تمرین سری ششم بینایی کامپیوتر

دکتر محمدی

هلیا شمس زاده

۴۰۰۵۲۱۴۸۶

سوال اول).....

الف- ابعاد خروجی و تعداد پارامترهای هر لایه را محاسبه کنید.

لایهٔ اول: Input(shape=(512, 512, 3))

خروجی این لایه برابر یک ماتریس 3×512×512 می‌باشد.

لایهٔ دوم: Conv2D(32, (9, 9), strides=2, padding='same', activation='relu')

چون padding= ‘same’، حین convolution اندازهٔ ماتریس تغییر نمی‌کند، اما به دلیل stride=2 چون دو تا دو تا جلو می‌رود، در خروجی نهایی طول و عرض نصف شده و تعداد کانال‌ها نیز برابر تعداد کرنل‌ها یعنی 32 می‌باشد. پس خروجی این لایه دارای ابعاد 32×256×256 می‌باشد.

تعداد پارامترها برابر (1 + 3×9×9)×32 یعنی 7808 تا می‌باشد.

لایهٔ سوم: MaxPooling2D((4, 4), strides=4)

به دلیل ابعاد ۴ در ۴ pooling، طول و عرض خروجی تقسیم بر ۴ می‌شود، تعداد کانال‌ها هم تغییری نمی‌کند و خروجی این لایه دارای ابعاد 32×64×64 می‌باشد. این لایه پارامتر قابل آموزش ندارد.

لایهٔ چهارم: Conv2D(64, (5, 5), strides=1)

چون فرمت padding ذکر نشده است، یعنی مقدار دیفالت که ’valid‘ است دارد که یعنی padding انجام نمی‌شود و طول و عرض به اندازهٔ kernel_size – 1 یعنی ۴ تا کم می‌شوند. تعداد کانال‌ها هم برابر تعداد kernel ها می‌باشد. خروجی این لایه دارای ابعاد 64×60×60 می‌باشد.

تعداد پارامترها برابر (1 + 32×5×5)×64 یعنی 51264 تا می‌باشد.

لایهٔ پنجم: AveragePooling2D((2, 2), strides=2)

به دلیل ابعاد ۲ در ۲، طول و عرض نصف می‌شود، تعداد کانال‌ها تغییر نمی‌کند و خروجی این لایه برابر 64×30×30 می‌شود. این لایه پارامتر قابل آموزش ندارد.

لایهٔ ششم: Conv2D(128, (3, 3), strides=1, padding='valid', activation='relu')

به دلیل padding=’valid’، طول و عرض به اندازهٔ 1-3 یعنی 2 تا کمتر می‌شود، تعداد کانال‌ها هم برابر تعداد فیلترها یعنی ۱۲۸ تا می‌شود و خروجی این لایه برابر 128×28×28 می‌شود.

تعداد پارامترها برابر (1 + 64×3×3)×128 یعنی 73856 تا می‌باشد.

لایهٔ هفتم: Conv2D(128, (3, 3), strides=1, padding='same', activation='relu')

به دلیل padding=’same’ ابعاد ورودی تغییر نمی‌کند، تعداد کانال‌ها هم برابر ۱۲۸ تا می‌شود. خروجی این لایه برابر 128×28×28 می‌باشد. تعداد پارامترها برابر (1 + 128×3×3)×128 یعنی 147584 تا می‌باشد.

لایهٔ هشتم: MaxPooling2D((2, 2), strides=2)

به دلیل ابعاد pooling که ۲ در ۲ است، طول و عرض نصف شده و تعداد کانال‌ها نصف می‌شود. خروجی این لایه برابر 128×14×14 می‌باشد. این لایه پارامتر قابل آموزش ندارد.

لایهٔ نهم: Conv2D(512, (3, 3), strides=1, padding='valid', activation='relu')

به دلیل padding=’valid’ طول و عرض به اندازهٔ 1-3 یعنی 2 تا کمتر می‌شود و تعداد کانال‌ها برابر 512 می‌شود. خروجی این لایه برابر 512×12×12 می‌باشد.

تعداد پارامترها برابر (1 + 128×3×3)×512 یعنی 590336 تا می‌باشد.

لایهٔ دهم: GlobalAveragePooling2D()

این تابع طول و عرض را تبدیل به ۱ می‌کند. یعنی خروجی این لایه برابر 512×1 می‌شود. این لایه پارامتر قابل آموزش ندارد.

لایهٔ یازدهم: Dense(1024)

خروجی این لایه برابر 1024×1 می‌باشد. تعداد پارامترها برابر (1 + 512) × 1024 یعنی 525312 تا می‌باشد.

لایهٔ دوازدهم: Dense(10)

خروجی این لایه برابر 10×1 می‌باشد. یک classifier ۱۰ کلاسه است.

تعداد پارامترها برابر (1 + 1024)×10 یعنی 10250 تا می‌باشد.

مجموعا این شبکه دارای ۱۴۰۶۴۱۰ پارامتر قابل یادگیری است.

ب- تعداد اعمال ضرب و جمع در هر لایه که بر ورودی اعمال می شود را حساب کنید.

فقط برای لایه‌های کانوولوشنی، Average pooling، global pooling و تماما متصل ضرب و جمع داریم.

لایهٔ ۲: در کل این عمل محاسبه به ازای هر kernel، برای 256×256 تا پیکسل اعمال می‌شود که برای محاسبهٔ خروجی برای هر پیکسل 3×9×9 تا عمل ضرب و جمع انجام می‌شود. این تعداد برای هر 32 تا کرنل موجود انجام شده و در کل برای لایهٔ دوم تعداد (256×256×3×9×9)×32 ضرب و جمع انجام می‌شود.

لایهٔ ۴: در کل این عمل محاسبه به ازای هر kernel، برای 60×60 تا پیکسل اعمال می‌شود که برای محاسبهٔ خروجی برای هر پیکسل 32×5×5 تا عمل ضرب و جمع انجام می‌شود. این تعداد برای هر 64 تا کرنل موجود انجام شده و در کل برای لایهٔ چهارم تعداد (60×60×32×5×5)×64 ضرب و جمع انجام می‌شود.

لایهٔ ۵: در این لایه (average pooling)، پنجره‌های ۲ در ۲ با تعداد گام ۲، میانگین گرفته می‌شوند. یعنی برای هر کانال، 30×30بار میانگین گرفته می‌شود که برای میانگین گرفتن، ۴ تا جمع داریم. پس مجموعا برای هر کانال 4×30×30 تا ضرب و جمع داریم، و چون ۶۴ تا کانال داریم، مجموعا 4×30×30×64 تا عمل ضرب و جمع داریم.

لایهٔ ۶: در کل این عمل محاسبه به ازای هر kernel، برای 28×28 تا پیکسل (بخاطر padding='valid') اعمال می‌شود که برای محاسبهٔ خروجی برای هر پیکسل 64×3×3 تا عمل ضرب و جمع انجام می‌شود. این تعداد برای هر 128 تا کرنل موجود انجام شده و در کل برای لایهٔ ششم تعداد (28×28×64×3×3)×128 ضرب و جمع انجام می‌شود.

لایهٔ ۷: در کل این عمل محاسبه به ازای هر kernel، برای 28×28 تا پیکسل اعمال می‌شود که برای محاسبهٔ خروجی برای هر پیکسل 3×3×128 تا عمل ضرب و جمع انجام می‌شود. این تعداد برای هر 128 تا کرنل موجود انجام شده و در کل برای لایهٔ هفتم تعداد (28×28×128×3×3)×128 ضرب و جمع انجام می‌شود.

