دانشکده مهندسی کامپیوتر

تمرین سری ۱۳ بینایی کامپیوتر

نام درس

مبانی بینایی کامپیوتر

نام دانشجو

زهرا انوریان

نام استاد درس

دکتر محمدی

زمستان ۱۳۹۹

سوالات

۱. فرض کنید یک حجم ۳۲ در ۳۲ در ۱۰ وارد یک لایه کانوولوشنی که ۱۶ فیلتر ۹ در ۹ دارد، می‌شود. مقدار گسترش مرزها چقدر باید باشد تا طول و عرض خروجی این لایه تفاوتی نکند. تعداد پارامترهای این لایه را نیز به دست آورید.

پاسخ:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 = 32 - 9 + 2P + 1 = 32 \rightarrow 2P = 8 \rightarrow P = 4$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1 = 32 - 9 + 2P + 1 = 32 \rightarrow 2P = 8 \rightarrow P = 4$$

$$Parametes: F.F.K.D_1 + k(bias) = (9 * 9 * 16 * 10) + 16 = 12976$$

۲. اگر یک تصویر ۳ کاناله با ابعاد ۳۲ در ۳۲ وارد یک لایه کانوولوشنی با ۳ فیلتر ۵ در ۵ بدون صفر افزونه و با اندازه گام ۱ شوند ابعاد خروجی چه خواهد شد؟ اگر همان تصویر را به دو لایه کانولوشنی که هر دو ۳ فیلتر ۳ در ۳ بدون صفر افزونه و اندازه گام ۱ دارند بدهیم، ابعاد خروجی چه خواهد شد؟

پاسخ: خروجی اول: فیلتر (۵,۵)

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 = 32 - 5 + 0 + 1 = 28$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1 = 32 - 5 + 0 + 1 = 28 \rightarrow image.shape = (28, 28, 3)$$

خروجی دوم: فیلتر (۳,۳)

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 = 32 - 3 + 0 + 1 = 30$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1 = 32 - 3 + 0 + 1 = 30 \rightarrow image.shape = (30, 30, 3)$$

$$W_3 = \frac{W_2 - F + 2P}{S} + 1 = 30 - 3 + 0 + 1 = 28$$

$$H_3 = \frac{H_2 - F + 2P}{S} + 1 = 30 - 3 + 0 + 1 = 28 \rightarrow image.shape = (28, 28, 3)$$

۳. مطلع هستید که batch size، learning rate و epoch از سه پارامترهایی هستند که در شبکه‌های عمیق استفاده می‌شوند. تاثیر استفاده از هر کدام از این پارامترها را به صورت دقیق توضیح دهید.

پاسخ: دیتاست مورد نظر برای یادگیری را می‌توان به چند قسمت (batch) تقسیم کرد و جدا جدا پیش‌بینی کرد که به اندازه‌ی این قسمت‌ها batch_size می‌گویند. در واقع batch_size به تعداد نمونه‌های پردازش شده‌ی قبل به‌روز رسانی مدل گفته می‌شود. حال اگر batch_size برابر کل دیتاست یادگیری باشد الگوریتم یادگیری آن Batch Gradient Descent و اگر برابر ۱ باشد الگوریتم یادگیری آن Stochastic Gradient Descent و اگر برابر بیشتر از ۱ و کمتر از کل دیتاست باشد، الگوریتم آن برابر Mini-Batch Gradient Descent می‌باشد که این کار برای سریع‌تر کردن فرآیند یادگیری شبکه می‌باشد ولی epoch به تعداد دفعات پردازش کامل روی کل دیتاست گفته می‌شود که در واقع یک حلقه‌ایست که حلقه‌ی batch_size داخل آن قرار می‌گیرد و برای رسیدن به درصد دقت بهتر و یادگیری دقیق‌تر کمک می‌کند اما هردو هایپرپارامتر هستند و باید با آزمون و خطا و روش‌های موجود به مقدار مناسب برای آن‌ها برسیم. Learning rate نیز یکی دیگر از هایپرپارامترها و از مهم‌ترین آن‌ها می‌باشد. Learning rate کنترل می‌کند که مدل ما با چه سرعتی با مشکل سازگار شود و یاد بگیرد. هرچه این مقدار کم‌تر باشد تعداد epoch باید بیشتر باشد و طبیعتا برعکس، هرچه بیشتر باشد یعنی سرعت یادگیری زیادتر است و به تعداد epoch کمتری نیاز داریم اما اگر مقدار Learning rate بیش از حد زیاد باشند مدل به سرعت به یک راه حل غیربهینه همگرا شود و اگر خیلی مقدارش کوچک باشد، فرآیند گیر می‌کند.

