- نیکی مجیدی فر 98522382
- تمرین پنجم درس بینایی کامپیوتر

سوال دوم)

همان طور که میدانیم، می توان اشکال را با کمک معیار های مناسب دسته بندی کرد. در این نمونه ویژگی های هندسی اهمیت دارند. طبق گفته های استاد، سه توصیف گر فشردگی، چگالش ، گریز از مرکز می تواند توصیف گر خوبی برای مقایسه ی شکل هندسی اشکال باشد. در نتیجه برای مقایسه و دسته بندی اشکال از این سه ویژگی استفاده خواهیم کرد.

مراحل:

1- پیاده سازی توصیف گر ها

سه توصیف گر به ترتیب فشردگی، چگالش و گریز ار مرکز(فرمول ها طبق اسلاید)

```
def compactness(contour):
    P = cv2.arcLength(contour, True)
    area= cv2.contourArea(contour)
    P2 = np.power(P, 2)
    output = float((4 * np.pi * area) /P2)
    return output
```

```
def solidity(contour):
    area = cv2.contourArea(contour)
    hull = cv2.convexHull(contour)
    hull_area = cv2.contourArea(hull)
    output = float(area)/hull_area
    return output
```

```
def eccuntricity(contour):
    ratio = 1
    if len(contour) >= 5:
        (x,y),(MA,ma),angle = cv2.fitEllipse(contour)
        ratio =  np.power((1-(ma/np.power(MA , 2)) ) , 0.5)
    return ratio
```

2- پیاده سازی تابع محاسبه گر فاصله(اختلاف) بین دو شکل و برگرداندن یک عدد تحت عنوان فاصله دو شکل.

در این روش از روش محاسبه Manhattan استفاده کردیم. با این تفاوت که توان اختلاف فشردگی را برای تاثیر بیشتر 1 قرار دادیم. اختلاف بین تمام پارامتر ها را به توان dرسانده و با هم جمع می کنیم.

```
def distance_criteria(x,y):
    output = abs(np.power(x[0] - y[0] ,1)) + np.power(x[1] - y[1],2) +
np.power(x[2] - y[2],2)
    return np.power(output , 1)
```

3- محاسبه ی تمام کانتور ها در تصویر(پیاده سازی شده اولیه)

4- تابع گروه بندی و رنگ آمیزی اشکال در یک دسته(پیاده سازی اولیه)

5- محاسبه ی یک nparray برای تمام ویژگی های کانتور:

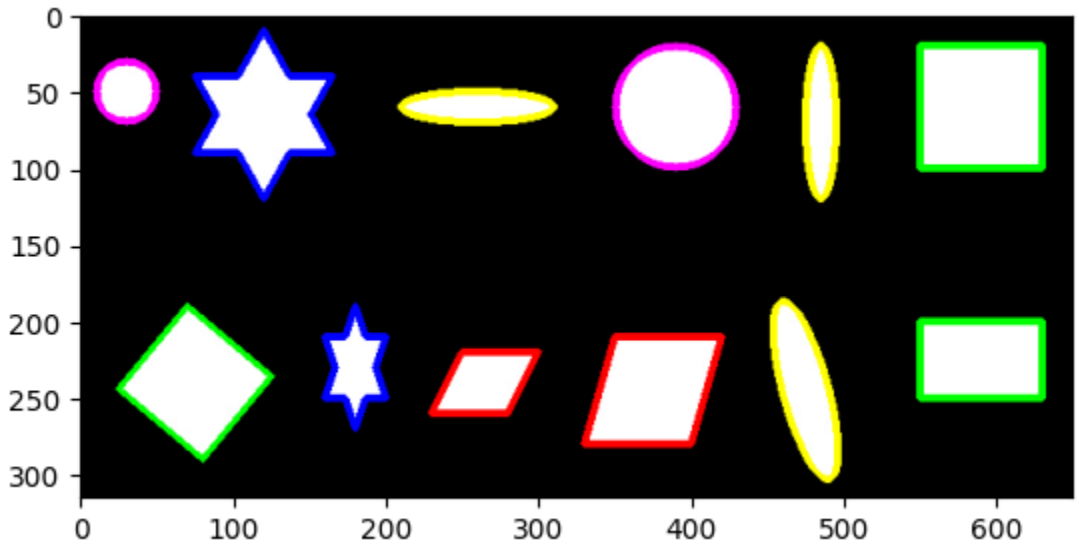
طبق خواسته ی سوال، باید برای هر کانتور شنایایی شده، یک لیست از تمام ویژگی هایش بسازیم. برای این کار یک nparray دو بعدی تعریف کردیم و در هر ایندکس برای هر کانتور،تمام توصیف گر هایش را محاسبه و قرار دادیم

```
def calculate_features(contours):
   n = len(contours)
   features = np.zeros((n , 3))
   for i, cnt in enumerate(contours):
      features[i , 0] = compactness(cnt)
      features[i ,1] = solidity(cnt)
      features[i,2] = eccuntricity(cnt)
   return features
```

