یاسمن توکلی:98522175

پروژه شبکه عصبی

X: نقاط ورودی که به صورت تصادفی و با توزیع یکنواخت در بازه [−2π,2π] تولید شده‌اند.

y: خروجی‌های متناظر با تابع کسینوس برای هر نقطه ورودی X.

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

# Generate training data
np.random.seed(1)
# Input features with 1000 samples
X = np.random.uniform(-2*np.pi, 2*np.pi, 1000)
y = np.cos(X)
```

ساخت مدل شبکه عصبی:

- مدل از نوع Sequential است که به معنای افزودن لایه‌ها به ترتیب است.
- اولین لایه پنهان با 16 نورون و تابع فعال‌سازی ReLU.
- لایه‌ی Dropout برای جلوگیری از بیش‌برازش.
- دومین لایه پنهان با 8 نورون.
- لایه خروجی برای پیش‌بینی نهایی.

```
model = Sequential()
# Hidden layer with 16 neurons
model.add(Dense(16, input_dim=1, activation='relu'))
model.add(Dropout(0.2))  # Dropout layer for regularization
model.add(Dense(8, activation='relu'))  # Additional hidden layer
model.add(Dense(1))  # Output layer
```

کامپایل و آموزش مدل:

تابع خطا به صورت میانگین مربعات خطاها.

بهینه‌ساز adam.

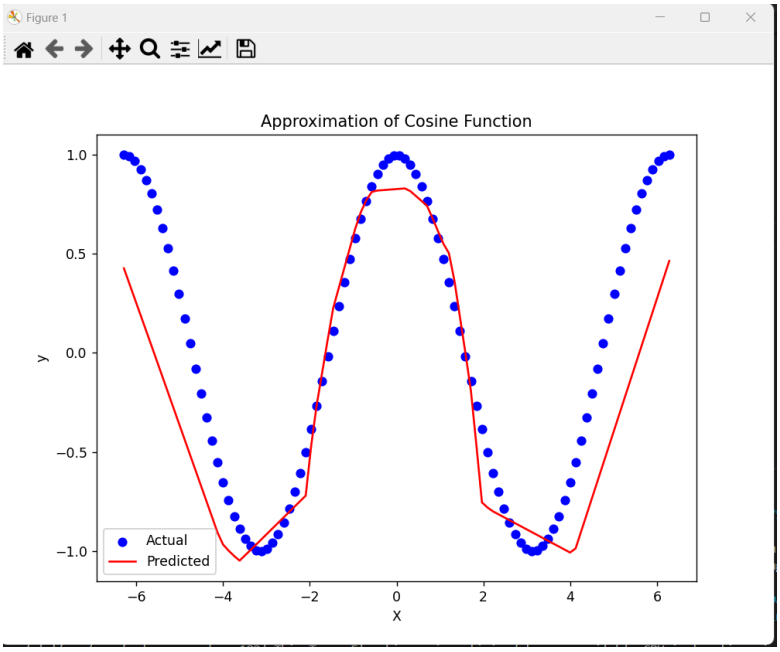
آموزش مدل با 200 دوره و اندازه دسته 32.

در انتها نیز تابع پیش بینی شده و تابع اصلی را کنار هم یکدیگر رسم میکنیم:

```
model.compile(loss='mean_squared_error', optimizer='adam')
# Train the model
history = model.fit(X, y, epochs=200, batch_size=32, verbose=0)  # epochs=100

