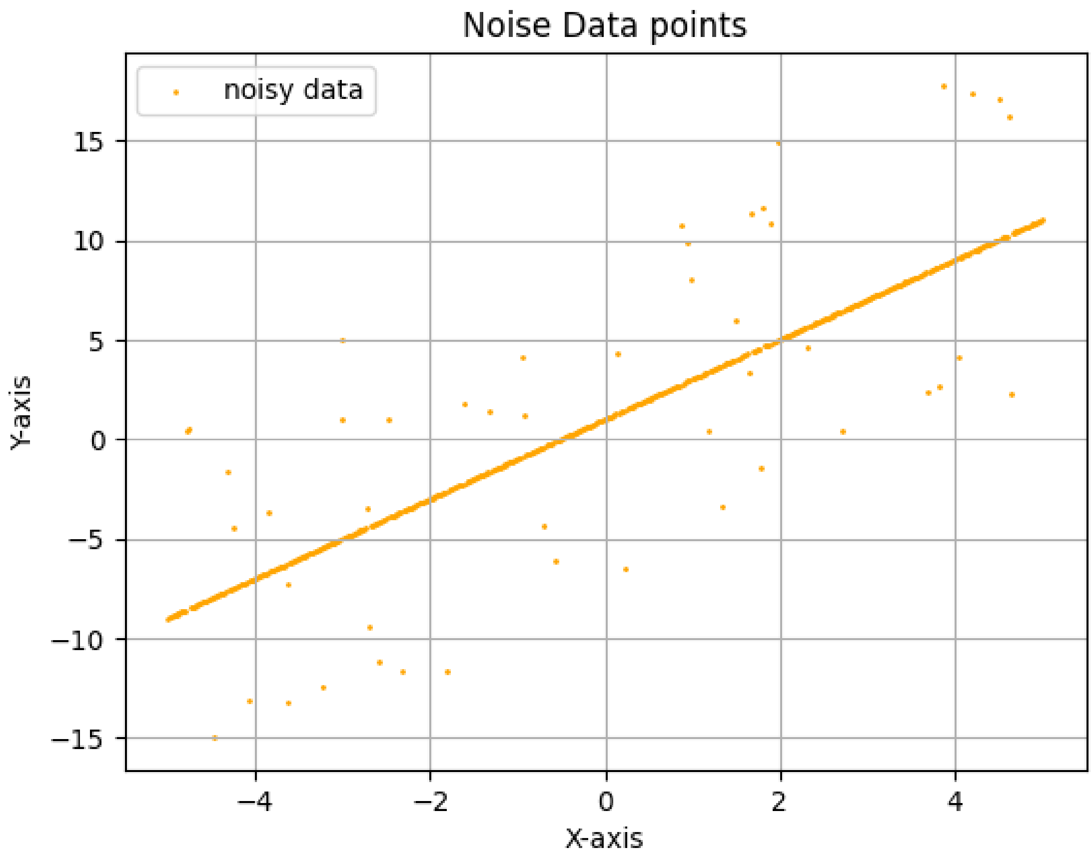


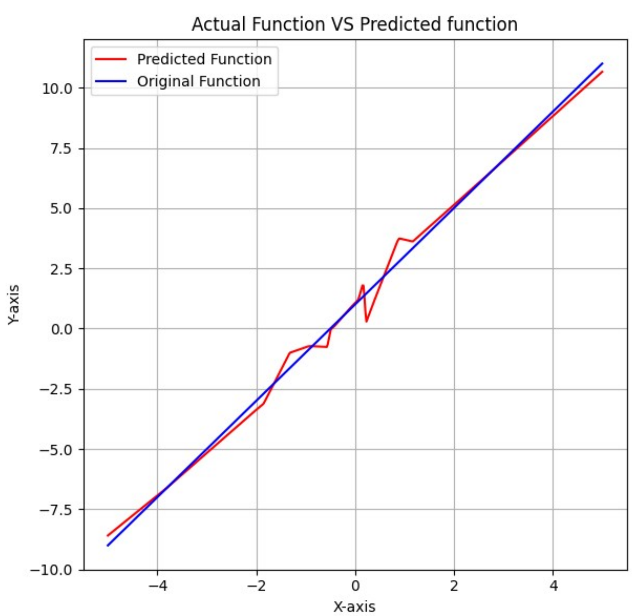


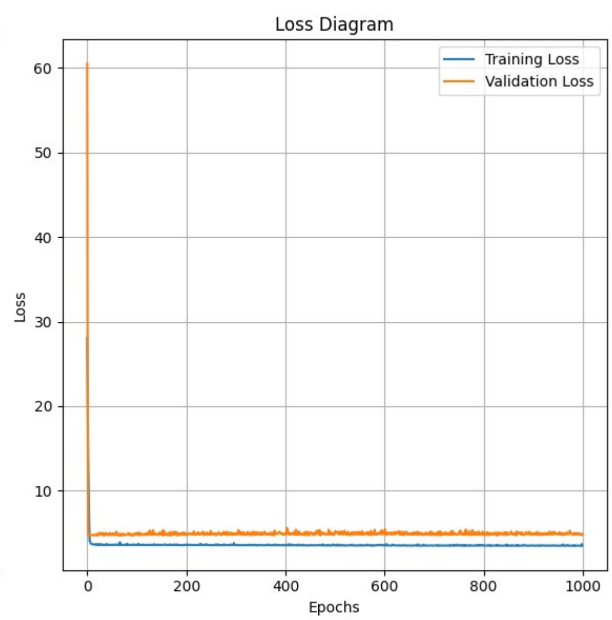

```
Noise Level: 5.0%, Mean Squared Error: 0.0082

functions = [linear, nonlinear, Complex]
noise_level = 0.1

for function in functions:
    X_train = np.linspace(-5, 5, 500).reshape(-1, 1)
    y_train = function(X_train)

     # Select a portion of points randomly and add noise to them
    num_noisy_points = int(len(y_train) * noise_level)
    noisy_indices = np.random.choice(len(y_train), num_noisy_points,
replace=False)
    y_train_noisy = y_train.copy()

    # Generate noise for the selected points
    noise =
np.random.randint(y_train.min(),y_train.max(),size=num_noisy_points)
    y_train_noisy[noisy_indices] += noise.reshape(-1, 1)

    model = tf.keras.Sequential([
    tf.keras.layers.Dense(20, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dense(1)
    ])

    model.compile(optimizer='adam', loss='mean_squared_error')

    history = model.fit(X_train, y_train_noisy,batch_size=16,
epochs=1000, verbose=0,validation_split=0.1)
```

```
    X_test = np.linspace(-5, 5, 500).reshape(-1, 1)
    y_pred_function = model.predict(X_test)

    plt.scatter(X_train,y_train_noisy,color='orange',label='noisy
data',s=1)

    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Noise Data points')
    plt.legend()
    plt.grid(True)
    plt.show()

    mse = np.mean(np.square(y_pred_function - function(X_test)))

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(X_test, y_pred_function, color='red', label='Predicted
Function')
    plt.plot(X_train, function(X_train), color='blue', label='Original
Function')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Actual Function VS Predicted function')
    plt.legend()
    plt.grid(True)

    plt.subplot(1, 2, 2)
    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('Loss Diagram')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()
    print(f"Noise Level: {noise_level*100}%, Mean Squared Error:
{mse:.4f}")
```

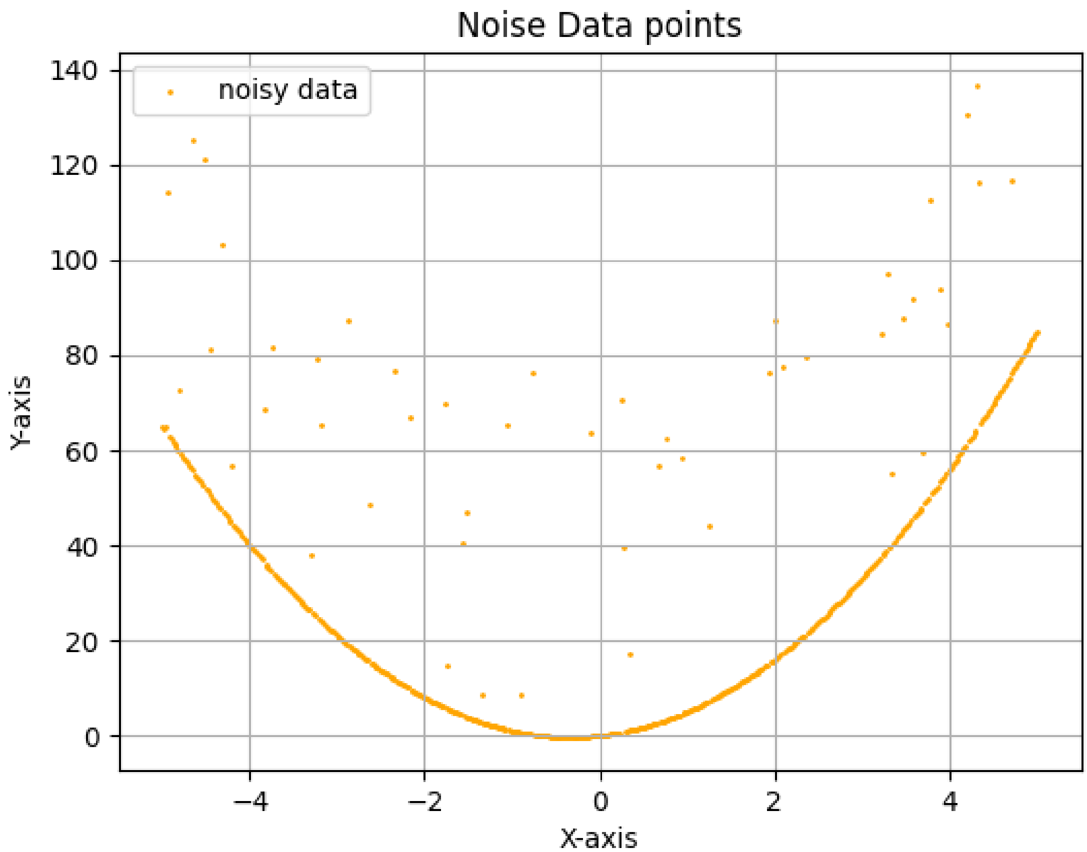
```
16/16 [==============================] - 0s 2ms/step
```