۴. مزایای استفاده از لایه‌های کانوولوشنی نسبت به لایه‌های کاملا متصل در پردازش تصویر چیست؟

پاسخ: اگر در تعداد لایه‌های شبکه‌های عصبی را با لایه‌های کاملا متصل زیاد کنیم، به درصد دقت مطلوبی نمی‌رسیم زیرا مقدار هر نورون در لایه‌ی خروجی به تمام وزن‌ها در لایه‌ی قبلی وابسته می‌باشد اما در لایه‌ی کانولوشنی مقدار هر نورون در لایه‌ی خروجی به بخشی از نورون‌های ورودی متصل است و ویژگی‌های محلی را بدست می‌آورد و دانشی که بدست می‌اورد را با تمام نورون‌ها به اشتراک می‌گذارد.

۵. دلایل استفاده از لایه‌های ادغام (pooling) چیست؟

پاسخ: لایه‌ی pooling بر سر خروجی لایه‌ی کانوولوشنی قرار می‌گیرد و پیکسل‌های همسایه را با یکدیگر ترکیب می‌کند و باعث کاهش ابعاد نورون‌ها و کاهش تعداد پارامترهای شبکه می‌شود.

۶. نشان دهید که می‌توان به جای یک لایه کانوولوشنی با فیلترهای (۷,۷) از سه لایه متوالی با فیلترهای (۳,۳) استفاده کرد و این کار محاسبات کمتری دارد.

پاسخ: با فرض padding = 0 و stride = 1 داریم:

$$W_3 = 32 - 7 + 1 = 26 \ , H_3 = 32 - 7 + 1 = 26$$

حاصل ۳ لایه با فیلتر (۳,۳):

$$W_2 = 32 - 3 + 1 = 30 \ , H_2 = 32 - 3 + 1 = 30$$

$$W_3 = 30 - 3 + 1 = 28 \ , H_3 = 30 - 3 + 1 = 28$$
$$W_4 = 28 - 3 + 1 = 26 \ , H_4 = 28 - 3 + 1 = 26$$

در می‌یابیم که حاصل هر دو یکسان است پس می‌توان به جای یک لایه با یک فیلتر (۷,۷) از سه لایه با یک فیلتر (۳,۳) استفاده کرد. در نتیجه برای اعمال فیلتر (۷,۷) در تصویر (۳۲,۳۲)ای باید به ازای هر پیکسل عمل ضرب انجام دهیم پس در کل دارای محاسبات $26 * 26 * 49 = 33124$ هستیم اما فیلتر (۳,۳) دارای محاسبات $30 * 30 * 9 + 28 * 28 * 9 + 26 * 26 * 9 = 21240$ می‌باشد. پس استفاده از سه لایه با فیلتر (۳,۳) بسیار بهتر می‌باشد.

۱. ⌨ در این تمرین، انجام یک کار طبقه بندی تصویر، مد نظر است (۷۰).

