دانشکده مهندسی کامپیوتر

مباحث ویژه ۱ (یادگیری عمیق)

گزارش نهایی پروژه

علی صداقی ۹۷۵۲۱۳۷۸

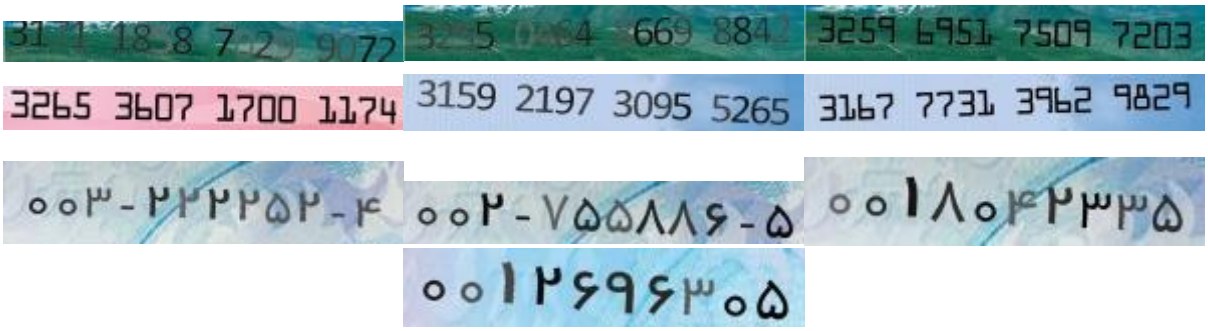
امید میرزاجانی ۹۷۵۲۲۰۶۷

۱ تهیه دیتاست

برای آموزش شبکه به تولید داده‌های مصنوعی پرداخته شد. پیاده‌سازی این بخش درون یک نوتبوک جداگانه صورت گرفت. داده‌های تولید شده در دو گروه بانکی و کارت ملی قرار می‌گیرند. در هر گروه از موارد زیر برای ایجاد داده‌های متنوع استفاده شده است:

- پس‌زمینه‌های متفاوت
- فونت‌های متفاوت
- زاویه‌های چرخش متفاوت

به تصاویر تولید شده نویز مصنوعی نیز اضافه شد تا کمی شبیه داده جهان واقع شود. Label هر تصویر در نام فایل تصویر قرار گرفت و در نهایت تصاویر با فرمت JPG و عرض 200 پیکسل ذخیره شدند. در ادامه تعدادی از تصاویر تولید شده آورده شده است:



با توجه به زمان‌بر بودن تولید دیتاست، تنها توانستیم ۲۰۰۰ تصویر (هر گروه ۱۰۰۰ تصویر) ایجاد کنیم. دیتاست تولید شده درون فایل Dataset.zip موجود است.

۲ شبکه دسته‌بند (Classifier)

با توجه به این که در مستند پروژه اشاره شده بود تصاویر در دو گروه کارت بانکی و کارت ملی هستند و همچنین فارسی بودن اعداد کارت ملی و انگلیسی بودن اعداد کارت بانکی و تفاوت طول رشته اعداد (کارت ملی ۱۰ رقم، کارت بانکی ۱۶ رقم) تصمیم گرفتیم که یک شبکه برای دسته‌بندی این دو کلاس ایجاد کنیم و سپس بر اساس کلاس تصویر آن را به دو شبکه جدا Feed کنیم.

✓ پیاده‌سازی این بخش درون نوتبوک National_Credit_Classifier.ipynb موجود است.

ساختار شبکه بکار رفته مشابه سایر شبکه‌ها به صورت زیر است:

```
Model: "Classifier_Model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 InputLayer (InputLayer)     [(None, 50, 200, 3)]      0

 Conv1 (Conv2D)              (None, 50, 200, 32)       896

 BN1 (BatchNormalization)    (None, 50, 200, 32)       128

 ReLU1 (ReLU)                (None, 50, 200, 32)       0

 Pool1 (MaxPooling2D)        (None, 25, 100, 32)       0

 dropout (Dropout)           (None, 25, 100, 32)       0

 Conv2 (Conv2D)              (None, 25, 100, 64)       18496

 BN2 (BatchNormalization)    (None, 25, 100, 64)       256

 ReLU2 (ReLU)                (None, 25, 100, 64)       0

 Pool2 (MaxPooling2D)        (None, 12, 50, 64)        0

 dropout_1 (Dropout)         (None, 12, 50, 64)        0

 Reshape (Reshape)           (None, 12, 3200)          0

 Dense1 (Dense)              (None, 12, 64)            204864

 Dropout1 (Dropout)          (None, 12, 64)            0

 bidirectional (Bidirectiona  (None, 12, 256)          197632
 l)

 bidirectional_1 (Bidirectio  (None, 128)              164352
 nal)

 Dense2 (Dense)              (None, 2)                 258

=================================================================
Total params: 586,882
Trainable params: 586,690
Non-trainable params: 192
_________________________________________________________________
```