# Generate test data
X_test = np.linspace(-2*np.pi, 2*np.pi, 100)
y_test = np.cos(X_test)

# Make predictions
predictions = model.predict(X_test)

# Plot the results
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 Cosine Function')
plt.legend()
plt.show()
```

نتیجه رسم:

یاسمن توکلی:98522175



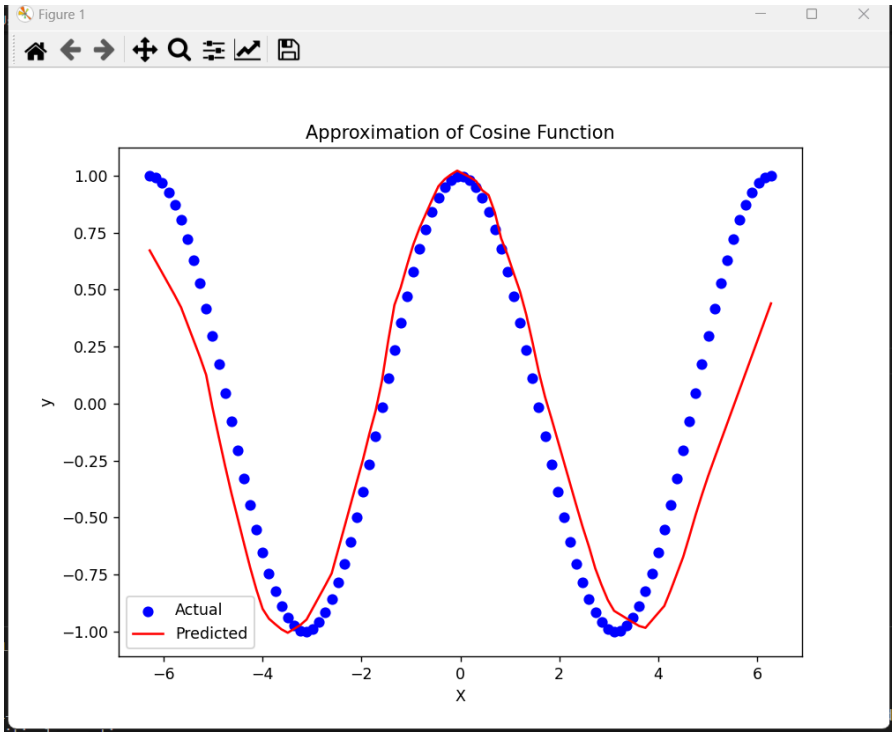
تحلیل پارامترها

- تعداد نقاط ورودی: افزایش تعداد نقاط ورودی می‌تواند به دقت بیشتر مدل کمک کند زیرا مدل تجربه بیشتری برای یادگیری الگوهای داده‌ها خواهد داشت.
- میزان پیچیدگی تابع: پیچیدگی بالاتر تابع مورد نظر می‌تواند نیازمند شبکه‌ای با لایه‌ها و نورون‌های بیشتری باشد تا بتواند الگوهای پیچیده‌تری را یاد بگیرد.
- تعداد لایه‌ها و نورون‌های هر لایه: افزایش تعداد لایه‌ها و نورون‌ها به معمولاً به بهبود توانایی مدل در یادگیری الگوهای پیچیده کمک می‌کند، اما ممکن است باعث بیش‌برازش شود.
- تعداد دوره‌های آموزشی (epochs): افزایش تعداد دوره‌های آموزشی می‌تواند به مدل اجازه دهد که الگوهای دقیق‌تری را یاد بگیرد، اما بعد از یک نقطه ممکن است تأثیر کمتری داشته باشد.
- وسعت دامنه ورودی (batch size): در توابع پیچیده‌تر، وسعت دامنه ورودی باید به گونه‌ای انتخاب شود که تمامی خصوصیات تابع قابل مشاهده باشد.

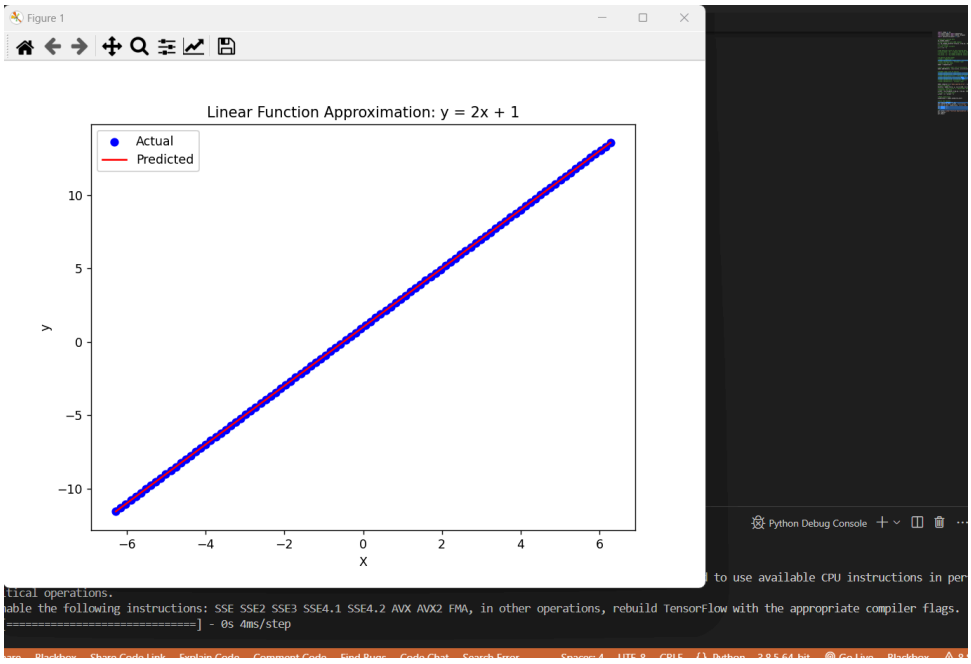
پس پارامترهای زیر را تغییر میدهیم تا نتیجه دقیق تری داشته باشیم:

- می‌توانیم یک لایه پنهان دیگر با 16 نورون اضافه کنیم.

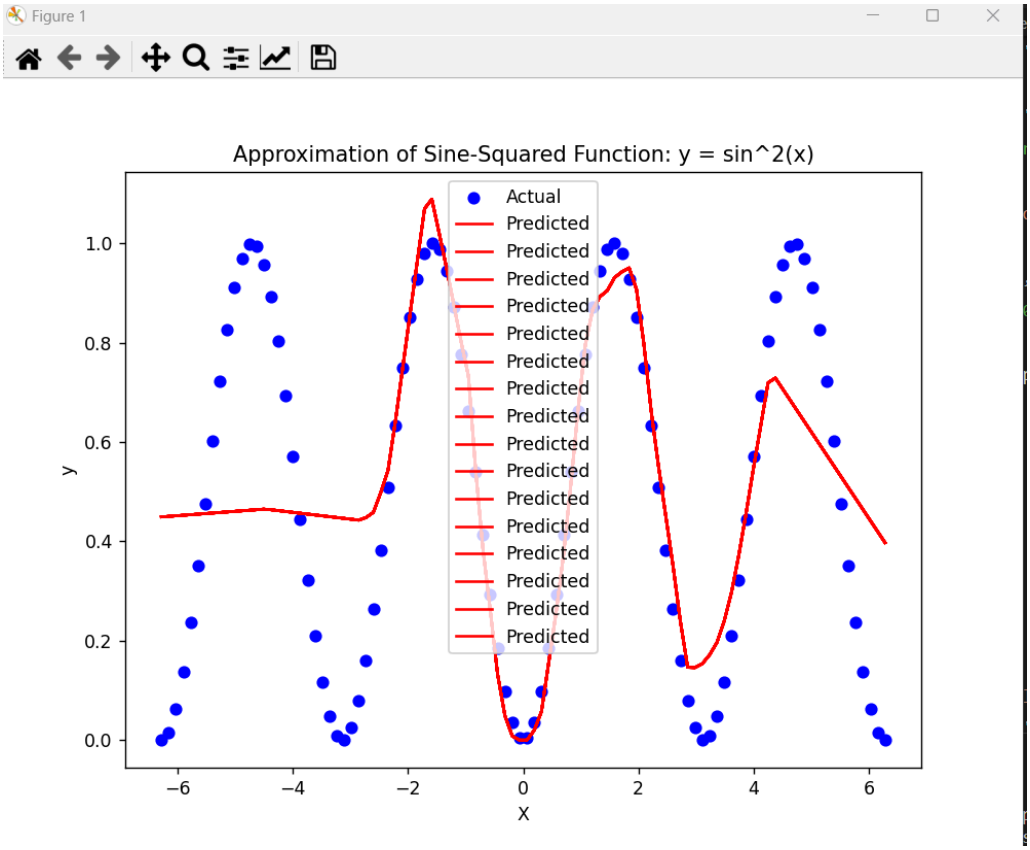
- می‌توان تعداد دوره‌ها را از 200 به 300 افزایش داد.
- اندازه دسته را از 32 به 16 تغییر دهیم.



با چند تابع دیگر:



یاسمن توکلی:98522175



بخش دوم:

کد این بخش، یک تصویر را می‌خواند و خطوط قرمز را در آن تشخیص داده و سپس یک منحنی ریاضی را بر روی مختصات آن خطوط فیت می‌کند. این کد از کتابخانه‌های SciPy، NumPy، OpenCV و Matplotlib برای پردازش تصویر و تجزیه و تحلیل داده استفاده می‌کند.

سپس تصویر از فضای رنگی BGR (استاندارد OpenCV) به HSV تبدیل می‌شود تا تشخیص رنگ‌ها ساده‌تر باشد.

```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

image = cv2.imread('red_line_image.jpg')
# Convert image to HSV color space
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

آستانه‌های رنگ قرمز در فضای رنگی HSV این امکان را می‌دهد تا رنگ‌هایی با تن قرمز را از سایر رنگ‌ها در تصویر تشخیص بدیم. در فضای رنگی HSV، رنگ قرمز دارای ویژگی‌های خاصی است که می‌توان از آن‌ها برای فیلتر کردن رنگ‌ها استفاده کرد.

ایجاد ماسک برای پیکسل‌های قرمز به معنای ساخت یک تصویر دودویی (باینری) است که در آن، پیکسل‌هایی که معیارهای تعریف‌شده برای رنگ قرمز را دارند به صورت سفید (یا 1) و بقیه به صورت سیاه (یا 0) نشان داده می‌شوند. این کار به ما امکان می‌دهد تا تنها بخش‌هایی از تصویر که قرمز هستند را شناسایی و پردازش کنیم.