لایهٔ ۹: در کل این عمل محاسبه به ازای هر kernel، برای 12×12 تا پیکسل اعمال می‌شود که برای محاسبهٔ خروجی برای هر پیکسل، 3×3×128 تا عمل ضرب و جمع انجام می‌شود. این تعداد برای هر 512 تا کرنل موجود انجام شده و در کل برای لایهٔ نهم تعداد (12×12×128×3×3)×512 ضرب و جمع انجام می‌شود.

لایهٔ ۱۰: این لایهٔ global pooling است که مقادیر کانال‌ها را میانگین می‌گیرد. یعنی 12×12×512 تا ضرب و جمع انجام می‌دهد.

لایهٔ ۱۱: برای هر نورون این لایه 512 تا ضریب w_i وجود دارد. تعداد نورون‌ها هم برابر 1024 تا است، پس مجموعا 1024×512 تا ضرب و جمع داریم.

لایهٔ ۱۲: برای هر نورون این لایه 1024 تا ضریب w_i وجود دارد. تعداد نورون‌ها هم برابر 10 تا است، پس مجموعا 10×1024 تا ضرب و جمع داریم.

ج- اگر به جای لایه GAP از flatten استفاده شود تعداد پارامترهای شبکه چند برابر می‌شود؟

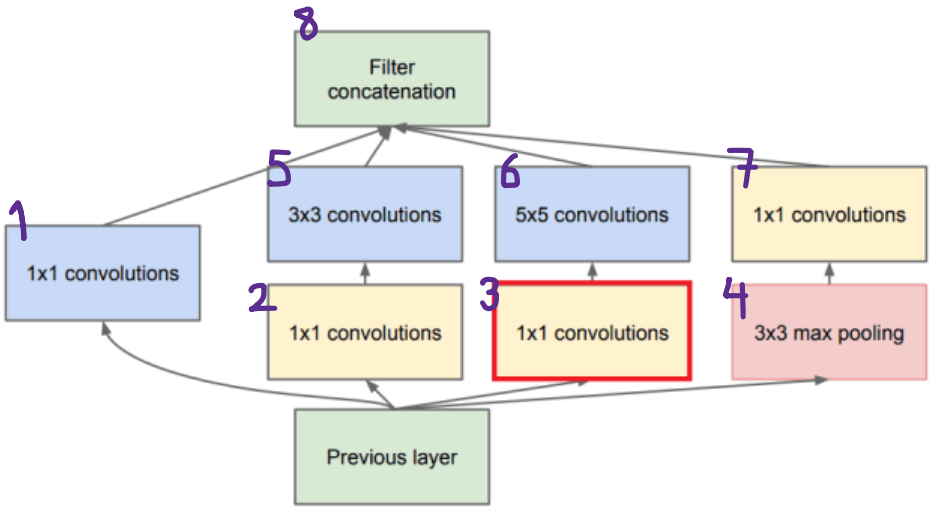
یعنی در لایهٔ دهم، ابعاد ورودی که معادل خروجی لایهٔ قبل که دارای ابعاد 12×12×512 است را flat می‌کنیم که در این صورت خروجی این لایه دارای ابعاد 1×73728 می‌شود پس در لایهٔ بعدی هم تعداد پارامترها برابر (1 + 73728) × 1024 یعنی برابر ۷۵۴۹۸۴۹۶ تا می‌شود. برای لایهٔ بعدی هم تعداد پارامترها برابر 10250 تا باقی می‌ماند.

مجموع کل پارامترهای شبکه در این حالت برابر ۷۶۳۷۹۵۹۴ تا می‌باشد که تقریبا **۵۳** برابر حالت قبلی می‌باشد.

سوال دوم).....

واضح است برای مقادیر ۲۰ و نزدیک آن، به ترتیب نمودارهای b، c و a تغییرات بیشتری دارند (تغییرات: b > c > a) ، یعنی مثلا با گذشتن یک epoch، تغییر loss برای نمودار b از تغییرات loss نمودار c، و تغییرات loss نمودار c از تغییرات loss نمودار a بیشتر است. طبق این استدلال، X = 14 را که نسبت به بقیهٔ موارد تغییری بیشتری در یک epoch داشته است می‌توانیم به نمودار b نسبت دهیم، X = 19.4 را به c نسبت داده و X = 19.94 را که نسبت به بقیه کمترین تغییرات را داشته ایم به a نسبت دهیم.

سوال سوم).....

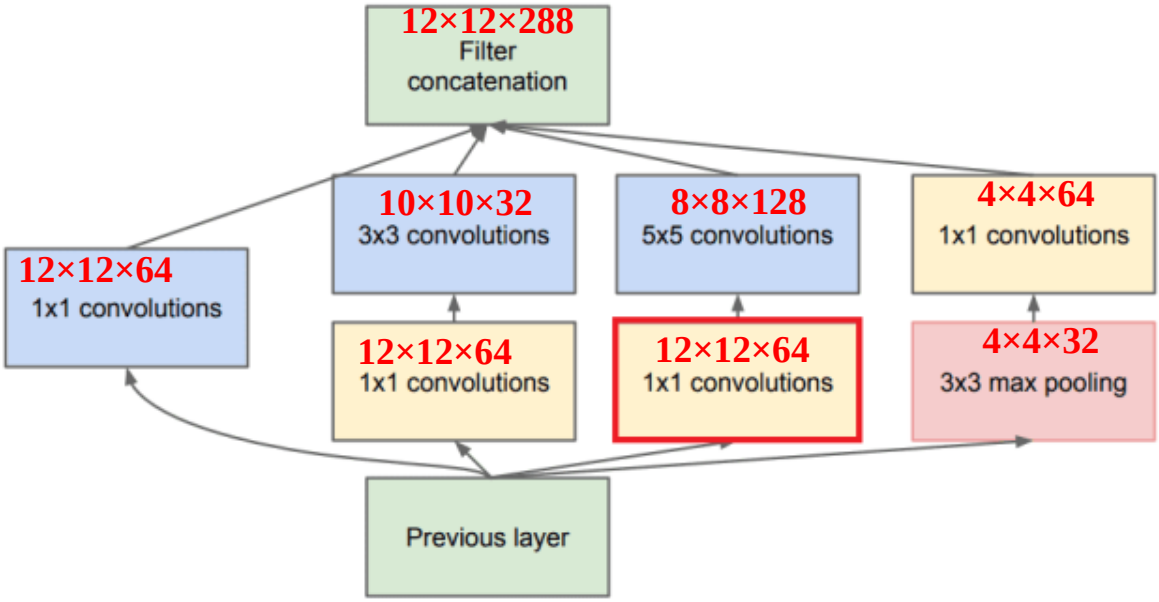


بخش اول) ۶۴ کرنل ۱×۱

براساس مطالب نوشته شده در این لینک، padding='valid' است و طول و عرض خروجی‌ها عوض می‌شود. خروجی عنصر ۱ و ۲ و ۳ (در شکل مشخص شده است) ماتریسی با ابعاد 64×12×12 می‌باشد (تعداد کانال‌ها به دلیل داشتن ۶۴ تا کرنل ۱ در ۱ برابر ۶۴ شد). خروجی عنصر ۴ که یک maxPooling 3*3 است (با Stride=3) یک ماتریس با ابعاد 32×4×4 است (طول و عرض تقسیم بر ۳ شده و تعداد کانال‌ها تغییری نمی‌کند). خروجی عنصر ۷ ماتریسی با ابعاد 64×4×4 می‌باشد (تعداد کانال‌ها به دلیل داشتن ۶۴ تا کرنل ۱ در ۱ برابر ۶۴ شد). خروجی عنصر ۵ یک ماتریس با ابعاد 32×10×10 می‌باشد (طول و عرض 2 تا کم می‌شوند و

تعداد کانال‌ها برابر تعداد کرنل های ۳ در ۳ یعنی ۳۲ تا است). خروجی عنصر ۶ ماتریسی با ابعاد 8×8×128 است (طول و عرض 4 تا کم می‌شوند و تعداد کانال‌ها برابر تعداد کرنل های ۵ در ۵ یعنی ۱۲۸ تا است).

