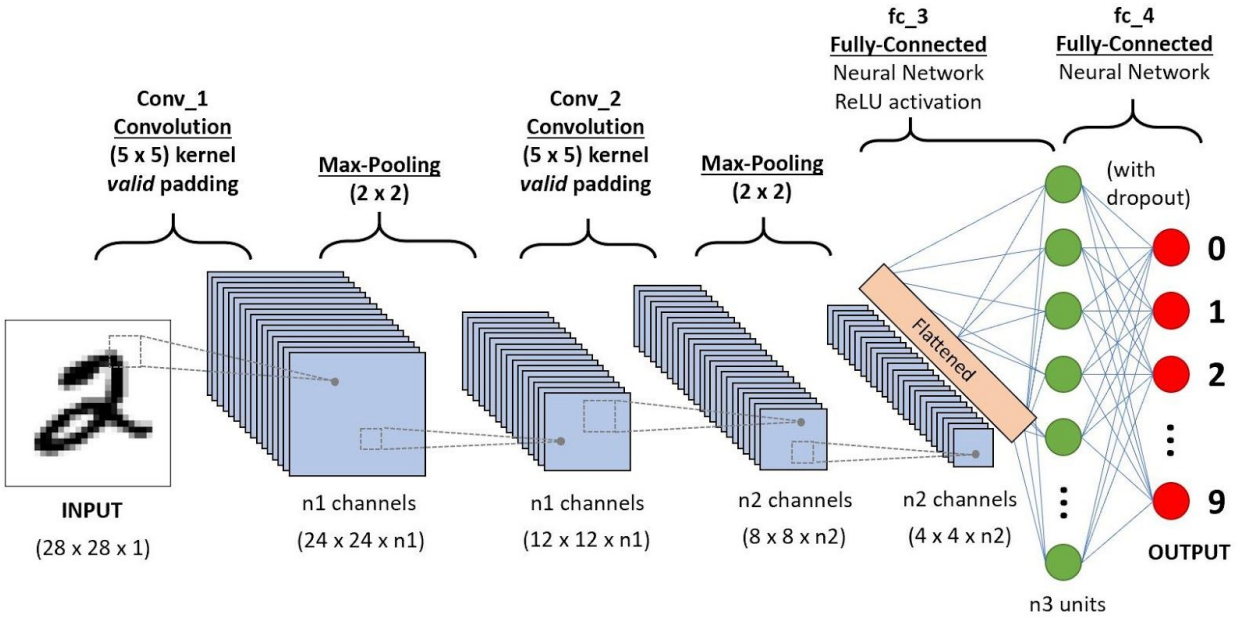
گزارش پروژه دوم یادگیری عمیق (CNN)

21 آذر ماه 1399

غزاله محمودی

96522249



منابع :

https://blog.keras.io/building-autoencoders-in-keras.html

لینک کدهای اجرایی:

- Section 1
  https://colab.research.google.com/drive/1QnSBoomMOvMCcFACAl3FGO-VeirzQOWZ?usp=sharing
- Section 2
  https://colab.research.google.com/drive/1B2O5yGCXF0tQJ0nswCZ2FPKjyYCiqPbG?usp=sharing
- Section 3
  https://colab.research.google.com/drive/1F1I7jNkfp0NyOG7_o2sQ_UAX6a-xVWbh?usp=sharing

بخش اول

در این بخش قصد داریم با استفاده از ماژول کراس با دیتاست mnist کار کنیم . ابتدا به کمک ماژول های کراس دیتا ست را ذخیره می کنیم .

```
from keras.datasets import mnist
from keras.utils import np_utils, to_categorical
```

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

همچنین با توجه به مواردی که در درس یاد گرفتیم برای اینکه تاثیر همه ورودی ها تقریبا یکسان و در نرمال باشد داده ها نرمال کرده (بر 255 تقسیم میکنیم) تا مقدارشان عددی بین صفر و یک شود.

تسک ما در واقع classification است. داده های خروجی شامل اعداد بین 0 تا 9 هستند که نشان دهنده کلاس آن داده می باشد. اگر از خوده اعداد 0 تا 9 استفاده کنیم این اختلاف اعداد روی مدل تاثیر می گذارد. بنابراین داده ها را one hot encoding می کنیم تا از این تاثیر جلوگیری کنیم .
و این کد گذاری به صورت مقابل است . برای مثال عدد یک به صورت [0, 0, 0, 0 , 0, 0, 0, 0, 1, 0] کد گذاری می شود.

```
y_train =  to_categorical(train_labels) #one hot encoding
y_test = to_categorical(test_labels)
```

همچنین لایه خروجی 10 نورون دارد که احتمال هر عدد را به ما نمایش میدهد.