در این شبکه ابتدا ۲ بلاک کانولوشنی مشابه داریم که درون هر بلاک توالی زیر مشاهده می‌شود:

Conv2D --- Batch Normalization --- ReLU --- MaxPool --- Dropout

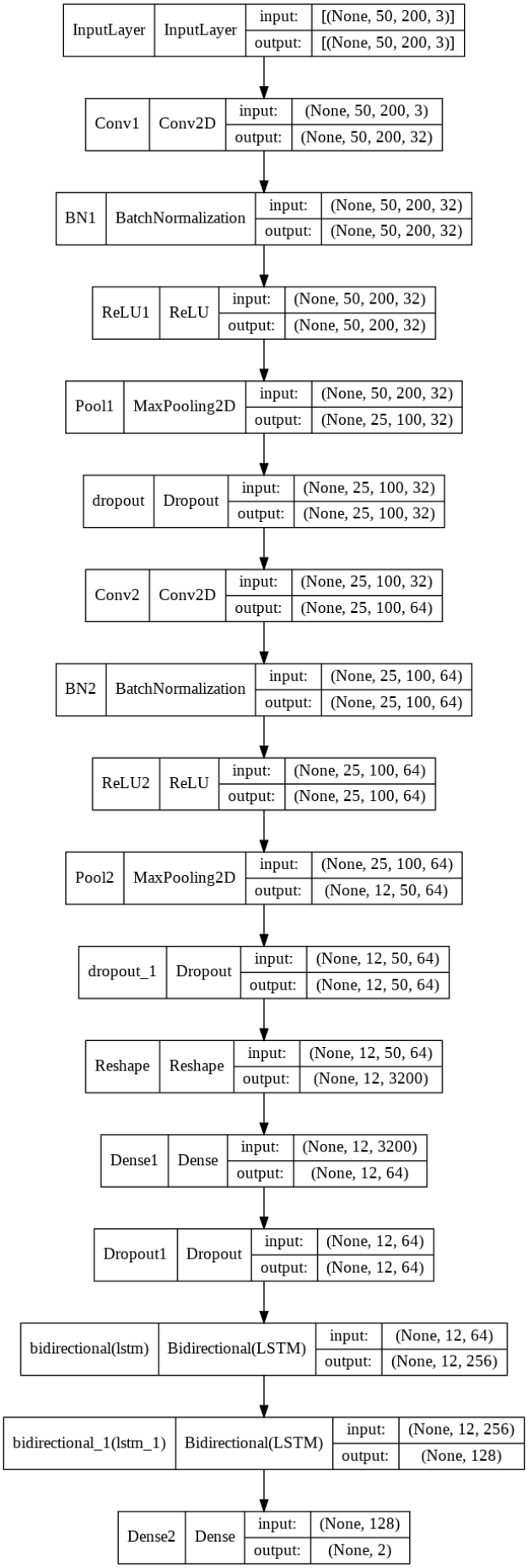
اندازه کرنل لایه کانولوشنی در دو بلاک یکسان و برابر 3 است. همچنین این دو لایه از Padding در حالت Same نیز استفاده می‌کنند. در لایه اول تعداد 32 فیلتر داریم و در لایه دوم 64 فیلتر موجود است. لایه‌های MaxPool نیز دارای Stride برابر 2 هستند. بنابراین تصویر درون هر بلاک کانولوشنی نصف می‌شود (2x Downsample) بنابراین بعد از این دو بلاک اندازه تصویر ربع تصویر ورودی است. از Dropout و Batch Normalization برای حل مشکل Overfit استفاده شده است.

پس از بلاک‌های کانولوشنی از یک لایه Dense با 64 نورون استفاده شده است. به دنبال آن یک ReLU و Dropout نیز داریم.

در ادامه از 2 لایه Bidirectional LSTM به ترتیب با 128 و 64 واحد استفاده شده است. لایه اول تمامی Sequenceها را بر می‌گرداند اما لایه دوم اینطور نیست.

در انتهای شبکه یک دسته‌بند 2 کلاسی با تابع فعال‌سازی Softmax قرار گرفته است.