عنصر ۸ که filter concatenation است، در واقع کانال‌ها را به هم می‌چسباند. طول و عرض‌ها را با padding یکی کرده و کانال‌ها را ادغام می‌کند. خروجی‌ها عنصرهای ۱، ۵، ۶ و ۷ یکی شده و به خروجی نهایی دارای ابعاد 12×12×288 می‌باشد (تعداد کانال‌ها برابر مجموع کانال‌های خروجی عناصر ۱، ۵، ۶ و ۷ است که برابر 64+64+32+128 می‌باشد). خروجی هر عنصر با رنگ قرمز در عکس زیر نوشته شده است.



بخش دوم) ۲۵۶ کرنل ۱×۱

خروجی عنصر ۳ برابر 12×12×256 می‌شود، چون تعداد کرنل‌ها برابر ۲۵۶ تا شد. ولی خروجی عنصر ۶ (که ورودی‌اش همان خروجی عنصر ۳ است) همان 8×8×128 باقی می‌ماند. خروجی نهایی هم همان 12×12×288 می‌شود.

سوال چهارم).....

بخش الف)

تعداد batch های موجود در هر epoch برابر است با $\left\lceil \frac{t}{batch\ size} \right\rceil$ (تعداد نمونه‌های آموزشی تقسیم بر سایز batch).

اگر e تعداد دوره‌های آموزشی (epochs) باشد، تعداد کل به‌روزرسانی‌های وزن‌ها برابر خواهد بود با:

$$e \times \left\lceil \frac{t}{batch\ size} \right\rceil$$

به عبارت دیگر، در هر دوره آموزشی وزن‌ها $\left\lceil \frac{t}{batch\ size} \right\rceil$ بار به‌روزرسانی می‌شوند و اگر این فرآیند برای e دورهٔ آموزشی تکرار شود، تعداد کل به‌روزرسانی‌های وزن‌ها برابر مقدار بالا خواهد بود.

بخش ب)

این نمودار مربوط به افت خطا (loss) در طول دوره‌های آموزشی (epochs) است. با توجه به نوسانات و تغییرات کوچک در مقدار خطا در هر دوره آموزشی، به نظر می‌رسد این نمودار مربوط به استفاده از mini-batch Gradient Descent باشد. در Mini-batch Gradient Descent ، داده‌های آموزشی به چندین دسته کوچک تقسیم می‌شوند و به‌روزرسانی‌های وزن‌ها پس از هر دسته انجام می‌شود. این منجر به نوساناتی در نمودار خطا می‌شود که در تصویر مشاهده می‌شود. این نوسانات ناشی از تغییرات کوچک در مجموعه‌های داده کوچک است که در هر دسته وجود دارد. دلایل:

- **نوسانات زیاد در خطا**: نوسانات در خطا نشان‌دهنده استفاده از مجموعه‌های داده کوچک‌تر در هر به‌روزرسانی است، که خاصیت Mini-batch Gradient Descent است.
- **روند کلی کاهش خطا**: با وجود نوسانات، روند کلی کاهش خطا حفظ شده است که نشان‌دهنده آموزش موفقیت‌آمیز مدل است.

اگر از Batch Gradient Descent استفاده می‌شد، نوسانات خطا کمتر می‌شد چون به‌روزرسانی وزن‌ها بر اساس کل مجموعه داده در هر دوره انجام می‌شد و نمودار نرم‌تری به دست می‌آمد. در مقابل، اگر از Stochastic Gradient Descent استفاده می‌شد، نوسانات خیلی بیشتری در نمودار مشاهده می‌شد، زیرا هر نمونه به تنهایی برای به‌روزرسانی وزن‌ها استفاده می‌شود.

بخش د)

نمودار Curve A:

نمودار آموزشی (training) به طور قابل توجهی سریع‌تر از نمودار اعتبارسنجی (validation) افت می‌کند و تفاوت قابل توجهی بین خطای آموزش و اعتبارسنجی وجود دارد، به خصوص در اواخر epoch ها. این الگو نشان‌دهنده overfitting است، یعنی مدل به خوبی روی داده‌های آموزشی عملکرد دارد اما عملکرد آن روی داده‌های اعتبارسنجی کمتر از حد انتظار است.

- **افزایش داده‌های آموزشی:** می‌تواند کمک کند تا مدل به داده‌های بیشتری دسترسی داشته باشد و در نتیجه به بهبود تعمیم‌دهی کمک کند.
- **افزایش لایه‌های شبکه**: این روش منجر به افزایش پیچیدگی مدل می‌شود و کمک می‌کند که مدل بتواند ویژگی‌های بیشتری را یاد بگیرد. اما در این حالت که مدل overfitting دارد، این موضوع می‌تواند منجر به بدتر شدن شرایط نیز بشود، زیرا ممکن است در حال حاضر مدل زیادی پیچیده باشد.
- **کاهش تعداد ویژگی‌های ورودی**: مفید است زیرا ممکن است مدل ویژگی‌های غیرمهم را یاد می‌گیرد که منجر به overfitting می‌شود.

نمودار Curve B:

نمودارهای آموزشی و اعتبارسنجی نزدیک‌تر به هم هستند و تفاوت کمتری بین خطای آموزش و اعتبارسنجی وجود دارد. این الگو نشان‌دهنده underfitting است، یعنی مدل نه تنها روی داده‌های اعتبارسنجی بلکه روی داده‌های آموزشی نیز عملکرد خوبی ندارد.

- **افزایش داده‌های آموزشی**: ممکن است کمکی نکند زیرا مدل حتی با داده‌های فعلی نیز عملکرد خوبی ندارد، یعنی مدل ساده است و نمی‌تواند برای داده‌های فعلی نیز ویژگی‌های خوب استخراج کند و افزایش تعداد داده‌ها کمک زیادی نمی‌کند.
- **افزایش لایه‌های شبکه:** می‌تواند مفید باشد چون ممکن است مدل به اندازه کافی پیچیدگی نداشته باشد تا ویژگی‌های موجود در داده‌ها را به خوبی یاد بگیرد.
- **کاهش تعداد ویژگی‌های ورودی**: به بهبود مدل کمک نمی‌کند زیرا مشکل اصلی عدم پیچیدگی مدل است.