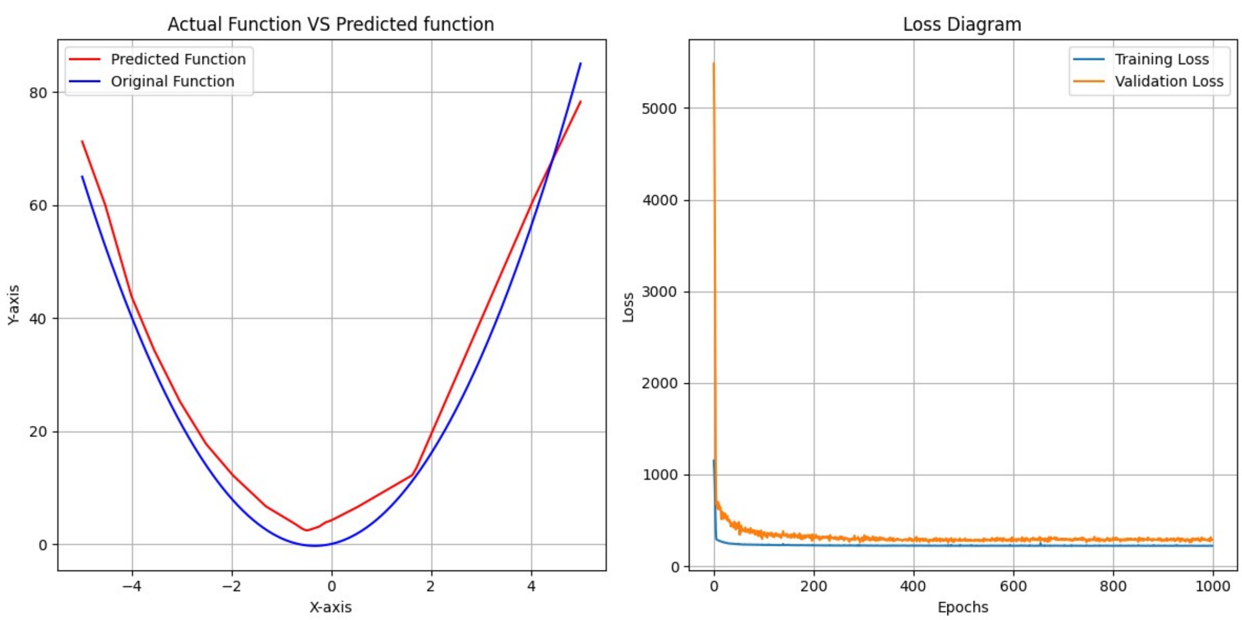




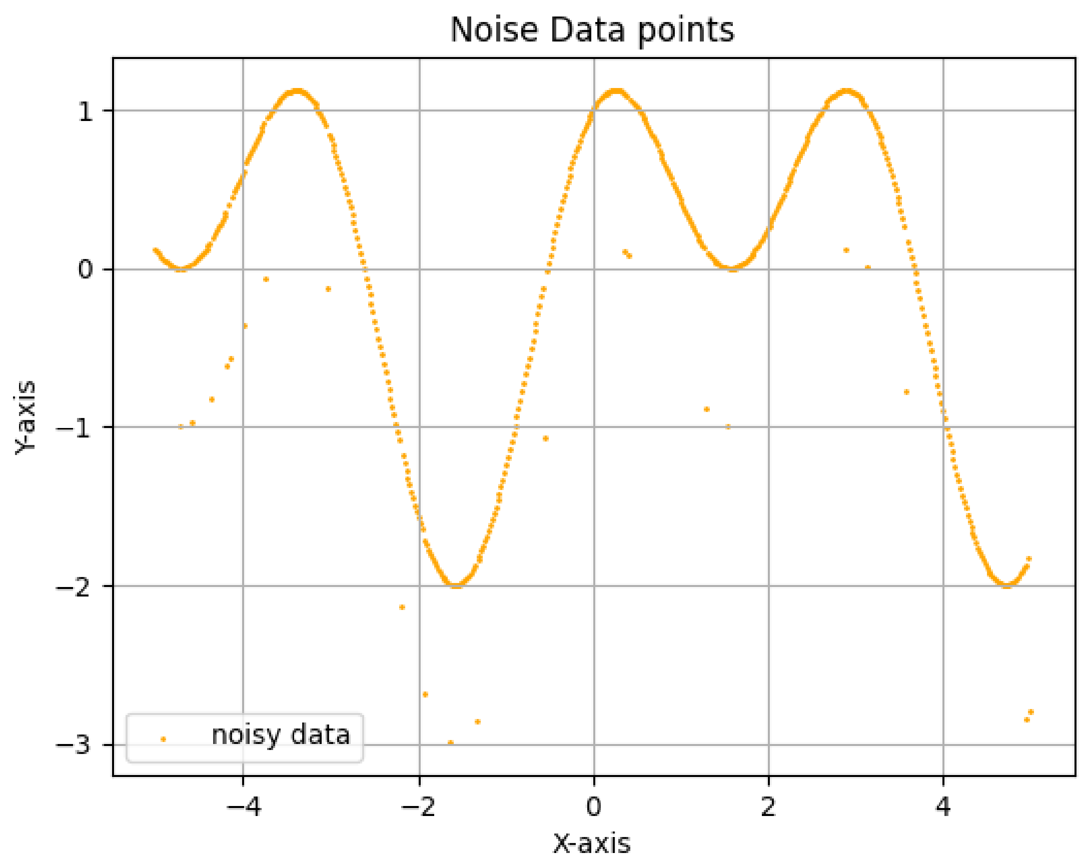


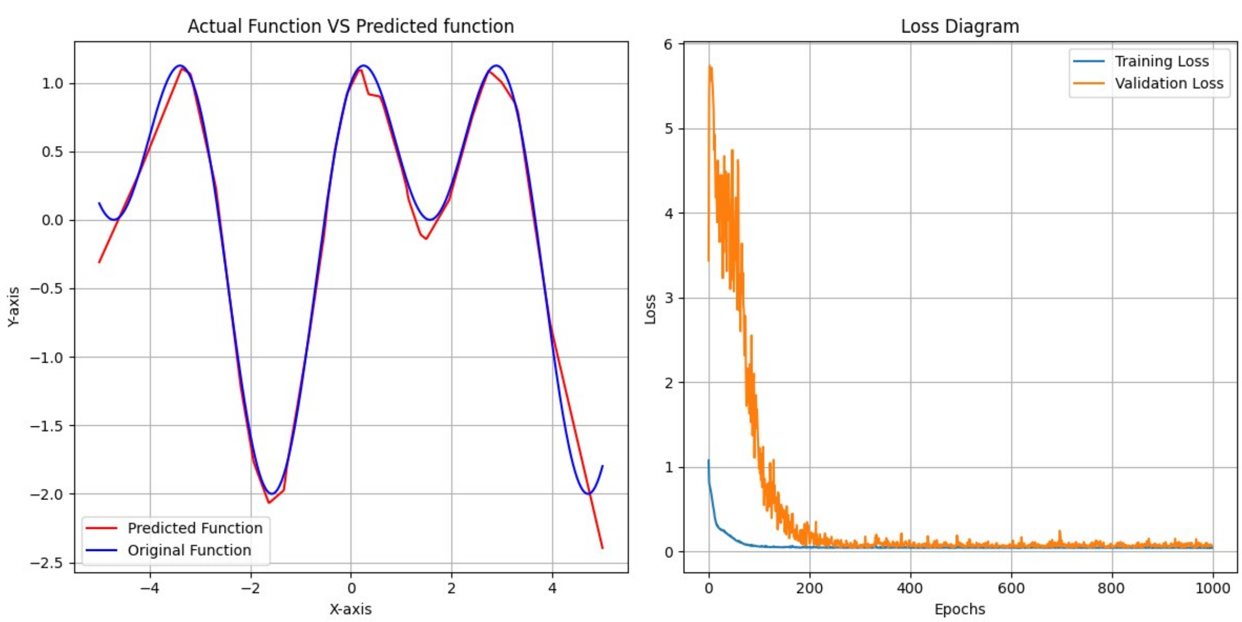

```
Noise Level: 10.0%, Mean Squared Error: 0.0893
16/16 [==============================] - 0s 2ms/step
```






```
Noise Level: 10.0%, Mean Squared Error: 20.1808
16/16 [==============================] - 0s 2ms/step
```



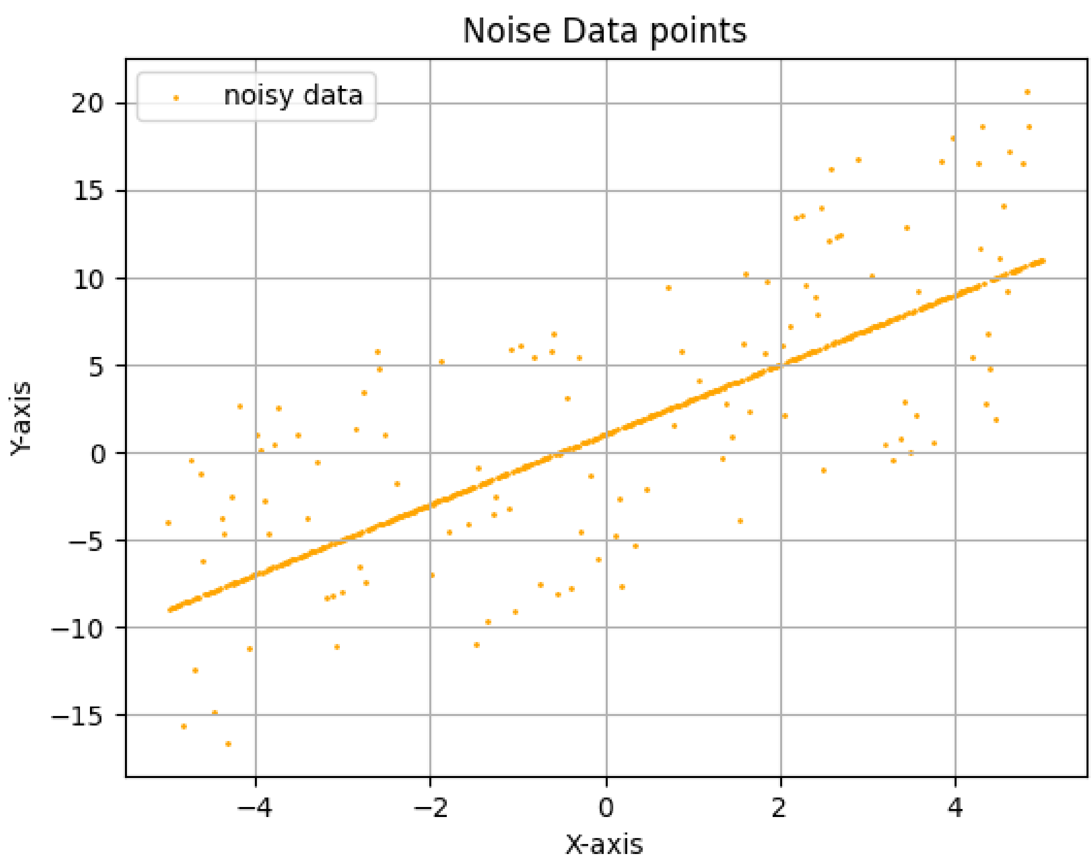


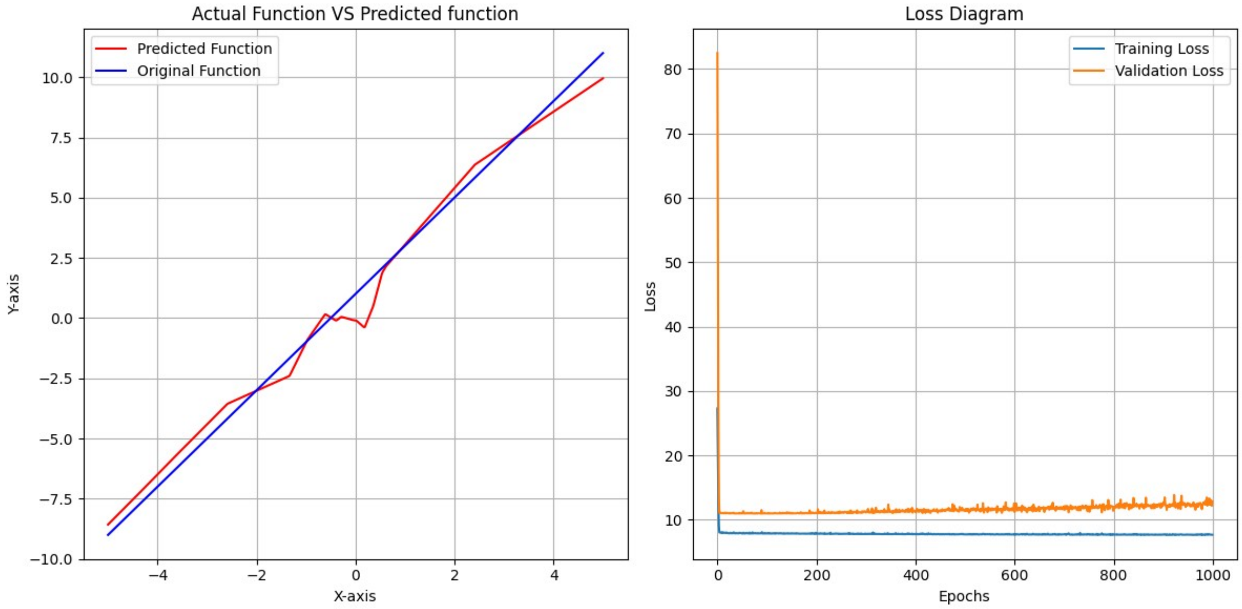

```
Noise Level: 10.0%, Mean Squared Error: 0.0133

functions = [linear, nonlinear, Complex]
noise_level = 0.25
```

```
for function in functions:
    X_train = np.linspace(-5, 5, 500).reshape(-1, 1)
    y_train = function(X_train)

     # Select a portion of points randomly and add noise to them
    num_noisy_points = int(len(y_train) * noise_level)
    noisy_indices = np.random.choice(len(y_train), num_noisy_points,
replace=False)
    y_train_noisy = y_train.copy()

    # Generate noise for the selected points
    noise =
np.random.randint(y_train.min(),y_train.max(),size=num_noisy_points)
    y_train_noisy[noisy_indices] += noise.reshape(-1, 1)

    model = tf.keras.Sequential([
    tf.keras.layers.Dense(20, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dense(20, activation='relu'),
    tf.keras.layers.Dense(1)
    ])

    model.compile(optimizer='adam', loss='mean_squared_error')

    history = model.fit(X_train, y_train_noisy,batch_size=16,
epochs=1000, verbose=0,validation_split=0.1)

    X_test = np.linspace(-5, 5, 500).reshape(-1, 1)
    y_pred_function = model.predict(X_test)

    plt.scatter(X_train,y_train_noisy,color='orange',label='noisy
data',s=1)

    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Noise Data points')
    plt.legend()
    plt.grid(True)
    plt.show()

    mse = np.mean(np.square(y_pred_function - function(X_test)))

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(X_test, y_pred_function, color='red', label='Predicted
Function')
    plt.plot(X_train, function(X_train), color='blue', label='Original
Function')
```

