HW5:

1)

«به نام خدا»

الف) بله زیرا برای پیکسل های گوشه ، (اطراف) منظور مروری به ۸ پیکسل در اطرافشان با فاصله 1 وجود ندارد

ب)

① Padding و LBP :
Reflect

۱۰	۱۰	۱۰	۱۰	۲۵۰	۲۵۰	۲۵۰	۲۵۰
۱۰	۱۰	۱۰	۱۰	۲۵۰	۲۵۰	۲۵۰	۲۵۰
۱۰	۱۰	۱۰	۱۰	۲۵۰	۲۵۰	۲۵۰	۲۵۰
۱۰	۱۰	۱۰	۱۰	۲۵۰	۲۵۰	۲۵۰	۲۵۰
۱۰	۱۰	۱۰	۱۰	۲۵۰	۲۵۰	۲۵۰	۲۵۰
۱۰	۱۰	۱۰	۱۰	۲۵۰	۲۵۰	۲۵۰	۲۵۰
۱۰	۱۰	۱۰	۱۰	۲۵۰	۲۵۰	۲۵۰	۲۵۰

در یکنواخت مستقل حالت ها را از ۰ تا ۸ در نظر می گیریم و روشن را یک در نظر می گیریم (سفید = ۱، سیاه = ۰)

کلاس ها را از پایین به بالا مطابق الگو از ۰ تا ۸ شماره گذاری می کنیم

۸	۸	۸	۵	۸	۸
۸	۸	۸	۵	۸	۸
۸	۸	۸	۵	۸	۸
۸	۸	۸	۵	۸	۸
۸	۸	۸	۵	۸	۸

2)

دو فیچر solidity , compactness را از تصویر استفاده می کنیم تا اشکال را تفکیک کنیم:

```
def solid(contour):
    '''
    You should implement one of the descriptors
    You can change name of this method.
    You can copy this cell and implement anothe
    You can create as many descriptor as you wa
    For more information, refer to https://docs
    input(s):
    contour (ndarray): contour of the shape
    output(s):
    output (float): computed feature value by a
    '''
    convHull=cv2.convexHull(contour,False)
    AreaHull=cv2.contourArea(convHull)
    AreaCountur= cv2.contourArea(contour)
    output= AreaCountur/AreaHull
    return output
```

```
def solid(contour):
    '''
    You should implement one of the descriptors
    You can change name of this method.
    You can copy this cell and implement anothe
    You can create as many descriptor as you wa
    For more information, refer to https://docs
    input(s):
    contour (ndarray): contour of the shape
    output(s):
    output (float): computed feature value by a
    '''
    convHull=cv2.convexHull(contour,False)
    AreaHull=cv2.contourArea(convHull)
    AreaCountur= cv2.contourArea(contour)
    output= AreaCountur/AreaHull
    return output
```

در این بخش ابتدا شکل محدب شکل اولیه را با تابع cv2.convexHull به دست اورده و در دو خط بعد با تابع conturArea مساحت ها را محاسبه و بر هم تقسیم میکنیم

```python
def compactmess(contour):
    '''
    You should implement one of the descriptors in this method
    You can change name of this method.
    You can copy this cell and implement another descriptor in
    You can create as many descriptor as you want.
    For more information, refer to https://docs.opencv.org/4.x
    input(s):
    contour (ndarray): contour of the shape
    output(s):
    output (float): computed feature value by applying the des
    '''
    AreaCountur= cv2.contourArea(contour)
    pcontoure= cv2.arcLength(contour,True)

    output=((4 *np.pi)* AreaCountur)/(pcontoure)**2
    return output
```

با فرمول موجود در اسلاید ها و محاسبه مساحت و محیط فشردگی را به دست می اوریم

برای پیاده سازی تابع eccentricity هم تلاش کردم ولی موفق نشدم و نیازی هم به این فیچر نبود

```python
def Eccentricity(contour):
    '''
    You should implement one of the descriptors in this method (compactness,
    You can change name of this method.
    You can copy this cell and implement another descriptor in next cell.
    You can create as many descriptor as you want.
    For more information, refer to https://docs.opencv.org/4.x/d1/d32/tutoria
    input(s):
    contour (ndarray): contour of the shape
    output(s):
    output (float): computed feature value by applying the descriptor on the
    '''
    if len(contour) >= 5:
        _, (major_axis, minor_axis), _ = cv2.fitEllipse(contour)
        eccentricity = np.sqrt(1 - (minor_axis / major_axis) ** 2)
    else:
        # Handle square separately
        eccentricity = 0.0
    return eccentricity
```

برای اینکه بفهمیم یک فیچر در دو تصویر بهم نزدیک است از تابع distance_creteria استفاده میکنیم که با تابع norm پیاده اش کردیم که می اید یک نوع نماینده یک ارایه را پیدا میکند تا اختلاف این عدد در دو ارایه میزان نزدیکی دو ارایه را نشان دهد

```python
def distance_criteria(x,y):
    """
    You should implement your distance criteria here.
    This method is used for comparing features of shapes.
    input(s):
    x (ndarray): feature vector of first shape with the shap
    y (ndarray): feature vector of second shape with the sha
    output(s):
    output (float): Distance between features of two shapes
    """
    normsX = np.linalg.norm(x)
    normsY = np.linalg.norm(y)
    return np.abs(normsX - normsY)
```

در نهایت یک ارایه برای ذخیره عدد فیچر های هر شکل ساختیم که ایندکس 0 انها فشردگی و ایندکس یک انها solidity انها است

```
features=np.ones(shape=(len(contours),2))
for i in range(len(contours)):
    features[i][0]= compactmess(contours[i])
    features[i][1]= solid(contours[i])
```

3)

توابع فعال سازی در شبکه های عصبی برای وارد کردن غیر خطی به محاسبات شبکه استفاده می شود. آنها به شبکه عصبی کمک می کنند تا روابط پیچیده بین ورودی ها و خروجی ها را بیاموزد و مدل کند. بدون توابع فعال‌سازی، یک شبکه عصبی اساساً یک مدل خطی خواهد بود که فقط می‌تواند روابط خطی را تقریبی کند و قادر به گرفتن الگوهای غیرخطی در داده‌ها نخواهد بود.

تابع فعال سازی sigmoid:

تابع سیگموئید که به عنوان تابع لجستیک نیز شناخته می شود، به صورت زیر تعریف می شود:

f(x) = 1 / (1 + exp(-x))

هر مقدار واقعی را به عنوان ورودی می گیرد و آن را در محدوده ای بین 0 و 1 scale می کند. منحنی S شکل دارد و برای تولید خروجی احتمال مانند استفاده می شود. در زمینه شبکه‌های عصبی، تابع سیگموئید در گذشته به‌عنوان تابع فعال‌سازی لایه‌های پنهان به‌طور گسترده مورد استفاده قرار می‌گرفت، اما به دلیل برخی محدودیت‌ها، استفاده از آن کاهش یافته است. یکی از اشکالات اصلی مشکل گرادیان ناپدید شدن است که می تواند روند آموزش شبکه های عمیق را مختل کند.

عملکرد فعال سازی Tanh:

تابع مماس هذلولی یا tanh به صورت زیر تعریف می شود:

f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x))

مشابه تابع سیگموئید، tanh همچنین ورودی را در محدوده ای بین -1 و 1 ترسیم می کند. در اطراف مبدا متقارن است و در مقایسه با تابع سیگموئید دارای شیب تندتری است. Tanh به موضوع خروجی های صفر-مرکز می پردازد و آن را برای آموزش شبکه های عصبی عمیق در مقایسه با تابع سیگموئید مناسب تر می کند.

تابع فعال سازی ReLU (واحد خطی اصلاح شده):

ReLU به صورت زیر تعریف می شود:

f(x) = max(0, x)

ReLU برای ورودی های منفی 0 برمی گرداند و به سادگی از ورودی های مثبت عبور می کند. این تابع به دلیل سادگی و کارایی، متداول ترین تابع فعال سازی در شبکه های عصبی مدرن است. همگرایی را در طول تمرین تسریع می کند. یکی از مزایای اصلی ReLU این است که مقادیر مثبت را اشباع نمی کند، که به شبکه اجازه می دهد تا به سرعت یاد بگیرد.

تابع فعال سازی پارامتریک (PRELU) ReLU:

PRELU توسعه تابع فعال سازی ReLU است که یک پارامتر قابل یادگیری را معرفی می کند. به این صورت تعریف می شود:

f(x) = max(0, x) + alpha * min(0, x)

در اینجا، آلفا یک پارامتر قابل یادگیری است که شیب قسمت منفی تابع را تعیین می کند. با اجازه دادن مقادیر منفی برای آلفا، PRELU می تواند عملکرد فعال سازی بهینه برای هر نورون را در طول تمرین یاد بگیرد. این سازگاری با ورودی‌های منفی می‌تواند در سناریوهایی که ReLU ممکن است منجر به نورون‌های مرده شود که هرگز فعال نمی‌شوند، سودمند باشد.

به طور خلاصه، توابع فعال‌سازی غیرخطی‌ها را به شبکه‌های عصبی وارد می‌کنند و آنها را قادر می‌سازد تا الگوهای پیچیده را یاد بگیرند. Sigmoid، tanh، ReLU و PRELU برخی از توابع فعال سازی رایج هستند که هر کدام ویژگی ها و مزایای خاص خود را دارند.

4)

ابتدا دیتا را میگیریم:

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

دیتا را در مرحله بعد اناده میکنیم :

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

در اینجا با توجه به ابعاد عکس ها . اینکه ورودی لایه بعد تک ستونه است reshape انجام میدهیم و مقادیر را بین 0و1 میاوریم

اینجا برای حالت sequential مانند مثال عمل میکنیم و مدل زیر را میسازیم:

```python
model = Sequential([
    Input(shape=(None, 784)),
    Dense(512, activation='relu'),
    Dense(10,activation='softmax')
])
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
model.summary()

hist = model.fit(train_images, train_labels, epochs=20, validation_split=0.2, batch_size=128)
```

MagicPython

برای مدل functional:

```python
inputs = keras.Input(shape=(784,))
```

```python
x = layers.Dense(512, activation="relu")(inputs)
outputs = layers.Dense(10,activation='softmax')(x)
```

```python
model = keras.Model(inputs=inputs, outputs=outputs, name="mnist_model")
```

```python
model.summary()
```

در این حالت میتوان لایه ها را مستقل تعریف کرد و انها را به شکل ورودی یکدیگر بهم داد در حالت بالا ابتدا ورودی را گرفته و به لایه های dense میدهیم و در اخر به خروجی و برای مدل ورودی و خروجی را معلوم میکنیم

از اینجا به بعد مشابه حالت قبل است

```
(variable) x_train: Any st, y_test) = keras.datasets.mnist.load_data()

x_train = x_train.reshape(60000, 784).astype("float32") / 255
x_test = x_test.reshape(10000, 784).astype("float32") / 255

model.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=keras.optimizers.RMSprop(),
    metrics=["accuracy"],
)

history = model.fit(x_train, y_train, batch_size=64, epochs=20, validation_split=0.2)
```

ابتدا دیتا را میگیریم و عملیات های لازم را انجام میدهیم و بعد مدل را کامپایل میکنیم و بعد مدل را اموزش میدهیم در epoch 20 و validation را 20 درصد دیتا ها در نظر میگیریم تا مدل را علاوه بر اموزش بیازماییم

5)

خیر زیرا ممکن نیست چون مدل functional قابلیت تعریف لایه های متعدد ومشترک را به ما میدهد به ضورت جدا از هم پس قابلیت استفاده در معماری های پیچیده تر را دارد و منعطف تر است در حالی که در sequential ما به راحتی نمیتوانیم این کار ها را بکنیم چون در واقع یک صف خطی از لایه ها است

6)

الف) نتیجه تضویری یک در یک است

ب) خروجی مانند حالت قبل یک در یک است(1*1) -> (3*3) <-(5و5) -> (7*7):

M’=M-n

ج) دومی چون در چند مرحله است به ویژگی های عمیق تری میرسد و در دومی چون سه بار تابع activation اجرا میشود به جای یک بار در نتیجه غیر خطی تر است

پارامتر های اولی : 148=1+3*7*7

زیرا یک bias داریم و 3 کانال در تصویر 7*7

پارامتر های دومی : چون کرنل 3*3 و در لایه اول 3 کانال و در بعدی ها یک کانال داریم و در هر یک یک عدد bias:
48=(3*3*3+1)+ (3*3*1+1) +(3*3*1+1)