# پروژه MNIST :

این پروژه براي تشخيص handwritten digits (شماره های دست نویس) است. دیتاست این پروژه شامل 60000 عکس 28*28 از اعداد 0-9 است که به صورت دست نویس هستند از بین آنها 50000 تا برای train کردن و ۱۰۰۰۰ تا براي تست کردن است.

تشخیص ارقام دست‌نویس توانایی رایانه برای تشخیص ارقام دست‌نویس انسان از منابع مختلف مانند تصاویر، کاغذها، صفحه‌های لمسی و غیره است و آنها را به ۱۰ کلاس از پیش تعریف شده (۰-۹) طبقه‌بندی می‌کند..

من براي پياده سازي اين پروژه ۳روش را در پیش گرفتم که کد هاي آنها به فايل ضميمه شده و شامل موارد زير هستند:

- شبکه عصبی به صورت sequential
- شبکه عصبی با کمک LSTM
- شبکه عصبی pretrained شده

## شبکه عصبی به صورت sequential :

در این قسمت ما شبکه عصبی را به صورت دستی پیاده سازي میکنیم به این صورت که موارد مختلف مثل تعداد و اندازه لایه ها، activation functionها، dropoutها و موارد دیگر کامل تحت اختیار ما هستند.

دراین بخش ما دیتاست MNIST را import کرده وعکس ها لیبل هاي آن را ذخیره میکنیم.
سپس براي خروجی بهتر اطلاعات را normalize کرده و شبکه را پیاده سازي میکنیم.
براي train کردن شبکه تنها کافی است از دستور model.fit استفاده کنیم در ادامه دیتاست test را به شبکه میدهیم که خروجی به شکل زیر خواهد بود.

```
Epoch 1/5
938/938 [==============================] - 5s 4ms/step - loss: 0.2220 - accuracy: 0.9320
Epoch 2/5
938/938 [==============================] - 3s 3ms/step - loss: 0.0984 - accuracy: 0.9692
Epoch 3/5
938/938 [==============================] - 2s 3ms/step - loss: 0.0727 - accuracy: 0.9771
Epoch 4/5
938/938 [==============================] - 2s 2ms/step - loss: 0.0581 - accuracy: 0.9816
Epoch 5/5
938/938 [==============================] - 3s 3ms/step - loss: 0.0510 - accuracy: 0.9837
313/313 [==============================] - 1s 1ms/step - loss: 0.0642 - accuracy: 0.9810
313/313 [==============================] - 0s 1ms/step
```

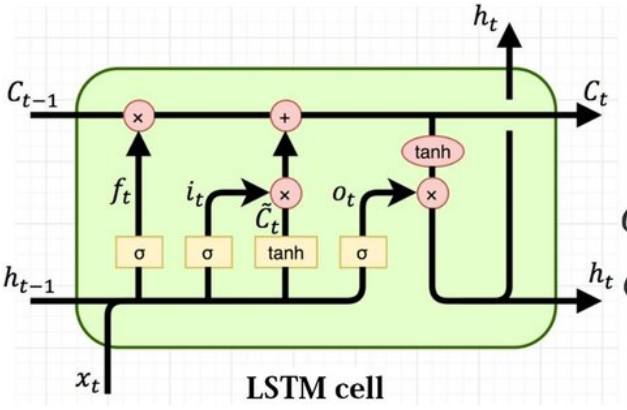
```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.98      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.99      0.98      0.98      1010
           4       0.97      0.98      0.98       982
           5       0.98      0.98      0.98       892
           6       0.99      0.99      0.99       958
           7       0.99      0.97      0.98      1028
           8       0.97      0.98      0.97       974
           9       0.99      0.96      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

Test score: 0.0641925260424614
Test accuracy: 0.9810000061988831
```

همان طور که پیداست مدت زمان اجرا شدن این برنامه با توجه به حجم دیتاست آن نسبتا پایین است و accuracy بسیار بالایی دارد.

همچنین مثلا میتوان فهمید از آنجایی که recall عدد 9 پایین تر است این عدد دفعات پیشتري پیش بینی اشتباهی داشته اند.

## شبکه عصبی با کمک LSTM :

این شبکه با کمک LSTM(Long Short Term Memory) پیاده سازي شده این شبکه در بسیاري دیتاست ها عکس العمل خیلی خوبی از خود نشان داده که معماري آن به صورت زیر است:




$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

این معماري شامل ۳بخش forgrt gate, output gate و input gate است به این طریق میتواند دیتاست را منظم تر کند و برخی را به نوعی فراموش کند تا نتیجه بهتري داشته باشیم.

براي پیاده سازي این شبکه از tenserflow وkeras استفاده میکنیم .این کتابخانه در پیاده سازي شبکه بسیار به ما کمک میکند و سبب سادگی کار میشود. میزان accuracy خروجی این برنامه به صورت زیر است:

```
accuracy=  0.9805999994277954
```

## شبکه عصبی pretrained شده:

این معماري از قبل train شده، دیتاست که براي این یادگیري اولیه در این نوع شبکه ها به کار میرودمیتواند متفاوت باشد که ما از دیتاست مخصوص image استفاده میکنیم.البته معماري این شبکه ها بسیار متفاوت است و با توجه به نیاز باید بهترین را انتخاب کرد که لیست معماري classification ها به شرح زیر است:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

ما در اینجا معماري RESNETرا انتخاب میکنیم چرا که نسبتا عمیق است و learning rate که دارد با دیتا ما سازگاري بهتري دارد .پس از پیاده سازي این شبکه حال با دیتا MNIST آن را آموزش میدهیم. براي این کار عملا تنها لایه آخر را در نظر میگیریم.و بقیه شبکه freeze میشود .

البته مشکل این شبکه این است که آز آنجایی که بسیار عمیق است (خود RESNETبه تنهایی 45 میلیون پارامتر دارد )مدت زمان زیاد و GPUبسیار قوي میطلبد لذا من تنها با 2 epoch این برنامه را اجرا کردم که خروجی به صورت زیر بدست آمد که با توجه به مقدار کم epoch نتیجه بسیار خوبی است.

```
Training....
Epoch: 0 Batch: 100 loss: 0.5547378659248352
Epoch: 0 Batch: 200 loss: 0.3358318507671356
Epoch: 0 Batch: 300 loss: 0.3978109359741211
Epoch: 0 Batch: 400 loss: 0.23702001571655273
Epoch: 0 Batch: 500 loss: 0.18744787573814392
Epoch: 0 Batch: 600 loss: 0.17843349277973175
Epoch: 0 Batch: 700 loss: 0.1693977415561676
Epoch: 0 Batch: 800 loss: 0.15776023268699646
Epoch: 0 Batch: 900 loss: 0.120021753013134
Epoch: 1 Batch: 100 loss: 0.16726051270961761
Epoch: 1 Batch: 200 loss: 0.03816727548837662
Epoch: 1 Batch: 300 loss: 0.04746527597308159
Epoch: 1 Batch: 400 loss: 0.16390030086040497
Epoch: 1 Batch: 500 loss: 0.07044760137796402
Epoch: 1 Batch: 600 loss: 0.09852813929319382
Epoch: 1 Batch: 700 loss: 0.09327635169029236
Epoch: 1 Batch: 800 loss: 0.16746655106544495
Epoch: 1 Batch: 900 loss: 0.13821925222873688
Training accuracy: 92.37083333333334 %
Predicting....
Batch 100 done
Test accuracy: 97.75 %
```