به نام خدا

پروژه دوم درس هوش مصنوعی – شبکه‌های عصبی بهاره کاوسی نژاد – 99431217

تکنیک Cross-Validation:

تکنیک Cross-Validation در یادگیری ماشین به منظور تخمین مهارت یک مدل یادگیری بر روی داده‌های unseen استفاده می‌شود؛ یعنی استفاده از یک نمونه محدود به منظور تخمین نحوه عملکرد کلی مدل، زمانی که برای پیش بینی داده‌هایی که در طول آموزش مورد استفاده قرار نگرفته‌اند.

بخش اول:

```
import numpy as np
import pandas as pd
import random
```

ابتدا کتابخانه های لازم را import می‌کنیم.

```
train = pd.read_csv('Airplane.csv')

train = train.drop(train.columns[0], axis=1)
train = train.drop("id", axis='columns')

print (train.info())
```

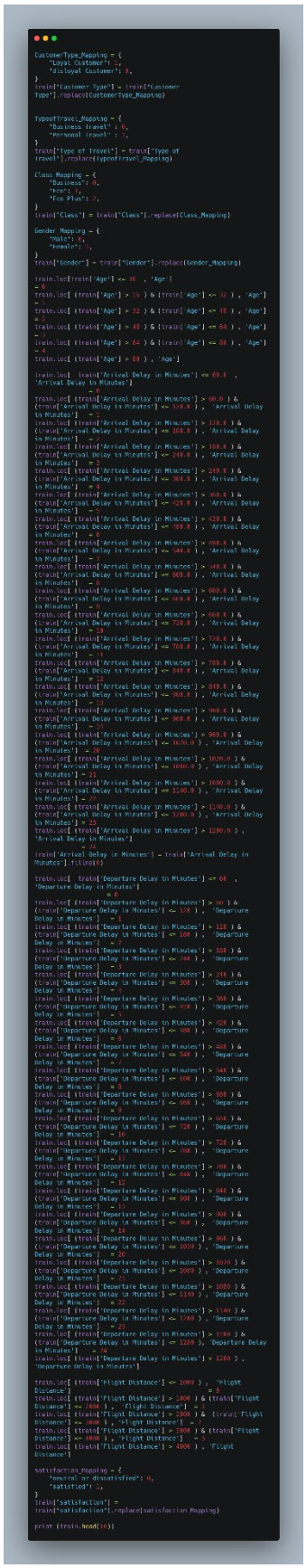
با دستور pd.read_csv فایل csv را می‌خوانیم و دو ستون اول آن را حذف می‌کنیم زیرا شماره سطر و id هستند که attribute نیستند و با دستور info اطلاعات attribute ها را چاپ می‌کنیم.

در مرحله بعد باید attribute ها را map کنیم و به آنها مقادیر عددی نسبت دهیم. به عنوان مثال mapping ویژگی Customer Type به صورت زیر خواهد بود:

```
CustomerType_Mapping = {
    "Loyal Customer": 1,
    "disloyal Customer": 0,
}
train["Customer Type"] = train["Customer
Type"].replace(CustomerType_Mapping)
```

همچنین attributeهایی مانند Age و Arrival Delay in Minutes را به صورت بازه map می‌کنیم:

```
train.loc[train['Age'] <= 16  , 'Age']
= 0
train.loc[ (train['Age'] > 16 ) & (train['Age'] <= 32 ) , 'Age']
= 1
train.loc[ (train['Age'] > 32 ) & (train['Age'] <= 48 ) , 'Age']
= 2
train.loc[ (train['Age'] > 48 ) & (train['Age'] <= 64 ) , 'Age']
= 3
train.loc[ (train['Age'] > 64 ) & (train['Age'] <= 80 ) , 'Age']
= 4
train.loc[ (train['Age'] > 80 ) , 'Age']
```



```python
def random_rows_after_2000(data_frame, num_rows):
    total_rows = data_frame.shape[0]
    eligible_indices = list(range(2001, total_rows))

    selected_indices = random.sample(eligible_indices, num_rows)
    selected_rows = data_frame.iloc[selected_indices]

    return selected_rows
```

از این تابع برای انتخاب نمونه‌های رندوم پس از نمونه 2000ام (زیرا 2000 نمونه اول برای تست هستند) برای train کردن استفاده می‌شود.

```python
class NeuralNet(object):
    def __init__(self, input_size):
        np.random.seed(1)
        self.synaptic_weights = 2 * np.random.rand(input_size,
1) - 1

    def __sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def __sigmoid_derivative(self, x):
        return x * (1 - x)

    def train(self, inputs, outputs, training_iterations):
        for iteration in range(0, training_iterations):
            output = self.learn(inputs)
            error = outputs - output
            factor = np.dot(inputs.T, error *
self.__sigmoid_derivative(output))
            factor = factor.mean(axis=1, keepdims=True)
            self.synaptic_weights += factor

    def learn(self, inputs):
        return self.__sigmoid(np.dot(inputs,
self.synaptic_weights))
```

در کلاس NeuralNet شبکه عصبی خود را می‌سازیم.