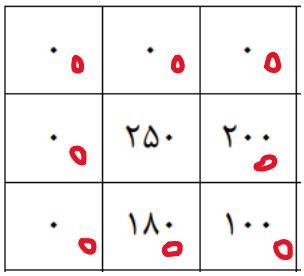
نتیجه‌گیری:

- برای **Curve A** پیشنهاد می‌شود **داده‌های آموزشی بیشتری استفاده شود** (**داده افزایی**) و یا **تعداد ویژگی‌های ورودی کاهش یابند**.
- برای **Curve B** پیشنهاد می‌شود **تعداد لایه‌های شبکه افزایش یابد**.

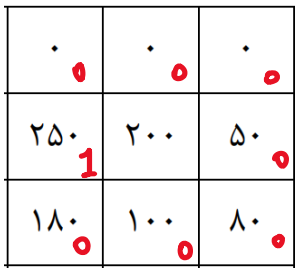
سوال پنجم).....

بخش الف)

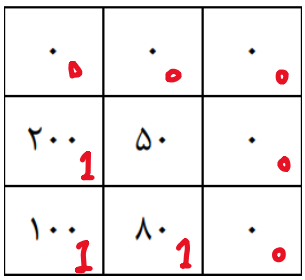
شکل رو به رو مقادیر LBP را برای پیکسل ۲۵۰ نشان می‌دهد که برابر 00000000 یعنی صفر می‌باشد.



شکل رو به رو مقادیر LBP را برای پیکسل ۲۰۰ نشان می‌دهد که برابر 00000001 یعنی ۱ می‌باشد.



شکل رو به رو مقادیر LBP را برای پیکسل ۵۰ نشان می‌دهد که برابر 00000111 یعنی ۷ می‌باشد.



۰ 0	۲۵۰ 1	۲۰۰ 1
۰ 0	۱۸۰	۱۰۰ 0
۰ 0	۲۰۰ 1	۴۰ 1

شکل رو به رو مقادیر LBP را برای پیکسل ۱۸۰ نشان می‌دهد که برابر 01101100 یعنی ۱۰۰ می‌باشد.

۲۵۰ 1	۲۰۰ 1	۵۰ 0
۱۸۰ 1	۱۰۰	۸۰ 0
۲۰۰ 1	۴۰ 0	۷۰ 0

شکل رو به رو مقادیر LBP را برای پیکسل ۱۰۰ نشان می‌دهد که برابر 11000011 یعنی ۱۹۵ می‌باشد.

۲۰۰ 1	۵۰ 0	۰ 0
۱۰۰ 1	۸۰	۰ 0
۴۰ 0	۷۰ 0	۰ 0

شکل رو به رو مقادیر LBP را برای پیکسل ۸۰ نشان می‌دهد که برابر 10000001 یعنی ۱۲۹ می‌باشد.

۰ 0	۱۸۰ 0	۱۰۰ 0
۰ 0	۲۰۰	۴۰ 0
۰ 0	۰ 0	۰ 0

شکل رو به رو مقادیر LBP را برای پیکسل ۲۰۰ نشان می‌دهد که برابر 00000000 یعنی صفر می‌باشد.

۱۸۰ 1	۱۰۰ 1	۸۰ 1
۲۰۰ 1	۴۰	۷۰ 1
۰ 0	۰ 0	۰ 0

شکل رو به رو مقادیر LBP را برای پیکسل ۴۰ نشان می‌دهد که برابر 11110001 یعنی ۲۴۱ می‌باشد.

۱۰۰ 1	۸۰ 1	۰ ۰
۴۰ ۰	۷۰	۰ ۰
۰ ۰	۰ ۰	۰ ۰

شکل رو به رو مقادیر LBP را برای پیکسل ۷۰ نشان می‌دهد که برابر 11000000 یعنی ۱۹۲ می‌باشد.

بخش ب)

اگر مقادیر با ثابت c جمع شوند، فواصل و اختلاف پیکسل‌ها حفظ می‌شود. مثلا اگر دو پیکسل اختلافشان ۵ است، بعد از جمع کردن با ثابت c این اختلاف برابر c + 5 می‌شود. (اگر a > b باشد، a + c > b + c نیز برقرار است) پس کدها فرقی نمی‌کنند؛ مگر اینکه c آنقدر بزرگ باشد که یک سری از مقادیر به ۲۵۵ برسند.

برای ضرب نیز همین استدلال برقرار است، زیرا اختلاف‌ها نیز در c ضرب می‌شوند (اگر a > b باشد، ac > bc نیز در صورت مثبت بودن c برقرار است). پس کدها تغییر نمی‌کنند، مگر اینکه c آنقدر بزرگ باشد که یک سری از مقادیر به ۲۵۵ برسند.

بخش ج)

تصویر اول (B):

تصویر اول شامل یک گل است که دارای بافت نسبتا پیچیده‌ای می‌باشد. هیستوگرام LBP این تصویر (هیستوگرامB) نشان‌دهندهٔ پراکندگی گسترده‌تری از مقادیر LBP است که به دلیل وجود بافت پیچیده و جزئیات زیاد در تصویر است.

تصویر دوم (C) :

این تصویر شامل خطوط سیاه و سفید ساده و بافت‌های هندسی واضح است. هیستوگرام LBP مربوط به این تصویر باید شامل تعداد کمی از پیک‌های بلند باشد که نشان‌دهنده الگوهای باینری تکراری و ساده است. در نمودارC ، هیستوگرام شامل تعداد کمی پیک‌های بلند و متمرکز است که نشان‌دهنده الگوهای ساده و تکراری است.

تصویر سوم(A) :

این تصویر شامل سنگ‌ریزه‌ها و بافت‌های تصادفی است که شامل تنوع بافتی بیشتری است نسبت به تصویر گل. هیستوگرام LBP مربوط به این تصویر باید دارای قله‌های بلند و پراکنده باشد که نشان‌دهنده تنوع بالا و نویز در الگوهای باینری محلی است. در نمودار A ، هیستوگرام شامل قله‌های بلند و پراکنده است که نشان‌دهنده توزیع یکنواخت‌تر و تنوع بیشتر در بافت تصویر است.

نتیجه‌گیری:

- **تصویر اول** با هیستوگرام **B** مطابقت دارد.
- **تصویر دوم** با هیستوگرام **C** مطابقت دارد.
- **تصویر سوم** با هیستوگرام **A** مطابقت دارد.

سوال ششم).....

بخش اول-

پس از import کردن کتابخانه‌های لازم دیتاست را load می‌کنیم:

Importing Required Libraries and Loading the Dataset

```
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras import layers, models, regularizers
import matplotlib.pyplot as plt
import numpy as np

# Load the Dogs vs. Cats dataset
(train_dataset, test_dataset), info = tfds.load('cats_vs_dogs', split=['train[:80%]', 'train[80%:]'], with_info=True, as_supervised=True)
```

```
Downloading and preparing dataset 786.67 MiB (download: 786.67 MiB, generated: 1.04 GiB, total: 1.81 GiB) to /root/tensorflow_datasets/cats_vs_dogs/4.0.1...
Dl Completed...: 100% 1/1 [00:13<00:00, 13.11s/ url]
Dl Size...: 100% 786/786 [00:13<00:00, 70.94 MiB/s]
WARNING:absl:1738 images were corrupted and were skipped
Dataset cats_vs_dogs downloaded and prepared to /root/tensorflow_datasets/cats_vs_dogs/4.0.1. Subsequent calls will reuse this data.
```

سپس تابع preprocess را می‌نویسیم که تصاویر ورودیم مدل را resize و نرمالایز می‌کند:

Preprocess the inputs in order to fit the model's input size and be normalized

```
# Preprocessing function to resize and normalize images
def preprocess(image, label):
    image = tf.image.resize(image, (128, 128))
    image = image / 255.0  # Normalize pixel values
    return image, label
```

به منظور بهتر کردن عملیات آموزش و جلوگیری از overfit شدن، تابع augment را می‌نویسیم که داده‌افزایی را انجام می‌دهد. این تابع عملیات flip، تغییر شدت روشنایی، تغییر کنتراست، تغییر مقیاس و random crop را انجام می‌دهد.

