تمرین پنجم بینایی

بنفشه قلی نژاد

سوال 1)

الف) در صورتی که از پدینگ استفاده نکنیم. تصویر خروجی کوچک تر می شود. (به اندازه ی (s//2 *2)) .

اما برای محاسبه و مقایسه بافت تصویر، از انجایی که یک الگوی تکرار شونده است ، نیازی به محاسبه ی پیکسل های دور تصوبر نیست و پیکسل های مرکزی تر تصویر الگوی اطراف را نیز مشخص میکنند اما اگر به طور جزئی مهم باشد، میتوان از یک پدینگ به اندازه 3//2 از هر طرف تصویر و پدینگ reflect اضافه کرد.

ب) روش lbp: در این روش، یک پنجره ی 3*3 را روی پیکسل مرکز قرار می دهیم و تمام پیکسل های همسایه مقدارشان با پیکسل مرکزی مقایسه می شود. در صورتی که مقدار آن ها بزرگ تر و یا مساوی باشد کد 1 ودر غیر این صورت کد 0 تعلق می گیرد. در نهایت این کد 8 تایی به یک رقم باینزی بین صفر تا 255 قرار می گیرد.

10	10	10	250	250	250
10	10	10	250	250	250
10	10	10	250	250	250
10	10	10	250	250	250
10	10	10	250	250	250

11111111	11111111	01111100	11111111
11111111	11111111	01111100	11111111
11111111	11111111	01111100	11111111

نتیجه :

255	255	124	255
255	255	124	255
255	255	124	255

سوال دوم)

الف)از منابع موجود در سوال استفاده کردیم و 5 توصیف گر را برای شکل تعریف کردیم.

- فشردگی(compactness)

```
Area = area(contour)
  Perimeter = perimeter(contour)
  output = float(4 * math.pi * Area) /( Perimeter)**2
  return output
```

- Aspect ratio

```
def aspect_ratio(contour):
    x,y,w,h = cv2.boundingRect(contour)
    return  float(w)/h
```

- Extent

```
def extent(cnt):
    area = cv2.contourArea(cnt)
    x,y,w,h = cv2.boundingRect(cnt)
    rect_area = w*h
    extent = float(area)/rect_area
    return extent
```

- Solidity (چگالش)

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

- گریز از مرکز

```
def eccentricity(contour):
    if len(contour) > 4:
        (x,y),(MA,ma),angle = cv2.fitEllipse(contour)
        return (1-(ma/MA **2) )** 0.5
    else:
        return 1
```

ب) distance criteria دراین مرحله، باید با کمک بردار ویژگی هر دو شکل به صورت دوبه دو، یک عدد را به غنوان تفاوت و تمایز شکل معرفی کنیم. طبق مقایسه انجام شده و توضیحات استاد سه ویژگی compsctess، solidity، eccentricity می توانند معیار خوبی برای تشخیص اشکال هندسی باشند و همچنین معیاری مستقل از چرخش هستند. بنابر این می توایم از جمع اختلاف این 3 معیار به عنوان عدد نهایی برای تمایز تصویر ارایه کنیم. (برای گریز مرکز و تاثیر گذاری آن را به توان دو می رسانیم)