- متد __init__: این متد constructor کلاس است و مقادیر تصادفی برای synaptic connectionهای بین نورون‌ها را تولید می‌کند.
- متد __sigmoid: این متد کمکی برای محاسبه sigmoid activation function به ازای ورودی x است. تابع sigmoid هر مقدار حقیقی را به بازه 0 تا 1 map می‌کند.
- متد __sigmoid_derivative: این متد کمکی برای محاسبه مشتق تابع sigmoid استفاده می‌شود. این مقدار، sigmoid curve gradient را تعیین می‌کند که در backpropagation برای تنظیم وزن‌ها مورد استفاده قرار می‌گیرد.
- متد train: این متد به تعداد training_iteration بار اجرا می‌شود و وزن‌ها را بر اساس error محاسبه شده تنظیم می‌کند. Error به صورت اختلاف بین خروجی مورد انتظار و خروجی تابع learn تعیین می‌شود.
- داخل حلقه train، وزن‌ها بر اساس error محاسبه شده و مشتق تابع sigmoid تنظیم می‌شوند. Adjusment factor بر اساس ضرب داخلی transpose ماتریس ورودی و element-wise product ارور و مشتق خروجی محاسبه می‌شود. سپس این factor به وزن‌های synaptic اضافه می‌شود.
- متد learn، به عنوان خروجی شبکه عصبی را برمی‌گرداند. این متد ضرب داخلی ورودی ها و وزن‌های synaptic را محاسبه کرده و نتیجه را به activation function می‌دهد تا خروجی به دست آید.

```
train_data = random_rows_after_2000(train, 10000)

target = train_data.iloc[:, -1]
train_data = train_data.iloc[:, :-1]

attributes = train.columns
attributes = list(attributes)
attributes = attributes[:-1]
```

داده‌های train و attributeها را مانند پروژه قبل مشخص می‌کنیم.

```
neural_network = NeuralNet(input_size=22)

inputs = np.array(train_data)
outputs = np.array(target).T

neural_network.train(inputs, outputs, 50)

NumberOfTestData = 3
for i in range(1, NumberOfTestData + 1):
    test_data_results = train.iloc[i, -1]
    test_data = train.iloc[i, :-1].values.reshape(1,-1)
    print(neural_network.learn(test_data))
```

شبکه عصبی را می‌سازیم و به عنوان input_size تعداد attributeها را می‌دهیم. سپس آن را train می‌کنیم.

دقت درخت تصمیم و شبکه عصبی:

دقت درخت تصمیم و شبکه عصبی بسیار نزدیک به هم است اما دقت شبکه عصبی اندکی بیشتر است.

بخش دوم و سوم:

```python
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
```

ابتدا کتابخانه‌های لازم را import می‌کنیم.

```python
np.random.seed(1)
X = np.random.uniform(0.1, 2, 1000)
y = 1 / X
```

مقادیر X را به صورت تصادفی و مقادیر y را بر اساس تابع 1/X تعیین می‌کنیم.

```python
noise_level = 0.1
X_noisy = X + np.random.normal(0, noise_level, size=X.shape)
y_noisy = y + np.random.normal(0, noise_level, size=y.shape)
```

برای بخش سوم به مقادیر x و y نویز اضافه می‌کنیم.

```
model = Sequential()
model.add(Dense(16, input_dim=1, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(8, activation='relu'))
model.add(Dense(1))
```

- ابتدا یک instance از کلاس Sequential می‌سازیم که یک stack خطی از لایه‌هاست و به ما اجازه می‌دهد که شبکه‌های عصبی را با اضافه کردن لایه‌ها به ترتیب بسازیم.
- یک لایه fully connected به مدل اضافه می‌کنیم. این Dense layer یک لایه است که در آن هر نورون به همه نورون‌های لایه قبل خود متصل شده است. در اینجا دارای 16 نورون است. پارامتر input_dim تعداد ابعاد داده ورودی را مشخص می‌کند. پارامتر activation=’relu’، تابع فعال سازی rectified linear unit (ReLU) را برای این لایه مشخص می‌کند. ReLU در لایه‌های پنهان برای non-linearity استفاده می‌شود.
- در خط بعدی یک لایه Dropout به مدل اضافه می‌کنیم. Dropout یک تکنیک regularization است که به صورت تصادفی بخشی از ورودی‌ها را در حین training، صفر قرار می‌دهد تا از overfitting و وابستگی بین نورون‌ها جلوگیری کند.
- سپس یک لایه fully connected دیگر با 8 نورون اضافه می‌کنیم.
- در انتها یک لایه خروجی به مدل اضافه می‌کنیم. این لایه نیز fully connected است و یک نورون دارد و خروجی پیش بینی شده را نشان می‌دهد. Activation Function به صورت پیش فرض خطی تعیین می‌شود.

```
model.compile(loss='mean_squared_error', optimizer='adam')
history = model.fit(X, y, epochs=200, batch_size=32, verbose=0)
```

مدل را کامپایل و train می‌کنیم.

```
X_test = np.linspace(0.1, 2, 100)
y_test = 1 / X_test
```

داده‌های test را generate می‌کنیم.