Function for Data Augmentation: Done for having better training process

```
[4]  # Data augmentation function to apply random transformations
     def augment(image, label):
         image = tf.image.random_flip_left_right(image)
         image = tf.image.random_brightness(image, max_delta=0.1)
         image = tf.image.random_contrast(image, 0.9, 1.1)

         # Random zoom
         scales = list(np.arange(0.8, 1.0, 0.1))
         boxes = np.zeros((len(scales), 4))

         for i, scale in enumerate(scales):
             x1 = y1 = 0.5 - 0.5 * scale
             x2 = y2 = 0.5 + 0.5 * scale
             boxes[i] = [x1, y1, x2, y2]

         def random_crop(img):
             crops = tf.image.crop_and_resize([img], boxes=boxes, box_indices=np.zeros(len(scales)), crop_size=(128, 128))
             return crops[tf.random.uniform(shape=[], minval=0, maxval=len(scales), dtype=tf.int32)]

         image = random_crop(image)

         return image, label
```

```
[5]  # Preprocess the training and test datasets
     train_dataset = train_dataset.map(preprocess, num_parallel_calls=tf.data.experimental.AUTOTUNE)
     test_dataset = test_dataset.map(preprocess, num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

سپس تابع augment را روی داده‌های train صدا می‌زنیم تا داده‌افزایی انجام شود. سپس داده‌های train و test را batch بندی می‌کنیم:

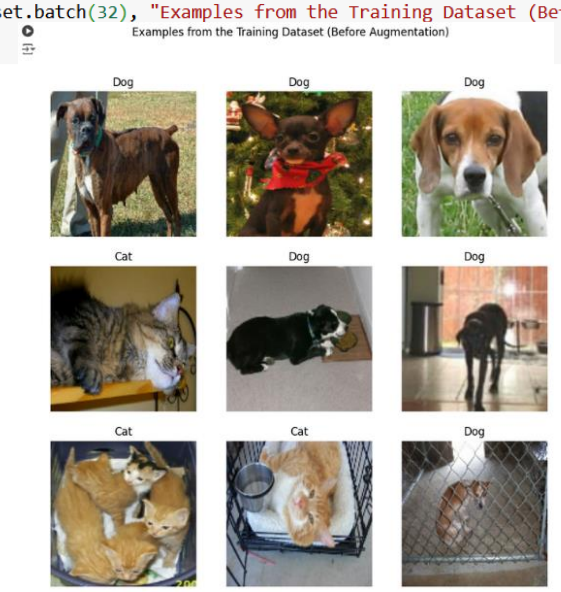
Apply data augmentation on training set and batch all data

```
[6]  # Apply data augmentation only to the training dataset
     train_dataset_augmented = train_dataset.map(augment, num_parallel_calls=tf.data.experimental.AUTOTUNE).batch(32).prefetch(tf.data.experimental.AUTOTUNE)

     # batch, and prefetch the datasets
     test_dataset = test_dataset.batch(32).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

بخشی از داده‌های train نشان داده شده است:

Show some examples of the training set

```
[7]  # Function to display some images from a dataset
     def show_examples(dataset, title):
         plt.figure(figsize=(10, 10))
         for batch in dataset.take(1):
             images, labels = batch
             for i in range(min(9, len(images))):
                 ax = plt.subplot(3, 3, i + 1)
                 plt.imshow(images[i].numpy())
                 plt.title('Dog' if labels[i].numpy() == 1 else 'Cat')
                 plt.axis("off")
         plt.suptitle(title)
         plt.show()

     # Show some examples from the original training dataset
     show_examples(train_dataset.batch(32), "Examples from the Training Dataset (Before Augmentation)")
```



سپس مدل را تعریف می‌کنیم. چند لایهٔ کانوولوشنی به همراه لایه‌های pooling تعریف می‌شود:

Create the model and give a summary of it

```
# Create the model with L2 regularization and increased dropout
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),  # Dropout to prevent overfitting
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Early stopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Display the model's summary
model.summary()
```

مدل تعریف شده به شکل زیر است:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2  (None, 63, 63, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 30, 30, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 12, 12, 128)       147584