تصویر دقیق تر مدل در شکل زیر آورده شده است:

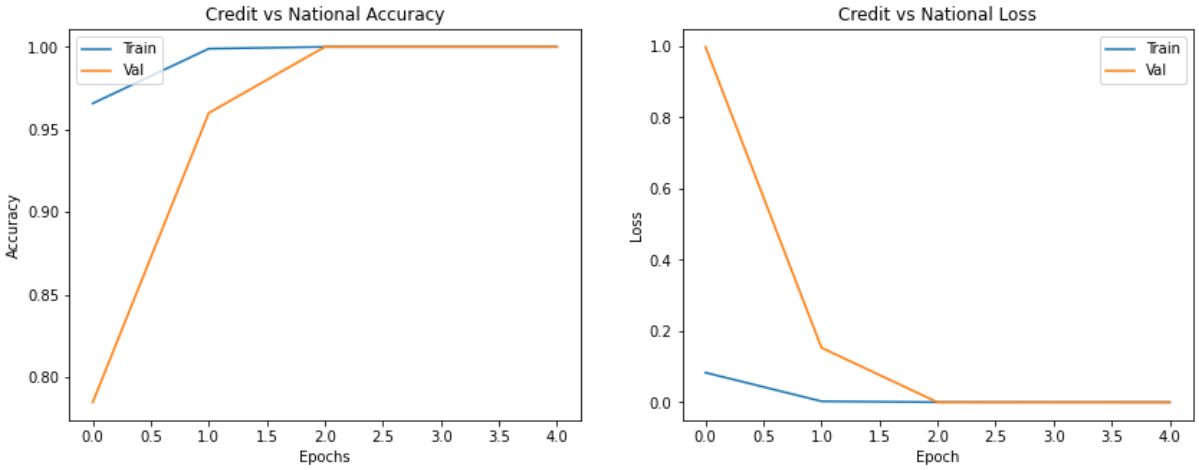

InputLayer | InputLayer | input: [(None, 50, 200, 3)] | output: [(None, 50, 200, 3)]
Conv1 | Conv2D | input: (None, 50, 200, 3) | output: (None, 50, 200, 32)
BN1 | BatchNormalization | input: (None, 50, 200, 32) | output: (None, 50, 200, 32)
ReLU1 | ReLU | input: (None, 50, 200, 32) | output: (None, 50, 200, 32)
Pool1 | MaxPooling2D | input: (None, 50, 200, 32) | output: (None, 25, 100, 32)
dropout | Dropout | input: (None, 25, 100, 32) | output: (None, 25, 100, 32)
Conv2 | Conv2D | input: (None, 25, 100, 32) | output: (None, 25, 100, 64)
BN2 | BatchNormalization | input: (None, 25, 100, 64) | output: (None, 25, 100, 64)
ReLU2 | ReLU | input: (None, 25, 100, 64) | output: (None, 25, 100, 64)
Pool2 | MaxPooling2D | input: (None, 25, 100, 64) | output: (None, 12, 50, 64)
dropout_1 | Dropout | input: (None, 12, 50, 64) | output: (None, 12, 50, 64)
Reshape | Reshape | input: (None, 12, 50, 64) | output: (None, 12, 3200)
Dense1 | Dense | input: (None, 12, 3200) | output: (None, 12, 64)
Dropout1 | Dropout | input: (None, 12, 64) | output: (None, 12, 64)
bidirectional(lstm) | Bidirectional(LSTM) | input: (None, 12, 64) | output: (None, 12, 256)
bidirectional_1(lstm_1) | Bidirectional(LSTM) | input: (None, 12, 256) | output: (None, 128)
Dense2 | Dense | input: (None, 128) | output: (None, 2)

داده‌ها به صورت RGB وارد این شبکه می‌شوند زیرا رنگ عامل تاثیرگذاری در دسته‌بندی کارت ملی و بانکی است. از Augmentation نیز استفاده شده است و میزان روشنایی تصویر بین دو عدد 0.5 و 0.1 تغییر می‌کند. 10 درصد داده‌ها در این بخش را برای Validation جدا کردیم. مدل را با هایپرپارامترهای زیر آموزش می‌دهیم.

- HEIGHT, WIDTH, CHANNELS = 50, 200, 3
- N_CLASSES = 2
- VALIDATION_SPLIT = 0.1
- BATCH_SIZE = 16
- LEARNING_RATE = 0.001
- EPOCHS = 5
- Adam