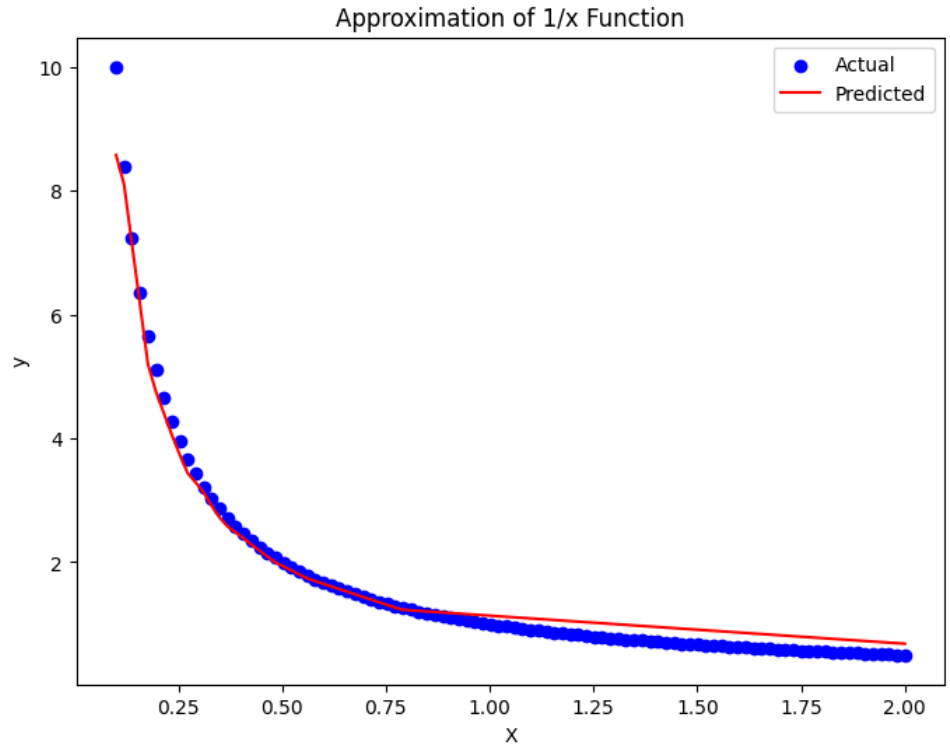
```
predictions = model.predict(X_test)
```

داده‌های پیش بینی شده را تولید می‌کنیم.

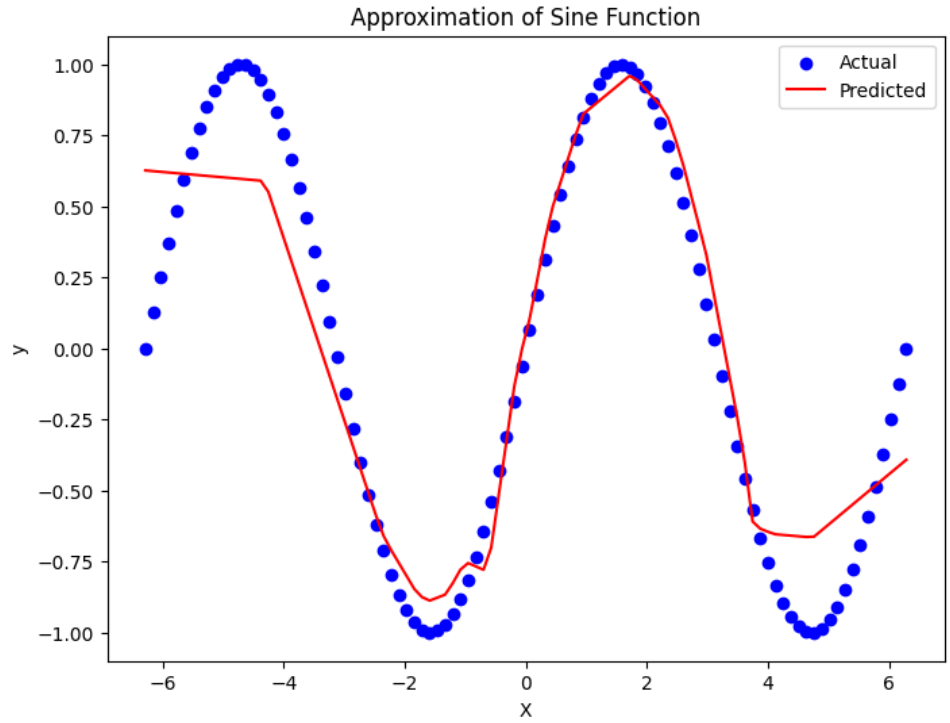
```
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, predictions, color='red', label='Predicted')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Approximation of 1/x Function')
plt.legend()
plt.show()
```

نتایج را رسم می‌کنیم.

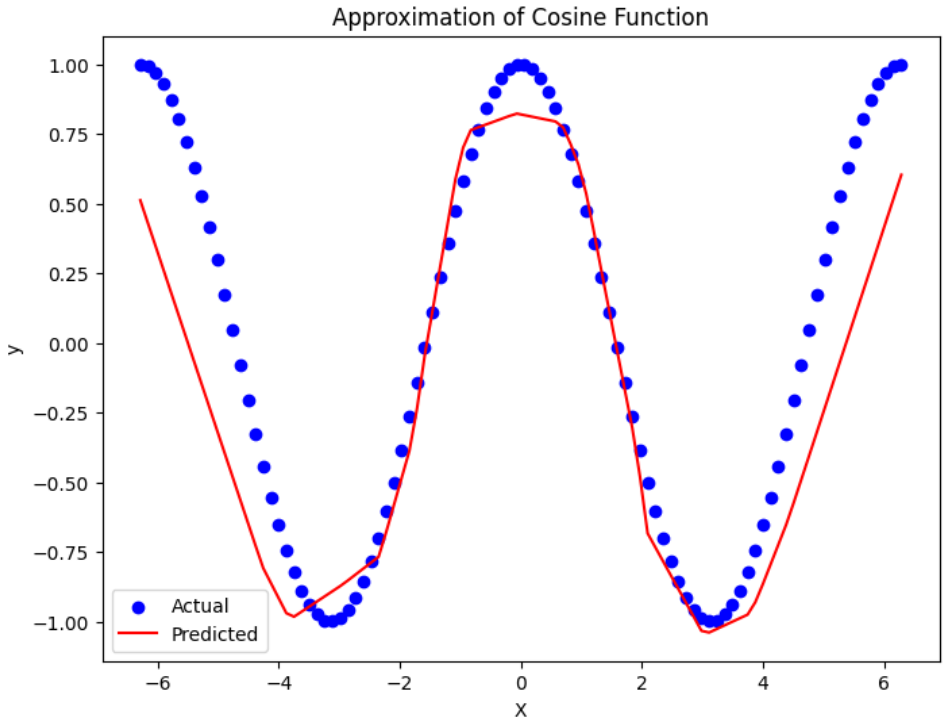
تابع 1/x:



تابع sin x:



تابع cos x:



هرچه تعداد نقاط ورودی و تعداد لایه‌های شبکه و نورون‌های هرلایه بیشتر باشد، دقت افزایش می‌یابد و هرچه تابع پیچیده‌تر باشد، دقت کاهش می‌یابد.