از آنجا که رنگ قرمز می‌تواند در دو بازه مختلف قرار گیرد، ماسک‌های ایجاد شده باید با هم ترکیب شوند تا تمامی تون‌های قرمز پوشش داده شوند.

استفاده از تابع cv2.bitwise_or() برای ترکیب دو ماسک انجام می‌شود، که نتیجه آن یک ماسک واحد است که تمامی پیکسل‌های قرمز را در بر می‌گیرد.

این فرآیند امکان دستکاری و تحلیل دقیق‌تر بخش‌هایی از تصویر که حاوی اطلاعات مورد نظر هستند را فراهم می‌آورد.ماسک قرمز برای تشخیص خطوط قرمز استفاده می‌شود و سپس از آن برای یافتن و فیت کردن خطوط به یک منحنی استفاده می‌شود.

```python
# Define lower and upper thresholds for red color in HSV
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])

# Create masks for red pixels using the thresholds
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)

# Apply Canny edge detection on the red mask
edges = cv2.Canny(red_mask, 50, 150, apertureSize=3)
```

کانتورها در پردازش تصویر، به معنای خطوطی هستند که شکل و مرزهای اشیاء در یک تصویر را نشان می‌دهند. در کتابخانه OpenCV، کانتورها به عنوان یک لیست از نقاطی که یک شکل را توصیف می‌کنند، ذخیره می‌شوند.

هر کانتور دارای نقاطی است که با هم متصل شده و حدود یک شکل معین را در تصویر مشخص می‌کنند.

پس کانتورها در تصویر لبه پیدا میکنیم چرا که نماینده‌ی خطوط و شکل‌های موجود هستند.

در صورت وجود کانتور، بزرگ‌ترین کانتور (که نماینده‌ی خط قرمز است) استخراج می‌شود.

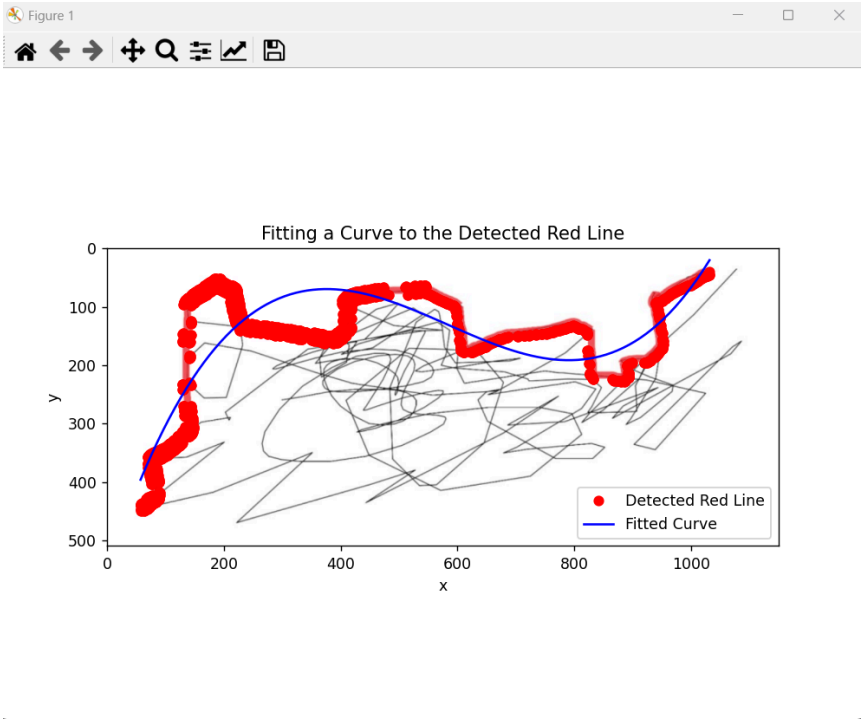
مختصات کانتور استخراج شده و یک منحنی چندجمله‌ای درجه سه بر روی آن فیت می‌شود.

```
contours, _ = cv2.findContours(
    edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Check if contours were found
if contours:
    # Extract the largest contour (assuming it corresponds to the red line)
    largest_contour = max(contours, key=cv2.contourArea)

    # Extract the coordinates of the red line
    x_coords = largest_contour[:, 0, 0]
    y_coords = largest_contour[:, 0, 1]

    # Print the coordinates of the red dots
    for x, y in zip(x_coords, y_coords):
        print(f"Red dot coordinates: ({x}, {y})")
```

```
# Fit a polynomial curve to the coordinates
degree = 3  # Degree of the polynomial curve
coeffs = np.polyfit(x_coords, y_coords, degree)
poly_func = np.poly1d(coeffs)

# Generate x values for plotting the curve
x_plot = np.linspace(min(x_coords), max(x_coords), 100)

# Evaluate the polynomial function for the plotted x values
y_plot = poly_func(x_plot)
```

در اخر هم منحنی پیش بینی شده را رسم میکنیم:

یاسمن توکلی:98522175



اگر نمودار ترسیم شده توسط کد دارای نقاط پرش ناگهانی مانند شکل‌های بالا باشد، این نشان‌دهنده چند احتمال می‌تواند باشد:

پراکندگی داده‌ها:

نقاط پرش ناگهانی می‌توانند نتیجه پراکندگی بالا در داده‌های واقعی باشند که کانتور بر اساس آنها شکل گرفته است. اگر داده‌ها نامرتب و با فاصله‌های ناگهانی وجود داشته باشند، منحنی فیت شده ممکن است دقت کمتری داشته باشد و به صورت پرشی نمایش داده شود.

محدودیت‌های الگوریتم فیت کردن:

الگوریتم‌های فیت کردن منحنی، مانند np.polyfit که برای فیت کردن منحنی‌های چندجمله‌ای استفاده می‌شوند، ممکن است در مواجهه با داده‌های پیچیده یا نامتعارف، نتوانند یک تخمین صاف و دقیق ارائه دهند. این موضوع به‌خصوص در داده‌هایی با تغییرات ناگهانی یا شدید بیشتر مشهود است.

کیفیت داده‌ها و پیش‌پردازش:

کیفیت داده‌های ورودی و نحوه پیش‌پردازش آن‌ها (مانند فیلتر کردن نویز، اعمال ماسک‌ها و تشخیص لبه‌ها) می‌تواند تأثیر زیادی بر نتیجه نهایی داشته باشد. اگر داده‌های ورودی شامل نویز یا خطاهایی باشند که به خوبی مدیریت نشده‌اند، ممکن است نتایج به دست آمده نیز دارای خطا و ناپایدار باشند.