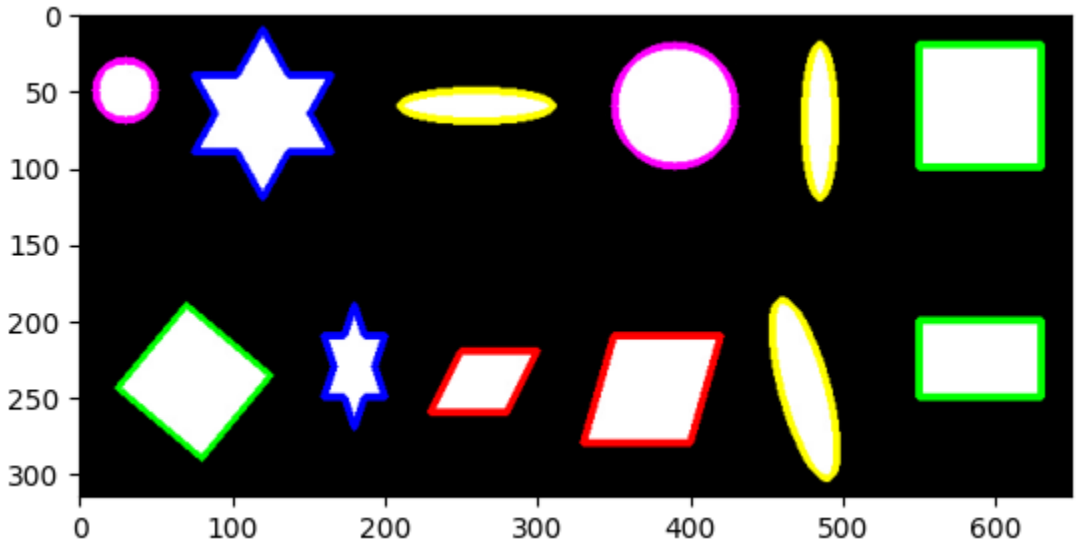
```
x1 , x2 , x3 ,x4, x5 =  x[0], x[1] , x[2] , x[3] , x[4]
    y1,y2,y3,y4,y5= y[0], y[1] , y[2] , y[3] , y[4]
    d1,d2,d3,d4,d5  =  abs(x1-y1), abs(x2-y2), abs(x3-y3), abs(x4-y4) ,
abs(y5-x5)
    ans =(d1) + (d4) +(d5**2)
    output= ans
```

قسمت گروه بندی و رنگ کردن تصاویر در یک گروه به صورت جدا پیاده سازی شده است. حال طبق توضیحات سوال تنها کافی است که بر روی تمام کانتور های شناسایی شده در شکل، یک فور بزنیم و به ازای تمام ویژگی ها برای هر کانتور یک ارایه بسازیم و هر ویژگی مربوط به هر شکل را در آن ذخیره کنیم.

```
features = np.zeros((len(contours) , 5) , dtype = np.float64)
i =0
for cnt in contours:
    d1 , d2 ,d3,d4,d5= descriptor1(cnt) , aspect_ratio(cnt) , extent(cnt) ,
solidity(cnt) , eccentricity(cnt)
```

```
    features[i][0] , features[i][1],  features[i][2], features[i][3],
features[i][4]  = d1 , d2 , d3 ,d4 ,d5
    i+=1
```

در نهایت آرایه مان را به تابع grouping می دهیم و نتیجه به صورت زیر است. (threshold : 0.09)



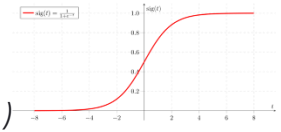
همان طور که مشخص است. فارغ از چرخش و اندازه اشکال به خوبی دسته بندی شده اند .

سوال 3)

یک تابع فعال سازی تصمیم می گیرد که آیا نورون باید فعال شود یا خیر. این بدان معناست که تصمیم خواهد گرفت که آیا ورودی نورون به شبکه در فرآیند پیش‌بینی با استفاده از عملیات ساده‌تر ریاضی مهم است یا خیر. نقش تابع فعالسازی استخراج خروجی از مجموعه مقادیر ورودی است که به یک گره (یا یک لایه) داده می شود.توابع فعال سازی با معرفی غیرخطی بودن نقشی جدایی ناپذیر در شبکه های عصبی ایفا می کنند. این غیرخطی بودن به شبکه‌های عصبی اجازه می‌دهد تا نمایش‌ها و توابع پیچیده‌ای را بر اساس ورودی‌هایی ایجاد کنند که با یک مدل رگرسیون خطی ساده امکان‌پذیر نیست.
تابع فعال سازی برای تولید یا تعریف یک خروجی خاص برای یک گره معین بر اساس ورودی ارائه شده استفاده می شود. به این معنی که تابع فعال سازی را روی نتایج جمع اعمال خواهیم کرد.
به طور کلی، توابع فعال سازی برای جلوگیری از خطی بودن ضروری هستند. بدون آنها، داده ها از طریق گره ها و لایه های شبکه تنها از طریق توابع خطی (a*x+b) عبور می کنند. ترکیب این توابع خطی دوباره یک تابع خطی است و بنابراین مهم نیست که داده ها از چند لایه عبور کنند، خروجی همیشه نتیجه یک تابع خطی است.