6- فراخوانی و پیاده سازی الگوریتم پیاده سازی شده:

```
7- # You should replace your features array with None
8- colors = [(255,0,0),(0,255,0),(0,0,255),(255,255,0),(255,0,255)]
9- groups = grouping(calculate_features(contours=contours),0.08)
10-     painting(groups, contours, colors)
```

نتیجه ی حاصل :



** اشکال به خوبی طبقه بندی و جدا سازی شده اند.

سوال سوم)

شبکه‌های عصبی از توابع فعال سازی برای رفتار غیرخطی استفاده می کند. اگر فقط از مدل خطی استفاده شود، شبکه مانند یک مدل خطی خواهد بود و قادر به یادگیری رفتارهای پیچیده نیست. با استفاده از توابعی مثل ReLU و sigmond شبکه قادر به یادگیری رفتارهای پیچیده‌تر خواهد بود. توابع فعال‌سازی می‌توانند روند بهتر انتقال داده را در شبکه فراهم کنند. با استفاده از آنها شبکه می‌تواند ویژگی‌های ورودی را به صورت مناسب برای استفاده در لایه‌های بعدی تبدیل کند.(ReLU نویزها را حذف می‌کند و ویژگی‌های مهم را به لایه‌ی بعدی منتقل می‌کند). همچنین توابع فعال‌سازی باعث سهولت در یادگیری می‌شوند. مثلا ReLU با ارائه نمودار ساده، یادگیری مدل را آسان‌تر و سریع‌تر می‌کند.

1. تابع فعال‌سازی سیگموئید (sigmoid)

تابع سیگموئید یک تابع غیرخطی است که با توجه به محدوده مقادیر ورودی، مقدار خروجی را در بازه (0، 1) تقسیم می‌کند.از مزایای آن همین است که مقدار خروجی را در بازه (1.0) تقسیم می‌کند که می تواند توصیف احتمالی برای خروجی شبکه فراهم کند. همچنین برای مسائل با دو کلاس استفاده می شود. از معایب آن باعث می‌شود در شبکه‌های عمیق گرادیان کاهش یابد و عملکرد یاذپدگیری آن نیز کاهش یابد.

2. تابع فعال‌سازی tanh

تابع تانژانت هایپربولیک نیز یک تابع غیرخطی است و مقدار خروجی را در بازه (-1، 1) تقسیم می‌کند. از مزایای آن همین است که خروجی تابع در بازه (-1، 1) قرار دارد که می‌تواند منجر به توزیع خروجی‌های بیشتری در دامنه مقادیر مختلف شود. همچنین همانطور که گفته شد تابع غیر خطی است و می‌تواند الگوهای پیچیده را نیز یاد بگیرد. اما این تابع در حالت نزولی و صعودی خود دارای روند خطی شدن می‌باشد که ممکن است منجر به مشکل کاهش گرادیان شود.

3. تابع فعال‌سازی ReLU:

ReLU تابعی غیرخطی است که مقدار صفر را برای ورودی‌های منفی و خروجی برابر با خود ورودی را برای ورودی‌های مثبت قرار می‌دهد. این تابع ساده و دارای محاسبه کمتر است. معمولا در شبکه‌های عصبی عمیق خوب عمل می‌کند . از مشکل کاهش گرادیان جلوگیری می‌کند. اما مقدار ثابت خروجی برای ورودی‌های مثبت و 0 برای منفی‌ها از عوامل محدودیت‌ساز این تابع است.

4. تابع فعال‌سازی PReLU:

تابع PReLU یا Parametric ReLU به شکل مشابهی با ReLU عمل می‌کند، اما با اضافه کردن یک پارامتر قابل یادگیری به تابع، امکان جابه‌جایی خطی بین منفی‌ها را نیز فراهم می‌کند. این پارامتر با شبکه عصبی آموزش می‌بیند و برای هر نرون مجزا قابل یادگیری است. اما اضافه کردن پارامترها می‌تواند باعث پیچیدگی مدل شود.

سوال چهارم)

ابتدا طبق خواسته ی سوال دیتا ست را با کمک لینک های داده شده import می کنیم و آن ها را برای استفاده در شبکه ی عصبی پیش رو کمی تغییر می دهیم.(برااای مثال داده های ما 3 بعد دارد(بعد تعداد و بعد طول و عرض تصویر))

برای ورودی قابل قبول به شبکه ی عصبی، آن را FLATTمی کنیم و شکل آن را به 784 پیکسل تبدیل می کنیم.