انتخاب درجه چندجمله‌ای:

انتخاب درجه چندجمله‌ای برای فیت کردن منحنی می‌تواند بر روی صافی و پیوستگی منحنی تأثیر بگذارد. اگر درجه چندجمله‌ای بسیار بالا باشد، ممکن است منجر به overfitting شود که در آن منحنی تلاش می‌کند تمام نوسانات کوچک در داده‌ها را مدل کند و نتیجه‌ای ناپایدار و پرپرش ایجاد کند.

راه‌حل‌ها:

- بهبود پیش‌پردازش داده‌ها: اطمینان از کیفیت بالای داده‌های ورودی و کاهش نویز قبل از فرآیند فیت کردن.

- استفاده از روش‌های فیت کردن منحنی مناسب‌تر: امکان استفاده از الگوریتم‌های پیچیده‌تر مانند splines یا regression models که می‌توانند داده‌های نامتعارف را بهتر مدیریت کنند.
- انتخاب درجه چندجمله‌ای مناسب: انتخاب درجه مناسب برای جلوگیری از overfitting و underfitting.
- استفاده از معیارهای آماری برای ارزیابی کیفیت فیت: استفاده از معیارهایی مانند R-squared برای سنجش کیفیت فیت منحنی به داده‌ها.

بخش سوم:

کد این بخش برای ساخت یک مدل شبکه عصبی کانولوشنی (CNN) برای تشخیص ده کلاس مختلف تصاویر از دیتاست CIFAR-10 استفاده می‌کند. این کد با استفاده از کتابخانه TensorFlow و ماژول Keras نوشته شده است.

بارگذاری دیتاست CIFAR-10:

دیتاست CIFAR-10 شامل 60,000 تصویر رنگی 32x32 در 10 کلاس مختلف است. تصاویر به دو بخش آموزشی و آزمایشی تقسیم شده‌اند.

نرمال‌سازی داده‌ها:

پیکسل‌های تصاویر برای قرارگیری در بازه 0 تا 1 نرمال‌سازی می‌شوند تا پردازش راحت‌تر شود.

```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
```

مدل شبکه عصبی کانولوشنی (CNN)، با استفاده از لایه‌های مختلف به منظور پردازش و تحلیل تصاویر برای تشخیص ده کلاس مختلف تصاویر طراحی شده است.

لایه‌های کانولوشنی (Conv2D)

این لایه‌ها اساس کار شبکه‌های کانولوشنی هستند. هر لایه از فیلترهایی با ابعاد 3x3 پیکسل تشکیل شده است که به صورت موازی بر روی تصویر ورودی اعمال می‌شوند. این فیلترها به دنبال الگوهای خاصی مانند لبه‌ها،

بافت‌ها و دیگر ویژگی‌های مربوط به تصویر هستند. فیلترها با حرکت روی تصویر (عملیات کانولوشن)، نقشه ویژگی‌هایی را ایجاد می‌کنند که جزئیات مهم تصویر را در بر دارند.

لایه فعال‌سازی ReLU

پس از هر لایه کانولوشنی، لایه فعال‌سازی ReLU (Rectified Linear Unit) قرار دارد. این لایه به غیرفعال کردن تمامی مقادیر منفی در نقشه ویژگی‌ها (خروجی کانولوشن) کمک می‌کند و فقط مقادیر مثبت را حفظ می‌کند. این فرآیند به بهبود عملکرد و سرعت آموزش شبکه کمک می‌کند.

لایه‌های MaxPooling

این لایه‌ها به کاهش ابعاد نقشه‌های ویژگی کمک می‌کنند. MaxPooling عموماً مقادیر بیشینه را در یک پنجره مشخص (معمولاً 2x2) انتخاب می‌کند و با این کار، تعداد ویژگی‌هایی که باید توسط شبکه پردازش شوند را کاهش می‌دهد. این فرآیند همچنین به جلوگیری از بیش‌برازش کمک می‌کند و مقاومت مدل در برابر جابه‌جایی مکانی را افزایش می‌دهد.

لایه‌های Dropout

این لایه‌ها به صورت تصادفی برخی از نورون‌ها را در هر دوره آموزش غیرفعال می‌کنند، که این امر به کاهش بیش‌برازش در شبکه کمک می‌کند. Dropout مطمئن می‌شود که شبکه بر روی داده‌های آموزشی بیش از حد وابسته نشود و توانایی تعمیم به داده‌های جدید را داشته باشد.

لایه Flatten

پس از استخراج ویژگی‌ها و کاهش ابعاد، لایه Flatten تمام ویژگی‌های استخراج شده را به یک بردار یک‌بعدی تبدیل می‌کند که می‌تواند به عنوان ورودی به لایه‌های بعدی داده شود.

لایه‌های Dense

این لایه‌ها به عنوان بخش تصمیم‌گیری شبکه عمل می‌کنند. اولین لایه Dense معمولاً تعداد زیادی نورون دارد (در این مورد 512) و لایه نهایی که تعداد نورون‌های آن برابر با تعداد کلاس‌ها است (10 برای CIFAR-10)، با استفاده از تابع فعال‌سازی softmax احتمال تعلق هر تصویر به هر کلاس را محاسبه می‌کند.

کامپایل مدل:

مدل با بهینه‌ساز Adam، تابع هزینه sparse_categorical_crossentropy و معیار دقت کامپایل می‌شود.

```
# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

آموزش مدل:

مدل با داده‌های آموزشی برای 20 دوره (epoch) آموزش داده می‌شود و داده‌های آزمایشی برای ارزیابی استفاده می‌شوند تا بفهمیم به کدام دسته تعلق دارند:

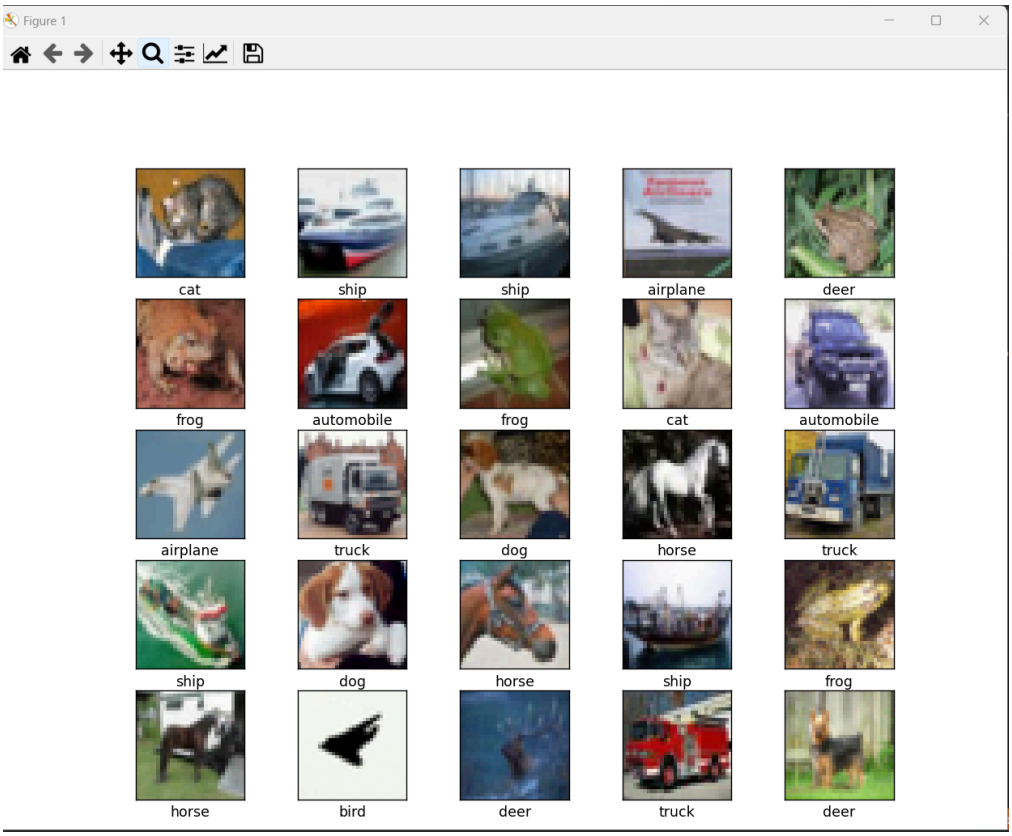
```
history = model.fit(x_train, y_train, epochs=20,
                    validation_data=(x_test, y_test))
```

یاسمن توکلی:98522175

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

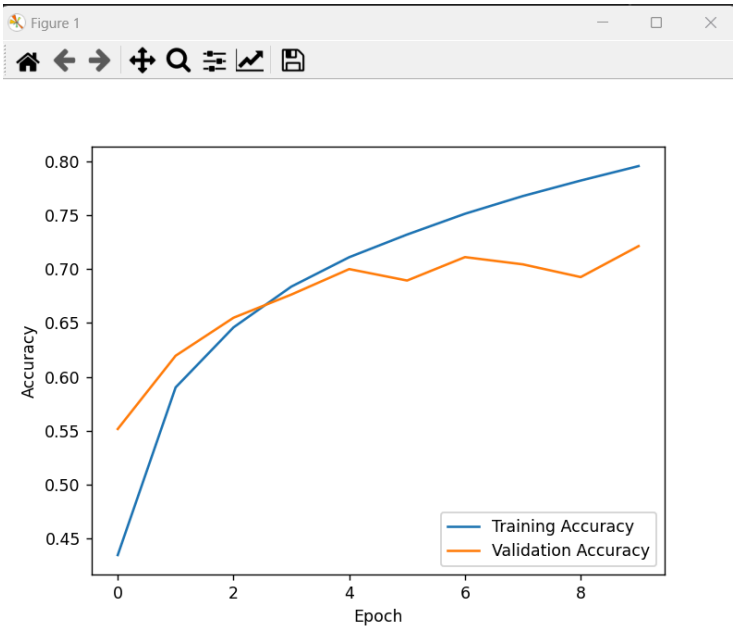
نتایج آزمایش بعد از دوبار تست:

تست اول:



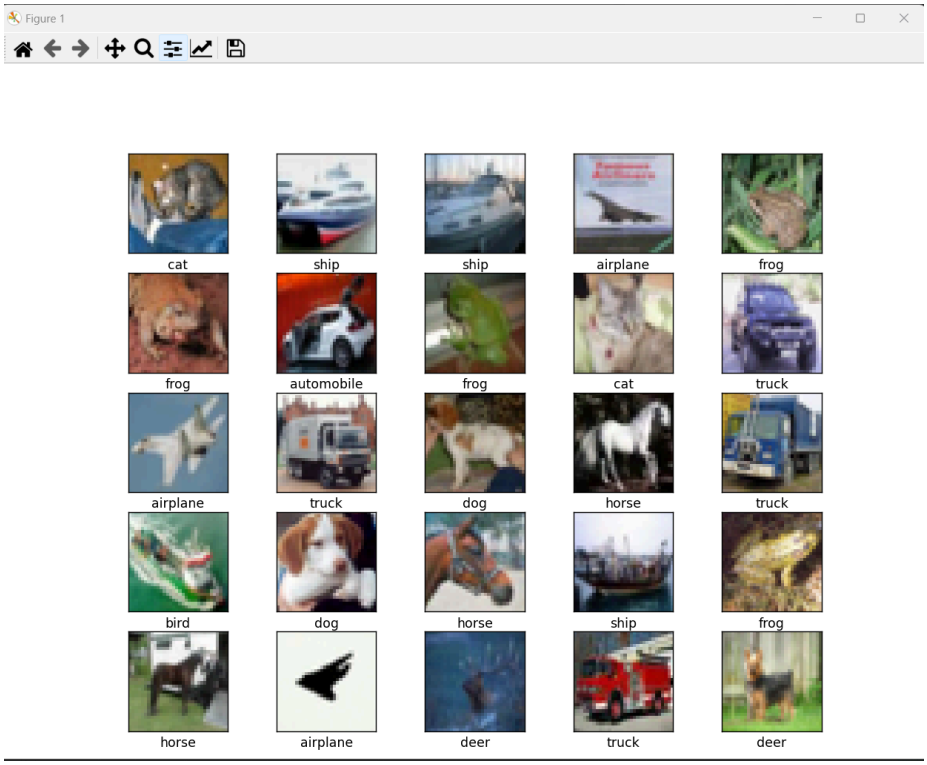
یاسمن توکلی:98522175

```
PROBLEMS 44  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  JUPYTER                Python Debug Console