- **تابع sigmoid:**

تابع Sigmoid به عنوان تابع لجستیک شناخته می شود که به عادی سازی خروجی هر ورودی در محدوده بین 0 تا 1 کمک می کند. هدف اصلی تابع فعال سازی حفظ خروجی یا مقدار پیش بینی شده در محدوده خاص است که باعث بازدهی خوب و دقت مدل می شود

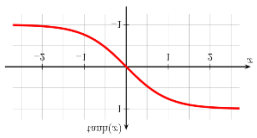


$$y = 1/(1+e^{(-x)})$$

در اینجا Y می تواند هر چیزی برای یک نورون بین محدوده –بی نهایت تا + بی نهایت باشد. بنابراین، ما باید خروجی خود را محدود کنیم تا پیش‌بینی مورد نظر یا نتایج تعمیم‌یافته را به دست آوریم. اشکال اصلی تابع فعال سازی سیگموئید ایجاد یک مشکل گرادیان ناپدید شدن است. این تابع فعال سازی مرکز غیر صفر است نرخ یادگیری مدل کند است یک مشکل گرادیان ناپدید کننده ایجاد میکند.

- تابع tanh:

تابع فعال سازی tanh نسبت به تابع فعال سازی سیگموئید برتری دارد زیرا دامنه این تابع فعال سازی از تابع فعال سازی سیگموئید بیشتر است. این تفاوت عمده بین تابع فعال سازی Sigmoid و Tanh است. عملکرد استراحت مانند تابع سیگموئید است که هر دو را می توان در شبکه فید فوروارد استفاده کرد.

$y=tanh(x)$ 

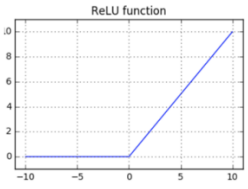
خروجی تابع فعال سازی tanh در مرکز صفر است. از این رو می‌توانیم به راحتی مقادیر خروجی را به صورت کاملاً منفی، خنثی یا به شدت مثبت ترسیم کنیم. معمولاً در لایه های پنهان شبکه عصبی استفاده می شود زیرا مقادیر آن بین –1 تا قرار دارد. بنابراین، میانگین لایه پنهان 0 یا بسیار نزدیک به آن است. این به متمرکز کردن داده ها کمک می کند و یادگیری لایه بعدی را بسیار آسان تر می کند.

اما همچنین با همان مشکل از بین رفتن گرادیان مانند یک تابع سیگموئید روبرو هستیم.

- تابع RelU :

ReLu بهترین و پیشرفته ترین تابع فعال سازی در حال حاضر در مقایسه با سیگموئید و TanH است زیرا تمام ایراداتی مانند مشکل گرادیان ناپدید شده به طور کامل در این تابع فعال سازی حذف شده است که این عملکرد فعال سازی را در مقایسه با سایر عملکردهای فعال سازی پیشرفته تر می کند.

محدوده: 0 تا بی نهایت معادله را می توان به صورت زیر ایجاد کرد:

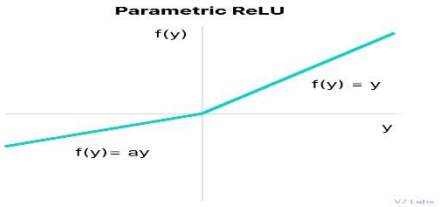


- { xi if x >=0
- 0 if x <=0 }

در اینجا تمام مقادیر منفی به 0 تبدیل می شوند، بنابراین هیچ مقدار منفی در دسترس نیست. مقادیر حداکثر آستانه Infinity هستند، بنابراین مشکلی در مورد گرادیان ناپدید وجود ندارد، بنابراین دقت پیش‌بینی خروجی و بازده حداکثر است. سرعت در مقایسه با سایر عملکردهای فعال سازی سریع است

- تابع Prelu:

پارامتریک relu نوع دیگری از ReLU است که هدف آن حل مشکل صفر شدن گرادیان برای نیمه چپ محور است.این تابع شیب قسمت منفی تابع را به عنوان آرگومان a ارائه می کند. با انجام پس انتشار، مناسب ترین مقدار a آموخته می شود.



fx = max(ax ,x)

جایی که "a" پارامتر شیب مقادیر منفی است. تابع ReLU پارامتر شده زمانی استفاده می‌شود که تابع ReLU نشت‌کننده هنوز در حل مشکل نورون‌های مرده شکست می‌خورد و اطلاعات مربوطه با موفقیت به لایه بعدی منتقل نمی‌شود. محدودیت این تابع این است که بسته به مقدار پارامتر شیب a ممکن است برای مسائل مختلف عملکرد متفاوتی داشته باشد.