ساختار شبکه :

- شبکه cnn با احتساب لایه ورودی و خروجی به صورت زیر طراحی کردم
  - لایه convolution با filter size = 32 , kernel size = (3, 3)
  - لایه max pooling با pool size = (2, 2)
  - batch normalization
  - لایه convolution با filter size = 32, kernel size = (3, 3)
  - لایه max pooling با pool size = (2, 2)
  - batch normalization
  - flatten
  - لایه fully connected با 64 نورون
  - لایه خروجی classification با 10 نورون با تابع فعال سازی softmax
  - برای همه لایه ها به جز لایه خروجی از تابع فعال ساز relu استفاده کردم.
  - چون تسک classification است در لایه خروجی از تابع فعال ساز softmax استفاده کردم.
- تعداد نورون های این شبکه :
  - تعداد پارامتر های cnn برابر 61379 می باشد.
- زمان اجرا در صورتی از GPU استفاده کنیم بسیار خوب است.
- این شبکه 6 لایه می باشد.
- بهترین تابع فعال ساز که نتیجه مناسب تری می داد relu بود.(نسبت به سایر تابع ها , sigmoid (tanh, softmax
- از k-fold cross validation استفاده کردم و k = 3 گرفتم.
- آموزش را با 10 ایتریشن انجام دادم.
- از مقدار batch = 2048 استفاده کردم. (batch size به معنا تعداد داده آموزشی هستند که یم ایتریشن حساب می شوند)
- همچنین در model.fit از validation استفاده کردم که علاوه بر داده تست پایانی بعد پایان هر ایتریشن دقت مدل را به داده ها روی داده ها تستی (k-fold) به ما بدهد.

نتایج و تحلیل ها :

- سرعت اجرایی cnn از mlp روی cpu کمتر است و اجرای آن زمان بیشتری می گیرد. البته در صورتی که از GPU استفاده کنیم، سرعت به حد بسیار خوبی افزایش می یابد.
- در cnn با یک شبکه با اندازه نسبتا کوچک، تعداد ایتریشن های معقول می توان به دقت خیلی خوبی دست پیدا کرد. در حالی که برای رسیدن به همین دقت باید از شبکه mlp بزرگتری با ایتریشن بیشتر استفاده کنیم تا به نتایج یکسان برسیم.
- تعداد پارامتر ها در cnn به وضوح خیلی کمتر از mlp می باشد و همین عامل از overfit شدن جلوگیری میکند.
- به کمک max pooling هم می توان اندازه را تا حدودی کاهش داد هم از overfit کردن جلوگیری کرد. همچنین ذات شبکه convolution کار استخراج فیچررا انجام می دهد و نسبت به نویز مقاوم تر است.
- استفاده از batch normalization باعث بهبود نتایج شد.
- با توجه به اینکه تسک ما کلاسه بندی چند کلاسه بود categorical cross entropy گزینه مناسبی برای محاسبه loss بود.
- همچنین طبق تجربه و توصیه از Optimizer عه adam برای محاسبه گرادیان استفاده شد.
- تعداد پارامتر های cnn برابر 61379 می باشد در حالی که با یک شبکه 6 لایه مواجه هستیم. شبکه mlp تمرین قبل من 109386 پارامتر دارد در حالی با احتساب لایه خروجی 4 لایه است.