313/313 [==============================] - 2s 5ms/step
PS C:\Users\Jasmine\Desktop\TenthSemester\AI\second-project> ^C
PS C:\Users\Jasmine\Desktop\TenthSemester\AI\second-project>
PS C:\Users\Jasmine\Desktop\TenthSemester\AI\second-project>  c:; cd 'c:\Users\Jasmine\Desktop\TenthSemester\AI\second-project'; & 'c:\Users\Jasmine\AppData\Local\Programs\Python\Python38\python.exe' 'c:\Users\Jasmine\.vscode\extensions\ms-python.debugpy-2024.4.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '7523' '--' 'C:\Users\Jasmine\Desktop\TenthSemester\AI\second-project\q3.py'
2024-04-12 18:07:47.797475: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/10
1563/1563 [==============================] - 25s 15ms/step - loss: 1.5505 - accuracy: 0.4350 - val_loss: 1.2474 - val_accuracy: 0.5518
Epoch 2/10
1563/1563 [==============================] - 24s 15ms/step - loss: 1.1580 - accuracy: 0.5903 - val_loss: 1.0837 - val_accuracy: 0.6197
Epoch 3/10
1563/1563 [==============================] - 23s 15ms/step - loss: 1.0080 - accuracy: 0.6460 - val_loss: 0.9938 - val_accuracy: 0.6549
Epoch 4/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.9050 - accuracy: 0.6838 - val_loss: 0.9328 - val_accuracy: 0.6764
Epoch 5/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.8272 - accuracy: 0.7110 - val_loss: 0.8697 - val_accuracy: 0.7000
Epoch 6/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.7646 - accuracy: 0.7320 - val_loss: 0.8986 - val_accuracy: 0.6894
Epoch 7/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.7092 - accuracy: 0.7513 - val_loss: 0.8495 - val_accuracy: 0.7111
Epoch 8/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.6620 - accuracy: 0.7677 - val_loss: 0.8842 - val_accuracy: 0.7044
Epoch 9/10
1563/1563 [==============================] - 23s 15ms/step - loss: 0.6198 - accuracy: 0.7821 - val_loss: 0.9049 - val_accuracy: 0.6926
Epoch 10/10
1563/1563 [==============================] - 24s 15ms/step - loss: 0.5808 - accuracy: 0.7955 - val_loss: 0.8494 - val_accuracy: 0.7213
313/313 - 1s - loss: 0.8494 - accuracy: 0.7213 - 1s/epoch - 4ms/step
```



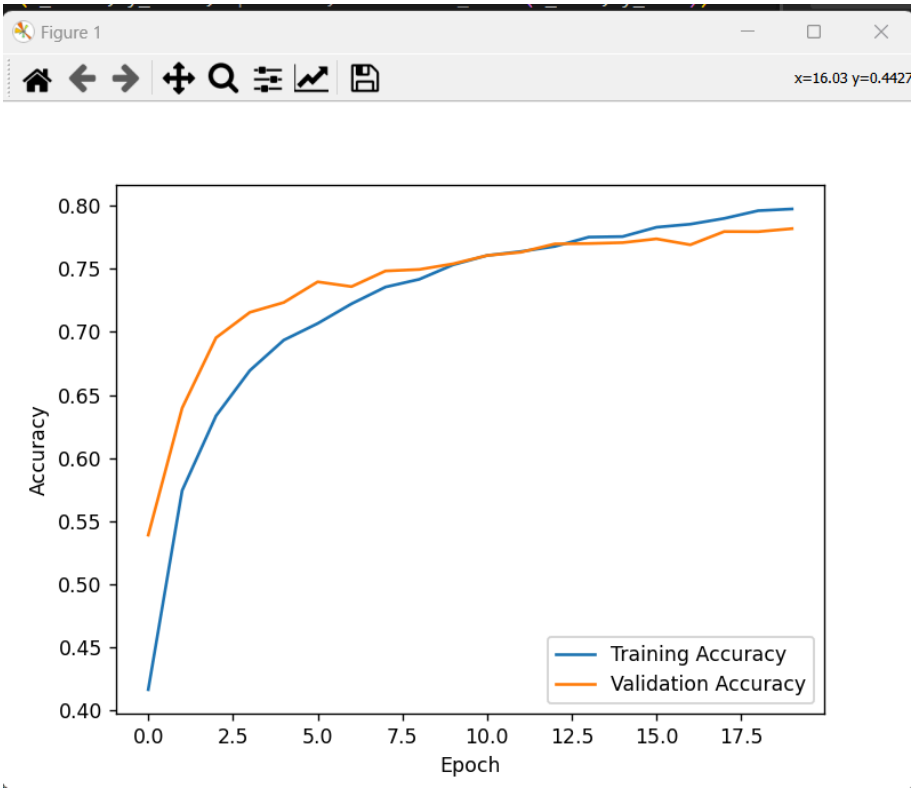
تست دوم:

یاسمن توکلی:98522175



همانطور که میبنید بعضی از تصاویر لیبل اشتباهی دارند.

```
PROBLEMS 47  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR  JUPYTER        Python Debug Console

Epoch 9/20
1563/1563 [==============================] - 75s 48ms/step - loss: 0.7303 - accuracy: 0.7418 - val_loss: 0.7370 - val_accuracy: 0.7496
Epoch 10/20
1563/1563 [==============================] - 73s 47ms/step - loss: 0.7017 - accuracy: 0.7533 - val_loss: 0.7235 - val_accuracy: 0.7541
Epoch 11/20
1563/1563 [==============================] - 73s 46ms/step - loss: 0.6828 - accuracy: 0.7607 - val_loss: 0.7064 - val_accuracy: 0.7607
Epoch 12/20
1563/1563 [==============================] - 74s 47ms/step - loss: 0.6718 - accuracy: 0.7639 - val_loss: 0.6992 - val_accuracy: 0.7633
Epoch 13/20
1563/1563 [==============================] - 73s 47ms/step - loss: 0.6552 - accuracy: 0.7679 - val_loss: 0.6748 - val_accuracy: 0.7699
Epoch 14/20
1563/1563 [==============================] - 73s 47ms/step - loss: 0.6363 - accuracy: 0.7753 - val_loss: 0.6860 - val_accuracy: 0.7702
Epoch 15/20
1563/1563 [==============================] - 75s 48ms/step - loss: 0.6310 - accuracy: 0.7757 - val_loss: 0.6770 - val_accuracy: 0.7709
Epoch 16/20
1563/1563 [==============================] - 73s 47ms/step - loss: 0.6178 - accuracy: 0.7831 - val_loss: 0.6689 - val_accuracy: 0.7739
Epoch 17/20
1563/1563 [==============================] - 73s 47ms/step - loss: 0.6087 - accuracy: 0.7856 - val_loss: 0.6818 - val_accuracy: 0.7692
Epoch 18/20
1563/1563 [==============================] - 73s 47ms/step - loss: 0.5977 - accuracy: 0.7901 - val_loss: 0.6706 - val_accuracy: 0.7797
Epoch 19/20
1563/1563 [==============================] - 72s 46ms/step - loss: 0.5826 - accuracy: 0.7963 - val_loss: 0.6848 - val_accuracy: 0.7796
Epoch 20/20
1563/1563 [==============================] - 72s 46ms/step - loss: 0.5728 - accuracy: 0.7976 - val_loss: 0.6707 - val_accuracy: 0.7820
313/313 - 2s - loss: 0.6707 - accuracy: 0.7820 - 2s/epoch - 8ms/step