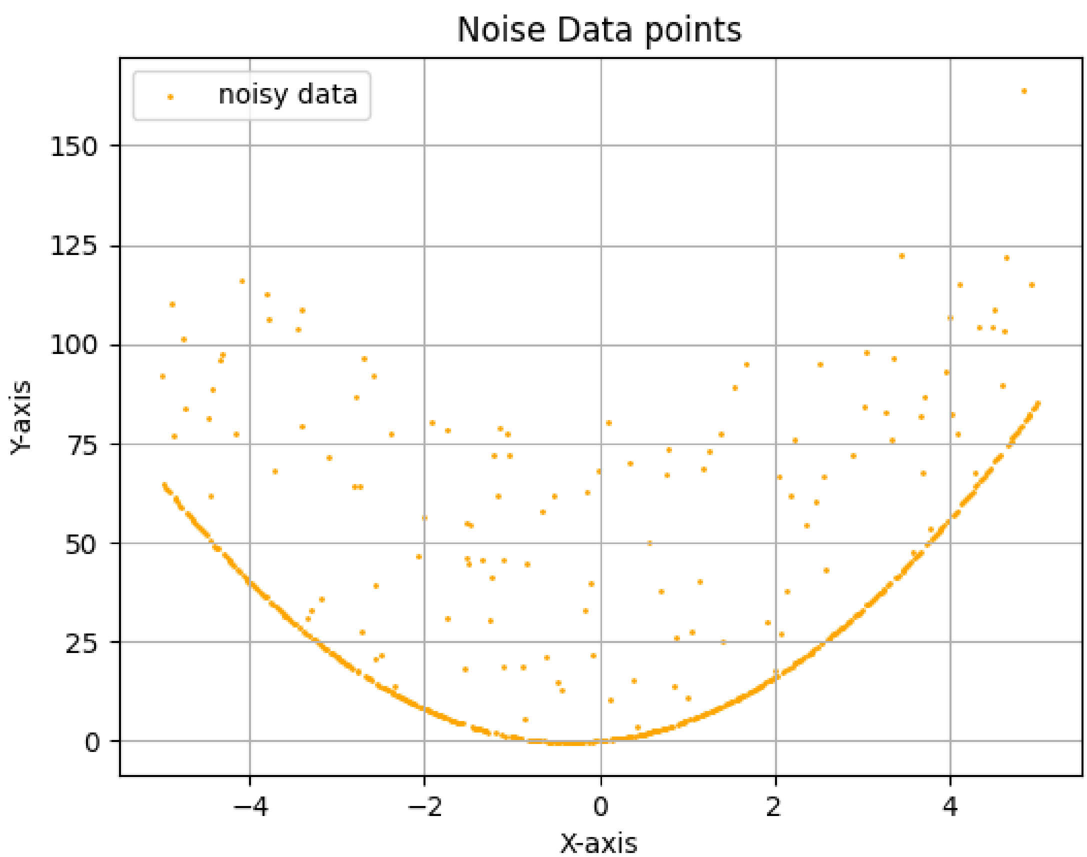
```
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')
    plt.title('Actual Function VS Predicted function')
    plt.legend()
    plt.grid(True)

    plt.subplot(1, 2, 2)
    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('Loss Diagram')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()
    print(f"Noise Level: {noise_level*100}%, Mean Squared Error:
{mse:.4f}")
```

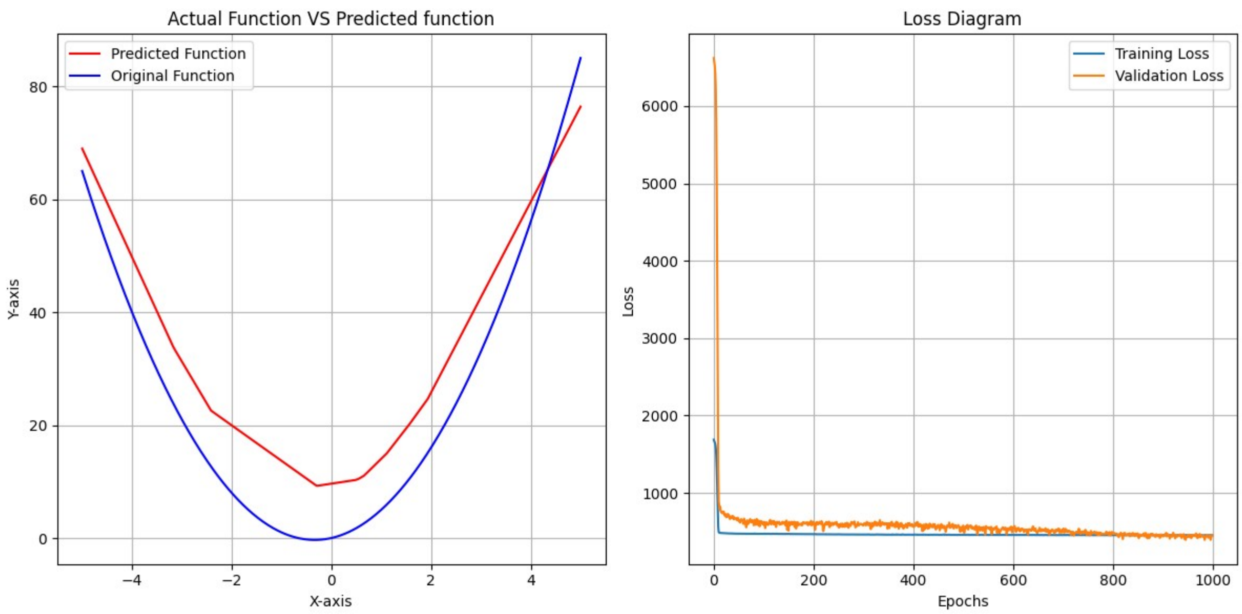
```
16/16 [==============================] - 0s 2ms/step
```



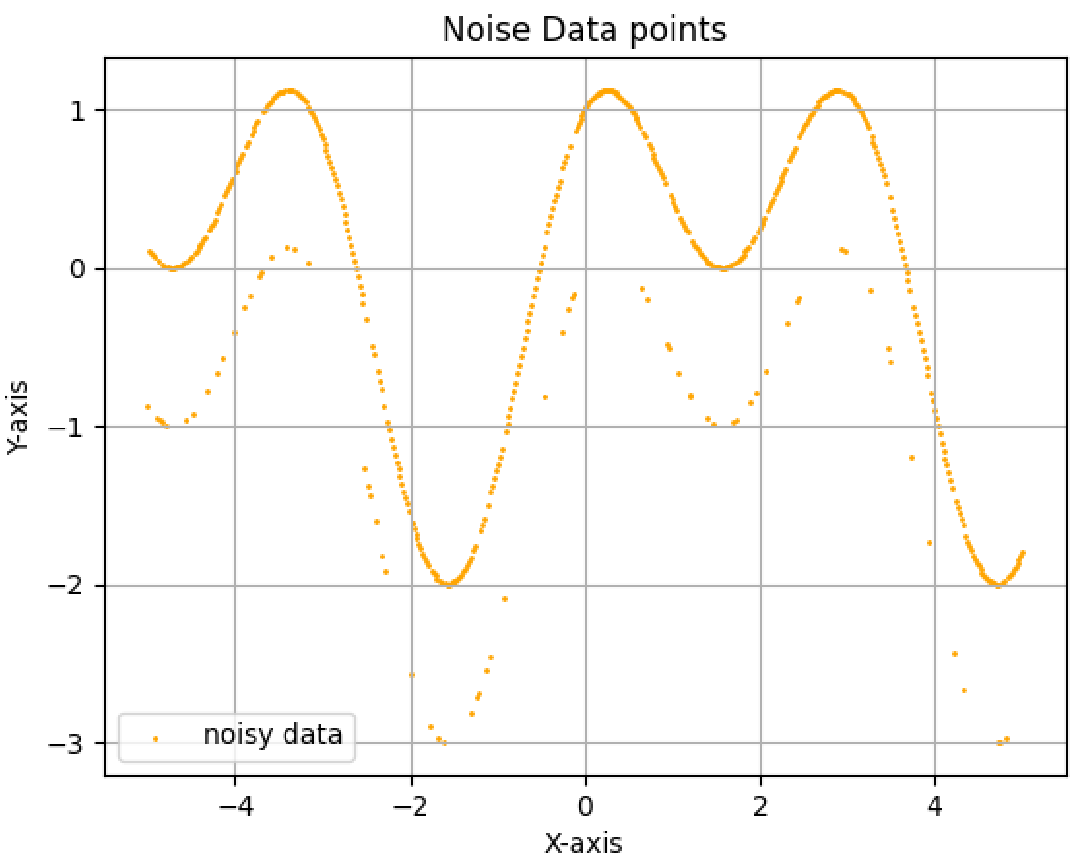


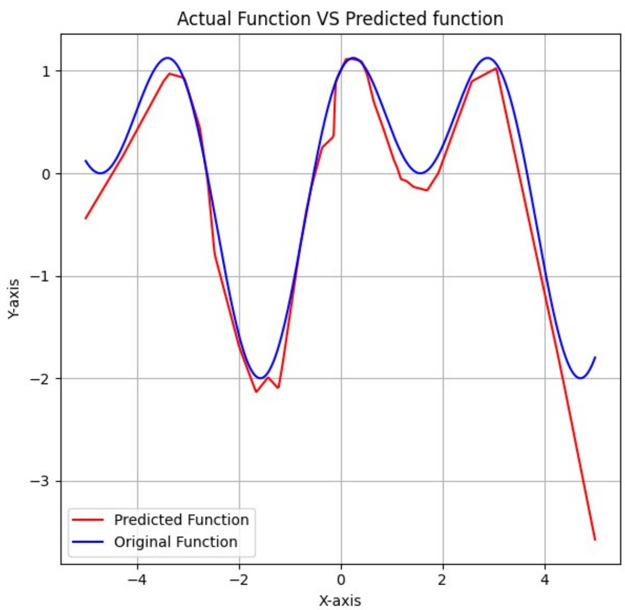

```
Noise Level: 25.0%, Mean Squared Error: 0.2899
16/16 [==============================] - 0s 2ms/step
```

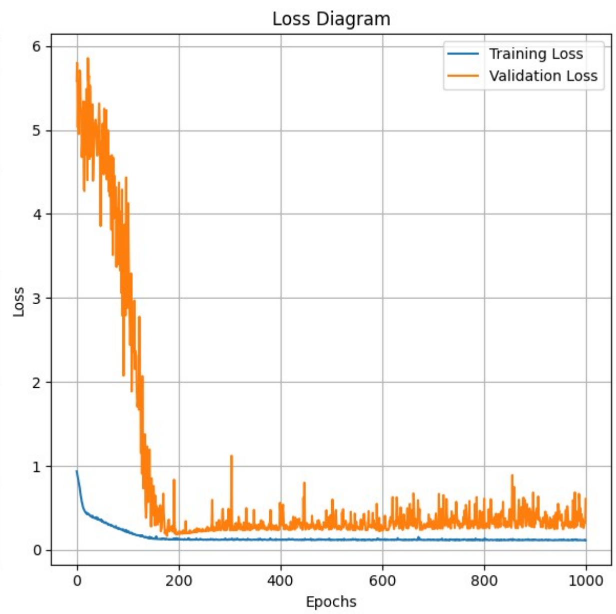





```
Noise Level: 25.0%, Mean Squared Error: 89.9824
16/16 [==============================] - 0s 2ms/step
```








```
Noise Level: 25.0%, Mean Squared Error: 0.1006
```