سوال 4)

در این سوال از ما خواسته شده است که یک دیتا ست را به کمک دو مدل sequentialو functionalپیاده سازی کنیم.

قسمت اول)

دانلود و استفاده از دیتا ست MNIST

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

از آن جای که دیتا ست ما شامل 60000 عکس در قالب 28 * 28 است، آن را تبدیل به یک وکتور می کنیم و مقادیر آن را بین 0-1 می بریم تا هر پیکسل را به عنوان ورودی مدل در نظر بگیرد (FULL CONNECTED)

```
train_images = train_images.reshape((60000, 28 * 28))
test_images = test_images.reshape((10000, 28 * 28))
print(test_images.shape)
train_images = train_images / 255.
test_images = test_images.astype("float32") / 255
```

حال ابتدا، مدل sequentialخود را تعریف می کنیم. شبکه عصبی ما 784 نورون را به عنوان ورودی می گیرد. در لایه ی پنهان آن یک لایه ی dense با 256 نورون قرار می دهیم و اکتیویشن آن را نیز reluمی گذاریم. در نهایت خروجی ما نیز به علت 10 کلاسه بودن دیتا ست، شامل 10 نورون نهاییی است.

```
model = keras.models.Sequential([
    keras.layers.Input(shape=(None, 784)),
    keras.layers.Dense(256, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, None, 256)         200960

 dense_3 (Dense)             (None, None, 10)          2570
```

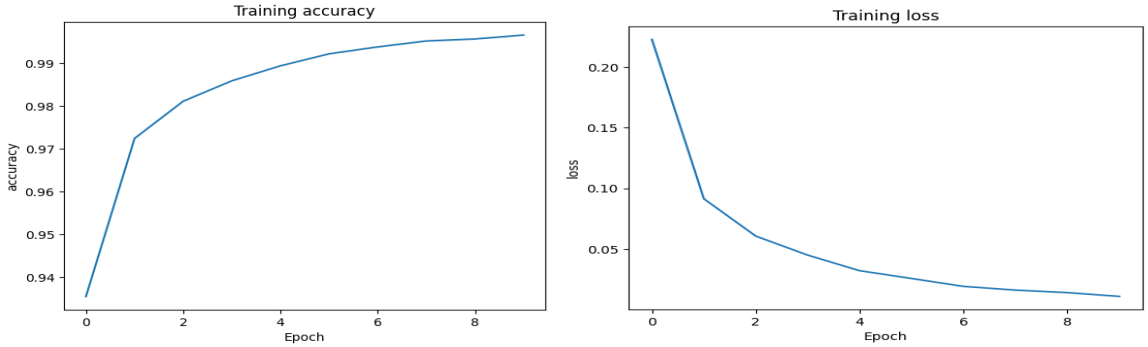
حال لازم است که توابع optimizer , loss را تعریف می کنیم که همانند نمونه ی قرار داده شده می گذازیم.

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

در نهایت، بعد از تعریف مدل ، آن را با 10 دور train می کنیم تا آموزش ببینند.

```
history = model.fit(train_images,train_labels, epochs=10)
```

نمودار accuracy , lossدر هر اپوک در فرایند یادگیری:



همان طور که مشاهده می شود، در هر دوره lossکم تر می شود و سرعت آموزش بالا می رود.

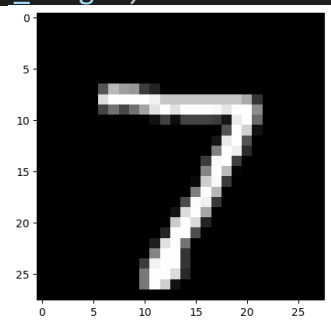
نتیجه بر روی تست:

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

```
313/313 - 1s - loss: 0.0757 - accuracy: 0.9795 - 958ms/epoch - 3ms/step
```

نتیجه ی پیش بینی شده توسط مدل:

```
y_pred_M = model.predict(test_images)
```



```
label : 7 , real number = 7
```

قسمت : functional model

مراحل بالا را عینا برای این مدل تعریف می کنیم.