همان طور که در بالا هم دیده شد، دیتا هایمان را در بازه 0-1 قرار می دهیم.

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
assert x_train.shape == (60000, 28, 28)
assert x_test.shape == (10000, 28, 28)
assert y_train.shape == (60000,)
assert y_test.shape == (10000,)
```

```
train_data = x_train.reshape((60000, 28 * 28))
test_data = x_test.reshape((10000, 28 * 28))
train_data = train_data/ 255
test_data = test_data / 255
```

پیاده سازی مدل : sequential

یک مل سه لایه با یک لایه پنهانی و 128 نورون. لایه ی ورودی ، دیتا هایمان و خروجی بسته به هرر عدد یک نورن یعنی به طور کلی 10 نورون نیز درخروجب پیاده سازی شده است .

مدلمان را طبق مراحل طی شده ، یک تابع optimizer و یک تابع loss تعریف می کنیم و با تعدادی ایپوک خاص آن را ترین می کنیم.

پیاده سازی مدل :

```
#define a simple sequential model
Seqmodel = keras.models.Sequential([
    keras.layers.Input(shape=(None, 784)),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
```

```
#optimizer and loss function
Seqmodel.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentr
opy(from_logits=True), metrics=['accuracy'])
```

```
#training
history = Seqmodel.fit(train_data, y_train, epochs=10)
```

پیاده سازی مدل functional :

- model deifinition

```
from keras import layers
inputs = keras.Input(shape=(784,))
dense = layers.Dense(128, activation="relu")
x = dense(inputs)
outputs = layers.Dense(10 , activation = 'softmax')(x)
#define model
modelF = keras.Model(inputs=inputs, outputs=outputs, name="mnist_model")
```

- model optimizer and loss function

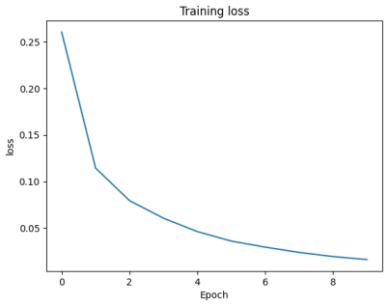
```
#compile model
modelF.compile(optimizer='adam',
               loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logi
ts=True),
               metrics=['accuracy'])
```

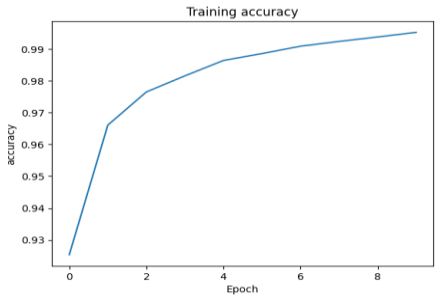
- model training in 10 epochs

```
historyF = modelF.fit(train_data,y_train, epochs=10)
```

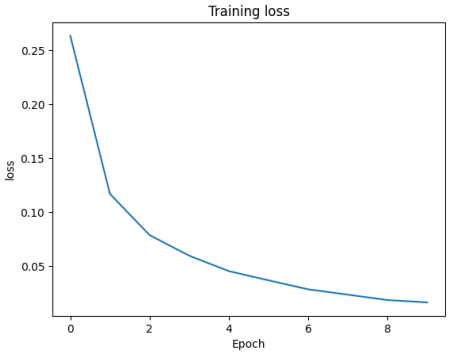
plots difference:

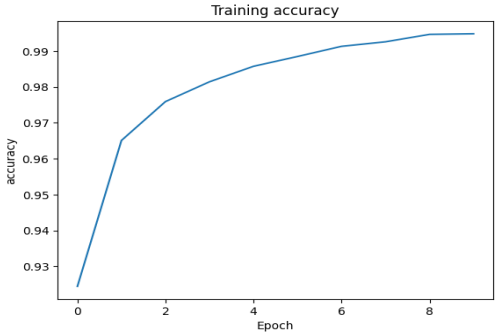
seq :





Functional :





همان طور که دیده می شود با تخمین خوبی شبیه هستند.

نتیجه trainدیتا روی data-tests :

Sequentional:

```
test_loss, test_acc = Seqmodel.evaluate(test_data ,y_test, verbose=2)
```

313/313 - 2s - loss: 0.0838 - accuracy: 0.9774 - 2s/epoch - 5ms/step

Functional speed:

```
test_loss, test_acc = modelF.evaluate(test_data ,y_test, verbose=2)
```

313/313 - 1s - loss: 0.0900 - accuracy: 0.9771 - 867ms/epoch - 3ms/step

همان طور که مشاهده می شود train, loss هر دو مدل به صورت تقریبا یک شکل است.

پیش بینی:(prerdictions)

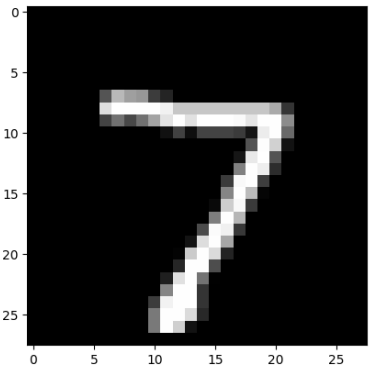
حال مانند نمونه ای که در نوت بوک اورده شده ، برای هر یک از داده ها یک داده ی تست را برای پیش بینی به مدل می دهیم و انتظار داریم بالا ترین احتمال کلاس برای نتیجه ی اصلی باسد.

Sequentional:

```
y_predseq = Seqmodel.predict(test_data)
print(y_predseq[0])
```

```
plt.imshow(x_test[0],cmap='gray')
label = np.argmax(y_predseq[0])
print(f'label : {label} , class = {y_test[0]}')
```

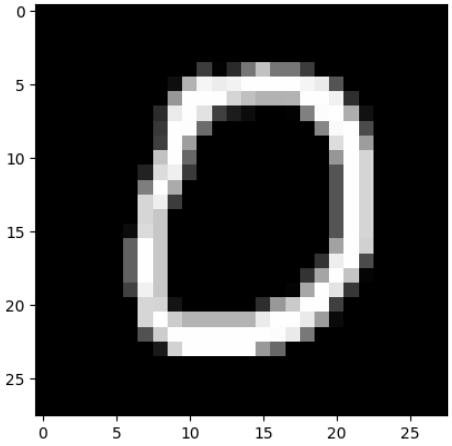
label : 7 , class = 7



Functional :

```
y_predfunc = modelF.predict(test_data)
print(y_predfunc[10])
```

```
plt.imshow(x_test[10],cmap='gray')
label = np.argmax(y_predfunc[10])
print(f'label : {label} , class = {y_test[10]}')
```



Result: label : 0 , class = 0

سوال پنجم)

در مدل sequential، استفاده از آن بسیار ساده و آسان است. اما اشتراک گذاری لایه ها یا انشعاب لایه ها مجاز نیست. همچنین، نمی توانید چندین ورودی یا خروجی داشته باشید.

functionalنسبت به sequential انعطاف پذیرتر است. اما قدرتمندتر از API متوالی است زیرا در اینجا انشعاب یا اشتراک گذاری لایه ها مجاز است. و همچنین می تواند چندین ورودی و خروجی داشته باشد.

در نتیجه می توانیم نتیجه بگیریم که به علت قدرتمند بودن fumctional، مدل sequentialبا یک سری محدودیت ها مواجه است و در نتیجه نمی توان هر مدل فانکشنال را به صورت sequentialپیاده کرد اما به صورت برعکس ممکن است.

سوال ششم)

الف)

Convolve(7*7 , kernel = 7*7)

In each channel: the result will be one pixel

Result : 3* (1 convolved pixel)

در یک مرحله انجام می شود و نتیجه سه کانال 3*1 است.

ب)

کرمل 3*3 است و در هر کانال سه مرتبه کانولوشن گرفته می شود.

اندازه تضویر در هر بار کانولوشن به ترتیب کوچک تر می شود (به اندزه padding)

1- convolve(7*7 image , kernel -3*3) = => 5*5 out put
2- convolve(5*5 image , kernel, 3*3) = => 3*3 out put
3- convolve(3*3 image , kernel, 3*3) = => `1*1 out put

همان طور که مشخص است، در هر کانال یک تصویر ا*1 باقی میماند در نتیجه 3*1*

پ)

خطی/ غیرخطی:

در حالت اول خطی تر است و میزان محاسبات کم تر است. و در یک مرحله انجام می پذیرد اما در حالت دوم محاسبات بیشتر است و چند مرحله ای است و غیر خطی تر است .

تعداد پارامتر و ویژگی :

در مدل اول به علت یک بارکانواو شدن، پیچبدگی شبکه کم تر است در نتیجه تعداد پارامتر و ویژگی های کم تری را به دست می آورد. اما در حالت دوم به علت چند بار کانوالو شدن و در هر مرحله یک سری فیچر و پارامتر های بیشتری مشخص میشود و کیفیت آن ها بهتر است.

عمق/سطح :

به طور واضح شبکه حالت دوم عمیق تر است و شبکه حالت اول سطحی تر است چرا که شبکه ی دوم چند مرحله ای است.

به طور کلی استفاده از شبکه ی دوم به طور کلی می توان ویژگی های بیشتری در اورد پس می تواند بهتر باشد.