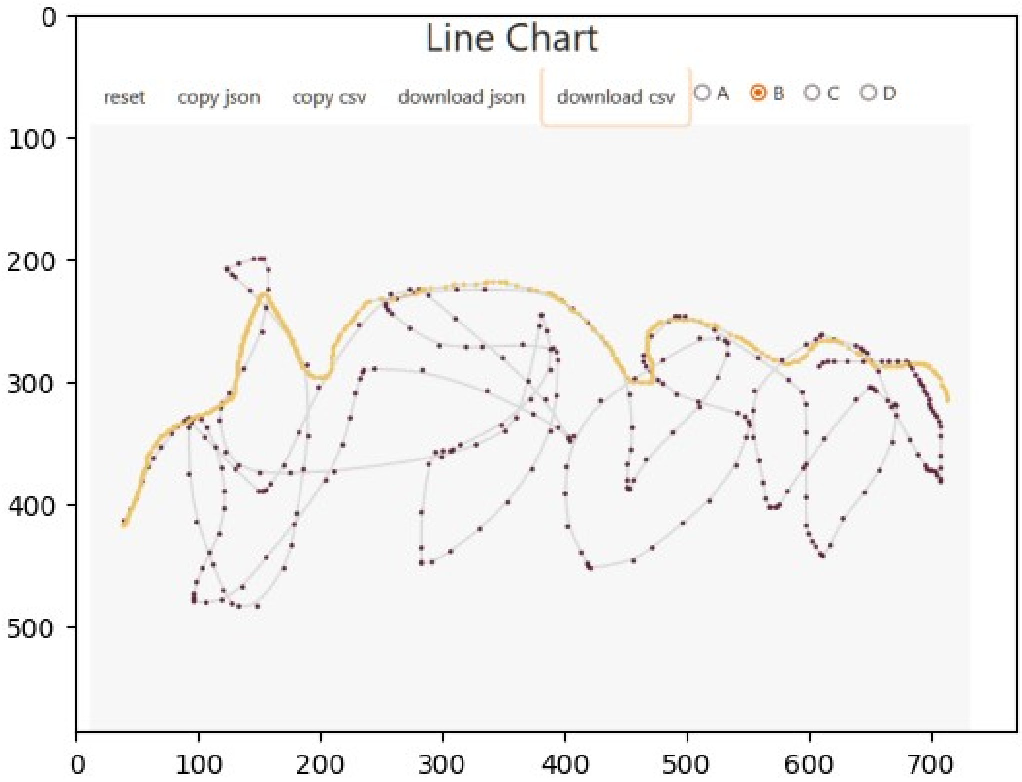
After adding noise to my initial data points I realized that my MLP can keep its robustness even after 25% noise adding, but there is also a bit less accuraccy with my simpler functions.

I have use a three layer MLP with 500 data points and 20 neurons in each layer. also the X_range is between -5 and 5.

#Question 3

I have used a webpage which url is drawdata.xyz to generate data points from my mouse movement. then I have used the approximate top convex of my Dummy drawing.

```
import cv2
# this only to show the picture of my dummy drawing.
image = cv2.imread('/content/draw.png')
plt.imshow(image)

<matplotlib.image.AxesImage at 0x7979778dcd90>
```

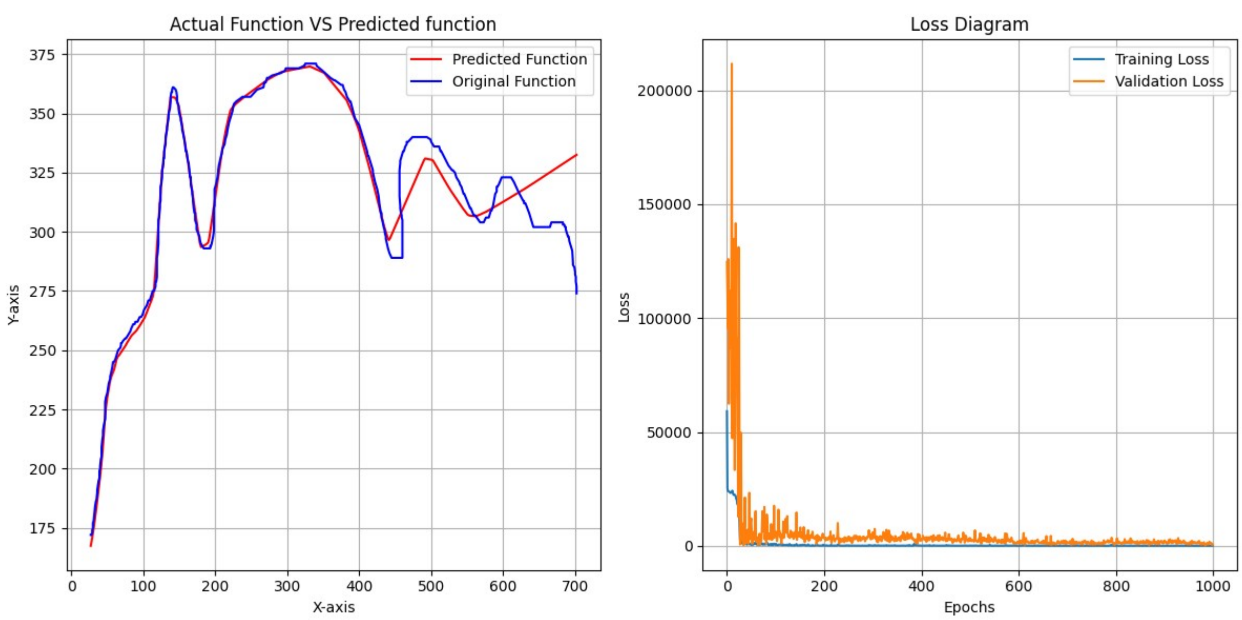


- In the above dummy drawing I have chosen the orange data points and used them as a function to be predicted.

```
dummy_data = pd.read_csv('/content/Draw.csv')

model = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(100, activation='relu'),
    tf.keras.layers.Dense(1)
    ])

model.compile(optimizer='adam', loss='mean_squared_error')

history = model.fit(dummy_data['x'], dummy_data['y'],batch_size=16,
epochs=1000, verbose=0,validation_split=0.2)

X_test = np.linspace(dummy_data['x'].min(), dummy_data['x'].max(),
415).reshape(-1, 1)
y_pred = model.predict(X_test)
```

```
mse = np.mean(np.square(y_pred - dummy_data['y'].values))

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(X_test, y_pred, color='red', label=f'Predicted Function')
plt.plot(dummy_data['x'], dummy_data['y'], color='blue',
label='Original Function')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Actual Function VS Predicted function')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
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('Loss Diagram')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

print(f"MSE: {mse:.4f}")
```

```
13/13 [==============================] - 0s 2ms/step
```



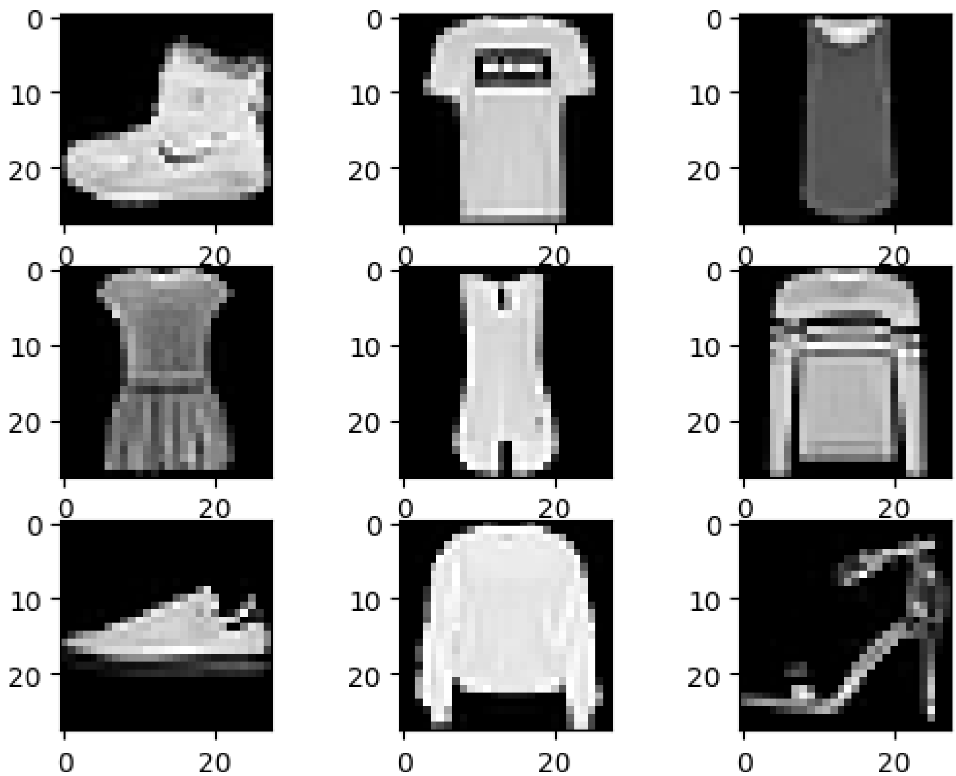
```
MSE: 3818.0826
```

After too many tries I found out that an MLP with 9 layer of neurons can approximate my dummy function with an MSE of 3818. As far as I know we can not caclulate the accuracy for regression, but orally I can see that my function is more than 70% accurate.

#**Part 2**

#Question 1

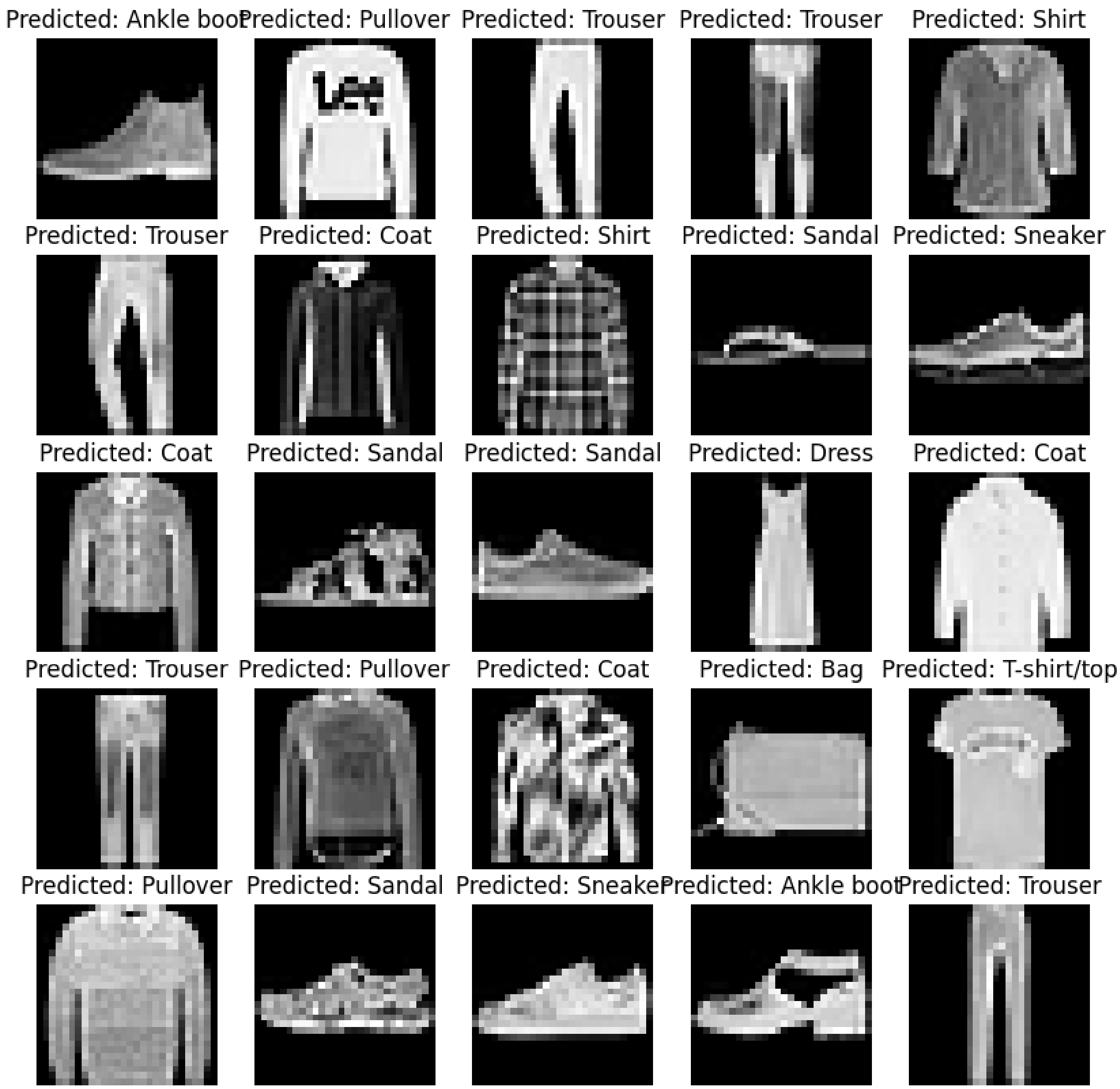
```
from tensorflow.keras.datasets import fashion_mnist
import tensorflow as tf

(trainX, trainy), (testX, testy) = fashion_mnist.load_data()

print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))

for i in range(9):

 plt.subplot(330 + 1 + i)

 plt.imshow(trainX[i], cmap=plt.get_cmap('gray'))

plt.show()

Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
```