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 512)               2359808

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 2601153 (9.92 MB)
Trainable params: 2601153 (9.92 MB)
Non-trainable params: 0 (0.00 Byte)
```

سپس مدل را با داده‌های train و validation آموزش می‌دهیم و نتایج هر epoch را نشان می‌دهیم:

Train model on the training set and validation set

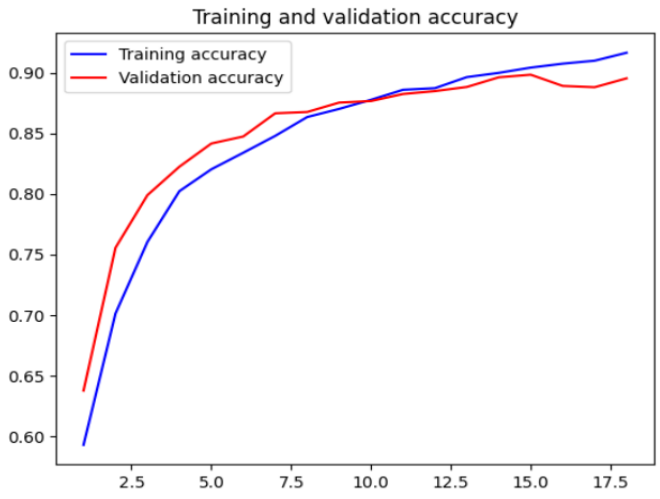
```
] # Train the model
  history = model.fit(train_dataset_augmented,
                      epochs=20,
                      validation_data=test_dataset,
                      callbacks=[early_stopping])
```

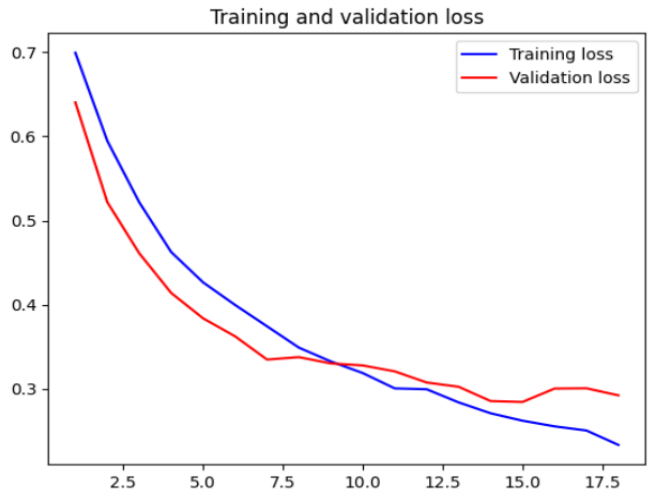
```
Epoch 1/20
582/582 [==============================] - 58s 86ms/step - loss: 0.6992 - accuracy: 0.5930 - val_loss: 0.6403 - val_accuracy: 0.6378
Epoch 2/20
582/582 [==============================] - 47s 82ms/step - loss: 0.5946 - accuracy: 0.7013 - val_loss: 0.5218 - val_accuracy: 0.7556
Epoch 3/20
582/582 [==============================] - 54s 92ms/step - loss: 0.5217 - accuracy: 0.7604 - val_loss: 0.4611 - val_accuracy: 0.7990
Epoch 4/20
582/582 [==============================] - 53s 91ms/step - loss: 0.4626 - accuracy: 0.8023 - val_loss: 0.4140 - val_accuracy: 0.8224
Epoch 5/20
582/582 [==============================] - 50s 86ms/step - loss: 0.4265 - accuracy: 0.8203 - val_loss: 0.3838 - val_accuracy: 0.8416
Epoch 6/20
582/582 [==============================] - 48s 82ms/step - loss: 0.3998 - accuracy: 0.8340 - val_loss: 0.3625 - val_accuracy: 0.8474
Epoch 7/20
582/582 [==============================] - 53s 92ms/step - loss: 0.3744 - accuracy: 0.8479 - val_loss: 0.3349 - val_accuracy: 0.8665
Epoch 8/20
582/582 [==============================] - 48s 83ms/step - loss: 0.3491 - accuracy: 0.8634 - val_loss: 0.3378 - val_accuracy: 0.8676
Epoch 9/20
582/582 [==============================] - 47s 81ms/step - loss: 0.3327 - accuracy: 0.8701 - val_loss: 0.3303 - val_accuracy: 0.8753
Epoch 10/20
582/582 [==============================] - 53s 91ms/step - loss: 0.3188 - accuracy: 0.8776 - val_loss: 0.3279 - val_accuracy: 0.8766
Epoch 11/20
582/582 [==============================] - 50s 87ms/step - loss: 0.3006 - accuracy: 0.8859 - val_loss: 0.3209 - val_accuracy: 0.8824
Epoch 12/20
582/582 [==============================] - 47s 81ms/step - loss: 0.2996 - accuracy: 0.8872 - val_loss: 0.3077 - val_accuracy: 0.8848
Epoch 13/20
582/582 [==============================] - 49s 84ms/step - loss: 0.2840 - accuracy: 0.8963 - val_loss: 0.3026 - val_accuracy: 0.8882
Epoch 14/20
582/582 [==============================] - 47s 80ms/step - loss: 0.2711 - accuracy: 0.8998 - val_loss: 0.2856 - val_accuracy: 0.8962
Epoch 15/20
582/582 [==============================] - 53s 91ms/step - loss: 0.2623 - accuracy: 0.9042 - val_loss: 0.2846 - val_accuracy: 0.8983
Epoch 16/20
582/582 [==============================] - 55s 94ms/step - loss: 0.2556 - accuracy: 0.9074 - val_loss: 0.3005 - val_accuracy: 0.8891
Epoch 17/20
582/582 [==============================] - 53s 91ms/step - loss: 0.2507 - accuracy: 0.9099 - val_loss: 0.3008 - val_accuracy: 0.8880
Epoch 18/20
582/582 [==============================] - 53s 90ms/step - loss: 0.2336 - accuracy: 0.9164 - val_loss: 0.2924 - val_accuracy: 0.8953
```

سپس نمودار loss و accuracy داده‌های train و validation را نشان می‌دهیم:

Plotting the loss and accuracy of train and validation data

```
# Plot training and validation loss and accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.figure(figsize=(14, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
```





سپس مدل ساخته شده را روی داده‌های تست evaluate کرده و نتیجه را پرینت می‌کنیم:

Evaluate the model on test set and print the result

```
# Evaluate the model on the test dataset
test_loss, test_acc = model.evaluate(test_dataset)
print(f'Test accuracy: {test_acc:.2f}')
```

```
146/146 [==============================] - 9s 60ms/step - loss: 0.2846 - accuracy: 0.8983
Test accuracy: 0.90
```

بخش دوم-

مراحل این بخش همانند بخش قبل است، فقط بخش ساختن مدل فرق می‌کند. ابتدا مدل InceptionV3 را با وزن‌های imagenet لود کرده ولی لایهٔ top آن را لود نمی‌کنیم. این لایه‌ها را فریز کرده و یک لایهٔ GAP و یک لایهٔ prediction یک کلاسه اضافه می‌کنیم. و نتایج آموزش را بعد از هر epoch نشان می‌دهیم:

```
# Load the InceptionV3 model without the top layer
base_model = InceptionV3(weights='imagenet', include_top=False)

# Add new top layers for our specific problem
x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(1, activation='sigmoid')(x)  # For binary classification

# Define the new model
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(
    train_dataset,
    epochs=10,
    validation_data=test_dataset
)
```

```
Epoch 1/10
582/582 [==============================] - 60s 95ms/step - loss: 0.2034 - accuracy: 0.9331 - val_loss: 0.0724 - val_accuracy: 0.9860
Epoch 2/10
582/582 [==============================] - 52s 90ms/step - loss: 0.0613 - accuracy: 0.9844 - val_loss: 0.0454 - val_accuracy: 0.9886
Epoch 3/10
582/582 [==============================] - 52s 90ms/step - loss: 0.0454 - accuracy: 0.9869 - val_loss: 0.0366 - val_accuracy: 0.9908
Epoch 4/10
582/582 [==============================] - 53s 90ms/step - loss: 0.0386 - accuracy: 0.9888 - val_loss: 0.0325 - val_accuracy: 0.9910
Epoch 5/10
582/582 [==============================] - 52s 89ms/step - loss: 0.0347 - accuracy: 0.9893 - val_loss: 0.0302 - val_accuracy: 0.9914
Epoch 6/10
582/582 [==============================] - 54s 92ms/step - loss: 0.0321 - accuracy: 0.9898 - val_loss: 0.0288 - val_accuracy: 0.9914
Epoch 7/10
582/582 [==============================] - 51s 88ms/step - loss: 0.0302 - accuracy: 0.9901 - val_loss: 0.0279 - val_accuracy: 0.9914
Epoch 8/10
582/582 [==============================] - 52s 89ms/step - loss: 0.0288 - accuracy: 0.9902 - val_loss: 0.0273 - val_accuracy: 0.9916
Epoch 9/10
582/582 [==============================] - 52s 89ms/step - loss: 0.0276 - accuracy: 0.9906 - val_loss: 0.0268 - val_accuracy: 0.9918
Epoch 10/10
582/582 [==============================] - 52s 90ms/step - loss: 0.0265 - accuracy: 0.9909 - val_loss: 0.0265 - val_accuracy: 0.9916
<keras.src.callbacks.History at 0x798b8011bfa0>
```

سپس مدل را روی داده‌های test ارزیابی می‌کنیم و نتیجه را گزارش می‌کنیم:

Evaluate model on the test set

```
# Evaluate the model
loss, accuracy = model.evaluate(test_dataset)
print(f'Test accuracy: {accuracy * 100:.2f}%')
```

```
146/146 [==============================] - 10s 70ms/step - loss: 0.0265 - accuracy: 0.9916
Test accuracy: 99.16%
```

سوال هفتم).....

ابتدا کتابخانه‌های لازم را import کرده و دیتاست را لود می‌کنیم:

Import libraries and load dataset

```
7] import numpy as np
   import cv2
   from tensorflow import keras
   from matplotlib import pyplot as plt

   # Load dataset
   (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

سپس داده‌های مربوط به صفر، ۱ و ۲ را جدا کرده و نرمالایز می‌کنیم:

Extract data for classed 0, 1, 2 and preprocess them

```
[3] train_indices = np.where((y_train == 0) | (y_train == 1) | (y_train == 2))
    test_indices = np.where((y_test == 0) | (y_test == 1) | (y_test == 2))

    x_train, y_train = x_train[train_indices], y_train[train_indices]
    x_test, y_test = x_test[test_indices], y_test[test_indices]

    # Preprocess images (normalize)
    x_train = x_train.astype(float) / 255.
    x_test = x_test.astype(float) / 255.
```

ویژگی‌های HuMoment را برای دیتاست حساب می‌کنیم:

Calculate HuMoments for dataset

```
[9] # Calculate Hu moments descriptors for each image
    def calculate_hu_moments(images):
        hu_moments = []
        for image in images:
            moments = cv2.moments(image)
            hu = cv2.HuMoments(moments).flatten()
            hu_moments.append(hu)
        return np.array(hu_moments)

    x_train_hu = calculate_hu_moments(x_train)
    x_test_hu = calculate_hu_moments(x_test)
```

Label هر دیتا را به شکل categorical در اورده تا احتمال کلاس ان به شکل vector در اید.

Convert class vectors to binary class matrices

```
[30] # Convert class vectors to binary class matrices
     num_classes = 10
     y_train_cat = keras.utils.to_categorical(y_train, num_classes)
     y_test_cat = keras.utils.to_categorical(y_test, num_classes)
```

مدل را ساخته و کامپایل می‌کنیم. یکی لایهٔ dense به همراه dropout می‌گذاریم. در نهایت summary آن را پرینت می‌کنیم:

Create model and compile it

```
1] # Define model
   model = keras.Sequential([
       keras.layers.Input(shape=x_train_hu[0].shape),
       keras.layers.Flatten(),
       keras.layers.Dense(units=num_classes*80, activation='relu'),
       keras.layers.Dropout(0.5),  # Adding dropout
       keras.layers.Dense(units=num_classes, activation='softmax')
   ])

   model.summary()

   # Compile model
   model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_20 (Flatten)        (None, 7)                 0

 dense_43 (Dense)            (None, 800)               6400

 dropout_17 (Dropout)        (None, 800)               0

 dense_44 (Dense)            (None, 10)                8010

=================================================================
Total params: 14410 (56.29 KB)
Trainable params: 14410 (56.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

مدل را روی داده‌های train آموزش می‌دهیم و نتیجهٔ هر epoch را نشان می‌دهیم:

Train model on training data

```
32] # Train model
    history = model.fit(x_train_hu, y_train_cat,
                        batch_size=100,
                        epochs=20,
                        validation_data=(x_test_hu, y_test_cat),
                        shuffle=True)
```

