داکیومنت تکلیف شماره ۵

■ سوال ۱

الف) در الگوریتم LBP ، هر نقطه در تصویر با همسایه‌ای از اطراف خود مقایسه می‌شود. برای محاسبه LBP ، نقاط همسایگی مستقیم برای هر پیکسل بررسی می‌شوند. به عنوان مثال در الگوی ۸ تایی با شعاع ۱، نقاط همسایگی بالا، پایین، چپ و راست، و همچنین نقاط همسایگی قطری مورد استفاده قرار می‌گیرند. بنابراین، اگر تصویر موردنظر به اندازه کافی بزرگ باشد و هیچ یک از نقاط همسایگی بر روی لبه‌های تصویر قرار نگیرند، می‌توان بدون استفاده از Padding عمل LBP را روی تصویر اعمال کرد. اما اگر لبه‌های تصویر نیاز به بررسی دارند، باید از Padding استفاده کنیم تا نقاط همسایگی لبه‌ها نیز در نظر گرفته شوند و اطلاعات آنها در محاسبه LBP لحاظ شود.

ب) برای حل این سوال، از تابع feature.local_binary_pattern استفاده خواهیم کرد. این تابع مقدار LBP را برای هر پیکسل محاسبه می‌کند، با مقایسه پیکسل مرکزی با پیکسل‌های اطراف و انتخاب ۰ یا ۱ براساس این مقایسه. اگر پیکسل مرکزی بیشتر یا مساوی با همسایگانش باشد، به آن جایگزین ۱ می‌شود و در غیر این صورت ۰.

در اینجا، با استفاده از این تابع، LBPبرای تصویر مورد نظر را محاسبه خواهیم کرد. اما به دلیل نبود همسایه‌هایی در مواضع خارج از تصویر، باید تصویر را با استفاده از Padding به اندازه کافی بزرگتر کنیم تا همسایه‌های لبه نیز در نظر گرفته شوند. بنابراین، برای حل این سوال، ابتدا تصویر را با استفاده از Padding گسترش می‌دهیم و سپس الگوریتم LBP را روی تصویر گسترده شده اعمال می‌کنیم.

```python
import numpy as np
from skimage import feature

image = np.array([
    [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]
])

image_gray = np.uint8(image)
res = feature.local_binary_pattern(image_gray, 8, 1, method='uniform')
print(res)
```

✓ 4.1s

```
[[3. 5. 6. 3. 5. 3.]
 [5. 8. 8. 5. 8. 5.]
 [5. 8. 8. 5. 8. 5.]
 [5. 8. 8. 5. 8. 5.]
 [3. 5. 6. 3. 5. 3.]]
```

■ سوال ۳

توابع فعال‌ساز (Activation functions) در شبکه‌های عصبی برای اضافه کردن قابلیت غیرخطی به نورون‌ها استفاده می‌شوند. این توابع، وظیفه‌ای مشابه عملگرهای غیرخطی در سیستم‌های غیرخطی دارند و به شبکه عصبی این امکان را می‌دهند تا الگوهای پیچیده‌تر را یاد بگیرند و توانایی بهتری در تقریب و پیش‌بینی داشته باشند.

۱. تابع فعالساز :Sigmoid تابع فعالساز sigmoid یک تابع غیرخطی است که مقادیر ورودی را به مقادیر بین ۰ و ۱ نگاشت می‌دهد. فرمول آن به صورت زیر است:

f(x) = 1 / (1 + e^(-x))

مزیت استفاده از تابع sigmoid این است که خروجی آن به صورت احتمال است، و به خاطر مقدار خروجی محدود بین ۰ و ۱، برای مسائلی مانند مسائل دسته‌بندی باینری مفید است. با این حال، مشکلی که در این تابع وجود دارد این است که برای ورودی‌های بزرگ، مشتق آن به صفر نزدیک می‌شود و این موجب می‌شود که در فرآیند بهینه‌سازی شبکه، گرادیان‌ها بسیار کوچک شده و مسئله گرادیان کاهشی ناپدید شود.

۲. تابع فعالساز Tanh :تابع فعالساز Tanh نیز مشابه تابع sigmoid عمل می‌کند، با این تفاوت که خروجی آن بین -۱ و ۱ قرار دارد. فرمول آن به صورت زیر است:

f(x) = (e^x - e^(-x)) / (e^x + e^(-x))

تابع Tanh نیز قابلیت غیرخطی را به شبکه می‌دهد و در برخی موارد، بهتر از تابع sigmoid عمل می‌کند. با این حال، همچنان با مشکل ناپدید شدن گرادیان روبرو است.

۳. تابع فعالساز ReLU: تابع فعالساز ReLU (Rectified Linear Unit) یک تابع غیرخطی است که مقادیر ورودی مثبت را بدون تغییر باقی می‌گذارد و مقادیر منفی را به صفر تبدیل می‌کند. فرمول آن به صورت زیر است

f(x) = max(0, x)

تابع ReLU به دلیل سادگی محاسباتی و عدم مشکل ناپدید شدن گرادیان، در بسیاری از شبکه‌های عصبی استفاده می‌شود و عملکرد خوبی در بسیاری از وظایف دارد. با این حال، مشکلی که در این تابع وجود دارد این است که واحدهایی که به صفر تبدیل می‌شوند، دیگر در فرآیند آموزش شبکه مشارکت نخواهند کرد و این موجب می‌شود که برخی از ویژگی‌ها و الگوها از دست روند.

۴. تابع فعالساز PReLU: تابع PReLU (Parametric Rectified Linear Unit) یک توسعه از تابع ReLU است و در آن پارامترهایی وجود دارند که به شبکه اجازه می‌دهند خروجی تابع ReLU را به طور خطی تغییر دهند برای مقادیر منفی. فرمول آن به صورت زیر است

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

در تابعPReLU ، پارامتر a قابل یادگیری است و توسط الگوریتم بهینه‌سازی بروزرسانی می‌شود. تابع PReLU از مزیت‌های تابع ReLU بهره می‌برد و در برخی موارد می‌تواند عملکرد بهتری نسبت به ReLU داشته باشد.

به طور خلاصه، هر یک از توابع فعالسازsigmoid ، Tanh، ReLUو PReLU ویژگی‌ها و مشکلات خاص خود را دارند. انتخاب تابع فعالساز به میزان وظیفه، ظرفیت محاسباتی، پیچیدگی داده‌ها و سایر عوامل بستگی دارد. امروزه، ReLUو توسعه‌های آن مانند PReLUبه طور گسترده در بسیاری از شبکه‌های عصبی استفاده می‌شوند به دلیل عملکرد خوب و سادگی محاسباتی که ارائه می‌کنند.

■ سوال ۵

نه، هر شبکه‌ای که به صورت **Functional** قابل پیاده‌سازی باشد، به صورت **Sequential** نیز قابل پیاده‌سازی نیست. دلیل این امر از ماهیت این دو روش پیاده‌سازی است.

متد **Sequential** در کتابخانه **Keras** یک مدل خطی است که لایه‌ها را به ترتیب یکی پس از دیگری اضافه می‌کند. در این متد، خروجی لایه قبلی به عنوان ورودی لایه بعدی استفاده می‌شود و ارتباط بین لایه‌ها به صورت یکجا (یعنی یک به یک) است. این متد مناسب برای شبکه‌هایی است که ساختار ساده و خطی دارند.

از سوی دیگر، متد **Functional** در **Keras** امکان ایجاد شبکه‌های با ساختارهای پیچیده‌تر را فراهم می‌کند. در این متد، می‌توان لایه‌ها را به صورت غیرخطی با چندین ورودی و خروجی پیاده‌سازی کرد و از توابع غیرخطی، شرطی و حلقه‌ها در معماری شبکه استفاده کرد. این متد مناسب برای ساختارهای پیچیده‌تر مانند شبکه‌های با دو یا چند شاخه و مدل‌های دارای اشتراک وزن است.

بنابراین، یک شبکه‌ای که به صورت **Functional** پیاده‌سازی شده است، ممکن است دارای ارتباطات غیرخطی و پیچیده باشد که در روش **Sequential** قابل پیاده‌سازی نیست. بنابراین، نمی‌توان هر شبکه‌ای که به صورت **Functional** قابل پیاده‌سازی است، به صورت **Sequential** نیز پیاده‌سازی کرد.

■ سوال ۶

output_size = [(input_size – kernel_size + 2 * padding) / stride] + 1

الف) اگر تصویر ۷ در ۷ را با یک کرنل ۷ در ۷ کانوالو کنیم، با توجه به فرمول **output_size** که ذکر شده، خروجی همچنان یک تصویر ۱ در ۱ خواهد بود.

ب) اگر تصویر ۷ در ۷ را با یک کرنل ۳ در ۳ کانوالو کنیم، با استفاده از فرمول مذکور، خروجی دارای ابعاد ۵ در ۵ خواهد بود. این به این معنی است که ابعاد تصویر با اعمال کانوالوشن و استفاده از کرنل ۳ در ۳ کاهش یافته است.به طور خلاصه طبق رابطه، ابعاد بعد از اعمال کرنل اول: پنج در پنج، ابعاد بعد از اعمال کرنل دوم: سه در سه و ابعاد بعد از اعمال کرنل سوم: یک در یک خواهد شد.

ج) کرنل ۷ در۷: در این حالت یک کرنل ۷ در ۷ داریم که بر روی یک تصویر سه کاناله اعمال می شود. همچنین برای خروجی هر لایه یک متغیر **bias** تعریف میشود. به صورت زیر:

(7 * 7 * 3) + 3 = 150

سه کرنل ۳ در ۳: در این حالت برای هر کرنل که بر روی تصویر سه کاناله با **bias** های مجزا اعمال می شود ۳۰ متغیر داریم که در مجموع می شود:

3 * ((3 * 3 * 3) + 3) = 90

به علت استفاده از سه مرحله کرنل در حالت دوم و اعمال آنها بر روی مقادیر جدید تصویر، عمق و پیچیدگی شبکه افزایش می‌یابد. همچنین به دلیل عمق بیشتر و سایز کرنل کوچکتر، در حالت دوم روابط غیرخطی بیشتری کشف می‌شود. به طور کلی، استفاده از کرنل دوم نتایج بهتری ارائه می‌دهد. این به خاطر داشتن پارامترهای کمتر و محاسبات بهینه‌تر آن است، همچنین عمق بیشتر و شناسایی روابط غیرخطی به ما در آموزش بهتر مدل کمک می‌کند.