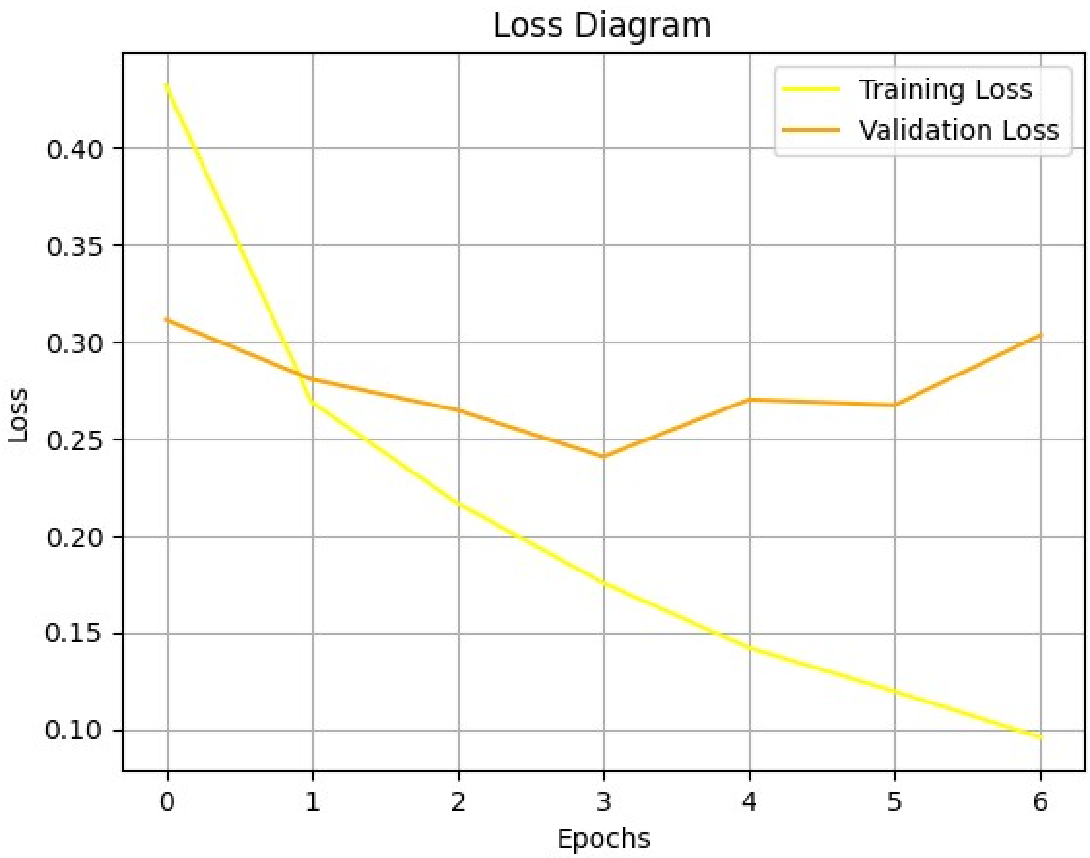


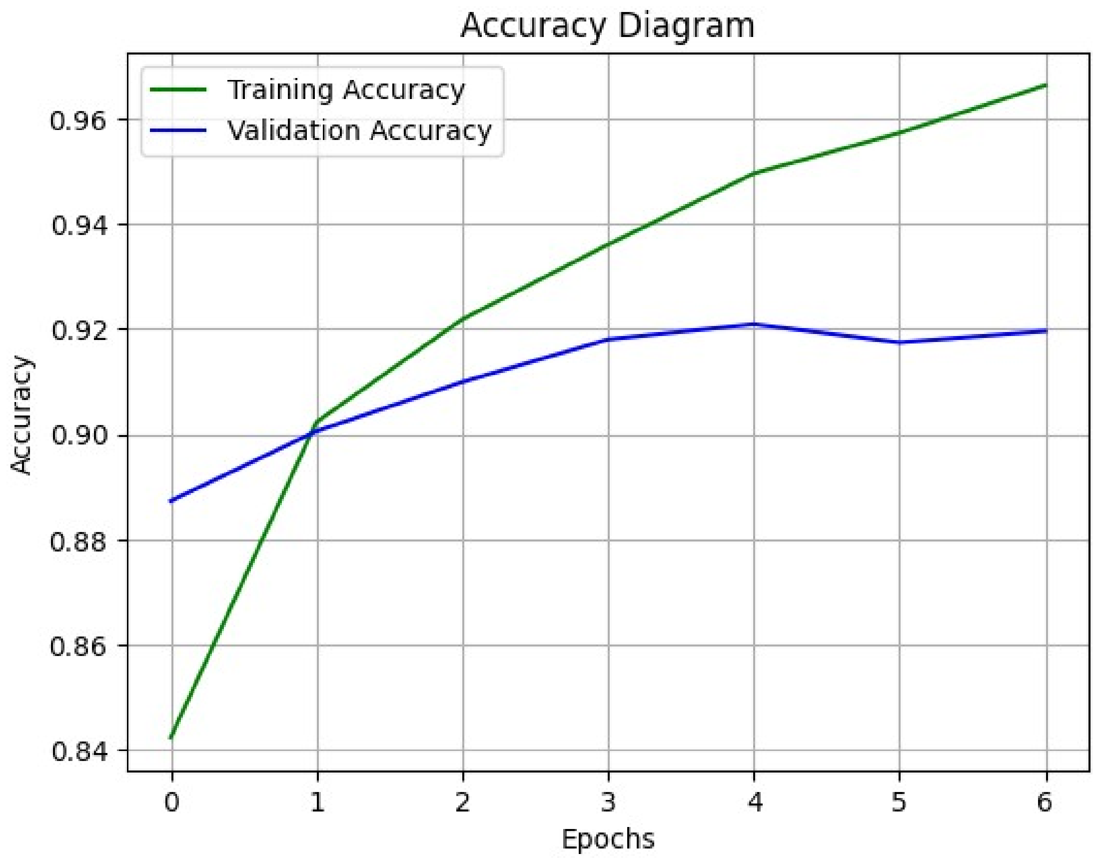

```
trainX = trainX / 255.0
testX = testX / 255.0

trainX = trainX.reshape(trainX.shape[0], 28, 28, 1)
testX = testX.reshape(testX.shape[0], 28, 28, 1)

trainy = tf.keras.utils.to_categorical(trainy, num_classes=10)
testy = tf.keras.utils.to_categorical(testy, num_classes=10)

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
     tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
```

```
metrics=['accuracy'])

history = model.fit(trainX, trainy, batch_size=32, epochs=7,
validation_split=0.2, verbose=1)



plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(testX[i].reshape(28, 28), cmap='gray')
    plt.title(f"Predicted: {predicted_class_names[i]}")
    plt.axis('off')
plt.show()

plt.plot(history.history['loss'], label='Training Loss',
color='yellow')
plt.plot(history.history['val_loss'], label='Validation Loss',
color='orange')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Diagram')
plt.legend()
plt.grid(True)
plt.show()

plt.plot(history.history['accuracy'], label='Training Accuracy',
color='green')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy',
color='blue')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy Diagram')
plt.legend()
plt.grid(True)
plt.show()

test_loss, test_acc = model.evaluate(testX, testy)
print(f"Test Accuracy: {test_acc}")

Epoch 1/7
1500/1500 [==============================] - 12s 6ms/step - loss:
0.4323 - accuracy: 0.8424 - val_loss: 0.3113 - val_accuracy: 0.8873
Epoch 2/7
1500/1500 [==============================] - 8s 5ms/step - loss:
0.2688 - accuracy: 0.9023 - val_loss: 0.2805 - val_accuracy: 0.9006
Epoch 3/7
1500/1500 [==============================] - 9s 6ms/step - loss:
0.2166 - accuracy: 0.9218 - val_loss: 0.2646 - val_accuracy: 0.9099
Epoch 4/7
```

```
1500/1500 [==============================] - 10s 6ms/step - loss:
0.1755 - accuracy: 0.9360 - val_loss: 0.2406 - val_accuracy: 0.9180
Epoch 5/7
1500/1500 [==============================] - 8s 5ms/step - loss:
0.1419 - accuracy: 0.9495 - val_loss: 0.2700 - val_accuracy: 0.9209
Epoch 6/7
1500/1500 [==============================] - 8s 6ms/step - loss:
0.1194 - accuracy: 0.9572 - val_loss: 0.2672 - val_accuracy: 0.9174
Epoch 7/7
1500/1500 [==============================] - 9s 6ms/step - loss:
0.0958 - accuracy: 0.9663 - val_loss: 0.3034 - val_accuracy: 0.9196
```




Loss Diagram
Training Loss
Validation Loss
0.40
0.35
0.30
0.25
0.20
0.15
0.10
Loss
0
1
2
3
4
5
6
Epochs




```
313/313 [==============================] - 1s 3ms/step - loss: 0.3165
- accuracy: 0.9173
Test Accuracy: 0.9172999858856201
```

For fashion_mnist dataset, I loaded the data first, then tried to print the shape of my data and also plot few images in it.

After that, I normalizd my data by dividing it by 255, and reshaped my data by adding the height dimension.

finally I used one-hot-encoding to convert labels to binary class matrices, and trained my model following by evaluating it and plotting first 25 predictinos. And I also plotted the loss and accuracy diagrams per epoch.

-> My model predicted the classes with an accuracy of 92%.