```
Epoch 1/20
187/187 [==============================] - 1s 4ms/step - loss: 1.0183 - accuracy: 0.5968 - val_loss: 0.6596 - val_accuracy: 0.7385
Epoch 2/20
187/187 [==============================] - 1s 3ms/step - loss: 0.5830 - accuracy: 0.7479 - val_loss: 0.5320 - val_accuracy: 0.8100
Epoch 3/20
187/187 [==============================] - 1s 3ms/step - loss: 0.4898 - accuracy: 0.8052 - val_loss: 0.4528 - val_accuracy: 0.8116
Epoch 4/20
187/187 [==============================] - 1s 3ms/step - loss: 0.4263 - accuracy: 0.8387 - val_loss: 0.3998 - val_accuracy: 0.8437
Epoch 5/20
187/187 [==============================] - 1s 3ms/step - loss: 0.3854 - accuracy: 0.8611 - val_loss: 0.3619 - val_accuracy: 0.8630
Epoch 6/20
187/187 [==============================] - 1s 3ms/step - loss: 0.3537 - accuracy: 0.8789 - val_loss: 0.3327 - val_accuracy: 0.8818
Epoch 7/20
187/187 [==============================] - 1s 3ms/step - loss: 0.3301 - accuracy: 0.8891 - val_loss: 0.3149 - val_accuracy: 0.9078
Epoch 8/20
187/187 [==============================] - 1s 3ms/step - loss: 0.3110 - accuracy: 0.8966 - val_loss: 0.2943 - val_accuracy: 0.8948
Epoch 9/20
187/187 [==============================] - 1s 3ms/step - loss: 0.2945 - accuracy: 0.9016 - val_loss: 0.2799 - val_accuracy: 0.9155
Epoch 10/20
187/187 [==============================] - 1s 3ms/step - loss: 0.2838 - accuracy: 0.9042 - val_loss: 0.2717 - val_accuracy: 0.8990
Epoch 11/20
187/187 [==============================] - 1s 3ms/step - loss: 0.2742 - accuracy: 0.9097 - val_loss: 0.2606 - val_accuracy: 0.9142
Epoch 12/20
187/187 [==============================] - 1s 3ms/step - loss: 0.2654 - accuracy: 0.9103 - val_loss: 0.2508 - val_accuracy: 0.9145
Epoch 13/20
187/187 [==============================] - 1s 3ms/step - loss: 0.2601 - accuracy: 0.9115 - val_loss: 0.2477 - val_accuracy: 0.9120
Epoch 14/20
187/187 [==============================] - 1s 3ms/step - loss: 0.2541 - accuracy: 0.9151 - val_loss: 0.2446 - val_accuracy: 0.9072
Epoch 15/20
187/187 [==============================] - 1s 3ms/step - loss: 0.2468 - accuracy: 0.9149 - val_loss: 0.2346 - val_accuracy: 0.9174
Epoch 16/20
187/187 [==============================] - 1s 5ms/step - loss: 0.2444 - accuracy: 0.9176 - val_loss: 0.2364 - val_accuracy: 0.9133
Epoch 17/20
187/187 [==============================] - 1s 6ms/step - loss: 0.2392 - accuracy: 0.9180 - val_loss: 0.2321 - val_accuracy: 0.9142
Epoch 18/20
187/187 [==============================] - 1s 6ms/step - loss: 0.2360 - accuracy: 0.9184 - val_loss: 0.2241 - val_accuracy: 0.9244
Epoch 19/20
```

مدل را روی داده‌های test ارزیابی می‌کنیم و نتیجه را پرینت می‌کنیم:

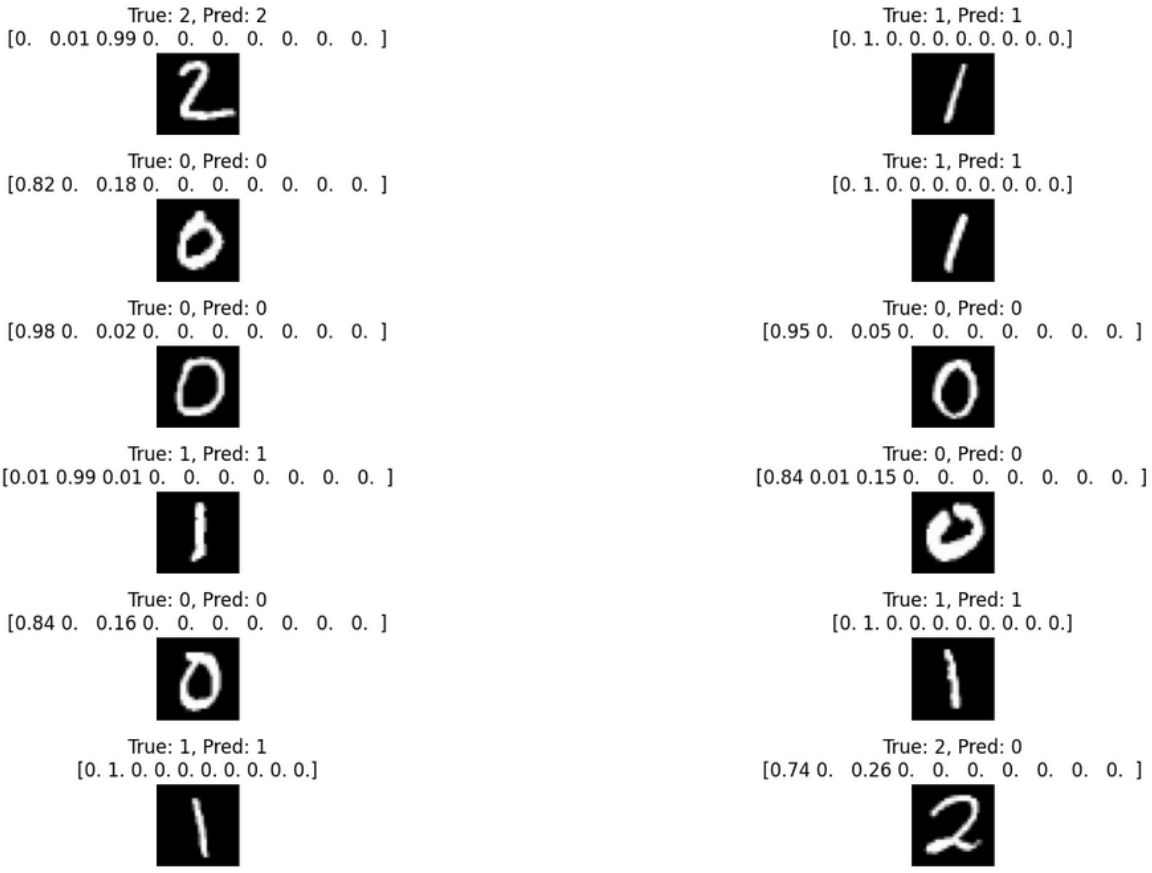
Evaluate model on the test set

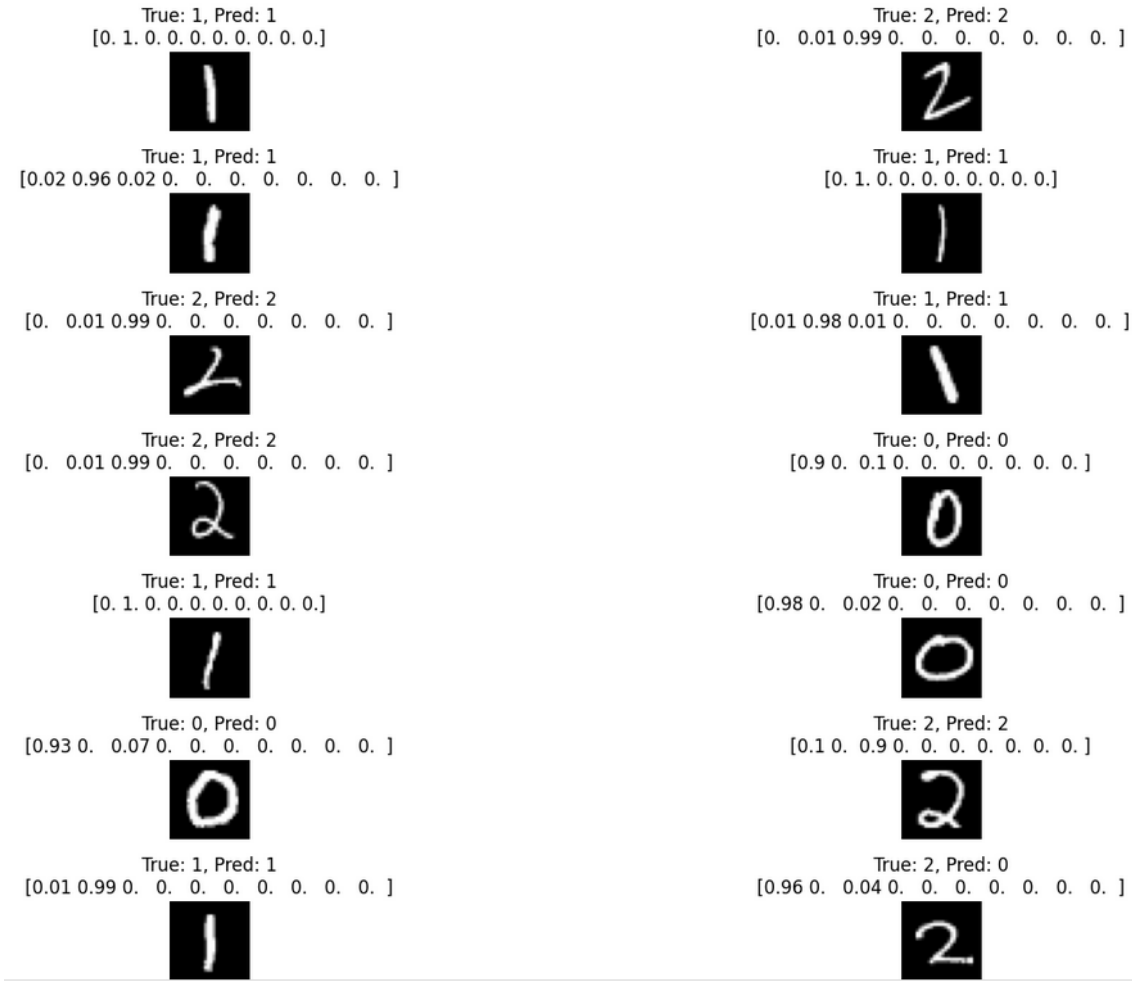
```
3] # Evaluate model on the test set
   test_loss, test_accuracy = model.evaluate(x_test_hu, y_test_cat)
   print(f'Test Accuracy: {test_accuracy:.4f}')

   99/99 [==============================] - 0s 3ms/step - loss: 0.2188 - accuracy: 0.9225
   Test Accuracy: 0.9225
```

۳۰ تا از داده‌های تست را به همراه prediction vector و label آن نشان می‌دهیم:

Show 30 images with predictions of the model

```
# Predict labels for test images
y_pred = model.predict(x_test_hu)

# Show 100 predictions with prediction vector
plt.figure(figsize=(20, 20))
count = 0
for i in range(len(x_test)):
    if count >= 30:
        break
    plt.subplot(15, 2, count + 1)
    plt.imshow(x_test[i], cmap='gray')
    pred_vector = np.round(y_pred[i], 2)
    plt.title(f'True: {np.argmax(y_test_cat[i])}, Pred: {np.argmax(y_pred[i])}\n{pred_vector}')
    plt.axis('off')
    count += 1

plt.tight_layout()
plt.show()
```





منابع:

- Link1: inceptionV1