بخش چهارم:

```python
import cv2
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

ابتدا کتابخانه‌های لازم را import می‌کنیم.

```python
image = cv2.imread('red_line_image.jpg')
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
lower_red1 = np.array([0, 50, 50])
upper_red1 = np.array([10, 255, 255])
lower_red2 = np.array([170, 50, 50])
upper_red2 = np.array([180, 255, 255])
red_mask1 = cv2.inRange(hsv_image, lower_red1, upper_red1)
red_mask2 = cv2.inRange(hsv_image, lower_red2, upper_red2)
red_mask = cv2.bitwise_or(red_mask1, red_mask2)
edges = cv2.Canny(red_mask, 50, 150, apertureSize=3)
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

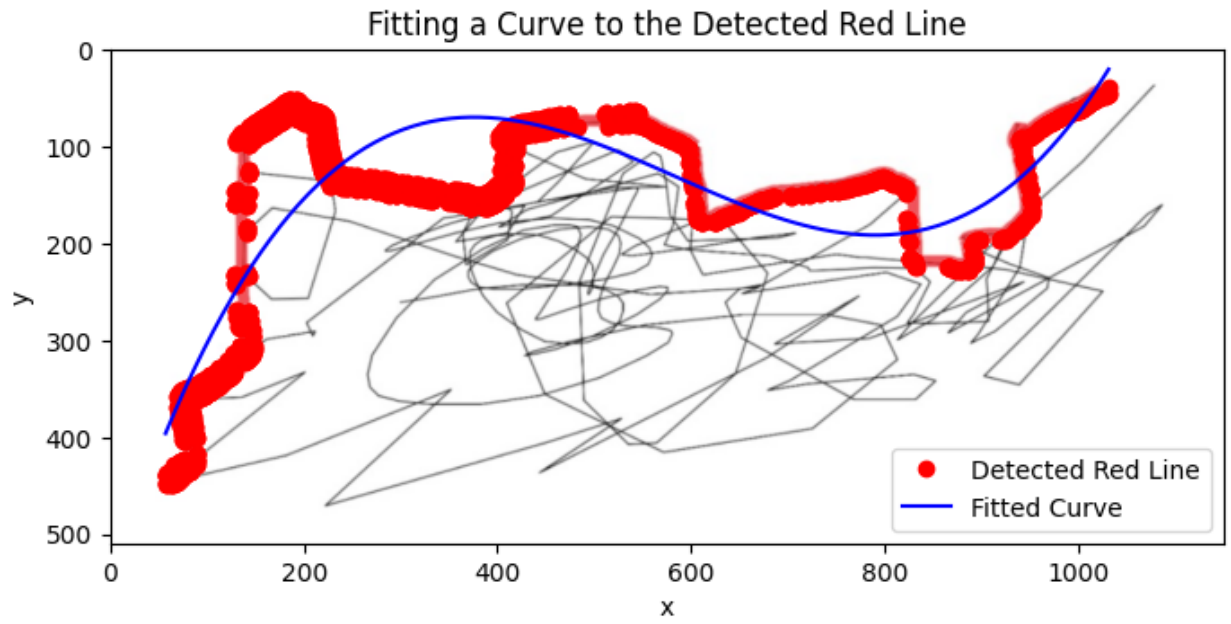
فرایند image processing را با کتابخانه OpenCV انجام می‌دهیم.

- تصویر مورد نظر را load می‌کنیم.

- تصویر load شده را از BGR به HSV تبدیل می‌کنیم. HSV معمولا برای image processingهایی که بر اساس رنگ هستند استفاده می‌شود.
- چند بازه برای رنگ قرمز تعریف می‌کنیم و سپس آنها را به صورت bitwise، or می‌کنیم.
- الگوریتم Canny edge را روی تصویر اعمال می‌کنیم. این الگوریتم بر اساس gradientها و thresholding عمل می‌کند و لبه‌ها را پیدا می‌کند.
- در مرحله بعد حدفاصل‌ها (contours) را پیدا می‌کنیم.

```
if contours:
    largest_contour = max(contours, key=cv2.contourArea)
    x_coords = largest_contour[:, 0, 0]
    y_coords = largest_contour[:, 0, 1]
    for x, y in zip(x_coords, y_coords):
        print(f"Red dot coordinates: ({x}, {y})")
    degree = 3
    coeffs = np.polyfit(x_coords, y_coords, degree)
    poly_func = np.poly1d(coeffs)
    x_plot = np.linspace(min(x_coords), max(x_coords), 100)
    y_plot = poly_func(x_plot)
    plt.figure(figsize=(8, 6))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.plot(x_coords, y_coords, 'ro', label='Detected Red
Line')
    plt.plot(x_plot, y_plot, 'b-', label='Fitted Curve')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Fitting a Curve to the Detected Red Line')
    plt.legend()
    plt.show()
else:
    print("No red line detected.")
```

- اگر حدفاصل‌ها پیدا شدند، ماکسیمم آنها را خط قرمز در نظر می‌گیریم و مختصات خط را پیدا می‌کنیم.
- سپس یک تابع polynomial برای خط پیدا شده fit می‌کنیم (درجه آن را 3 در نظر می‌گیریم).
- تابع پیدا شده را رسم می‌کنیم.