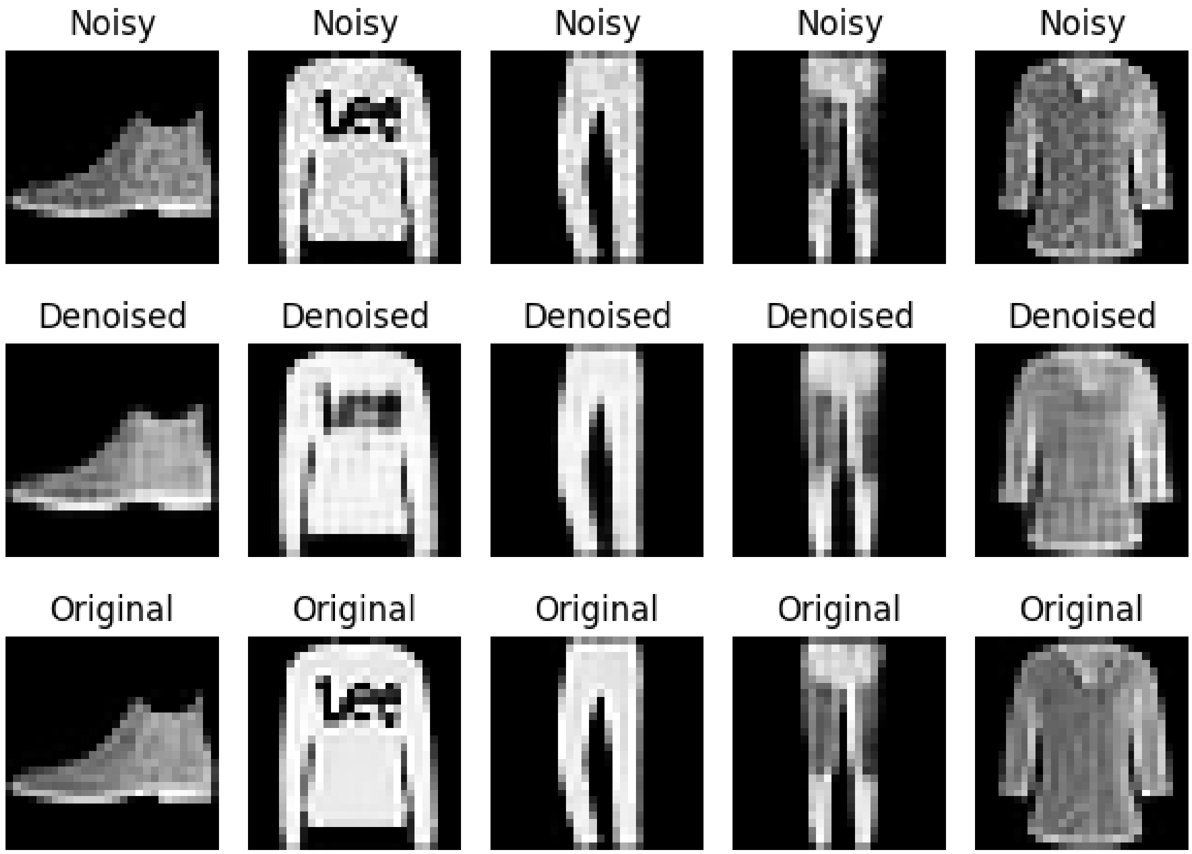
#Question 2

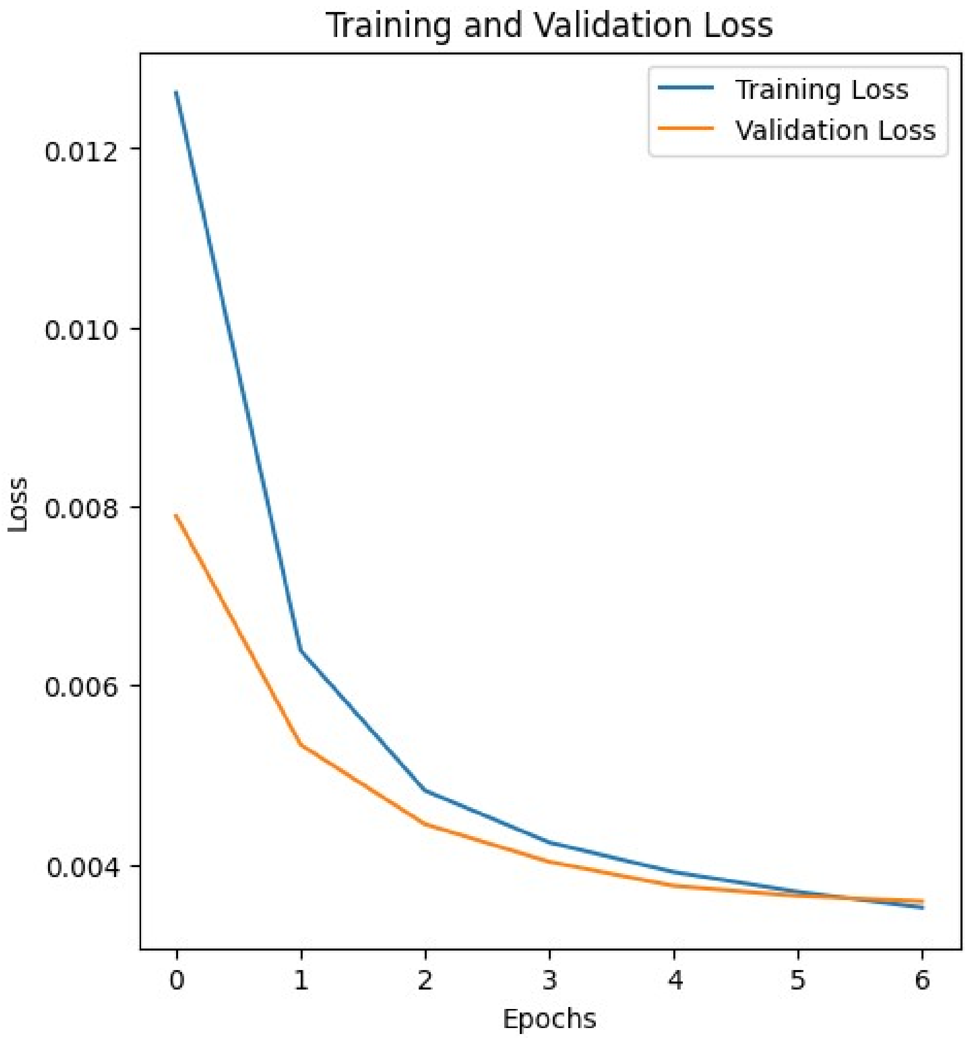
```
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
UpSampling2D
from tensorflow.keras.models import Model
(trainX, trainy), (testX, testy) = fashion_mnist.load_data()

trainX = trainX / 255.0
```

```
testX = testX / 255.0

trainX = trainX.reshape(trainX.shape[0], 28, 28, 1)
testX = testX.reshape(testX.shape[0], 28, 28, 1)

noise_factor = 0.1
noisy_trainX = trainX + noise_factor * np.random.randint(-1,1,
size=trainX.shape)
noisy_testX = testX + noise_factor * np.random.randint(-1,1,
size=testX.shape)

noisy_trainX = np.clip(noisy_trainX, 0., 255.)
noisy_testX = np.clip(noisy_testX, 0., 255.)

input_img = Input(shape=(28, 28, 1))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)

autoencoder.compile(optimizer='adam', loss='mean_squared_error')

noisy_trainX = noisy_trainX.reshape(-1, 28, 28, 1)
noisy_testX = noisy_testX.reshape(-1, 28, 28, 1)

history = autoencoder.fit(noisy_trainX, trainX, epochs=7,
batch_size=32, shuffle=True, validation_data=(noisy_testX,
testX),verbose=1)


decoded_images = autoencoder.predict(noisy_testX)

for i in range(5):
    plt.subplot(3, 5, i + 1)
    plt.imshow(noisy_testX[i].reshape(28, 28), cmap='gray')
    plt.title('Noisy')
    plt.axis('off')

    plt.subplot(3, 5, i + 6)
    plt.imshow(decoded_images[i].reshape(28, 28), cmap='gray')
    plt.title('Denoised')
    plt.axis('off')
```

```python
    plt.subplot(3, 5, i + 11)
    plt.imshow(testX[i], cmap='gray')
    plt.title('Original')
    plt.axis('off')

plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
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()

test_loss = autoencoder.evaluate(decoded_images, testX)
print(f"Test Loss: {test_loss}")
```

```
Epoch 1/7
1875/1875 [==============================] - 14s 5ms/step - loss:
0.0126 - val_loss: 0.0079
Epoch 2/7
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0064 - val_loss: 0.0053
Epoch 3/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0048 - val_loss: 0.0044
Epoch 4/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0042 - val_loss: 0.0040
Epoch 5/7
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0039 - val_loss: 0.0038
Epoch 6/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0037 - val_loss: 0.0036
Epoch 7/7
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0035 - val_loss: 0.0036
313/313 [==============================] - 1s 2ms/step
```


Noisy
Noisy
Noisy
Noisy
Noisy
Denoised
Denoised
Denoised
Denoised
Denoised
Original
Original
Original
Original
Original