Test accuracy: 0.7820000052452087
313/313 [==============================] - 3s 9ms/step
PS C:\Users\Jasmine\Desktop\TenthSemester\AI\second-project>
```

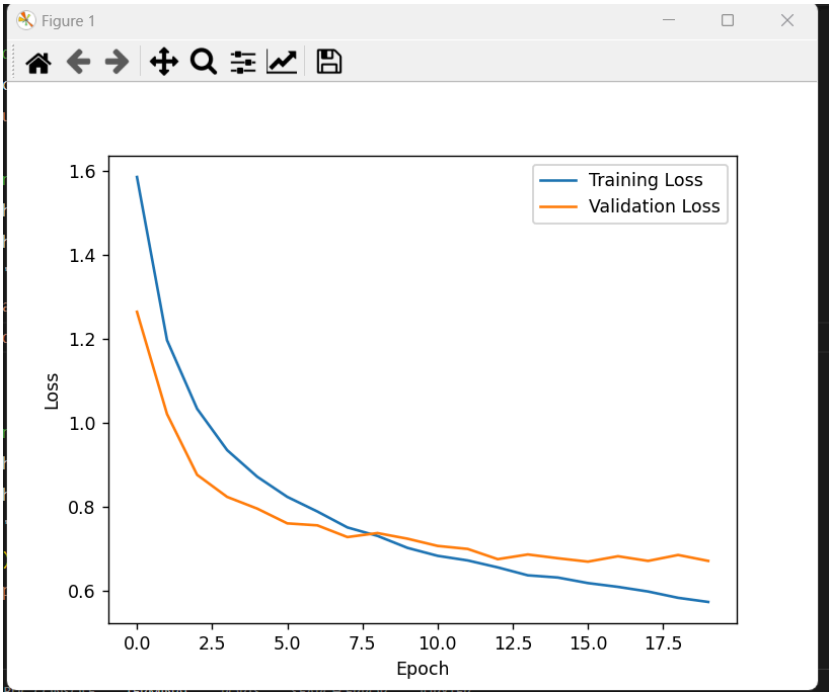


نمودار دقت (Accuracy)

دقت آموزشی (Training Accuracy) و دقت اعتبارسنجی (Validation Accuracy):

دقت آموزشی با گذشت زمان به طور پیوسته افزایش می‌یابد، که نشان‌دهنده بهبود توانایی مدل در شناسایی و طبقه‌بندی تصاویر داده آموزشی است.

دقت اعتبارسنجی نیز به طور مشابه افزایش می‌یابد، اما نرخ افزایش پس از حدود 10 دوره (epoch) کمی آهسته‌تر می‌شود. این می‌تواند نشان‌دهنده این باشد که مدل شروع به دستیابی به حد نهایی خود می‌کند و یا که داده‌های اعتبارسنجی برخی ویژگی‌هایی را دارند که مدل به خوبی آموزش دیده برای آنها تطبیق پیدا نکرده است.



نمودار زیان (Loss)

زیان آموزشی (Training Loss) و زیان اعتبارسنجی (Validation Loss):

هر دو معیار زیان با گذشت زمان کاهش می‌یابند، که نشان‌دهنده بهبود عملکرد مدل است. این کاهش زیان به خصوص در دوره‌های اولیه سریع‌تر است.

زیان اعتبارسنجی نیز کاهش می‌یابد اما مشابه با نمودار دقت، نرخ کاهش پس از حدود 10 دوره کمتر شده و بیشتر نوسان دارد. این می‌تواند نشان‌دهنده بیش‌برازش باشد، جایی که مدل بیش از حد به داده‌های آموزشی خاص وابسته شده و توانایی تعمیم به داده‌های جدید را ندارد.

نتیجه گیری:

از نمایش تصاویر و برچسب‌های پیش‌بینی شده، می‌توان دید که مدل تا حد زیادی موفق به شناسایی صحیح دسته‌های تصاویر شده است. این نشان‌دهنده این است که مدل کارایی خوبی در آموزش و اعتبارسنجی دارد، با این حال ممکن است اندکی بیش‌برازش رخ داده باشد که می‌تواند با تنظیمات بیشتر روی پارامترهای مدل مانند نرخ یادگیری یا استفاده از تکنیک‌های منظم‌سازی بهتر کنترل شود.

تکنیک های image generation:

تکنیک‌های تولید تصویر (Image Generation) در یادگیری عمیق، به خصوص در پردازش تصاویر، اغلب برای افزایش دقت مدل‌ها و بهبود توانایی تعمیم آن‌ها استفاده می‌شوند. این تکنیک‌ها با ایجاد تنوع در داده‌های آموزشی، به مدل کمک می‌کنند تا ویژگی‌های مختلف تصاویر را در شرایط متفاوت بهتر یاد بگیرند.

1. Data Augmentation

افزایش داده یا Data Augmentation یکی از رایج‌ترین روش‌ها برای افزایش کمیت و کیفیت داده‌های آموزشی بدون نیاز به جمع‌آوری داده‌های جدید است. این روش شامل تغییراتی در تصاویر موجود است که می‌تواند شامل موارد زیر باشد:

- چرخش (Rotation): چرخاندن تصاویر به میزان تصادفی در یک بازه مشخص، مانند 0 تا 30 درجه.
- تغییر مقیاس (Scaling): تغییر اندازه تصاویر برای آموزش مدل در شناسایی اشیاء در اندازه‌های مختلف.
- انتقال (Translation): جابجایی تصویر در جهات مختلف.
- بازتاب (Flipping): وارونه کردن تصاویر افقی یا عمودی.
- برش (Cropping): برداشتن بخش‌هایی از تصویر.
- تغییر روشنایی و کنتراست: تغییر در نور و کنتراست تصاویر برای آموزش مدل با شرایط نوری متفاوت.

2. Generative Adversarial Networks (GANs)

شبکه‌های مولد تخاصمی (GANs) می‌توانند برای تولید تصاویر جدید آموزش دیده شوند که از داده‌های واقعی قابل تمیز دادن نیستند. این شبکه‌ها معمولا شامل دو بخش هستند: یک مولد که تصاویر جدید تولید می‌کند و یک تمیزدهنده که تلاش می‌کند تصاویر واقعی را از تصاویر تولیدی تشخیص دهد. استفاده از GANs می‌تواند به تولید داده‌های آموزشی بیشتر و تنوع بخشیدن به داده‌های موجود کمک کند.

بخش چهارم:

کد زیر تصاویر را از پوشه original images با لیبل هرعکس(که نام فایل آن است) میخواند و در یک آرایه میریزد.

همچنین توجه داشته باشید که برای ساده سازی تصاویر کوچکتر شده و آن ها را flatten میکنیم:

```python
def load_images(dataset_folder):
    images = []
    labels = []
    for subdir, _, files in os.walk(os.path.join(dataset_folder, "Faces")):
        for file in files:
            filepath = os.path.join(subdir, file)
            label = os.path.splitext(os.path.basename(filepath))[0]
            label = re.sub(r'_\d+', '', label)
            image = imread(filepath, as_gray=True)
            image = resize(image, (50, 50))  # Resize the image
            images.append(image.flatten())  # Flatten the image
            labels.append(label)
    # Convert labels to numeric values
    # label_encoder = LabelEncoder()
    # encoded_labels = label_encoder.fit_transform(labels)
    return np.array(images), np.array(labels)
```

سپس داده ها را از فایل csv داده شده میخوانیم و مانند قسمت های قبلی مدل شبکه عصبی را train و test میکنیم.

