به نام خدا

گزارش پروژه درس یادگیری عمیق

موضوع پروژه: OCR

دکتر محمدرضا محمدی

اعضای گروه:

محمد یارمقدم

صدرا حیدری مقدم

هدف پروژه:

هدف در این پروژه یادگیری شبکه OCR برای یادگیری اعداد موجود در کارت ملی و کارت های بانکی است. برای این امر لازم است ابتدا داده های مورد نیاز برای آموزش شبکه را آموزش دهیم. سپس شبکه موردنظر را تشکیل دهیم و بر روی داده های موجود شبکه را ران کنیم و آموزش دهیم. در نهایت بر روی داده های تست دیده نشده شبکه را مورد ازریابی قرار دهیم.

نحوه تولید داده‌ها

طبیعتا برای آموزش دادن شبکه‌ای که توانایی تشخیص کد ملی یا شماره کارت از روی عکس را دارد، باید یک مجموعه داده داشته باشیم که شبکه با دیدن این مجموعه داده و با تلاش برای کم کردن خطا و افزایش دقت بر روی هر عکس مجموعه داده، آموزش ببیند. اما مشکل اصلی آن است که این داده ها همیشه در دسترس نیستند و یا بسیار پراکنده هستند به صورتی که جمع‌آوری آن‌ها کار دشواری است. به همین دلیل مجموعه داده را به کمک یک ریپازیتوری معتبر به نام Text Recognition Data Generator یا به اختصار TRDG ساختیم که این ریپازیتوری و نحوه کارش را در این لینک می‌توانید مشاهده کنید.

برای تولید داده به کمک TRDG به یک فایل تکست نیاز داریم که همه‌ی داده‌ها باید به صورت متن در هر خط این فایل تکست نوشته شوند. کاری که این سرویس انجام می‌دهد آن است که با توجه به فیلتر هایی که برای آن در نظر می گیریم، عکس متن هر خط را با ارتفاع 32 تولید می‌کند و نام فایل را برابر شماره خط و خود متن قرار می‌دهد. با این کار، برای آنکه به لیبل یک عکس دسترسی داشته باشیم کافی است نام آن عکس را در نظر بگیریم چون متن درون هر تصویر، همان نام آن تصویر است.

کدهای مربوط به تولید داده در فایل DataGenerator.ipynp قابل مشاهده هستند که در آن در ابتدا TRDG نصب شده و سپس 10000 داده‌ی رندوم که شامل 5000 شماره کارت و 5000 کد ملی است، برای داده‌های ترین تولید شده و در فایل تکست ریخته شده. همین کار برای داده‌های تست و اعتبارسنجی هم انجام شده با این تفاوت که 1250 داده رندوم، شامل 625 شماره کارت و 25 کد ملی، به صورت متن تولید شده و در فایل‌های تکست مربوط به خودشان ذخیره شده‌اند و بعد از آن این داده‌ها shuffle شده‌اند. دقت شود که تقسیم داده‌ها به صورت 80، 10، 10 بوده یعنی 80٪ داده‌های آموزش هستند و داده‌های تست و اعتبارسنجی هرکدام 10٪ کل داده‌ها را تشکیل می‌دهند که این تقسیم بندی یکی از پراستفاده ترین دسته بندی‌ها است.

تعدادی از خطوط تکست فایل مورد نیاز برای تولید داده را در زیر می‌بینید.

داده‌های کارت ملی

```
6240189938
0090181415
0729199709
3413476670
8417337993
7084977188
1512671919
```

داده های کارت بانکی

```
7939 6943 8789 0986
0306 8588 3952 2729
7982 9158 9262 2489
3018 0935 1147 0566
3051 8773 0598 4726
1547 2738 5487 1747
4765 2411 8588 4549
7523 6178 2880 4085
```

تکست فایل ها در فایل درایو شیر شده با نام‌های validation_data_melli, train_data_bank, train_data_melli, test_data_bank, test_data_melli و validation_data_bank قابل مشاهده هستند.

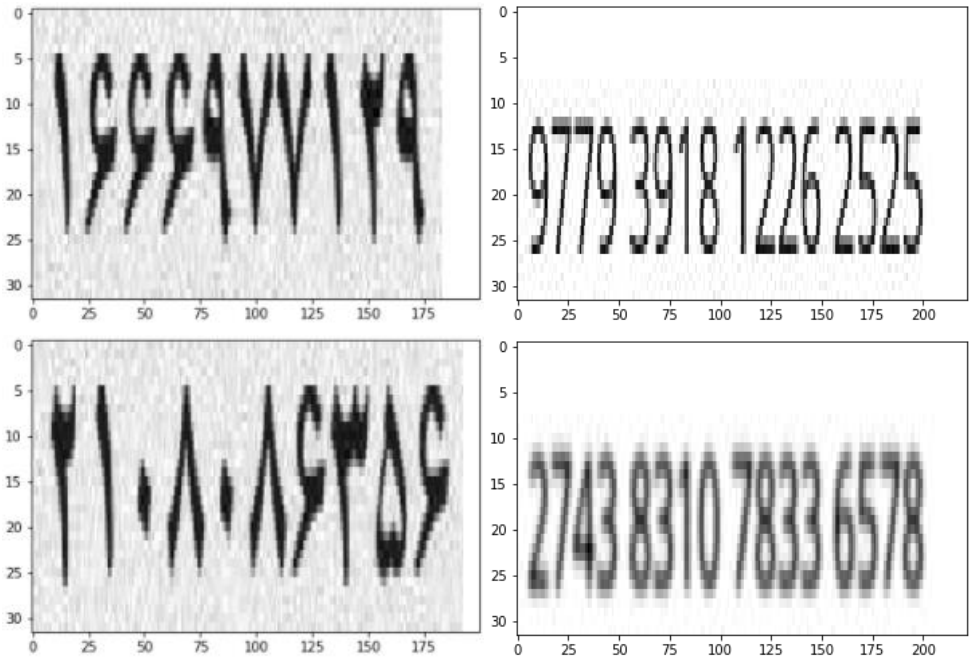
بعد از تولید فایل های تکست، نوبت به تولید داده‌های ورودی شبکه به کمک TRDG است. برای این کار باید فونت هایی که می‌خواهیم داده‌ها با آن فونت ها تولید شوند را در خود پکیج TRDG و در قسمت فونتها قرار دهیم. سپس با اجرا کردن دستور مربوط به TRDG می توان داده‌ها را تولید کرد. البته برای اجرای آن باید محل فایل تکست، فولدر ذخیره‌ی داده‌های عکس، فونت و پردازش‌های عکس مشخص شوند، مثلا اگر بخواهیم تصویر به صورت رندوم تار شود از rbl– 2 bl– استفاده می کنیم و اگر حالتی را در نظر نگیریم، تصاویر با نویز تولید می‌شوند.

برای تولید داده از augmentation استفاده شده و 60000 داده آموزش از 10000 تا داده تکست موجود و 7500 داده عکس تست و اعتبار سنجی تولید شد و داده افزایی به کمک 4 فونت شامل دو فونت انگلیسی و دو فونت فارسی انجام شده که بر روی هر فونت هم سه حالت بر روی عکس‌ها پردازش شده که سه حالت به صورت زیر هستند. (توضیحات و کد هر بخش را در فایل نوتبوک می‌توانید ببینید)

1) عکس‌های با نویز
2) عکس‌های بدون نویز و با تار شدگی رندوم(ممکن است تار شود و ممکن هست نشود)
3) عکس‌های با نویز که به صورت رندوم در زوایای مختلف کج شده اند

مشکل دیگری که وجود داشت overwrite شدن عکس ها روی هم بود چون اسم‌هایشان مشابه بود و فقط عکس تفاوت داشت، به همین دلیل برای حل این مشکل، قسمتی از کد فایل run.py در پکیج TRDG نیز تغییر داده شد.

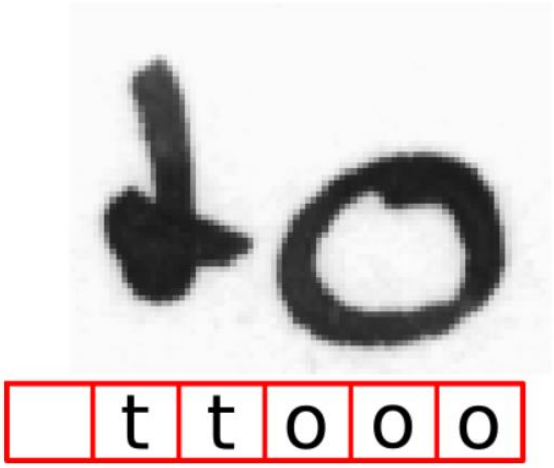
پس با توجه به موارد گفته شده 6 نوع عکس از هر داده متنی تولید شده که در زیر نمونه‌ای از آنها را می‌بینید. همچنین این داده ها در فولدرهای train_dataset, test_dataset و validation_dataset قابل مشاهده هستند.



https://github.com/Belval/TextRecognitionDataGenerator

CTC Loss

برای تابع ضرر از تابع ضرر Connectionist Temporal Classification یا به اختصار CTC استفاده شده. اگر از این تابع ضرر استفاده نکنیم، هر قسمتی که لایه کانولوشنی بررسی می کند و به لایه RNN خروجی می‌دهد، خروجی خاص خود را خواهد داشت یعنی مثلا ممکن است کلمه‌ی to مانند شکل زیر، به صورت ttooo– شناسایی شود چون مثلا حرف O چند بار در فیلترهای نهایی لایه آخر کانولوشنی قرار گرفته که باعث شده شبکه RNN انتها نیز چند بار این حرف را شناسایی کند و کنار هم قرار دهد.



شاید در این حالت بگوییم که مشکلی نیت و هر حرف خروجی را یک بار در نظر می گیریم و کنار هم قرار می دهیم تا به کلمه to برسیم اما اگر کلمه ی ما too بود چه اتفاقی می‌افتاد؟ طبیعتا به مشکل می خوردیم. تابع ضرر CTC این مشکل را برای ما حل می کند و کافی است خروجی مد نظر عکس را به CTC بدهیم تا خودش مشکلات محلی و عرضی برای هر حرف یک تصویر را حل کند. همچنین به کمک این تابع ضرر، پس پردازشی بر روی خروجی نیاز نیست. به این دلایل تابع ضرر CTC loss انتخاب شده و عموما هم برای text recognition از این تابع ضرر استفاده می شود.

در این تمرین نتوانستیم از این loss استفاده کنیم.

https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c

توضیح مراحل کد:

در قسمت اول ابتدا کولب را به محیط درایو متصل می‌کنیم تا اطلاعات موجود در فولدر های موجود در درایو تا بتوانیم بخوانیم. همانطور که در قسمت قبل بیان شد داده ها در سه دسته آموزش و تست و اعتبارسنجی در فولدر های جداگانه در درایو ذخیره شده است که خوانده می‌شود.

سپس alphabet مورداستفاده که برای ما اعداد 0 تا 9 و فاصله(space) است راتشکیل می دهیم.

در کلاس SynthDataset هدف تشکیل دیتای موردنیاز برای آموزش شبکه است. در درایو هر عکس به فرمتی ذخیره شده است که نام آن در واقعا لیبل مورد نیاز برای شبکه است. بنابراین برای تشکیل داده های موردنیاز دیتای هر عکس را خوانده و به عنوان ورودی در نظر میگیریم و نام عکس را خوانده و به عنوان لیبل قرار می‌دهیم.

در تابع getitem عملیاتی که بالا گفته شد، انجام می‌شود و در تابع get_all_items ابعاد عکس ها تغییر میکند. به دلیل اینکه طول اعداد در دو کارت موردنظر برابر نیست نیاز است که طول داده در داده کوچک‌تر با فضای خالی پر شود تا شبکه راحت‌تر آموزش ببیند. هم چنین نیاز است که خروجی 20 تایی در نظر گرفته شود و لیبل در کارت ملی با 10 فضای خالی و لیبل در کارت بانکی با 1 فضای خالی همراه می‌شود. اینکار به این منظور انجام می‌شود که طول الفبای هر تصویر یکسان شود. برای یکی کردن width عکس های موجود نیز ابتدا یک تصویر تمام سفید توسط np.ones تشکیل می‌دهیم و سپس تصویر به عرض مشترک 222 تبدیل میکنیم و آنرا روی تصویر جدید نگاشت می‌کنیم. در این قسمت برای تبدیل داده ها به داده قابل فهم برای شبکه ابتدا یک آرایه 20 تایی از اعضای 11 تایی تشکیل می‌دهیم و برای هر قسمت با استفاده از تابع find ایندکس آن حرف در آرایه را یک قرار می‌دهیم تا یک مپ one hot تشکیل شود. این کار برای آن استفاده شد که categorical_crossentrophy بتوان استفاده کرد.

سپس برای تشکیل دیتای نهایی تست و آموزش در سه دسته آموزش و تست و ولیدیشن یک شی از این کلاس می‌سازیم و بر روی آن این متد ها را در هر حالت فراخوانی می‌کنیم.

برای قسمت تشکیل شبکه از لایه های LSTM و کانولوشن استفاده کردیم تا در ابتدا ویژگی های عکس که شامل نوع اعداد می‌شود یادگیری شود و سپس با استفاده از LSTM قواعد ترکیب آنها آموزش دیده شود.

در نهایت آز لایه های batch normalization و dropout در بین این لایه ها استفاده کردیم تا احتمال overfit را کاهش دهیم.

نکته حائز اهمیت در تشکیل شبکه این است که برای تعیین تعداد ورودی لایه LSTM می بایست تعداد 32 را به 20 که طول داده ورودی است reshape کنیم. به این منظور ابتدا از maxpooling استفاده میکنیم تا طول ورودی 16 شود و سپس با padding دوبعدی 4 صفر به انتهای آن اضافه میکنیم تا shape با هم مچ شود.

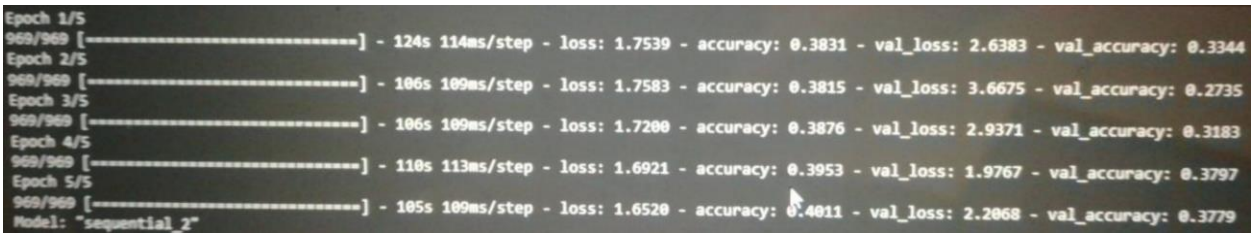
```
tf.keras.layers.Input(shape=(train_x.shape[1], train_x.shape[2], 1)),
tf.keras.layers.Conv2D(16, kernel_size=(3, 3), activation="relu", padding="same"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu", padding="same"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu", padding="same"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu", padding="same"),
tf.keras.layers.BatchNormalization(),
tf.keras.layers.ZeroPadding2D(padding=(2,2)),
tf.keras.layers.Dropout(0.3),
tf.keras.layers.TimeDistributed(tf.keras.layers.Flatten()),
tf.keras.layers.MaxPooling1D(),
tf.keras.layers.LSTM(64, activation='relu', return_sequences=True),
tf.keras.layers.LSTM(11, activation='softmax', return_sequences=True)
```

این شبکه را با بهینه‌ساز adam و تابع خطا categorical در پنج epoch ران کردیم. یکی از مشکلات این بود که قصد استفاده از CTC_Loss را داشتیم اما به علت اینکه داده های ورودی ما one hot هستند امکان استفاده از آن را پیدا نکردیم. اگر امکان استفاده داشتیم با احتمال بالا به نتایج بهتری دست میافتیم. مشکلی که در عدم استفاده از این نوع loss ممکن است ایجاد شود این است که هر قسمتی که لایه کانولوشنی بررسی می کند و به لایه RNN خروجی می‌دهد، خروجی خاص خود را خواهد داشت یعنی مثلا ممکن است کلمه‌ی to مانند شکل زیر، به صورت ttooo– شناسایی شود چون مثلا حرف O چند بار در فیلترهای نهایی لایه آخر کانولوشنی قرار گرفته که باعث شده شبکه RNN انتها نیز چند بار این حرف را شناسایی کند و کنار هم قرار دهد.

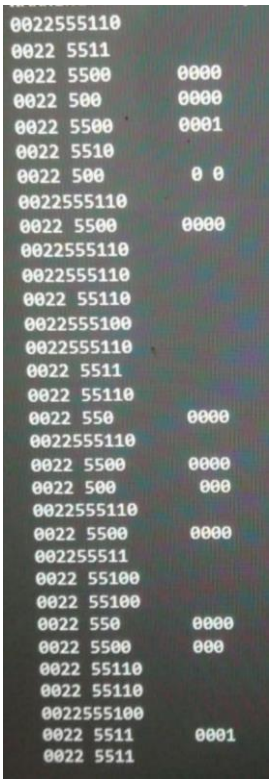
در ابتدا تعداد داده 20000 بود که تست اولیه انجام دهیم. در این حالت به نتایج زیر دست یافتیم:

```
Epoch 1/5
625/625 [==============================] - 59s 71ms/step - loss: 2.1859 - accuracy: 0.3455 - val_loss: 2.3726 - val_accuracy: 0.3308
Epoch 2/5
625/625 [==============================] - 46s 74ms/step - loss: 2.0641 - accuracy: 0.3548 - val_loss: 2.0993 - val_accuracy: 0.3431
Epoch 3/5
625/625 [==============================] - 44s 70ms/step - loss: 2.0428 - accuracy: 0.3551 - val_loss: 2.0664 - val_accuracy: 0.3535
Epoch 4/5
625/625 [==============================] - 44s 70ms/step - loss: 2.0356 - accuracy: 0.3551 - val_loss: 2.0780 - val_accuracy: 0.3494
Epoch 5/5
625/625 [==============================] - 44s 70ms/step - loss: 2.0297 - accuracy: 0.3570 - val_loss: 2.0387 - val_accuracy: 0.3631
```

سپس از تمامی داده آموزش استفاده کردیم تا بهبود در شبکه را شاهد باشیم. در اینجا مشکلی حاصل شد که تعداد 60000 داده به طور مثال در درایو تولید و برای آموزش آماده سازی شده بود اما در انتقال به محیط کولب در نهایت به 32000 میرسید و به هیچ وجه بیشتر نمیشد. در ران شدن این تعداد داده در پنج ایپاک به نتایج زیر دست یافتیم:



پس از این مرحله از تابع convert_output_to_text استفاده کردیم تا اعداد خروجی شبکه را به نماد های موجود بر روی کارت ها تبدیل کنیم و بتوانیم نتیجه شبکه را مقایسه کنیم تا میزان یادگیری شبکه را مشاهده کنیم:



همانطور که در تصویر مشخص است میزان کمی در اعداد کد ملی یادگیری صورت گرفته است که از آنجا که چنج ایپاک بوده نتیجه قابل قبولی است.

پس از تلاش های فراوان بالاخره به آمار 40000 داده دست یافتیم. اما باز هم از انتظار ما داده کمتری بدست آمد. به همین دلیل در کنار شبکه قبلی یک شبکه با تغییرات زیر را نیز تست کردیم تا بهترین نتیجه را بدست آوریم:

```
model = Sequential(
  [
   tf.keras.layers.Input(shape=(train_x.shape[1], train_x.shape[2], 1)),
   tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu", padding="same"),
   tf.keras.layers.BatchNormalization(),
   tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu", padding="same"),
   tf.keras.layers.BatchNormalization(),
   tf.keras.layers.Dropout(0.2),
  #  tf.keras.layers.MaxPooling2D(pool_size=(2, 2), padding="same"),
   tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu", padding="same"),
   tf.keras.layers.BatchNormalization(),
   tf.keras.layers.Dropout(0.2),
   tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation="relu", padding="same"),
   tf.keras.layers.BatchNormalization(),
   tf.keras.layers.MaxPooling2D(pool_size=(2, 2), padding="SAME"),
   tf.keras.layers.ZeroPadding2D(padding=(2,2)),
   tf.keras.layers.Dropout(0.2),
   tf.keras.layers.TimeDistributed(tf.keras.layers.Flatten()),
   tf.keras.layers.LSTM(64, activation='relu', return_sequences=True),
   tf.keras.layers.LSTM(11, activation='softmax', return_sequences=True)
  ])
```

این شبکه را 40000 داده آموزش و در پنج ایپاک آموزش دادیم که نتیجه زیر حاصل شد:

```
Epoch 1/5
1000/1000 [==============================] - 90s 69ms/step - loss: 1.7253 - accuracy: 0.3924 - val_loss: 1.8538 - val_accuracy: 0.3853
Epoch 2/5
1000/1000 [==============================] - 65s 65ms/step - loss: 1.6523 - accuracy: 0.3998 - val_loss: 1.8116 - val_accuracy: 0.3911
Epoch 3/5
1000/1000 [==============================] - 69s 69ms/step - loss: 1.5843 - accuracy: 0.4042 - val_loss: 1.7125 - val_accuracy: 0.4019
Epoch 4/5
1000/1000 [==============================] - 66s 66ms/step - loss: 1.5633 - accuracy: 0.4127 - val_loss: 1.7350 - val_accuracy: 0.4016
Epoch 5/5
1000/1000 [==============================] - 67s 67ms/step - loss: 1.8715 - accuracy: 0.3820 - val_loss: 2.1427 - val_accuracy: 0.3644
```

بازهم به علت پیچیدگی شبکه و تعداد کم داده ها دقت خوبی حاصل نشد اما روند بهتری در رشد دقت شاهد بودیم.

بنابراین این مورد را با ده ایپاک مجدد تست کردیم. نتیجه به طرز عجیبی ضعیف شد:

```
Epoch 1/10
1000/1000 [==============================] - 75s 72ms/step - loss: 1.9506 - accuracy: 0.3873 - val_loss: 2.1300 - val_accuracy: 0.3747
Epoch 2/10
1000/1000 [==============================] - 65s 65ms/step - loss: 1.9234 - accuracy: 0.3891 - val_loss: 2.1033 - val_accuracy: 0.3764
Epoch 3/10
1000/1000 [==============================] - 65s 65ms/step - loss: 1.6191 - accuracy: 0.4080 - val_loss: 1.7135 - val_accuracy: 0.4003
Epoch 4/10
1000/1000 [==============================] - 68s 68ms/step - loss: 1.5326 - accuracy: 0.4128 - val_loss: 1.6811 - val_accuracy: 0.4025
Epoch 5/10
1000/1000 [==============================] - 68s 68ms/step - loss: nan - accuracy: 0.3708 - val_loss: nan - val_accuracy: 0.0619
Epoch 6/10
1000/1000 [==============================] - 68s 68ms/step - loss: nan - accuracy: 0.0654 - val_loss: nan - val_accuracy: 0.0619
Epoch 7/10
1000/1000 [==============================] - 65s 65ms/step - loss: nan - accuracy: 0.0654 - val_loss: nan - val_accuracy: 0.0619
Epoch 8/10
1000/1000 [==============================] - 65s 65ms/step - loss: nan - accuracy: 0.0654 - val_loss: nan - val_accuracy: 0.0619
Epoch 9/10
1000/1000 [==============================] - 67s 67ms/step - loss: nan - accuracy: 0.0654 - val_loss: nan - val_accuracy: 0.0619
Epoch 10/10
1000/1000 [==============================] - 66s 66ms/step - loss: nan - accuracy: 0.0654 - val_loss: nan - val_accuracy: 0.0619
Model: "sequential_2"
```

خروجی ها هم کاملا اشتباه شد و هیچ عددی تشخیص داده نشد:

```
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
0000000000000000000
```

در ادامه شبکه را روی دو حالت برای کارت بانکی و کارت ملی طراحی کردیم و مقداری کد را تغییر دادیم اما بن کد همان کدی است که توضیح داده شد.

شبکه و کد نهایی:

همانطور که بیان شد، داده‌های کارت ملی و کارت بانکی را جدا کردیم تا مجبور نباشیم که لیبل را گسترش دهیم و لیبل برای شبکه‌ی کارت ملی 10 حرفی و لیبل برای شبکه کارت بانکی 20 حرفی قرار داده شد و دو شبکه مختلف با توجه به تفاوت ابعاد دو نوع داده طراحی شد که در زیر هر کدام از شبکه ها و نتایج خروجی آنها را مشاهده می‌کنید.

1) شبکه مرتبط با داده‌های کارت بانکی: Numbers_OCR_Tensorflow_bank.ipynb

این شبکه متشکل از چندین لایه کانولوشنی به همراه دو لایه LSTM هست که ساختار آن را در زیر می‌بینید.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 222, 16)       160

batch_normalization_12 (Bat  (None, 32, 222, 16)       64
chNormalization)

conv2d_13 (Conv2D)           (None, 32, 222, 32)       4640

batch_normalization_13 (Bat  (None, 32, 222, 32)       128
chNormalization)

dropout_9 (Dropout)          (None, 32, 222, 32)       0

conv2d_14 (Conv2D)           (None, 32, 222, 32)       9248

batch_normalization_14 (Bat  (None, 32, 222, 32)       128
chNormalization)

dropout_10 (Dropout)         (None, 32, 222, 32)       0

conv2d_15 (Conv2D)           (None, 32, 222, 64)       18496

batch_normalization_15 (Bat  (None, 32, 222, 64)       256
chNormalization)

max_pooling2d_3 (MaxPooling  (None, 16, 111, 64)       0
2D)

zero_padding2d_3 (ZeroPaddi  (None, 20, 115, 64)       0
ng2D)

dropout_11 (Dropout)         (None, 20, 115, 64)       0

time_distributed_3 (TimeDis  (None, 20, 7360)          0
tributed)

lstm_6 (LSTM)                (None, 20, 32)            946304

lstm_7 (LSTM)                (None, 20, 11)            1936

=================================================================
Total params: 981,360
Trainable params: 981,072
Non-trainable params: 288
```

با آموزش این شبکه روی مجموعه داده‌ی بانکی بعد از 15 ایپوک به نتیجه‌ی زیر می‌رسیم. دقت شود که برای آموزش شبکه از ایپوک های 5 تایی یا 10 تایی استفاده کردیم و در هر مرحله نتایج را در model_bank.h5 ذخیره و برای ادامه، نتایج را لود کردیم.

```
Epoch 1/10
469/469 [==============================] - 56s 114ms/step - loss: 1.5703 - accuracy: 0.3903 - val_loss: 1.4873 - val_accuracy: 0.4259
Epoch 2/10
469/469 [==============================] - 52s 110ms/step - loss: 1.5470 - accuracy: 0.3988 - val_loss: 1.5213 - val_accuracy: 0.4245
Epoch 3/10
469/469 [==============================] - 52s 110ms/step - loss: 1.5403 - accuracy: 0.4002 - val_loss: 1.4409 - val_accuracy: 0.4358
Epoch 4/10
469/469 [==============================] - 51s 108ms/step - loss: 1.5322 - accuracy: 0.4040 - val_loss: 1.4182 - val_accuracy: 0.4549
Epoch 5/10
469/469 [==============================] - 50s 106ms/step - loss: 1.5058 - accuracy: 0.4197 - val_loss: 1.3919 - val_accuracy: 0.4590
Epoch 6/10
469/469 [==============================] - 51s 109ms/step - loss: 1.4817 - accuracy: 0.4256 - val_loss: 1.3666 - val_accuracy: 0.4662
Epoch 7/10
469/469 [==============================] - 51s 109ms/step - loss: 1.4704 - accuracy: 0.4279 - val_loss: 1.3563 - val_accuracy: 0.4633
Epoch 8/10
469/469 [==============================] - 51s 108ms/step - loss: 1.4518 - accuracy: 0.4332 - val_loss: 1.3265 - val_accuracy: 0.4722
Epoch 9/10
469/469 [==============================] - 51s 108ms/step - loss: 1.4294 - accuracy: 0.4363 - val_loss: 1.2687 - val_accuracy: 0.4883
Epoch 10/10
469/469 [==============================] - 52s 112ms/step - loss: 1.3830 - accuracy: 0.4469 - val_loss: 1.2321 - val_accuracy: 0.4975
```

ایپوک 20 تا 25:

```
Epoch 1/5
469/469 [==============================] - 55s 113ms/step - loss: 1.3259 - accuracy: 0.4704 - val_loss: 1.1699 - val_accuracy: 0.5267
Epoch 2/5
469/469 [==============================] - 52s 110ms/step - loss: 1.3160 - accuracy: 0.4731 - val_loss: 1.1670 - val_accuracy: 0.5227
Epoch 3/5
469/469 [==============================] - 50s 107ms/step - loss: 1.3163 - accuracy: 0.4725 - val_loss: 1.1615 - val_accuracy: 0.5237
Epoch 4/5
469/469 [==============================] - 51s 110ms/step - loss: 1.3055 - accuracy: 0.4783 - val_loss: 1.1568 - val_accuracy: 0.5311
Epoch 5/5
469/469 [==============================] - 52s 111ms/step - loss: 1.2996 - accuracy: 0.4805 - val_loss: 1.1620 - val_accuracy: 0.5286
```

ایپوک 25 تا 30

```
Epoch 1/5
469/469 [==============================] - 54s 113ms/step - loss: 1.3043 - accuracy: 0.4789 - val_loss: 1.1529 - val_accuracy: 0.5244
Epoch 2/5
469/469 [==============================] - 51s 109ms/step - loss: 1.2967 - accuracy: 0.4807 - val_loss: 1.1563 - val_accuracy: 0.5326
Epoch 3/5
469/469 [==============================] - 52s 112ms/step - loss: 1.3059 - accuracy: 0.4788 - val_loss: 1.1425 - val_accuracy: 0.5391
Epoch 4/5
469/469 [==============================] - 52s 110ms/step - loss: 1.2849 - accuracy: 0.4874 - val_loss: 1.1298 - val_accuracy: 0.5397
Epoch 5/5
469/469 [==============================] - 52s 110ms/step - loss: 1.2713 - accuracy: 0.4914 - val_loss: 1.1200 - val_accuracy: 0.5426
```

همانطور که مشاهده می شود، هنوز شبکه overfit نشده و همچنان در حال یادگیری است

ایپوک 30 تا 35:

```
Epoch 1/5
469/469 [==============================] - 57s 116ms/step - loss: 1.2706 - accuracy: 0.4904 - val_loss: 1.1222 - val_accuracy: 0.5455
Epoch 2/5
469/469 [==============================] - 51s 108ms/step - loss: 1.2591 - accuracy: 0.4945 - val_loss: 1.1549 - val_accuracy: 0.5415
Epoch 3/5
469/469 [==============================] - 51s 109ms/step - loss: 1.2572 - accuracy: 0.4947 - val_loss: 1.1092 - val_accuracy: 0.5404
Epoch 4/5
469/469 [==============================] - 51s 108ms/step - loss: 1.2579 - accuracy: 0.4940 - val_loss: 1.1104 - val_accuracy: 0.5445
Epoch 5/5
469/469 [==============================] - 51s 108ms/step - loss: 1.2527 - accuracy: 0.4959 - val_loss: 1.1164 - val_accuracy: 0.5419
```

همانطور که مشاهده می کنید در این ایپوک ها دیگر شبکه رو به overfitting رفته و دقت اعتبار سنجی حدود 54٪ و دقت آموزش حدود 49٪ گیر کرده که این احتمالا به دلیل مشکل ظرفیت شبکه اتفاق افتاده.

در نهایت خروجی برخی داده‌های تست به صورت زیر درآمد:

```
0283 8596 0699 6665
0283 5948 1896 0962
0283 8088 0644 6162
0283 5819 8033 4662
0283 8666 6562 6665
0283 5806 3596 8920
```

2) شبکه مرتبط با داده‌های کارت ملی(داده‌های فارسی): Numbers_OCR_Tensorflow_melli.ipynb

این شبکه هم متشکل از چندین لایه کانولوشنی به همراه دو لایه LSTM هست با این تفاوت که یک لایه پولینگ قرار داده شده تا ابعاد آن با ابعاد شبکه LSTM متناسب باشد و همچنین لایه پدینگ هم به صورت (1،1) قرار داده شد که ساختار آن را در زیر می‌بینید.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 200, 16)       160

batch_normalization_12 (Bat  (None, 32, 200, 16)       64
chNormalization)

conv2d_13 (Conv2D)           (None, 32, 200, 32)       4640

batch_normalization_13 (Bat  (None, 32, 200, 32)       128
chNormalization)

dropout_7 (Dropout)          (None, 32, 200, 32)       0

max_pooling2d_6 (MaxPooling  (None, 16, 100, 32)       0
2D)

conv2d_14 (Conv2D)           (None, 16, 100, 32)       9248

batch_normalization_14 (Bat  (None, 16, 100, 32)       128
chNormalization)

conv2d_15 (Conv2D)           (None, 16, 100, 64)       18496

batch_normalization_15 (Bat  (None, 16, 100, 64)       256
chNormalization)

max_pooling2d_7 (MaxPooling  (None, 8, 50, 64)         0
2D)

zero_padding2d_3 (ZeroPaddi  (None, 10, 52, 64)        0
ng2D)

dropout_8 (Dropout)          (None, 10, 52, 64)        0

time_distributed_3 (TimeDis  (None, 10, 3328)          0
tributed)

lstm_6 (LSTM)                (None, 10, 50)            675800

lstm_7 (LSTM)                (None, 10, 11)            2728

=================================================================
Total params: 711,648
Trainable params: 711,360
Non-trainable params: 288
_________________________________________________________________
```