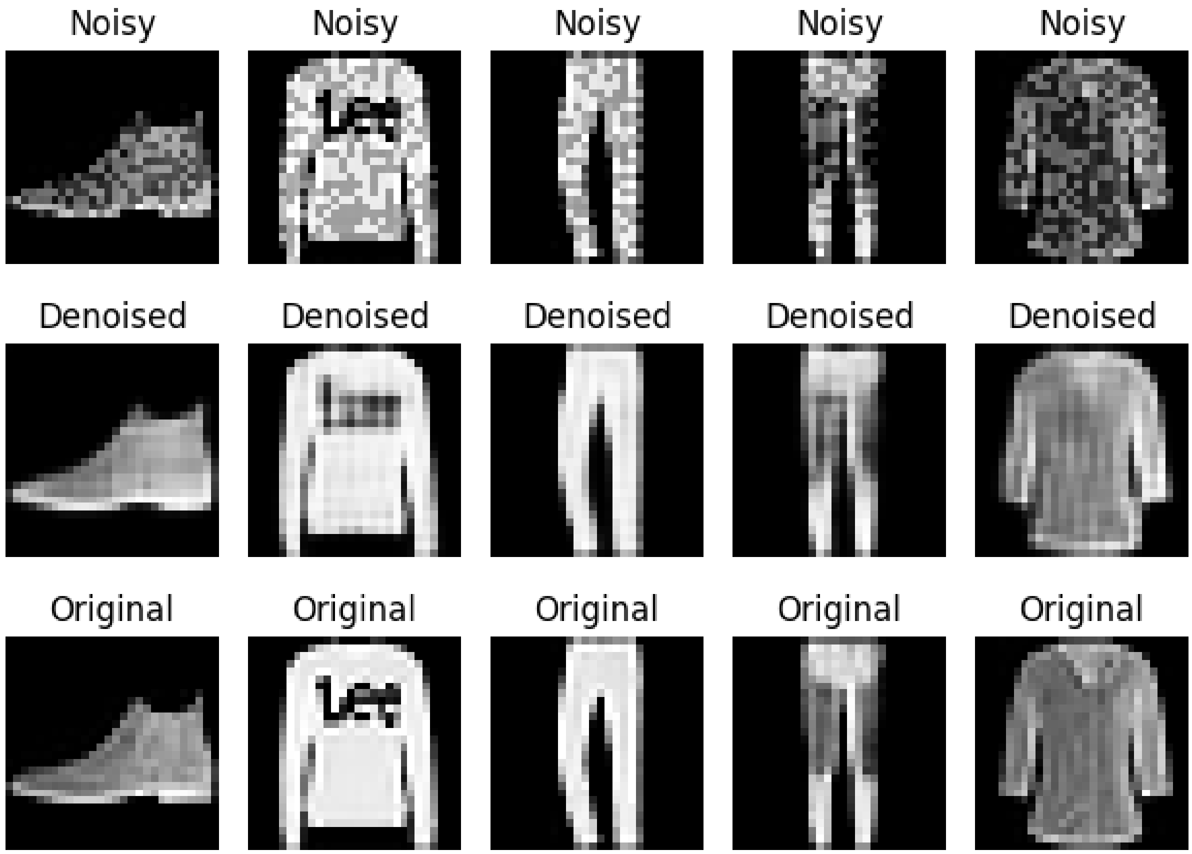


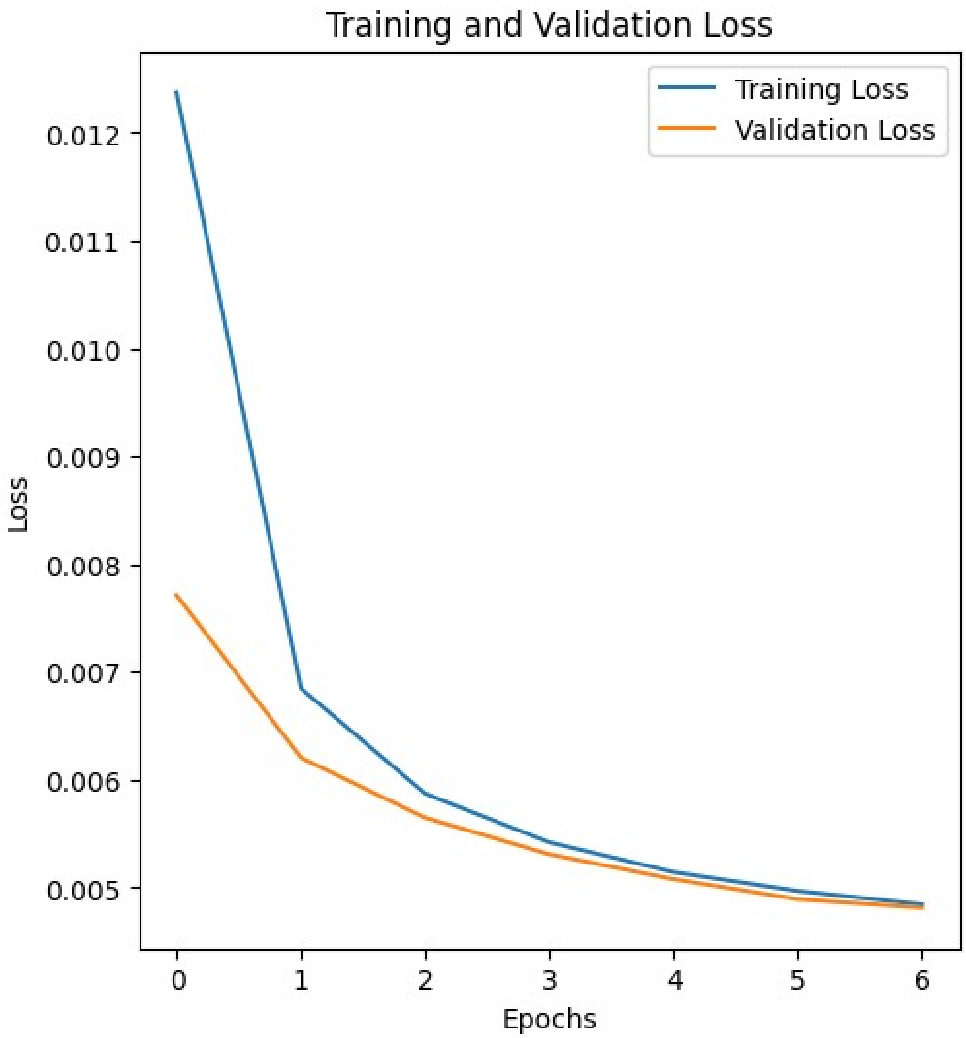

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0084
Test Loss: 0.008392498828470707

(trainX, trainy), (testX, testy) = fashion_mnist.load_data()

trainX = trainX / 255.0
testX = testX / 255.0

trainX = trainX.reshape(trainX.shape[0], 28, 28, 1)
testX = testX.reshape(testX.shape[0], 28, 28, 1)

noise_factor = 0.3
noisy_trainX = trainX + noise_factor * np.random.randint(-1,1,
size=trainX.shape)
noisy_testX = testX + noise_factor * np.random.randint(-1,1,
size=testX.shape)

noisy_trainX = np.clip(noisy_trainX, 0., 255.)
```

```
noisy_testX = np.clip(noisy_testX, 0., 255.)

input_img = Input(shape=(28, 28, 1))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)

autoencoder.compile(optimizer='adam', loss='mean_squared_error')

noisy_trainX = noisy_trainX.reshape(-1, 28, 28, 1)
noisy_testX = noisy_testX.reshape(-1, 28, 28, 1)

history = autoencoder.fit(noisy_trainX, trainX, epochs=7,
batch_size=32, shuffle=True, validation_data=(noisy_testX,
testX),verbose=1)


decoded_images = autoencoder.predict(noisy_testX)

for i in range(5):
    plt.subplot(3, 5, i + 1)
    plt.imshow(noisy_testX[i].reshape(28, 28), cmap='gray')
    plt.title('Noisy')
    plt.axis('off')

    plt.subplot(3, 5, i + 6)
    plt.imshow(decoded_images[i].reshape(28, 28), cmap='gray')
    plt.title('Denoised')
    plt.axis('off')

    plt.subplot(3, 5, i + 11)
    plt.imshow(testX[i], cmap='gray')
    plt.title('Original')
    plt.axis('off')

plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
```

```python
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()

test_loss = autoencoder.evaluate(decoded_images, testX)
print(f"Test Loss: {test_loss}")
```

```
Epoch 1/7
1875/1875 [==============================] - 10s 5ms/step - loss:
0.0124 - val_loss: 0.0077
Epoch 2/7
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0068 - val_loss: 0.0062
Epoch 3/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0059 - val_loss: 0.0056
Epoch 4/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0054 - val_loss: 0.0053
Epoch 5/7
1875/1875 [==============================] - 11s 6ms/step - loss:
0.0051 - val_loss: 0.0051
Epoch 6/7
1875/1875 [==============================] - 10s 5ms/step - loss:
0.0050 - val_loss: 0.0049
Epoch 7/7
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0048 - val_loss: 0.0048
313/313 [==============================] - 1s 2ms/step
```


Noisy
Noisy
Noisy
Noisy
Noisy
Denoised
Denoised
Denoised
Denoised
Denoised
Original
Original
Original
Original
Original

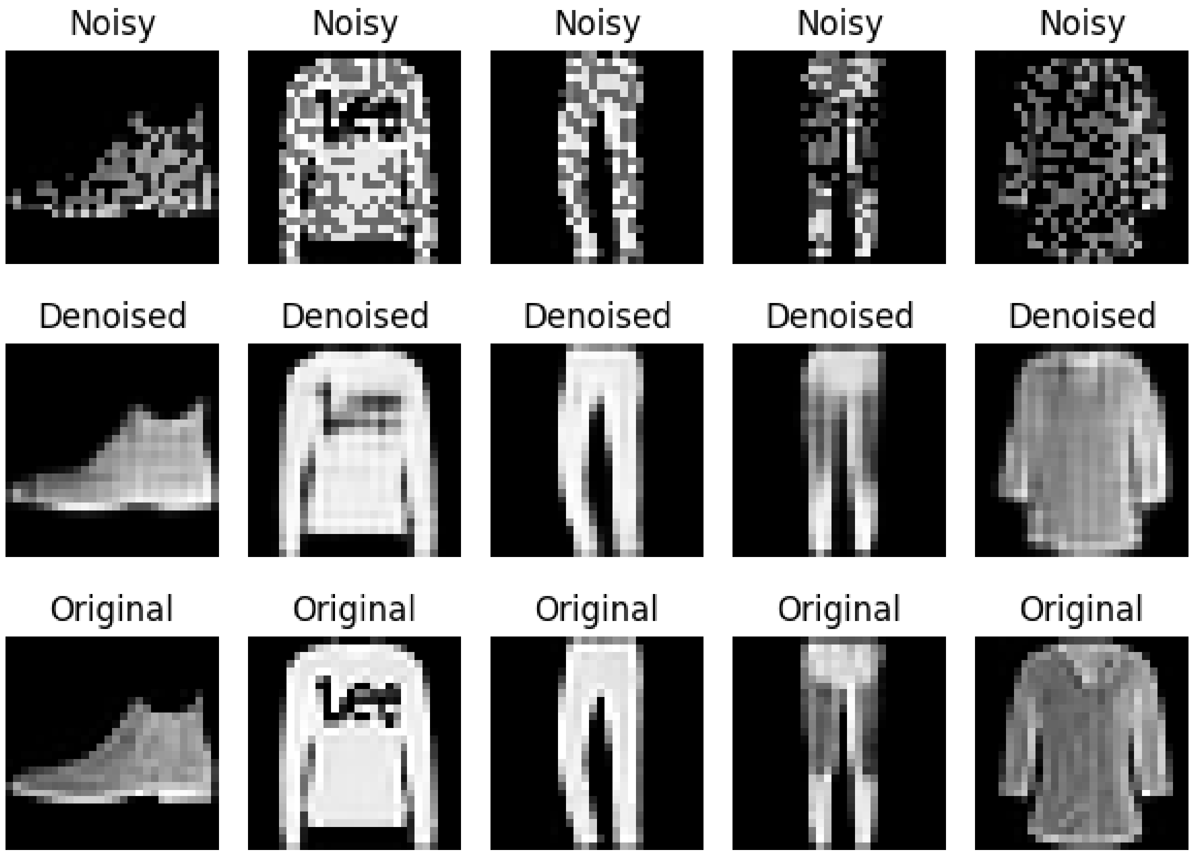



```
313/313 [==============================] - 1s 2ms/step - loss: 0.0131
Test Loss: 0.013126650825142860

(trainX, trainy), (testX, testy) = fashion_mnist.load_data()

trainX = trainX / 255.0
testX = testX / 255.0

trainX = trainX.reshape(trainX.shape[0], 28, 28, 1)
testX = testX.reshape(testX.shape[0], 28, 28, 1)

noise_factor = 0.5
noisy_trainX = trainX + noise_factor * np.random.randint(-1,1,
size=trainX.shape)
noisy_testX = testX + noise_factor * np.random.randint(-1,1,
size=testX.shape)

noisy_trainX = np.clip(noisy_trainX, 0., 255.)
```

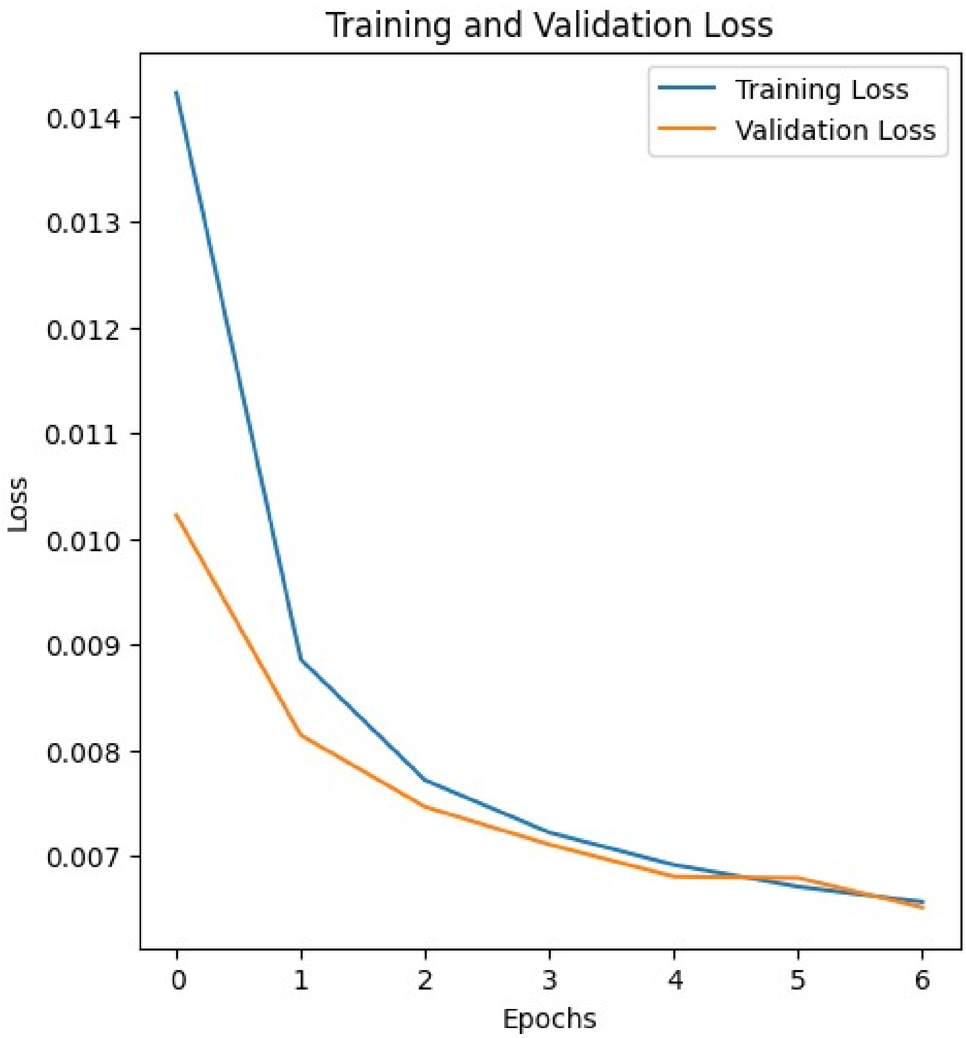
```
noisy_testX = np.clip(noisy_testX, 0., 255.)

input_img = Input(shape=(28, 28, 1))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)

autoencoder.compile(optimizer='adam', loss='mean_squared_error')

noisy_trainX = noisy_trainX.reshape(-1, 28, 28, 1)
noisy_testX = noisy_testX.reshape(-1, 28, 28, 1)

history = autoencoder.fit(noisy_trainX, trainX, epochs=7,
batch_size=32, shuffle=True, validation_data=(noisy_testX,
testX),verbose=1)


decoded_images = autoencoder.predict(noisy_testX)

for i in range(5):
    plt.subplot(3, 5, i + 1)
    plt.imshow(noisy_testX[i].reshape(28, 28), cmap='gray')
    plt.title('Noisy')
    plt.axis('off')

    plt.subplot(3, 5, i + 6)
    plt.imshow(decoded_images[i].reshape(28, 28), cmap='gray')
    plt.title('Denoised')
    plt.axis('off')

    plt.subplot(3, 5, i + 11)
    plt.imshow(testX[i], cmap='gray')
    plt.title('Original')
    plt.axis('off')

plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
```

```python
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()

test_loss = autoencoder.evaluate(decoded_images, testX)
print(f"Test Loss: {test_loss}")
```

```
Epoch 1/7
1875/1875 [==============================] - 15s 7ms/step - loss:
0.0142 - val_loss: 0.0102
Epoch 2/7
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0089 - val_loss: 0.0081
Epoch 3/7
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0077 - val_loss: 0.0075
Epoch 4/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0072 - val_loss: 0.0071
Epoch 5/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0069 - val_loss: 0.0068
Epoch 6/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0067 - val_loss: 0.0068
Epoch 7/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0066 - val_loss: 0.0065
313/313 [==============================] - 1s 2ms/step
```



Noisy
Noisy
Noisy
Noisy
Noisy
Denoised
Denoised
Denoised
Denoised
Denoised
Original
Original
Original
Original
Original

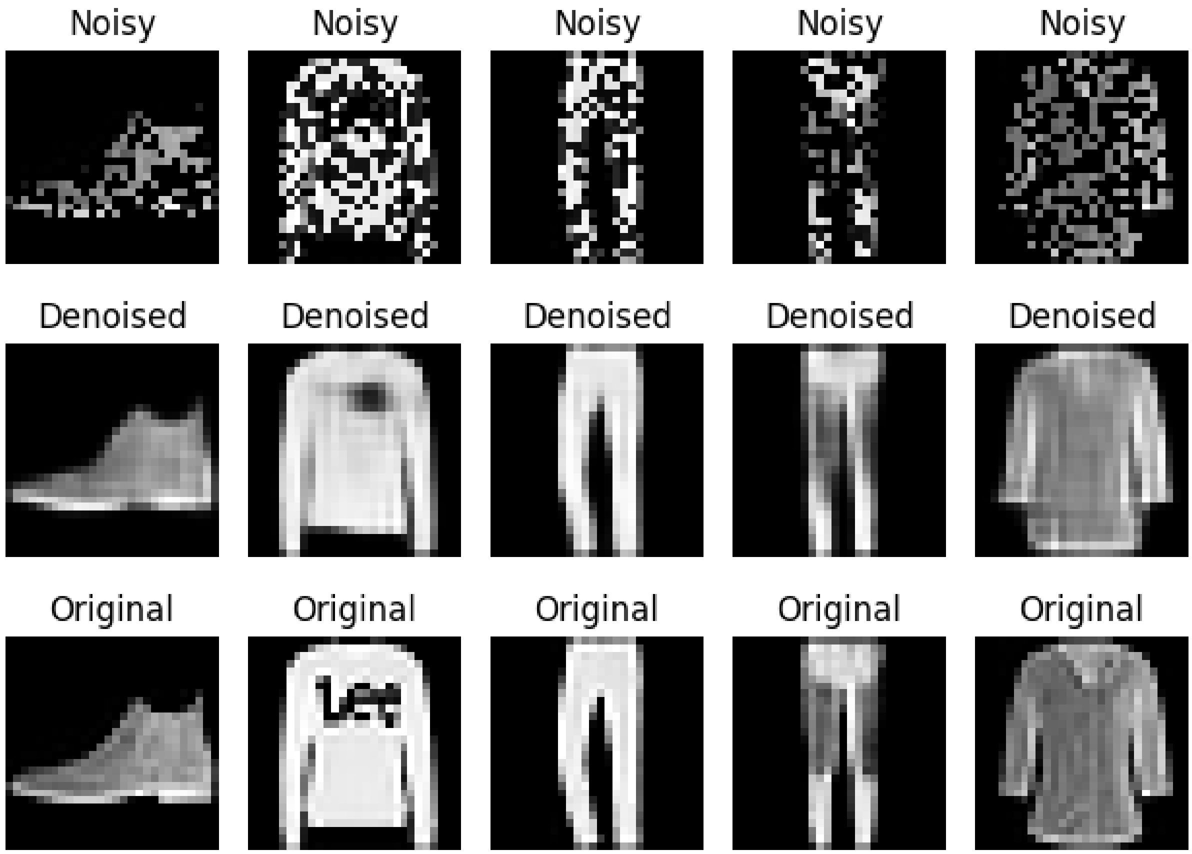


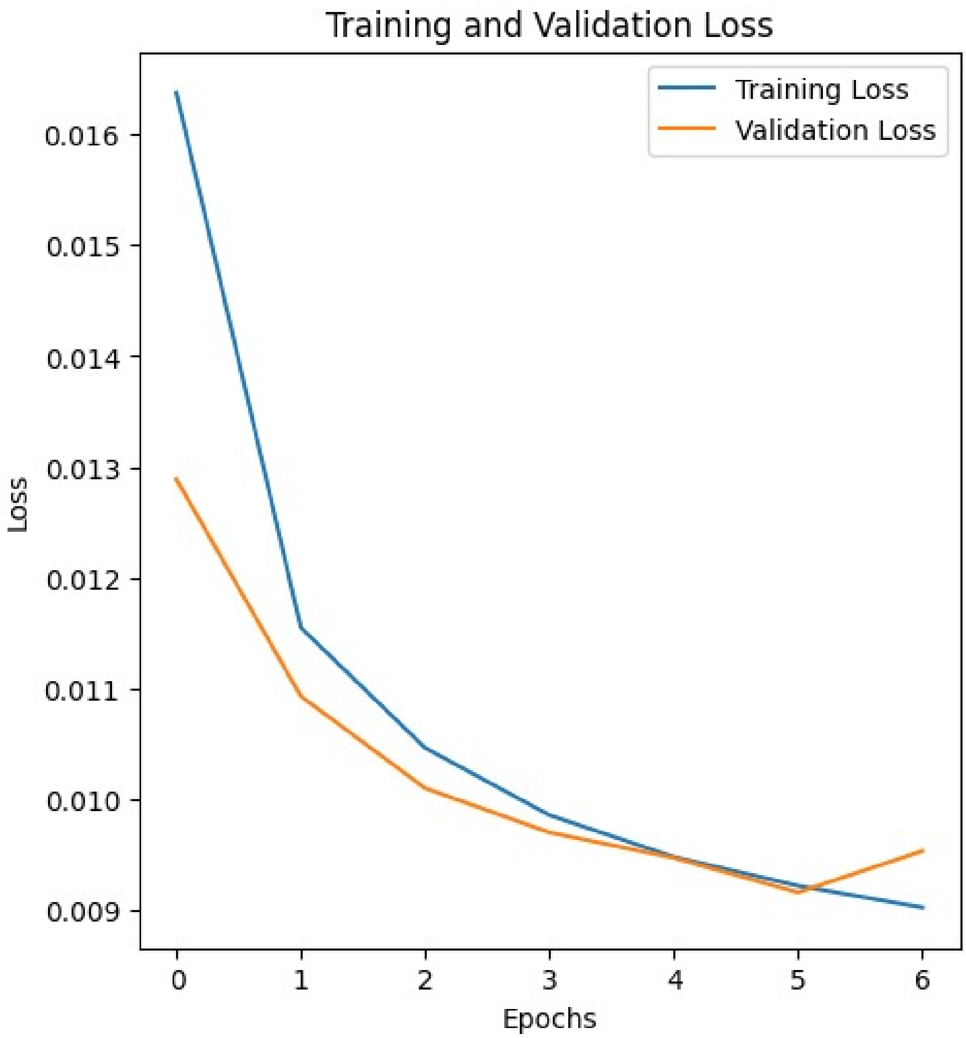

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0208
Test Loss: 0.020835930481553078

(trainX, trainy), (testX, testy) = fashion_mnist.load_data()

trainX = trainX / 255.0
testX = testX / 255.0

trainX = trainX.reshape(trainX.shape[0], 28, 28, 1)
testX = testX.reshape(testX.shape[0], 28, 28, 1)

noise_factor = 0.85
noisy_trainX = trainX + noise_factor * np.random.randint(-1,1,
size=trainX.shape)
noisy_testX = testX + noise_factor * np.random.randint(-1,1,
size=testX.shape)

noisy_trainX = np.clip(noisy_trainX, 0., 255.)
```

```python
noisy_testX = np.clip(noisy_testX, 0., 255.)

input_img = Input(shape=(28, 28, 1))

x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)

autoencoder.compile(optimizer='adam', loss='mean_squared_error')

noisy_trainX = noisy_trainX.reshape(-1, 28, 28, 1)
noisy_testX = noisy_testX.reshape(-1, 28, 28, 1)

history = autoencoder.fit(noisy_trainX, trainX, epochs=7,
batch_size=32, shuffle=True, validation_data=(noisy_testX,
testX),verbose=1)


decoded_images = autoencoder.predict(noisy_testX)

for i in range(5):
    plt.subplot(3, 5, i + 1)
    plt.imshow(noisy_testX[i].reshape(28, 28), cmap='gray')
    plt.title('Noisy')
    plt.axis('off')

    plt.subplot(3, 5, i + 6)
    plt.imshow(decoded_images[i].reshape(28, 28), cmap='gray')
    plt.title('Denoised')
    plt.axis('off')

    plt.subplot(3, 5, i + 11)
    plt.imshow(testX[i], cmap='gray')
    plt.title('Original')
    plt.axis('off')

plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
```

```python
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()

test_loss = autoencoder.evaluate(decoded_images, testX)
print(f"Test Loss: {test_loss}")
```

```
Epoch 1/7
1875/1875 [==============================] - 10s 4ms/step - loss:
0.0164 - val_loss: 0.0129
Epoch 2/7
1875/1875 [==============================] - 10s 6ms/step - loss:
0.0115 - val_loss: 0.0109
Epoch 3/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0105 - val_loss: 0.0101
Epoch 4/7
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0099 - val_loss: 0.0097
Epoch 5/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0095 - val_loss: 0.0095
Epoch 6/7
1875/1875 [==============================] - 9s 5ms/step - loss:
0.0092 - val_loss: 0.0092
Epoch 7/7
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0090 - val_loss: 0.0095
313/313 [==============================] - 1s 3ms/step
```


Noisy
Noisy
Noisy
Noisy
Noisy
Denoised
Denoised
Denoised
Denoised
Denoised
Original
Original
Original
Original
Original




```
313/313 [==============================] - 1s 3ms/step - loss: 0.0145
Test Loss: 0.014474798925220966
```

Neural networks can learn to remove noise from data. This will be done by train the network using noisy inputs and comparing them to the original inputs.

in Learning process, I use the autoencoder structure. It has an encoder and an decoder, which the first one gets the noisy data and simplify that, and the second one tries to decode the simplified data into the original noiseless data.

It is normal for results to be different on train set and test sets, so in this example we have a slight difference of about 0.01 in data loss. Of course results on train set are better.

I started from a noise factore of 0.1 that means 10% of data pixels have noise, and finnally I endded up into 0.8. But the data loss did not changed much. It changed from almost 0.008 to 0.014.