```python
# Function to load labels from CSV file
def load_labels(csv_file):
    labels_df = pd.read_csv(csv_file)
    return labels_df['label'].values


# Path to the dataset folder and CSV file
dataset_folder = "C://Users//Jasmine//Desktop//TenthSemester//AI//second-project//q4//dataset//Faces"
csv_file = "C://Users//Jasmine//Desktop//TenthSemester//AI//second-project//q4//dataset//Dataset.csv"

# Load images and labels
X, y = load_images(dataset_folder)
labels = load_labels(csv_file)
# print("X\n", X)
# print("Y\n", Y)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Initialize MLP classifier with regularization
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000)
```

در نهایت از برخی تکنینک های cross-validation هم استفاده میکنیم.

:cv=5

این پارامتر مشخص می‌کند که باید از اعتبارسنجی متقابل 5 تایی استفاده شود. در اعتبارسنجی متقابل 5 تایی، داده‌های آموزشی (X_train و y_train) به 5 مجموعه کوچک‌تر یا "fold" تقسیم می‌شوند. مدل پنج بار آموزش داده می‌شود، هر بار با استفاده از 4 از 5 fold به عنوان مجموعه آموزشی و fold باقی‌مانده به عنوان مجموعه آزمایشی برای ارزیابی عملکرد مدل. این فرآیند به ارزیابی اینکه مدل چگونه احتمالاً روی داده‌های دیده نشده عمل می‌کند کمک می‌کند.

:(...)cross_val_score

این تابع یک امتیاز را از طریق اعتبارسنجی متقابل ارزیابی می‌کند. این تابع آرایه‌ای از امتیازات تخمین‌زننده (در این مورد mlp_classifier) را برای هر دور از اعتبارسنجی متقابل برمی‌گرداند. به طور پیش‌فرض، معیار امتیازدهی که استفاده می‌شود، معیار پیش‌فرض برای تخمین‌زننده است که معمولاً برای طبقه‌بندان، امتیاز دقت است.

:cv_scores

این متغیر آرایه‌ای از امتیازات بازگشتی توسط تابع cross_val_score را ذخیره می‌کند. این امتیازات به شما اجازه می‌دهند تا ارزیابی کنید مدل چقدر خوب عمل می‌کند. امتیازات بالاتر معمولاً نشان‌دهنده مدل بهتری هستند، و تفاوت‌ها در امتیازات در میان دسته‌ها می‌تواند به حساسیت مدل به تقسیم‌بندی مجموعه داده‌ها اشاره داشته باشد.

در آخر نیز نتایج(درصد خطا و دقت و لیبل عکس ها) را نشان میدهیم:

```
# Perform cross-validation
cv_scores = cross_val_score(mlp_classifier, X_train, y_train, cv=5)

# Train the classifier
mlp_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = mlp_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Accuracy:", np.mean(cv_scores))
```

نتایج:

یاسمن توکلی:98522175

آزمایش اول:

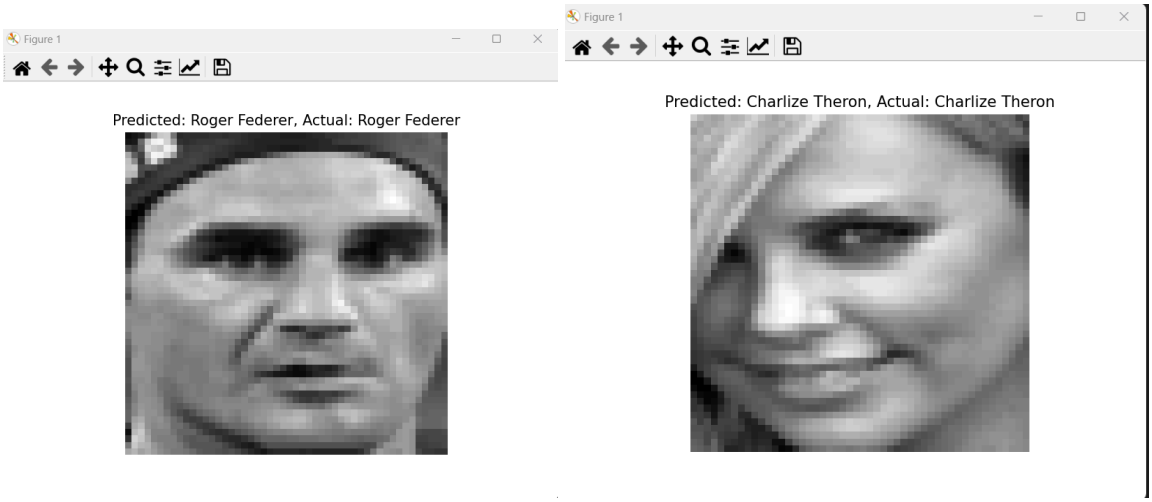
```
Test Accuracy: 0.5322896281800391
Classification Report:
                    precision    recall  f1-score   support

       Akshay Kumar       0.00      0.00      0.00         3
  Alexandra Daddario       0.62      0.59      0.61        17
         Alia Bhatt       0.54      0.68      0.60        19
   Amitabh Bachchan       0.93      0.78      0.85        18
      Andy Samberg       0.65      0.73      0.69        15
    Anushka Sharma       0.12      0.06      0.08        17
      Billie Eilish       0.40      0.24      0.30        25
         Brad Pitt       0.38      0.38      0.38        21
     Camila Cabello       0.50      0.60      0.55        10
    Charlize Theron       0.47      0.44      0.45        16
       Claire Holt       0.78      0.67      0.72        21
      Courtney Cox       0.60      0.50      0.55        12
     Dwayne Johnson       0.67      0.50      0.57        12
    Elizabeth Olsen       0.80      0.67      0.73        18
    Ellen Degeneres       0.60      0.80      0.69        15
       Henry Cavill       0.46      0.58      0.51        19
    Hrithik Roshan       0.38      0.50      0.43        20
       Hugh Jackman       0.79      0.55      0.65        20
      Jessica Alba       0.53      0.36      0.43        22
            Kashyap       0.57      0.50      0.53         8
       Lisa Kudrow       0.50      0.73      0.59        11
     Margot Robbie       0.33      0.45      0.38        11
            Marmik       0.50      0.75      0.60         8
    Natalie Portman       0.48      0.65      0.55        20
    Priyanka Chopra       0.67      0.65      0.66        31
   Robert Downey Jr       0.37      0.45      0.41        22
     Roger Federer       0.50      0.38      0.43        13
         Tom Cruise       0.57      0.57      0.57        14
  Vijay Deverakonda       0.62      0.64      0.63        28
```

آزمایش دوم:

```
  Vijay Deverakonda       0.62      0.64      0.63        28
       Virat Kohli       0.50      0.18      0.27        11
         Zac Efron       0.41      0.50      0.45        14

           accuracy                           0.53       511
          macro avg       0.52      0.52      0.51       511
       weighted avg       0.54      0.53      0.53       511

Cross-Validation Scores: [0.56968215 0.46568627 0.57352941 0.46078431 0.53676471]
Mean CV Accuracy: 0.521289371494319
```

لیبل های درست:

یاسمن توکلی:98522175



لیبل های نادرست: