۱)

الف) در حالت کلی نیازی به padding نیست. این الگوریتم به عنوان یک descriptor و با مقایسه هر پیکسل با مقادیر پیکسل‌های همسایه عمل می‌کند.

در صورتی که اندازه این همسایگی از ابعاد تصویر بزرگ‌تر باشد، برای انجام عملیات‌ها نیاز به افزودن padding هست.

افزودن padding می‌تواند بافت مناطق مرزی تصویر را تغییر دهد که این می‌تواند ناخوشایند باشد.

به همین دلیل در صورتی که پیکسل همسایه‌ای از یک پیکسل مرکزی، در محدوده تصویر قرار نداشت (در آن جایگاه مقداری نداشتیم)، آن را نادیده می‌گیریم و صرفا الباقی پیکسل‌های همسایه را به حساب می‌آوریم.

ب) این الگوریتم با مقایسه پیکسل مرکزی با پیکسل‌های اطراف عمل می‌کند و در صورتی که مقدار کم‌تری داشته باشند مقدار صفر و در غیر این صورت مقدار یک به آن می‌دهد. سپس از پیکسل بالا سمت چپ شروع می‌کند و اعداد را کنار هم می‌چیند تا کد LBP آن به دست آید.

برای حل این سوال از تابع آماده feature.local_binary_pattern استفاده کردم (فایل آن در محتوای دیرکتوری تمرین گنجانده شده است) که از پدینگ استفاده نمی‌کند و در صورتی که پیکسل همسایه‌ای وجود نداشت آن را ایگنور می‌کند.
خروجی:

```
[[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.]]
```

۳)

هدف استفاده از توابع فعال‌ساز در شبکه‌های عصبی توانایی ایجاد لایه‌های غیرخطی است که باعث کشف روابط پیچیده‌تر در داده‌های ورودی و پیش‌بینی دقیق‌تر می‌شود.

این تابع روی خروجی نورون‌های هر لایه اعمال می‌شود و آن را به یک سیگنال تبدیل می‌کند.

Sigmoid: این تابع که به تابع logistic نیز معروف است، ورودی را به یک عدد بین صفر و یک نگاشت می‌کند. منحنی این تابع تقریبا شبیه حرف S است و در مدل‌هایی که خروجی آن‌ها به‌عنوان احتمال تفسیر می‌شود گزینه خوبی است. مشکل این تابع فعال‌سازی vanishing gradient است. به این معنی که در مرحله backpropagation مقادیر گرادیان محاسبه‌شده با برگشت در شبکه به سمت لایه‌های اول کوچک‌تر می‌شوند. این باعث می‌شود وزن‌های لایه‌های اولیه به‌خوبی آپدیت نشوند و فرآیند یادگیری کند شود.

Tanh: این تابع مشابه تابع sigmoid است با این تفاوت که ورودی را به مقادیر بین منفی یک و یک نگاشت می‌کند. منحنی آن هم‌چنان مشابه حرف S است و مانند sigmoid می‌تواند در تفسیر خروجی‌ها به شکل احتمال استفاده شود (با کمی تغییر، چرا که باید به مقادیر بین صفر و یک مپ شود). مزیت آن نسبت به sigmoid این است که مقادیر آن در اطراف صفر قرار دارد که می‌تواند به فرآیند آموزش کمک کند.

ReLU: این تابع در صورتی که ورودی مثبت دریافت کند مقدار دقیق آن را خروجی می‌دهد و در صورتی که مقدار منفی دریافت کند مقدار صفر خروجی می‌دهد. این تابع فعال‌ساز دچار مشکل vanishing gradient نمی‌شود اما در صورتی که مقادیر اولیه آن درست مقداردهی نشوند می‌تواند باعث ایجاد نورون‌های خاموش شود. نورون‌هایی که به‌ازای هر ورودی احتمالی خروجی صفر می‌دهند و غیرفعال هستند.

PReLU: این تابع همانند تابع قبل است با این تفاوت که یک پارامتر قابل‌یادگیری

اضافه دارد و به مقادیر ورودی منفی مقدار صفر نمی‌دهد بلکه مقدار کوچکی به آن‌ها تخصیص می‌دهد (شیب کم در قسمت منفی نمودار) که می‌تواند مشکل نورون‌های خاموش را برطرف کند.

در کل دو تابع اول برای مدل‌های احتمالاتی مناسب هستند و دو تابع دوم برای ساختن شبکه‌های عمیق. بسته به مسئله می‌توان تابع مناسب را انتخاب کرد.

۵)

به طور کلی می‌توانیم یک شبکه sequential برای هر شبکه functional پیاده‌سازی کنیم.

شبکه عصبی sequential نوع پرکاربردی از معماری شبکه‌های عصبی است که پشته‌ای از لایه‌ها دارد و خروجی یک لایه به‌عنوان ورودی لایه بعدی استفاده می‌شود. در حالی که شبکه عصبی functional امکان بیش‌تری برای معماری‌های پیچیده‌تر شبکه دارد.

تبدیل مدل functional به sequential گاهی با دشواری کم و صرفا با چینش لایه‌ها در یک پشته انجام می‌شود. اما برخی اوقات برای شبکه‌های پیچیده نیاز به بازسازی ساختار شبکه داریم، کارهایی مانند ایجاد زیرمدل‌های جداگانه به‌طوری که بتوانند به صورت sequential مرتب شوند.

در برخی موارد این تبدیل امکان‌پذیر نیست. مثال‌ها:

- شبکه‌هایی که ورودی یا خروجی‌های چندگانه دارند
- شبکه‌هایی با اتصال پرش یا شاخه‌بندی‌های پیچیده
- شبکه‌هایی با لایه‌های مشترک
- شبکه‌هایی با ساختارهای پویا (ساختار بستگی به داده‌های ورودی یا یک سری محاسبات میانی خاص دارد)

در کل باید دانست که شبکه‌های sequential دارای محدودیت‌هایی هستند. به طور مثال نمی‌توانند ورودی یا خروجی‌های چندگانه را مدیریت کنند، محدود به پشته‌ای از لایه‌ها هستند و با ساختارهای پویا منطبق نمی‌شوند.

۶)

ابعاد خروجی از رابطه زیر به دست می‌آید:

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

الف) در این مثال ابعاد خروجی یک در یک می‌شود:

[(7 – 7 + 2 * 0) / 1] + 1

ب) در این مثال نیز ابعاد خروجی یک در یک می‌شود:

طبق رابطه بالا:

ابعاد بعد از اعمال کرنل اول: پنج در پنج

ابعاد بعد از اعمال کرنل دوم: سه در سه

ابعاد بعد از اعمال کرنل سوم: یک در یک

پ)

کرنل ۷ در ۷:

در این حالت یک کرنل ۷ در ۷ داریم که بر روی یک تصویر سه کاناله اعمال می‌شود. 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 biasهای مجزا اعمال می‌شود ۳۰ متغیر داریم که مجموعا برای سه کرنل می‌شود:

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

عمق و خطی بودن ویژگی‌ها:

- به دلیل اینکه در حالت دوم ما سه مرحله کرنل می‌زنیم و هر بار این کرنل روی مقادیر جدید تصویر اعمال می‌شود، عمق و پیچیدگی شبکه افزایش می‌یابد.

- هم‌چنین به دلیل عمق بیش‌تر و کوچک بودن سایز کرنل (جزئیات بیش‌تر تاثیرگذار هستند)، در حالت دوم ما روابط غیرخطی بیش‌تری کشف می‌کنیم.

در کل کرنل دوم نتیجه بهتری می‌دهد زیرا پارامترهای کم‌تری دارد و محاسبات آن بهینه‌تر است و هم‌چنین به دلیل عمق بیش‌تر و یافتن روابط غیرخطی، در آموزش بهتر مدل به ما کمک می‌کند.