تصویر 10 ایپوک اول را در زیر می‌بینید:

```
Epoch 1/10
480/480 [==============================] - 59s 117ms/step - loss: 2.2132 - accuracy: 0.1512 - val_loss: 2.0800 - val_accuracy: 0.1952
Epoch 2/10
480/480 [==============================] - 51s 106ms/step - loss: 1.8601 - accuracy: 0.2938 - val_loss: 1.6887 - val_accuracy: 0.3681
Epoch 3/10
480/480 [==============================] - 50s 105ms/step - loss: 1.5246 - accuracy: 0.4257 - val_loss: 1.3379 - val_accuracy: 0.4933
Epoch 4/10
480/480 [==============================] - 50s 103ms/step - loss: 1.2115 - accuracy: 0.5398 - val_loss: 1.0744 - val_accuracy: 0.5932
Epoch 5/10
480/480 [==============================] - 48s 101ms/step - loss: 1.0308 - accuracy: 0.6087 - val_loss: 0.9031 - val_accuracy: 0.6624
Epoch 6/10
480/480 [==============================] - 47s 98ms/step - loss: 0.8914 - accuracy: 0.6596 - val_loss: 0.7723 - val_accuracy: 0.7070
Epoch 7/10
480/480 [==============================] - 47s 98ms/step - loss: 0.7719 - accuracy: 0.7035 - val_loss: 0.7145 - val_accuracy: 0.7267
Epoch 8/10
480/480 [==============================] - 45s 95ms/step - loss: 0.7113 - accuracy: 0.7236 - val_loss: 0.6307 - val_accuracy: 0.7593
Epoch 9/10
480/480 [==============================] - 50s 104ms/step - loss: 0.6685 - accuracy: 0.7393 - val_loss: 0.5944 - val_accuracy: 0.7650
Epoch 10/10
480/480 [==============================] - 50s 103ms/step - loss: 0.6309 - accuracy: 0.7519 - val_loss: 0.7314 - val_accuracy: 0.7334
```

همانطور که می بینید مدل بسیار خوب آموزش دیده که به دلیل شبکه ی طراحی شده ی خوب است و همچنین تعداد ارقام کارت ملی کمتر از کارت بانکی است.

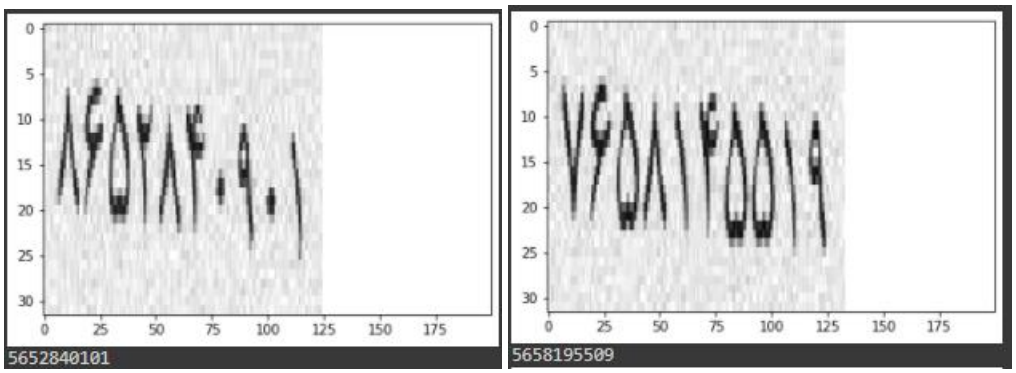
```
480/480 [==============================] - 58s 114ms/step - loss: 0.5569 - accuracy: 0.7913 - val_loss: 0.4967 - val_accuracy: 0.8120
```

بعد از ایپوک 12 با دقت اعتبارسنجی 81.20٪ و دقت آموزش 79.13٪، شبکه overfit می شود.

همچنین دقت روی داده تست نیز برابر 81.03٪ شد.

در نهایت خروجی برخی داده‌های تست به صورت زیر درآمد:

```
5131500226
5658195509
5566870801
5053511102
5157068022
5612168224
5849237306
5652840101
5274404906
5198280625
```



همانطور که می بینید، عمده ی اعداد به درستی تشخیص داده شده‌اند.

در نهایت با کمک فایل Predict.ipynp و با کمک تابع predict می توان از این شبکه استفاده کرد.

توضیحات نحوه اجرای فایل در ReadMe.txt ذکر شده.

منابع اصلی

تمرین 13

مقدار بسیار زیادی سرچ متفرقه

https://github.com/Deepayan137/Adapting-OCR/tree/master/src