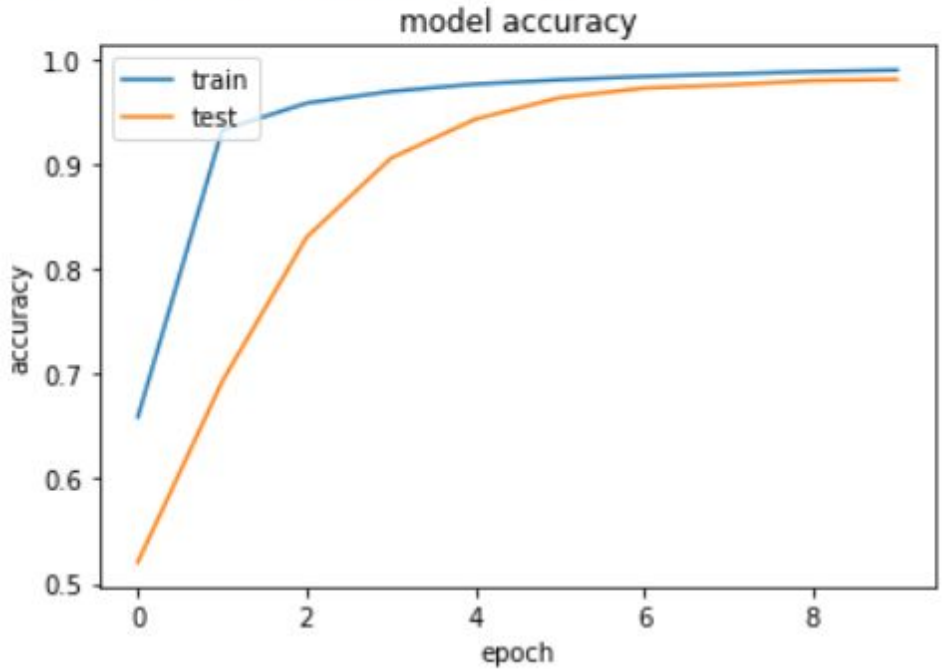
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
```

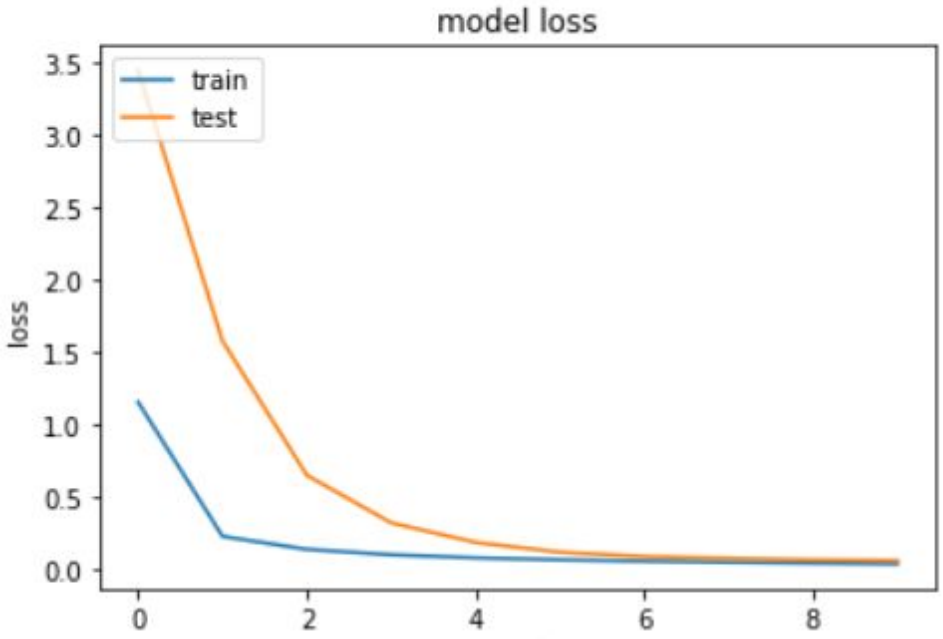
نتایج تصویری حاصل از آموزش شبکه :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_32 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
batch_normalization_15 (Batc (None, 26, 26, 32)        128
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 11, 11, 32)        9248
_________________________________________________________________
batch_normalization_16 (Batc (None, 11, 11, 32)        128
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 5, 5, 32)          0
_________________________________________________________________
flatten_16 (Flatten)         (None, 800)               0
_________________________________________________________________
dense_32 (Dense)             (None, 64)                51264
_________________________________________________________________
dense_33 (Dense)             (None, 10)                650
=================================================================
Total params: 61,738
Trainable params: 61,610
Non-trainable params: 128
```

دقت مدل و نمودار های تحلیلی :

```
Test loss : 0.059760645031929016
Test accuracy: 98.18999767303467
```





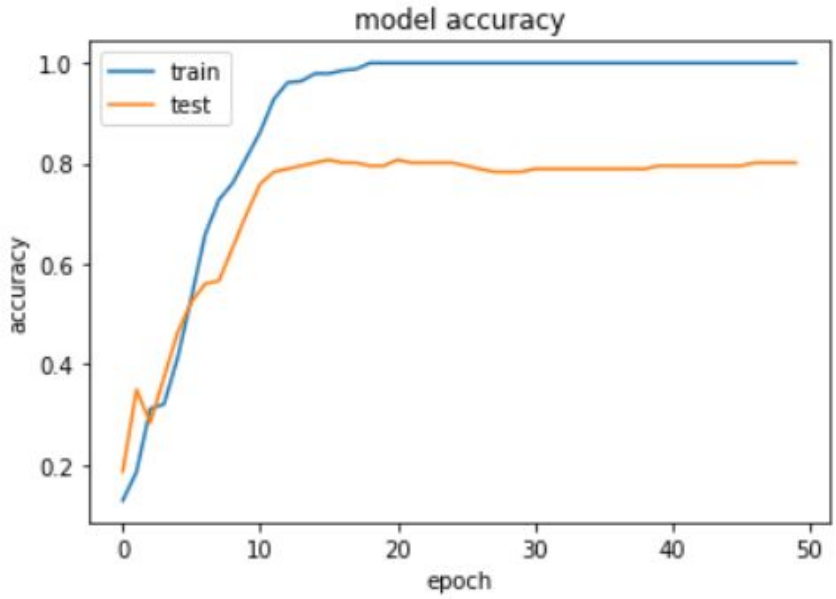
بخش دوم

با کاهش تعداد داده های آموزشی و ثابت نگه داشتن ساختار شبکه و افزایش تعداد epoch ها تا تعدادی که قطعا موجب overfit شود(به عنوان مثال 50)، زمینه بیش برازش را فراهم می کنیم.

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 24, 24, 64)	1664
conv2d_11 (Conv2D)	(None, 20, 20, 64)	102464
flatten_5 (Flatten)	(None, 25600)	0
dense_5 (Dense)	(None, 10)	256010

Total params: 360,138
Trainable params: 360,138
Non-trainable params: 0

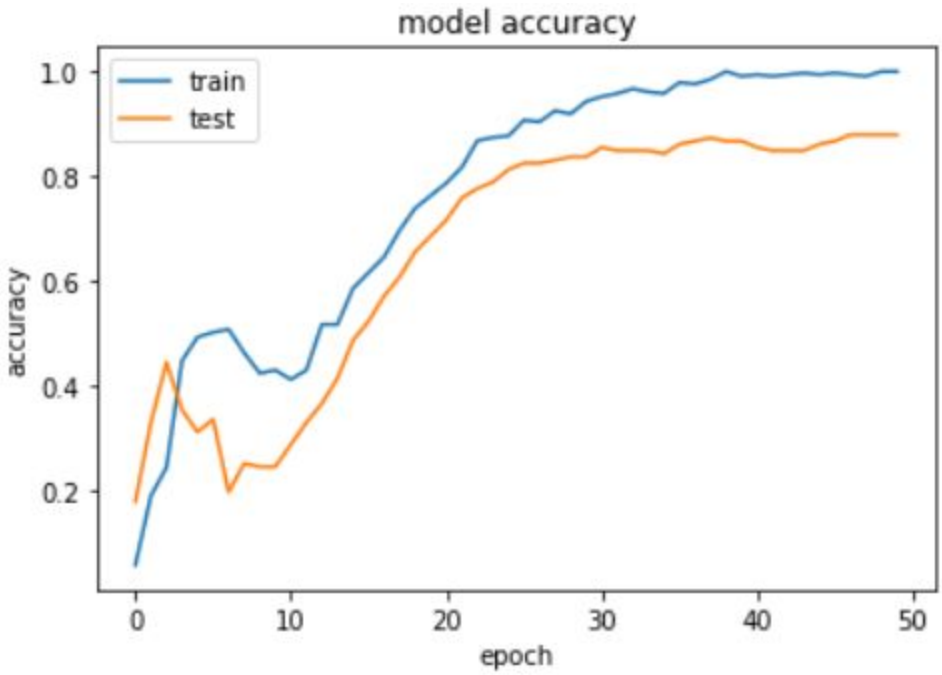
مشاهده می شود که از جایی به بعد دقت شبکه روی داده های آموزشی دقیقا 100 درصد می شود در حالی که نتیجه دقت داده تست حوالی 70 تا 80 درصد است.



با اضافه کردن فقط 1 لایه dropout با پارامتر 0.5 تا حدودی اثر منفی overfit را توانستم کم کنم . لازم به ذکر هنگام استفاده از drop out باید تعداد ایتریشن ها را کمی از حالت عادی بالاتر در نظر بگیریم.

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 24, 24, 64)	1664
dropout_6 (Dropout)	(None, 24, 24, 64)	0
conv2d_19 (Conv2D)	(None, 20, 20, 64)	102464
flatten_9 (Flatten)	(None, 25600)	0
dense_9 (Dense)	(None, 10)	256010

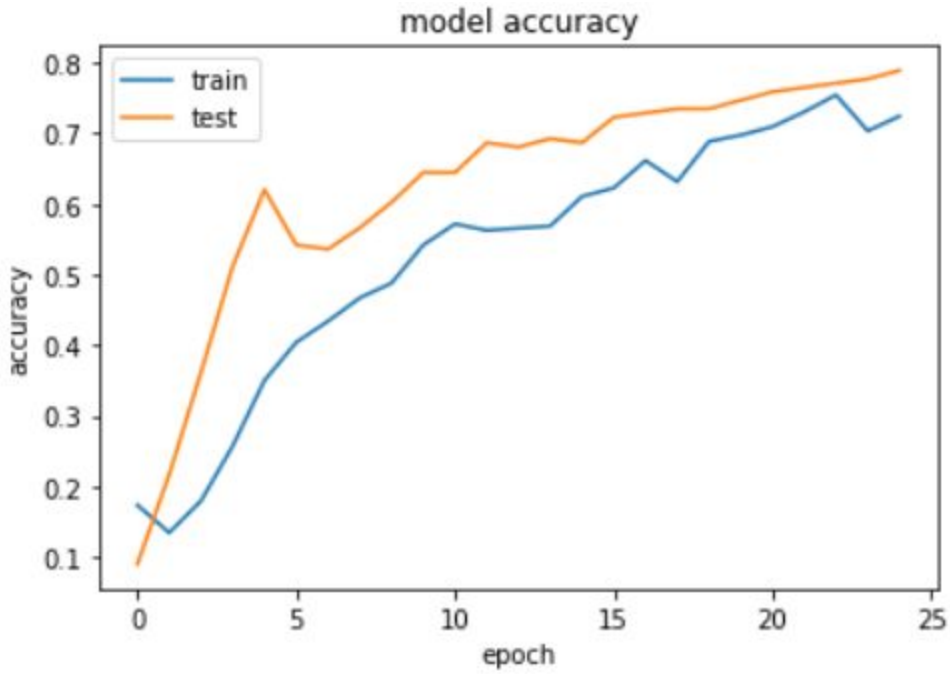
Total params: 360,138
Trainable params: 360,138
Non-trainable params: 0



اما این پارامتر به تنهایی قادر به اصلاح وضع موجود نیست. باید ساختار شبکه که شامل اندازه شبکه و تعداد epoch ها می شود را به طور مناسبی انتخاب کنم.
همچنین تعداد داده های آموزشی و تستی که پارامتر تاثیر گذاری هستند را به صورت درست تعیین کنم تا از overfit ممانعت کنم.
در ادامه علاوه بر اضافه کردن لایه drop out تعداد epoch ها را نیز کاهش داده و همچنین اندازه شبکه (سایز فیلتر ها) را هم عدد منطقی تری در نظر گرفتم. نتایج به شکل زیر حاصل شد.

Layer (type)	Output Shape	Param #
conv2d_29 (Conv2D)	(None, 24, 24, 8)	208
dropout_9 (Dropout)	(None, 24, 24, 8)	0
conv2d_30 (Conv2D)	(None, 20, 20, 8)	1608
flatten_14 (Flatten)	(None, 3200)	0
dense_14 (Dense)	(None, 10)	32010

Total params: 33,826
Trainable params: 33,826
Non-trainable params: 0

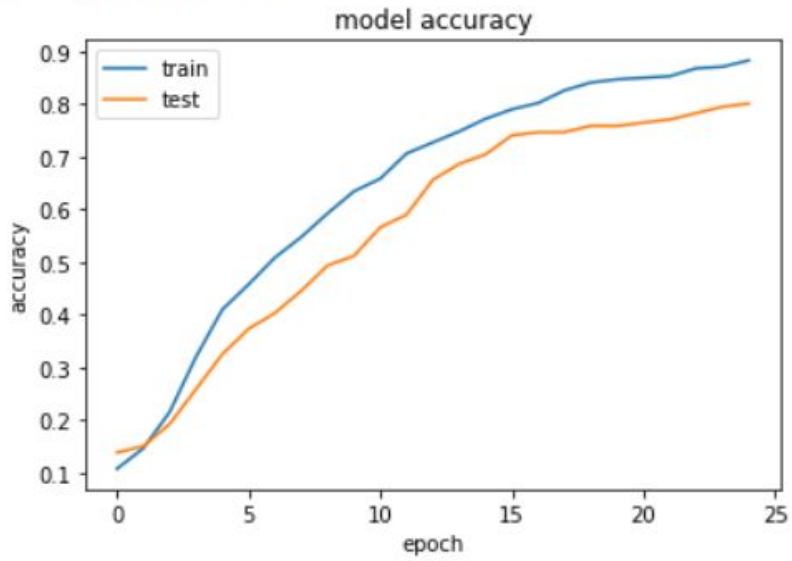


همچنین اضافه کردن لایه max pooling و batch normalization از overfit به خوبی جلوگیری می کند.

Layer (type)	Output Shape	Param #
conv2d_49 (Conv2D)	(None, 24, 24, 8)	208
max_pooling2d_11 (MaxPooling	(None, 12, 12, 8)	0
batch_normalization_2 (Batch	(None, 12, 12, 8)	32
conv2d_50 (Conv2D)	(None, 8, 8, 8)	1608
flatten_23 (Flatten)	(None, 512)	0
dense_23 (Dense)	(None, 10)	5130

Total params: 6,978
Trainable params: 6,962
Non-trainable params: 16

```
Test loss : 1.4153251647949219
Test accuracy: 75.0
```



بخش سوم

در این بخش ابتدا به تصویر نویز گوسی از مقدار کم تا مقدار بالا اضافه کردم.

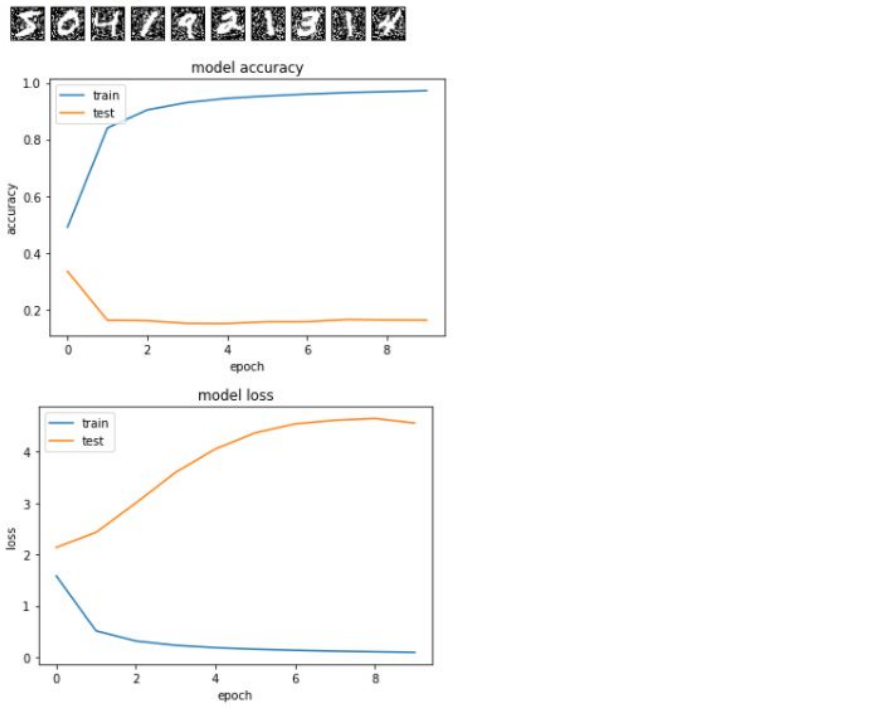
```
def add_noise(x_train, x_test, noise_factor):

    x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
    x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

    x_train_noisy = np.clip(x_train_noisy, 0., 1.)
    x_test_noisy = np.clip(x_test_noisy, 0., 1.)

    return x_train_noisy, x_test_noisy
```

سپس با شبکه قسمت اول آن را آموزش دادم.

در صورتی که از batch normalization در قسمت آموزش استفاده کنم نتایج نه چندان مناسبی به دست آمد.

نتایج به صورت زیر است :

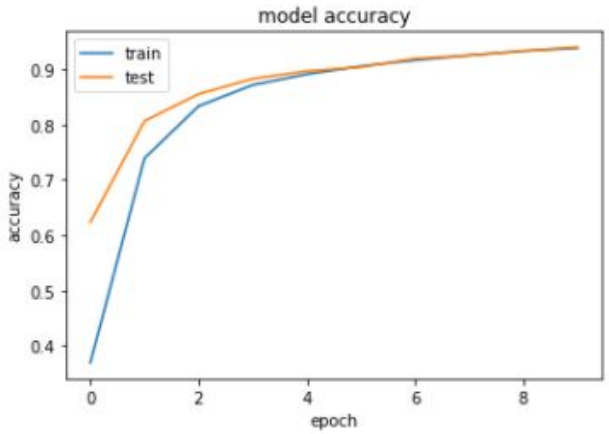
```
313/313 [==============================] - 1s 2ms/step - loss: 4.5535 - accuracy: 0.1639
Test loss : 4.553471565246582
Test accuracy: 16.39000028371811
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: MatplotlibDeprecationWarning: Ad
  if sys.path[0] == '':
```

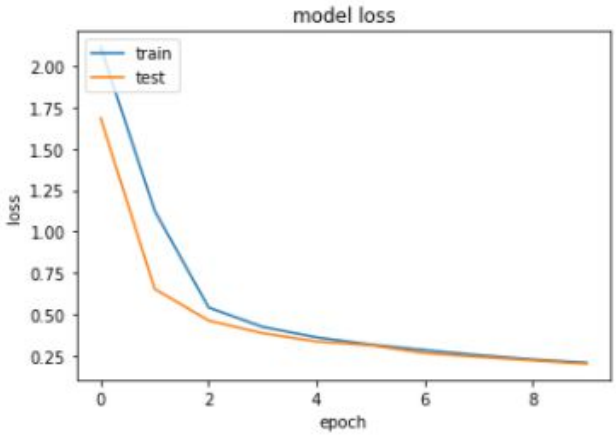


به عنوان اولین تلاش برای رفع نویز مدل سوال 1 را بدون batch normalization آموزش دادم و مشخص شد به دلیل لایه max pooling و لایه convolution مقاومت نسبتا خوبی در مقابل نویز دارد و از دقت حوالی 98 درصد روی داده های بدون نویز، در داده های نویزی با نویز 0.9 به دقت حوالی 91 درصد رسید. بدیهی است اگر درصد نویز کمتر باشد، به تناسب دقت بالاتر است.

نویز 0.9 برای داده آموزشی و داده تست :

```
_________________________________________________________________
313/313 [==============================] - 1s 2ms/step - loss: 0.1904 - accuracy: 0.9441
Test loss : 0.19036470353603363
Test accuracy: 94.4100022315979
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: MatplotlibDeprecationWarni
  if sys.path[0] == '':
```

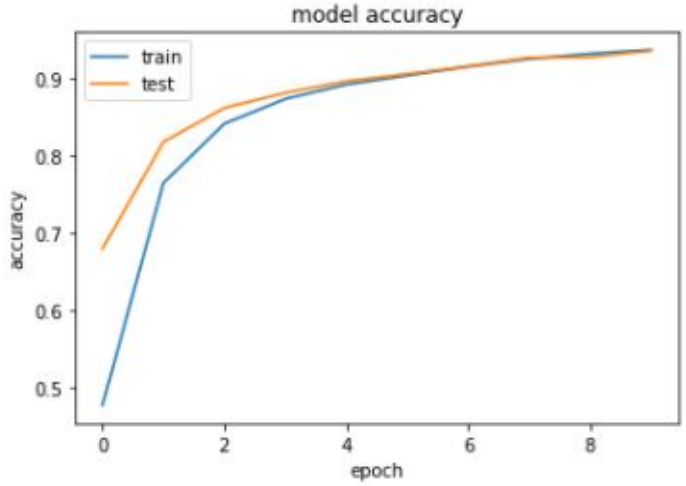


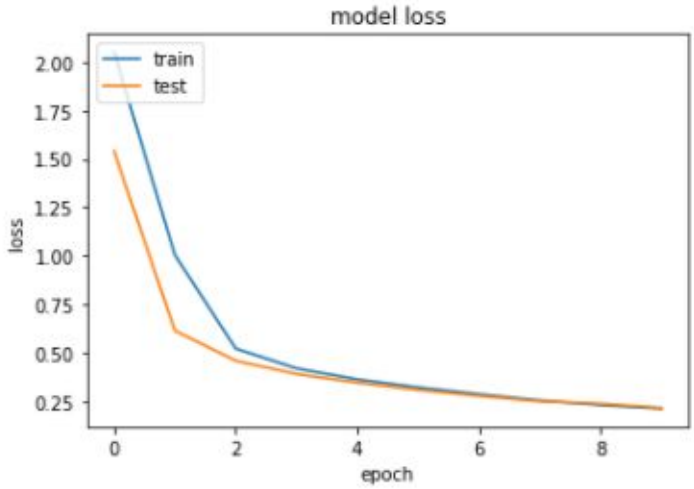


همچنین در صورتی که میزان نویز داده های تست بیشتر از ترین باشد، کاهش دقت و کارایی مدل را به دنبال خواهد داشت.

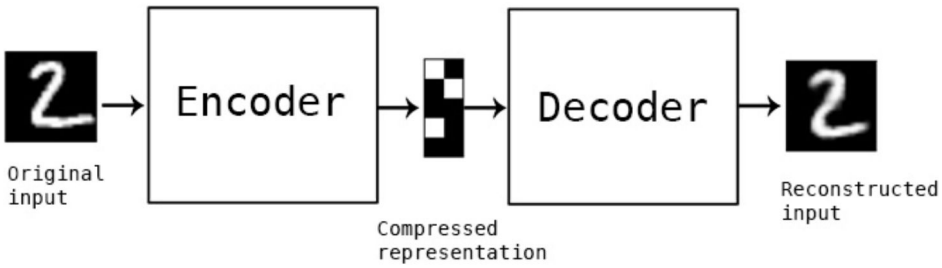
نتایج داده تست نویز 0.9 و داده آموزشی نویز 0.7

```
_________________________________________________________________
313/313 [==============================] - 1s 2ms/step - loss: 0.2152 - accuracy: 0.9383
Test loss : 0.21515026688575745
Test accuracy: 93.83000135421753
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: MatplotlibDeprecationWarr
  if sys.path[0] == '':
```





در این بخش قصد داریم از شبکه عصبی برای کاربرد رفع نویز استفاده کنیم. با تحقیقاتی که انجام دادم به این نتیجه رسیدم که برای این کار از شبکه های autoencoder استفاده می کنند که در این موارد بسیار خوب عمل می کنند. ساختار کلی این شبکه ها به صورت زیر می باشد.



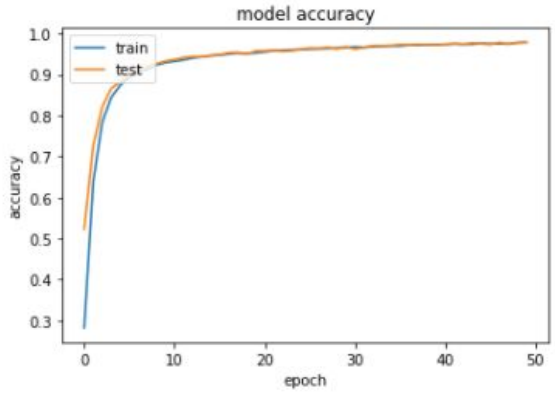
این شبکه به روش های مختلفی قابل پیاده سازی می باشد. به طور مثال به صورت fully connected , sequence to sequence Convolutional قابل پیاده سازی است. برای کاربرد رفع نویز تصویر شبکه Convolutional پیشنهاد می شود که از همان ساختار استفاده کردم.

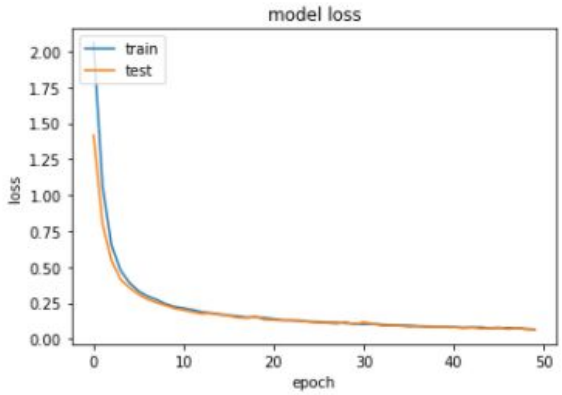
```
def model(x_train, y_train, x_test, y_test):

    model = Sequential()
    #encoder
    model.add(Conv2D(16, kernel_size=3, activation='relu',padding='same', input_shape=(28,28,1)))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Conv2D(8, kernel_size=3, padding='same',activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
    model.add(Conv2D(8, kernel_size=3,padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))

    #decoder
    model.add(Conv2D(8, kernel_size=3, activation='relu',padding='same'))
    model.add(UpSampling2D(size=(2,2)))
    model.add(Conv2D(8, kernel_size=3, padding='same',activation='relu'))
    model.add(UpSampling2D(size=(2,2)))
    model.add(Conv2D(16, kernel_size=3,padding='same', activation='relu'))
    model.add(UpSampling2D(size=(2,2)))

    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(10, activation='softmax'))

    model.summary()

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

    history = model.fit(x_train, y_train,
            batch_size=2048, epochs=50 , verbose=1,
            validation_data=(x_train, y_train))
```

این شبکه که از لایه های max pooling و upsampling و لایه های convolution متوالی تشکیل شده، مقاومت در برابر نویز را به حد خوبی بالا برد.

به طوری که برای داده های نویزی با نویز کمتر از 0.3 دقت شبکه با حالت بدون نویز تقریبا برابر است و برای نویز 0.9 دقت تا 96 درصد افزایش یافت.

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0970 - accuracy: 0.9699
Test loss : 0.09695956110954285
Test accuracy: 96.99000120162964
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: MatplotlibDeprecationWarnin
  if sys.path[0] == '':
```





نتایج و تحلیل ها :

- لایه max pooling به دلیل اینکه از بین پنجره تنها مقدار ماکزیمم را نگه می دارد به نظر می رسد در حال مناسب ترین فیچر است و از این رو در مواجه با نویز کمک می کند.
- لایه upsampling در واقع نوعی unpooling است که سایز شبکه را افزایش می دهد.
- مدل طراحی شده برای قسمت دوم به طرز خوبی تاثیر نویز را کاهش داد و مقاومت خوبی به نویز نشان داد. روند کلی عملکرد آن بدین صورت بود که ابتدا با max pooling سایز را کوچک کرده و نویز ها را تا حد امکان بی اثر کند (فیچر های اصلی تصویر را استخراج می کند)، سپس با upsampling به اندازه اولیه برگشته و بدین ترتیب در تصویر نویز را از بین می برد. حال که تصویر بدون اثر نویز داریم classification را انجام می دهد.
- البته می دانیم که اصل شبکه های CNN نسبت به mlp نسبت به نویز مقاوم تر هستند بنابراین بدون هر کدام از مراحل گفته شده اصل cnn تاثیر کمتری از نویز می پذیرد. به این دلیل که اصل کار این شبکه بر مبنا استخراج فیچر از تصویر می باشد.