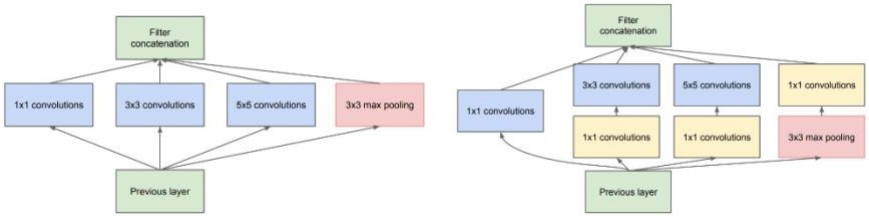
آزمایش‌های این تمرین با استفاده از مجموعه داده Fashion MNIST انجام میشود که یک مجموعه داده نسبتا بزرگ از تصاویر کوچک و شامل ۱۰ کلاس است. نیازی به دانلود این مجموعه داده نیست و با استفاده از دستور زیر به صورت خودکار دانلود می‌شود:

```
from keras.datasets import fashion_mnist

# The data, split between train and test sets:
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

با استفاده از یک شبکه عصبی کانوولوشنی این مجموعه داده را طبقه‌بندی کنید. در شبکه خود به تعداد مناسب از لایه‌های Conv2D، Pooling2D و Dense استفاده کنید. در شبکه خود میتوانید از inception module نیز استفاده کنید. برای پیاده سازی این ماژول به دو لینک زیر مراجعه کنید.

لینک ۱، لینک ۲



(a) Inception module, naïve version (b) Inception module with dimension reductions

چند پیشنهاد در طراحی شبکه:

- در عمل متداول است که بعد از هر دو لایه کانوولوشن یک لایه ادغام بیشینه استفاده شود و در لایه‌های کانوولوشن پس از هر ادغام تعداد فیلترها را ۲ برابر کنند.

- معمولاً از فیلتر با ابعاد (۳,۳) استفاده می‌شود زیرا هم بهتر عمل می‌کند و هم میتوان به جای یک فیلتر (۵,۵) از دو لایه متوالی (۳,۳) استفاده کرد که محاسبات کمتری دارد.
- در لایه اول می‌توان از فیلتر بزرگتر مانند (۷,۷) استفاده کرد.
- همچنین ایده‌های مربوط به شبکه‌های مطرح شده در کلاس درس نیز قابل استفاده هستند.
- لطفا توجه کنید که این موارد پیشنهاد هستند و از تجربه‌های آموزش شبکه‌های عصبی توسط کاربران به دست آمده‌اند. طراحی یک مدل مناسب به تلاش و هنر شما بستگی دارد.

پس از آموزش مدل، داده‌های آزمون را برای ارزیابی استفاده کنید و سپس دقت و ماتریس سردرگمی (confusion matrix) را برای نتایج گزارش کنید. مدل‌هایی که دقت‌های بالاتری را به دست بیاورند، نمره تشویقی خواهند گرفت. همچنین، ۱۰ تصویر آزمونی که بیشترین خطا را داشته‌اند (احتمال کلاس صحیح آن‌ها کمترین بوده است)، ترسیم کنید.

برای پیاده‌سازی کدهای خود می‌توانید از سرویس رایگان Google Colab استفاده کنید که نحوه استفاده از آن در لینک‌های زیر توضیح داده شده است:

https://mh-salari.me/google-colab/

https://deeplearning.ir/gpu.../

https://medium.com/better-programming/one-stop-guide-to-google-colab-d67c94d30516

همچنین مطالعه لینک زیر برای آشنایی با نحوه آموزش یک شبکه با استفاده از کتابخانه مفید Keras است.

https://towardsdatascience.com/writing-your-first-neural-net-in-less-than-30-lines-of-code-with-keras-18e160a35502

انتخاب‌های خود برای چگونگی تقسیم مجموعه آموزشی، ارزیابی، آزمایشی و... را به طور مختصر توضیح دهید. برای این سوال لطفا توضیحی تشریحی در مورد نحوه پیاده‌سازی کد، معماری شبکه طراحی شده و بررسی و مقایسه نتایج آورده شود.

پاسخ: معماری زیر که ورودی را ابتدا به یک لایه‌ی کانولوشن و سپس به لایه‌ی ادغام بیشینه می‌دهیم و سپس آن را تبدیل به یک بردار می‌کنیم و به لایه‌ی کاملا متصل که دارای ۱۰۰ نورون است می‌دهیم و در نهایت به لایه‌ی کاملا متصل بعدی به تعداد کلاس‌های دیتاست می‌دهیم، پیاده‌سازی می‌کنیم.

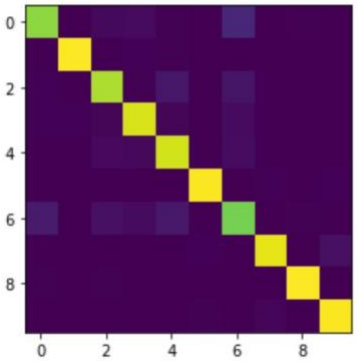
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 5408)              0
_________________________________________________________________
dense (Dense)                (None, 100)               540900
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

دقتی که از این معماری بدست می‌اوریم ۹۱.۸۵ درصد است.