در این تصویر تشخیص خط قرمز و fit کردن یک تابع درجه 3 نشان داده شده است.

```
y_coords = max(y_coords) - y_coords
model = Sequential()
model.add(Dense(32, input_dim=1, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
```

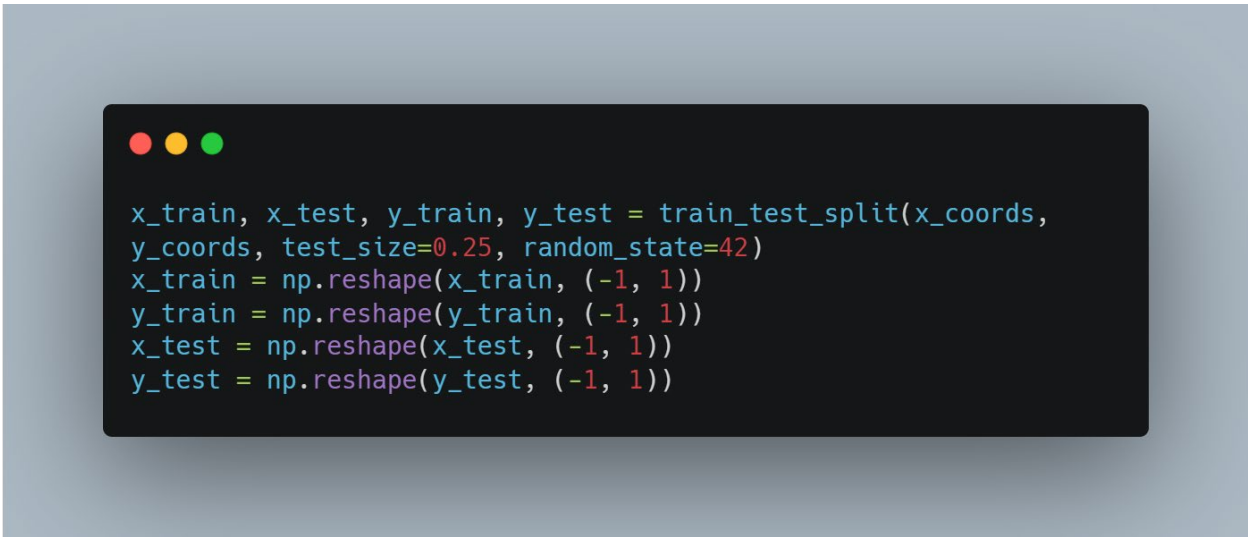
به ساخت شبکه عصبی می‌پردازیم:

- ابتدا یک instance از کلاس Sequential می‌سازیم که یک stack خطی از لایه‌هاست و به ما اجازه می‌دهد که شبکه‌های عصبی را با اضافه کردن لایه‌ها به ترتیب بسازیم.
- یک لایه fully connected به مدل اضافه می‌کنیم. این Dense layer یک لایه است که در آن هر نورون به همه نورون‌های لایه قبل خود متصل شده است. در اینجا دارای 32 نورون است. پارامتر input_dim تعداد ابعاد

داده ورودی را مشخص می‌کند. پارامتر activation='relu'، تابع فعال سازی rectified linear unit (ReLU) را برای این لایه مشخص می‌کند. ReLU در لایه‌های پنهان برای non-linearity استفاده می‌شود.

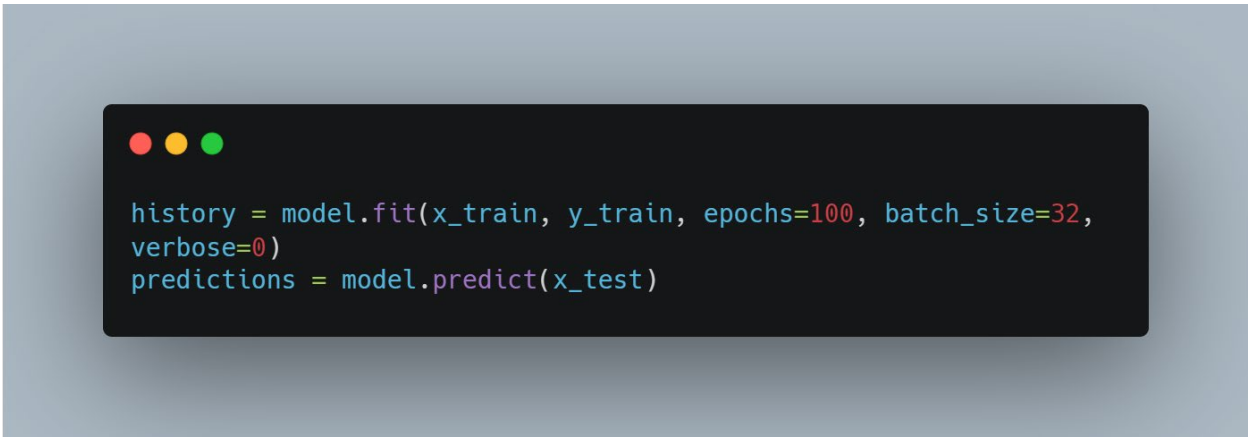
- یک لایه fully connected دیگر با 64 نورون اضافه می‌کنیم.
- یک لایه fully connected دیگر با 32 نورون اضافه می‌کنیم.
- در انتها یک لایه خروجی به مدل اضافه می‌کنیم. این لایه نیز fully connected است و یک نورون دارد و خروجی پیش بینی شده را نشان می‌دهد. Activation Function خطی تعیین می‌شود.