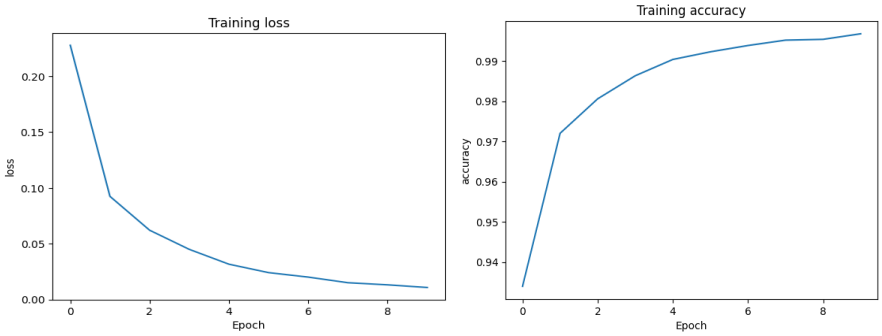
تفاوت آن در مدل در نحوه ی تعریف آن است.

```
inputs = keras.Input(shape=(784,))
img_inputs = keras.Input(shape=(32, 32, 3))
inputs.shape
dense = layers.Dense(256, activation="relu")
x = dense(inputs)
outputs = layers.Dense(10 , activation = 'softmax')(x)
model2 = keras.Model(inputs=inputs, outputs=outputs, name="mnist_model")
```

```
Model: "mnist_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense_2 (Dense)             (None, 256)               200960

 dense_3 (Dense)             (None, 10)                2570

=================================================================
Total params: 203,530
Trainable params: 203,530
```

مراحل کامپایل و ترین را عینا برای این مدل تکرار می کنیم.

نتیجه نمودار های loss , trainدر فرایند آموزش :



تست سرعت بر روی داده های تست:

```
313/313 - 1s - loss: 0.0758 - accuracy: 0.9807 - 684ms/epoch - 2ms/step
```

مرحله ی حدس نهایی :

```
label : 2 , real number = 2
```

همان طور که مشاهده میشود ، نتیجه برای هر دو حالت شبیه به هم و مطلوب است.

سوال 5)

Sequential network به شما این امکان را می دهد که مدل ها را لایه به لایه با چیدن آنها بسازید. از این جهت محدود است که به شما اجازه نمی‌دهد مدل‌هایی ایجاد کنید که لایه‌های مشترک داشته باشند یا ورودی یا خروجی‌های متعددی داشته باشند.

Functionalانعطاف پذیری بیشتری را فراهم می کند زیرا می توانید به راحتی مدل هایی را تعریف کنید که در آن لایه ها به بیش از لایه های قبلی و بعدی متصل می شوند و می توانید لایه ها را به هر لایه دیگری متصل کنید. در نتیجه می توانید شبکه های پیچیده ای مانند Residual Network ایجاد کنید.

هر شبکه ی seqمی تواند به صورت functionalپیاده سازی شود اما برعکس آن لزوما برقرار نیست.

ساخت مدل هایی با چند ورودی، چند خروجی یا لایه های اشتراکی دشوار است. برای شبکه هایی که نیاز به ادغام لایه ها، الحاق لایه ها، افزودن لایه ها دارند، انعطاف پذیر نیست.

سوال 6)

الف)

```
Channels = 3 , stride =1
```

از آن جایی که کرنل با تصویر هم انداره است، در هر کانال یک پیکسل باقی می ماند درنتیجه یک پیکسل در هر یک از این کانال ها خروجی کانولوشن ذکر شده است. 3 *1*1

ب) مرحله اول : 5*5*3 مرحله دوم : 3*3*3، مرحله سوم : 3 * 1 * 1

پ) در نهایت نتیجه هر دو عملگر ها یکسان است.

اگر بخواهیم که تعداد پارامتر هایمان زیاد نشوند، حالت اول حالت بهتری است زیرا فقط یکبار کانولوشن را اجرا میکند. اما در حالت دوم شبکه عمیق تری را می سازد، تعداد پارامتر ها بیشتر است اما به ازای آن تعداد ویژگی که به دست می آید بیشتر است و می تواند دقیق تر باشد. حالت اول نسبت به حالت دوم خطی تر و سطحی تر است اما حالت دوم غیر خطی تر است.

بستگی به وضعیت می تواند هر کدام به صرفه تر باشد اما به طور کلی نتیجه حالت دوم می تواند دقیق تر باشد.