```
Epoch 1/10
600/600 [==============================] - 9s 4ms/step - loss: 0.6231 - accuracy: 0.7841 - val_loss: 0.3402 - val_accuracy: 0.8739
Epoch 2/10
600/600 [==============================] - 2s 3ms/step - loss: 0.3036 - accuracy: 0.8933 - val_loss: 0.3034 - val_accuracy: 0.8899
Epoch 3/10
600/600 [==============================] - 2s 3ms/step - loss: 0.2557 - accuracy: 0.9083 - val_loss: 0.2872 - val_accuracy: 0.8958
Epoch 4/10
600/600 [==============================] - 2s 3ms/step - loss: 0.2187 - accuracy: 0.9216 - val_loss: 0.2742 - val_accuracy: 0.9010
Epoch 5/10
600/600 [==============================] - 2s 3ms/step - loss: 0.2015 - accuracy: 0.9260 - val_loss: 0.2645 - val_accuracy: 0.9048
Epoch 6/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1806 - accuracy: 0.9336 - val_loss: 0.2508 - val_accuracy: 0.9124
Epoch 7/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1617 - accuracy: 0.9406 - val_loss: 0.2487 - val_accuracy: 0.9127
Epoch 8/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1430 - accuracy: 0.9481 - val_loss: 0.2537 - val_accuracy: 0.9099
Epoch 9/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1293 - accuracy: 0.9540 - val_loss: 0.2535 - val_accuracy: 0.9131
Epoch 10/10
600/600 [==============================] - 2s 3ms/step - loss: 0.1202 - accuracy: 0.9554 - val_loss: 0.2458 - val_accuracy: 0.9185
```

و دارای ماتریس زیر می‌باشد.

```
[[820   1  23  27   6   2 114   0   7   0]
 [  0 984   0   8   5   0   2   0   1   0]
 [ 11   1 864   7  65   0  52   0   0   0]
 [  8   8  13 930  13   0  27   0   1   0]
 [  0   0  26  23 918   0  31   0   2   0]
 [  0   0   0   0   0 980   0   9   0  11]
 [ 70   0  45  29  67   0 783   0   6   0]
 [  0   0   0   0   0  10   0 946   1  43]
 [  4   2   5   3   0   1   1   2 982   0]
 [  1   0   0   0   0   4   0  17   0 978]]
[[0.82  0.001 0.023 0.027 0.006 0.002 0.114 0.    0.007 0.   ]
 [0.    0.984 0.    0.008 0.005 0.    0.002 0.    0.001 0.   ]
 [0.011 0.001 0.864 0.007 0.065 0.    0.052 0.    0.    0.   ]
 [0.008 0.008 0.013 0.93  0.013 0.    0.027 0.    0.001 0.   ]
 [0.    0.    0.026 0.023 0.918 0.    0.031 0.    0.002 0.   ]
 [0.    0.    0.    0.    0.    0.98  0.    0.009 0.    0.011]
 [0.07  0.    0.045 0.029 0.067 0.    0.783 0.    0.006 0.   ]
 [0.    0.    0.    0.    0.    0.01  0.    0.946 0.001 0.043]
 [0.004 0.002 0.005 0.003 0.    0.001 0.001 0.002 0.982 0.   ]
 [0.001 0.    0.    0.    0.    0.004 0.    0.017 0.    0.978]]
```



حال معماری inception module را نیز پیاده‌سازی و امتحان می‌کنیم.

```
Model: "model_3"
```

Layer (type)	Output Shape	Param #	Connected to
input_5 (InputLayer)	[(None, 28, 28, 1)]	0	
conv2d_37 (Conv2D)	(None, 28, 28, 96)	192	input_5[0][0]
conv2d_39 (Conv2D)	(None, 28, 28, 16)	32	input_5[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 28, 28, 1)	0	input_5[0][0]
conv2d_36 (Conv2D)	(None, 28, 28, 64)	128	input_5[0][0]
conv2d_38 (Conv2D)	(None, 28, 28, 128)	110720	conv2d_37[0][0]
conv2d_40 (Conv2D)	(None, 28, 28, 32)	12832	conv2d_39[0][0]
conv2d_41 (Conv2D)	(None, 28, 28, 32)	64	max_pooling2d_6[0][0]
concatenate_6 (Concatenate)	(None, 28, 28, 256)	0	conv2d_36[0][0] conv2d_38[0][0] conv2d_40[0][0] conv2d_41[0][0]
flatten_1 (Flatten)	(None, 200704)	0	concatenate_6[0][0]
dense (Dense)	(None, 100)	20070500	flatten_1[0][0]
dense_1 (Dense)	(None, 10)	1010	dense[0][0]

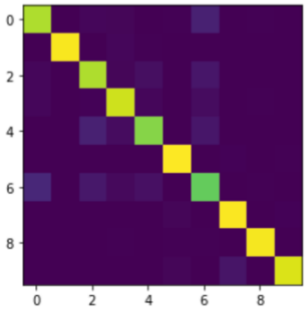
```
Total params: 20,195,478
Trainable params: 20,195,478
Non-trainable params: 0
```

که با استفاده از این معماری به درصد دقت ۹۰.۳۹ رسیدیم.

```
Epoch 1/10
600/600 [==============================] - 19s 27ms/step - loss: 0.5964 - accuracy: 0.7979 - val_loss: 0.3369 - val_accuracy: 0.8799
Epoch 2/10
600/600 [==============================] - 16s 27ms/step - loss: 0.2638 - accuracy: 0.9031 - val_loss: 0.3005 - val_accuracy: 0.8924
Epoch 3/10
600/600 [==============================] - 16s 27ms/step - loss: 0.1956 - accuracy: 0.9283 - val_loss: 0.2986 - val_accuracy: 0.8995
Epoch 4/10
600/600 [==============================] - 16s 27ms/step - loss: 0.1513 - accuracy: 0.9441 - val_loss: 0.2775 - val_accuracy: 0.9086
Epoch 5/10
600/600 [==============================] - 16s 27ms/step - loss: 0.1134 - accuracy: 0.9576 - val_loss: 0.2954 - val_accuracy: 0.9108
Epoch 6/10
600/600 [==============================] - 16s 27ms/step - loss: 0.0864 - accuracy: 0.9687 - val_loss: 0.3604 - val_accuracy: 0.9049
Epoch 7/10
600/600 [==============================] - 17s 28ms/step - loss: 0.0664 - accuracy: 0.9756 - val_loss: 0.3846 - val_accuracy: 0.9076
Epoch 8/10
600/600 [==============================] - 17s 28ms/step - loss: 0.0527 - accuracy: 0.9814 - val_loss: 0.4626 - val_accuracy: 0.9026
Epoch 9/10
600/600 [==============================] - 17s 28ms/step - loss: 0.0446 - accuracy: 0.9834 - val_loss: 0.4391 - val_accuracy: 0.9033
Epoch 10/10
600/600 [==============================] - 17s 28ms/step - loss: 0.0310 - accuracy: 0.9898 - val_loss: 0.5419 - val_accuracy: 0.9039
```

و دارای ماتریس زیر می‌باشد.

```
[[865   0  18  14   3   4  92   0   4   0]
 [  1 973   0  17   6   0   2   0   1   0]
 [ 16   2 868  13  38   1  61   0   1   0]
 [ 18   3  11 913  19   1  31   0   4   0]
 [  3   1  92  34 807   1  61   0   1   0]
 [  0   0   0   0   0 984   0  11   0   5]
 [112   1  63  25  43   1 750   0   5   0]
 [  0   0   0   0   0  13   0 977   0  10]
 [  1   0   2   7   2   5   7   3 971   2]
 [  0   0   0   0   0  13   2  54   0 931]]
[[0.865 0.    0.018 0.014 0.003 0.004 0.092 0.    0.004 0.    ]
 [0.001 0.973 0.    0.017 0.006 0.    0.002 0.    0.001 0.    ]
 [0.016 0.002 0.868 0.013 0.038 0.001 0.061 0.    0.001 0.    ]
 [0.018 0.003 0.011 0.913 0.019 0.001 0.031 0.    0.004 0.    ]
 [0.003 0.001 0.092 0.034 0.807 0.001 0.061 0.    0.001 0.    ]
 [0.    0.    0.    0.    0.    0.984 0.    0.011 0.    0.005]
 [0.112 0.001 0.063 0.025 0.043 0.001 0.75  0.    0.005 0.    ]
 [0.    0.    0.    0.    0.    0.013 0.    0.977 0.    0.01 ]
 [0.001 0.    0.002 0.007 0.002 0.005 0.007 0.003 0.971 0.002]
 [0.    0.    0.    0.    0.    0.013 0.002 0.054 0.    0.931]]
```

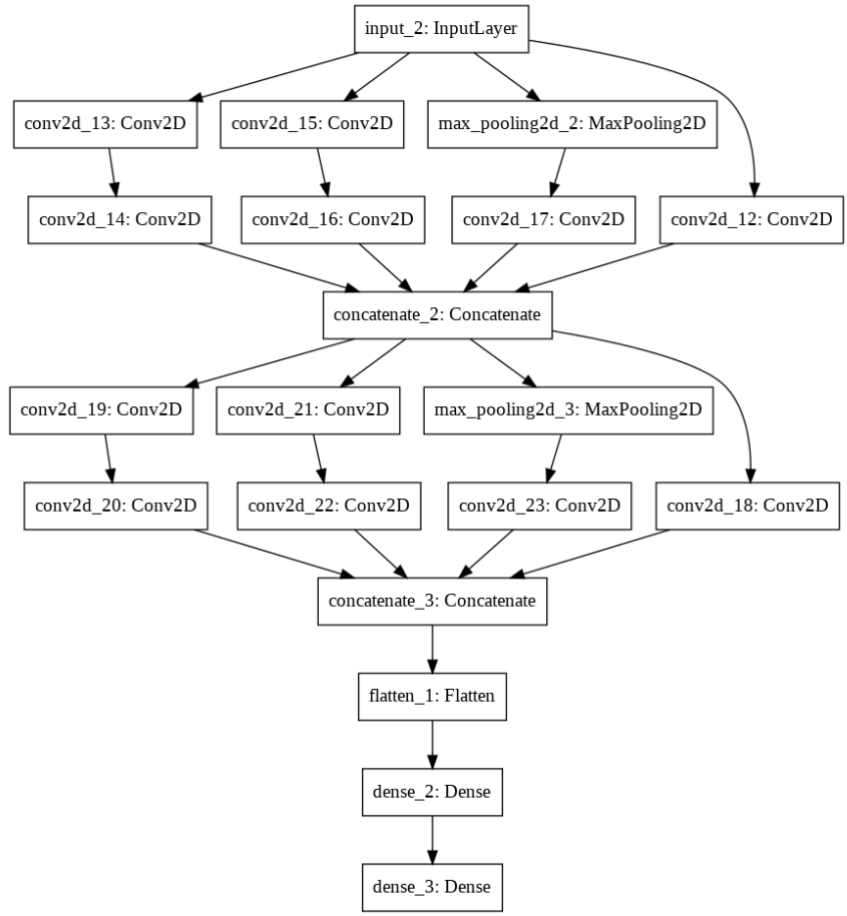


حال به معماری قبلی یک inception module دیگری اضافه می‌کنیم و آن را امتحان می‌کنیم.

Model: "model_4"

Layer (type)	Output Shape	Param #	Connected to
input_6 (InputLayer)	[(None, 28, 28, 1)]	0	
conv2d_43 (Conv2D)	(None, 28, 28, 96)	192	input_6[0][0]
conv2d_45 (Conv2D)	(None, 28, 28, 16)	32	input_6[0][0]
max_pooling2d_7 (MaxPooling2D)	(None, 28, 28, 1)	0	input_6[0][0]
conv2d_42 (Conv2D)	(None, 28, 28, 64)	128	input_6[0][0]
conv2d_44 (Conv2D)	(None, 28, 28, 128)	110720	conv2d_43[0][0]
conv2d_46 (Conv2D)	(None, 28, 28, 32)	12832	conv2d_45[0][0]
conv2d_47 (Conv2D)	(None, 28, 28, 32)	64	max_pooling2d_7[0][0]
concatenate_7 (Concatenate)	(None, 28, 28, 256)	0	conv2d_42[0][0] conv2d_44[0][0] conv2d_46[0][0] conv2d_47[0][0]
conv2d_49 (Conv2D)	(None, 28, 28, 128)	32896	concatenate_7[0][0]
conv2d_51 (Conv2D)	(None, 28, 28, 32)	8224	concatenate_7[0][0]
max_pooling2d_8 (MaxPooling2D)	(None, 28, 28, 256)	0	concatenate_7[0][0]
conv2d_48 (Conv2D)	(None, 28, 28, 128)	32896	concatenate_7[0][0]
conv2d_50 (Conv2D)	(None, 28, 28, 192)	221376	conv2d_49[0][0]
conv2d_52 (Conv2D)	(None, 28, 28, 96)	76896	conv2d_51[0][0]
conv2d_53 (Conv2D)	(None, 28, 28, 64)	16448	max_pooling2d_8[0][0]
concatenate_8 (Concatenate)	(None, 28, 28, 480)	0	conv2d_48[0][0] conv2d_50[0][0] conv2d_52[0][0] conv2d_53[0][0]
flatten_2 (Flatten)	(None, 376320)	0	concatenate_8[0][0]
dense_2 (Dense)	(None, 100)	37632100	flatten_2[0][0]
dense_3 (Dense)	(None, 10)	1010	dense_2[0][0]

Total params: 38,145,814
Trainable params: 38,145,814
Non-trainable params: 0



که با استفاده از این معماری به درصد دقت ۹۱.۸۳ رسیدیم.

```
Epoch 1/10
600/600 [==============================] - 49s 79ms/step - loss: 0.6355 - accuracy: 0.8020 - val_loss: 0.2888 - val_accuracy: 0.8930
Epoch 2/10
600/600 [==============================] - 48s 81ms/step - loss: 0.2266 - accuracy: 0.9161 - val_loss: 0.2384 - val_accuracy: 0.9150
Epoch 3/10
600/600 [==============================] - 49s 82ms/step - loss: 0.1551 - accuracy: 0.9418 - val_loss: 0.2432 - val_accuracy: 0.9151
Epoch 4/10
600/600 [==============================] - 50s 83ms/step - loss: 0.1029 - accuracy: 0.9626 - val_loss: 0.2408 - val_accuracy: 0.9201
Epoch 5/10
600/600 [==============================] - 50s 83ms/step - loss: 0.0656 - accuracy: 0.9768 - val_loss: 0.2884 - val_accuracy: 0.9200
Epoch 6/10
600/600 [==============================] - 50s 84ms/step - loss: 0.0414 - accuracy: 0.9853 - val_loss: 0.3403 - val_accuracy: 0.9169
Epoch 7/10
600/600 [==============================] - 50s 84ms/step - loss: 0.0266 - accuracy: 0.9910 - val_loss: 0.3782 - val_accuracy: 0.9191
Epoch 8/10
600/600 [==============================] - 50s 84ms/step - loss: 0.0180 - accuracy: 0.9937 - val_loss: 0.4499 - val_accuracy: 0.9158
Epoch 9/10
600/600 [==============================] - 50s 84ms/step - loss: 0.0178 - accuracy: 0.9939 - val_loss: 0.4833 - val_accuracy: 0.9173
Epoch 10/10
600/600 [==============================] - 50s 84ms/step - loss: 0.0129 - accuracy: 0.9962 - val_loss: 0.5238 - val_accuracy: 0.9183
```

و دارای ماتریس زیر می‌باشد.

```
[[859   2  18  18   4   1  92   0   6   0]
 [  2 979   1  12   3   0   3   0   0   0]
 [ 17   2 846  11  53   0  69   0   2   0]
 [ 14   2   8 932  20   0  23   0   1   0]
 [  2   1  35  20 884   0  57   0   1   0]
 [  0   0   0   0   0 985   0   7   0   8]
 [ 92   1  44  31  48   0 778   0   6   0]
 [  0   0   0   0   0   3   0 983   0  14]
 [  2   2   2   3   1   0   7   5 978   0]
 [  0   0   0   0   0   5   1  35   0 959]]
[[0.859 0.002 0.018 0.018 0.004 0.001 0.092 0.    0.006 0.   ]
 [0.002 0.979 0.001 0.012 0.003 0.    0.003 0.    0.    0.   ]
 [0.017 0.002 0.846 0.011 0.053 0.    0.069 0.    0.002 0.   ]
 [0.014 0.002 0.008 0.932 0.02  0.    0.023 0.    0.001 0.   ]
 [0.002 0.001 0.035 0.02  0.884 0.    0.057 0.    0.001 0.   ]
 [0.    0.    0.    0.    0.    0.985 0.    0.007 0.    0.008]
 [0.092 0.001 0.044 0.031 0.048 0.    0.778 0.    0.006 0.   ]
 [0.    0.    0.    0.    0.    0.003 0.    0.983 0.    0.014]
 [0.002 0.002 0.002 0.003 0.001 0.    0.007 0.005 0.978 0.   ]
 [0.    0.    0.    0.    0.    0.005 0.001 0.035 0.    0.959]]
```