مدل را کامپایل می‌کنیم.



```
x_train, x_test, y_train, y_test = train_test_split(x_coords,
y_coords, test_size=0.25, random_state=42)
x_train = np.reshape(x_train, (-1, 1))
y_train = np.reshape(y_train, (-1, 1))
x_test = np.reshape(x_test, (-1, 1))
y_test = np.reshape(y_test, (-1, 1))
```

داده‌های train و test را جدا می کنیم و برای تطابق با مدل آرایه‌های یک بعدی را به دو بعدی تبدیل می‌کنیم.



```
history = model.fit(x_train, y_train, epochs=100, batch_size=32,
verbose=0)
predictions = model.predict(x_test)
```

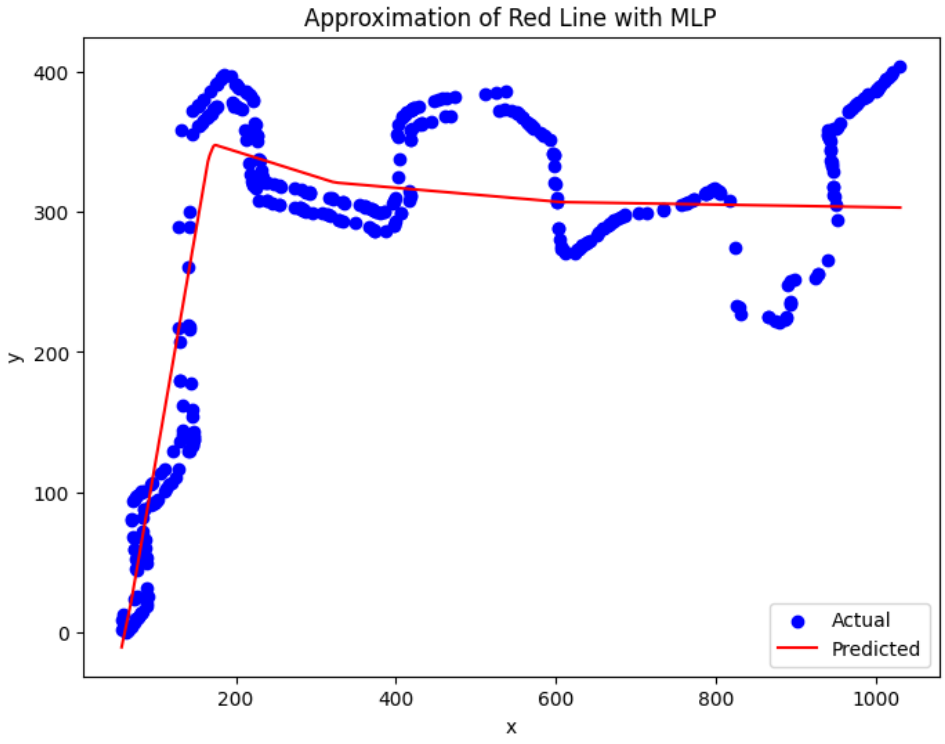
مدل را train کرده و پیش بینی ها را انجام می‌دهیم.

```python
sorted_indices = np.argsort(x_test.flatten())
x_test_sorted = x_test[sorted_indices]
y_test_sorted = y_test[sorted_indices]
predictions_sorted = predictions[sorted_indices]
mse = np.mean((predictions - y_test) ** 2)
print("Mean Squared Error (MSE):", mse)
plt.figure(figsize=(8, 6))
plt.scatter(x_test_sorted, y_test_sorted, color='blue',
label='Actual')
plt.plot(x_test_sorted, predictions_sorted, color='red',
label='Predicted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Approximation of Red Line with MLP')
plt.legend()
plt.show()
```

داده‌های تست را sort می‌کنیم و مدل را ارزیابی و رسم می‌کنیم.

مقدار MSE:

```
Mean Squared Error (MSE): 2458.833037601837
```



بخش پنجم:

در این بخش از dataset پیشنهادی USPS استفاده شده است.

```
import os
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from skimage.io import imread
from skimage.transform import resize
from skimage.feature import hog
from skimage import util
```

ابتدا کتابخانه‌های لازم را import می‌کنیم.

```
train_directory = "/content/drive/MyDrive/ANN/images/train"
test_directory = "/content/drive/MyDrive/ANN/images/test"
X_train = [] # Feature
y_train = [] # Label
X_test = []  # Feature
y_test = []  # Label
```

پوشه‌ای که داده‌های train و test در آن قرار دارند را مشخص کرده و آرایه‌هایی برای داده‌های test و train، initialize می‌کنیم.

```
orientations = 9
pixels_per_cell = (8, 8)
cells_per_block = (2, 2)
for filename in os.listdir(train_directory):
    if filename.endswith('.jpg'):
        digit = filename.split('_')[0]
        if digit.isdigit():
            file_path = os.path.join(train_directory, filename)
            image = imread(file_path, as_gray=True)
            inverted_image = util.invert(image)
            resized_image = resize(inverted_image, (32, 32))
            features = hog(resized_image,
orientations=orientations, pixels_per_cell=pixels_per_cell,
                           cells_per_block=cells_per_block,
transform_sqrt=True)

            X_train.append(features)
            y_train.append(int(digit))
X_train = np.array(X_train)
y_train = np.array(y_train)
```

- به ازای تمامی تصاویری که در پوشه train قرار دارند، برچسب آن تصویر را جدا می‌کنیم و مطمئن می‌شویم که رقم است.
- آدرس تصویر را به دست آورده و آن را به grayscale تبدیل می‌کنیم.
- تصویر را invert می‌کنیم یعنی زمینه سیاه آن را به سفید تبدیل کرده و رقم سفید را به سیاه تبدیل می‌کنیم تا مطمئن شویم که رقم در برابر زمینه مشخص است.
- همه تصاویر را به یک اندازه مشخص (32 x 32 pixels)، resize می‌کنیم.
- مقدار Histogram of Oriented Gradient (HOG) تصویر را به دست می‌آوریم. HOG یک ویژگی است که اطلاعات local gradient تصویر را ذخیره می‌کند. این اطلاعات برای تشخیص یک object در تصویر مورد استفاده قرار می‌گیرند.
- مقادیر HOG محاسبه شده را در X_train و برچسب رقم هر تصویر را در y_train ذخیره کرده و در انتها این دو لیست را به Numpy array تبدیل می‌کنیم.

```
for filename in os.listdir(test_directory):
    if filename.endswith('.jpg'):
        digit = filename.split('_')[0]
        if digit.isdigit():
            file_path = os.path.join(test_directory, filename)
            image = imread(file_path, as_gray=True)
            inverted_image = util.invert(image)
            resized_image = resize(inverted_image, (32, 32))
            features = hog(resized_image,
orientations=orientations, pixels_per_cell=pixels_per_cell,
                           cells_per_block=cells_per_block,
transform_sqrt=True)
            X_test.append(features)
            y_test.append(int(digit))
X_test = np.array(X_test)
y_test = np.array(y_test)
```

مراحل بالا را برای داده‌های تست نیز انجام می‌دهیم.

```
clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

test_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", test_accuracy)
```

یک MLP (Multi-Layer Perceptron) classifier می‌سازیم.

- تعداد نورون‌ها در هر لایه پنهان را 100 نورون تعیین می‌کنیم.
- حداکثر تعداد iterationها برای train کردن MLP را 1000 تعیین می‌کنیم.

دقت به دست آمده:

```
Accuracy: 0.9586447433981067
```

بخش ششم:

```
import os
import numpy as np
import tensorflow as tf
import skimage.io
from skimage.util import random_noise
from sklearn.model_selection import train_test_split
from skimage.metrics import peak_signal_noise_ratio,
structural_similarity
from skimage.transform import resize
import matplotlib.pyplot as plt
```

ابتدا کتابخانه‌های لازم را import می‌کنیم.

```
non_noisy_directory = "/content/drive/MyDrive/ANN/images/train"
noisy_directory = "/content/drive/MyDrive/ANN/images/noisy"

noise_type = 'gaussian'
variance = 0.01

for filename in os.listdir(non_noisy_directory):
   if filename.endswith('.jpg'):
       file_path_non_noisy = os.path.join(non_noisy_directory,
filename)
       non_noisy_image = skimage.io.imread(file_path_non_noisy)
       noisy_image = random_noise(non_noisy_image,
mode=noise_type, var=variance)
       noisy_filename = 'noisy_' + filename
       noisy_file_path = os.path.join(noisy_directory,
noisy_filename)
       skimage.io.imsave(noisy_file_path, (noisy_image *
255).astype(np.uint8))
```

- پوشه‌ای که در آن فایل‌های اصلی قرار دارند و پوشه‌ای که می‌خواهیم تصاویر noisy را در آن ذخیره کنیم را مشخص می‌کنیم.
- نوع نویز را Gaussian و واریاس آن را 0.01 در نظر می‌گیریم. با تغییر این مقدار می‌توان مقدار نویز را تغییر داد.
- نویز را روی تصاویر اعمال می‌کنیم و تصاویر noisy را ذخیره می‌کنیم.

```
noisy_images = []
non_noisy_images = []

for noisy_filename in os.listdir(noisy_directory):
    if noisy_filename.startswith('noisy_'):
        common_identifier = noisy_filename.split('noisy_')[1]
        non_noisy_filename = common_identifier
        file_path_noisy = os.path.join(noisy_directory,
noisy_filename)
        file_path_non_noisy = os.path.join(non_noisy_directory,
non_noisy_filename)
        noisy_image = skimage.io.imread(file_path_noisy)
        non_noisy_image = skimage.io.imread(file_path_non_noisy)
        noisy_images.append(noisy_image)
        non_noisy_images.append(non_noisy_image)
```

بر اساس نام تصاویر اصلی و noisy، ورودی (تصویر noisy) و خروجی مورد نظر (تصویر اصلی و بدون نویز) را مشخص می‌کنیم.

```
resized_noisy_images = []
resized_non_noisy_images = []
desired_shape = (32, 32)
for noisy_image, non_noisy_image in zip(noisy_images,
non_noisy_images):
    resized_noisy_image = resize(noisy_image, desired_shape,
mode='reflect', anti_aliasing=True)
    resized_non_noisy_image = resize(non_noisy_image,
desired_shape, mode='reflect', anti_aliasing=True)
    resized_noisy_images.append(resized_noisy_image)
    resized_non_noisy_images.append(resized_non_noisy_image)

noisy_images = np.array(resized_noisy_images)
non_noisy_images = np.array(resized_non_noisy_images)

noisy_images = noisy_images / 255.0
non_noisy_images = non_noisy_images / 255.0
```

- تمامی تصاویر را به ابعاد مشخصی (تمامی تصاویر را به ابعاد مشخصی (32 x 32 pixels)، resize می‌کنیم.
- لیست تصاویر اصلی و تصاویر noisy را به Numpy array تبدیل می‌کنیم.
- مقادیر پیکسل‌ها را به بازه 0 تا 1، normalize می‌کنیم.

```
X_train, X_test, y_train, y_test = train_test_split(noisy_images,
non_noisy_images, test_size=0.25)

X_train = np.reshape(X_train, (*X_train.shape, 1))
X_test = np.reshape(X_test, (*X_test.shape, 1))
```

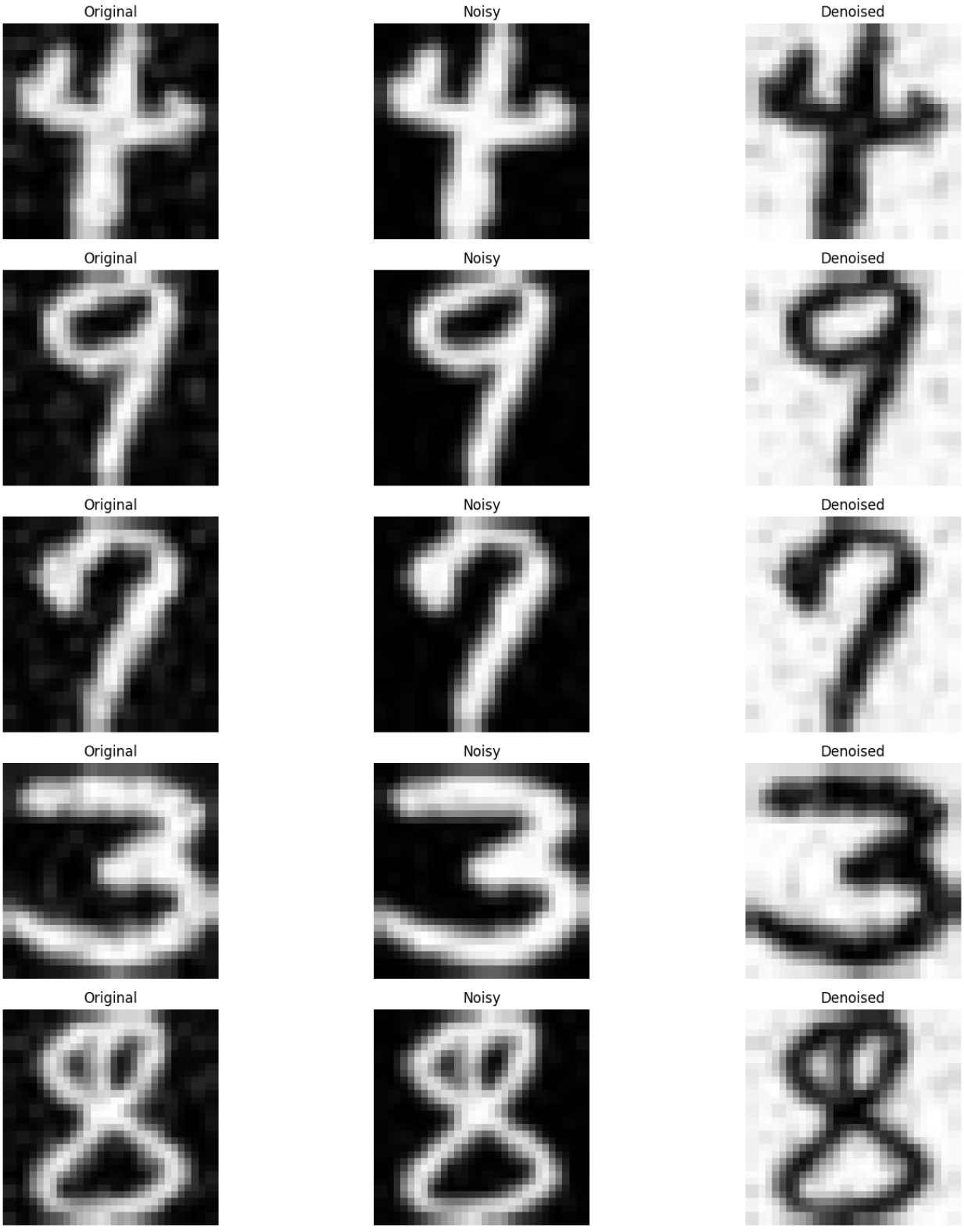
داده‌های train و test را جدا کرده و ابعاد داده‌های ورودی را مشخص می‌کنیم.

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(32,
32, 1)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(32, activation='sigmoid')
])

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

model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_test, y_test))
```

- همانند بخش‌های قبلی پروژه، لایه‌های شبکه عصبی را می‌سازیم.
- شبکه عصبی را کامپایل کرده و مدل را train می‌کنیم.

```python
denoised_images = model.predict(X_test)

print(denoised_images.shape)

print(np.unique(denoised_images[:,:,:,0]))

denoised_images = (denoised_images + 1) / 2.0

num_images = 5
random_indices = np.random.choice(range(len(X_test)), num_images,
replace=False)

fig, axes = plt.subplots(num_images, 3, figsize=(15, 15))

for i, idx in enumerate(random_indices):
    axes[i, 0].imshow(X_test[idx].squeeze(), cmap='gray')
    axes[i, 0].set_title('Original')
    axes[i, 0].axis('off')

    axes[i, 1].imshow(y_test[idx].squeeze(), cmap='gray')
    axes[i, 1].set_title('Noisy')
    axes[i, 1].axis('off')

    axes[i, 2].imshow(denoised_images[idx, :, :, 0], cmap='gray')
    axes[i, 2].set_title('Denoised')
    axes[i, 2].axis('off')

plt.tight_layout()
plt.show()
```

- تصاویر denoise شده را با استفاده از تابع predict در denoised_images ذخیره می کنیم.
- این تصاویر را از بازه -1 تا 1 به 0 تا 1، rescale می کنیم تا به بازه grayscale بازگردند.
- 5 مورد از داده های تست را به صورت تصادفی انتخاب می کنیم و آنها را نمایش می دهیم.


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