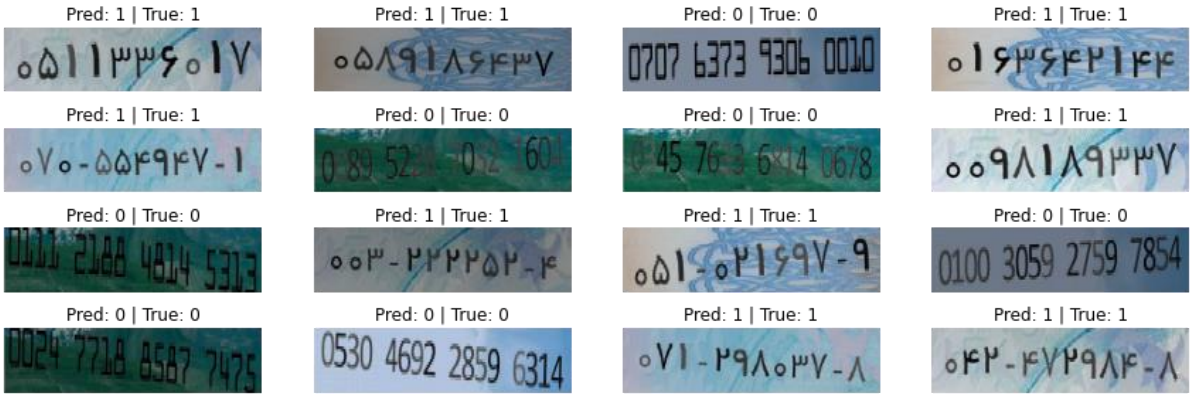
نتایج به صورت زیر است:

```
Epoch 1/5
114/114 [==============================] - 74s 512ms/step - loss: 0.0834 - accuracy: 0.9658 - val_loss: 0.9982 - val_accuracy: 0.7850
Epoch 2/5
114/114 [==============================] - 45s 396ms/step - loss: 0.0028 - accuracy: 0.9989 - val_loss: 0.1530 - val_accuracy: 0.9600
Epoch 3/5
114/114 [==============================] - 48s 420ms/step - loss: 3.0887e-04 - accuracy: 1.0000 - val_loss: 1.3389e-04 - val_accuracy: 1.0000
Epoch 4/5
114/114 [==============================] - 45s 391ms/step - loss: 1.3420e-04 - accuracy: 1.0000 - val_loss: 4.3613e-05 - val_accuracy: 1.0000
Epoch 5/5
114/114 [==============================] - 53s 463ms/step - loss: 6.4223e-05 - accuracy: 1.0000 - val_loss: 3.0459e-05 - val_accuracy: 1.0000
```



همانطور که مشاهده می‌شود مدل به سرعت توانسته دسته‌بندی را کامل یاد بگیرد و دقت آن به عدد 100 درصد رسیده است که حکایت از ساده بودن Task دارد.

در ادامه بهترین مدل (با توجه به Validation Loss) را درون فایل classifier_model.hdf5 ذخیره می‌کنیم و مدل ذخیره شده را روی 16 تصویر جدید Test می‌کنیم. نتیجه در صفحه بعدی آورده شده است.


Pred: 1 | True: 1
Pred: 1 | True: 1
Pred: 0 | True: 0
Pred: 1 | True: 1
Pred: 1 | True: 1
Pred: 0 | True: 0
Pred: 0 | True: 0
Pred: 1 | True: 1
Pred: 0 | True: 0
Pred: 1 | True: 1
Pred: 1 | True: 1
Pred: 0 | True: 0
Pred: 0 | True: 0
Pred: 0 | True: 0
Pred: 1 | True: 1
Pred: 1 | True: 1

۳ شبکه‌های OCR

شبکه‌ای بر پایه شبکی قبلی ایجاد می‌کنیم. این شبکه از تابع ضرر Connectionist Temporal Classification یا CTC استفاده می‌کند. این شبکه دارای دو کانال ورودی است که یکی برای تصاویر Grayscale و دیگری برای دریافت Label هر تصویر است. ساختار شبکه تا قبل از دومین لایه Bidirectional LSTM دقیقا مشابه شبکه قبلی است. اما اینبار در لایه مذکور تمامی Sequenceها را خروجی می‌دهیم. همچنین دسته‌بند نیز این بار 12 کلاس دارد (اعداد ۰ تا ۹ و دو کاراکتر ویژه).

این مدل از یک لایه CTC که توسط ما تعریف شده است به عنوان لایه خروجی استفاده می‌کند. دواقع این لایه یک Wrapper روی ctc_batch_cost است. ctc_batch_cost با دریافت label واقعی و پیشبینی شده مقدار خطای شبکه را محاسبه می‌کند تا بتوانیم با Backpropagation شبکه را طبق آن آموزش دهیم. لایه CTC مخصوص زمان آموزش است و در فاز ارزیابی این لایه از شبکه حذف می‌شود.

- ✓ پیاده‌سازی این بخش درون نوتبوک OCR_Network.ipynb موجود است.

ساختار این شبکه در صفحه بعدی آورده شده است.

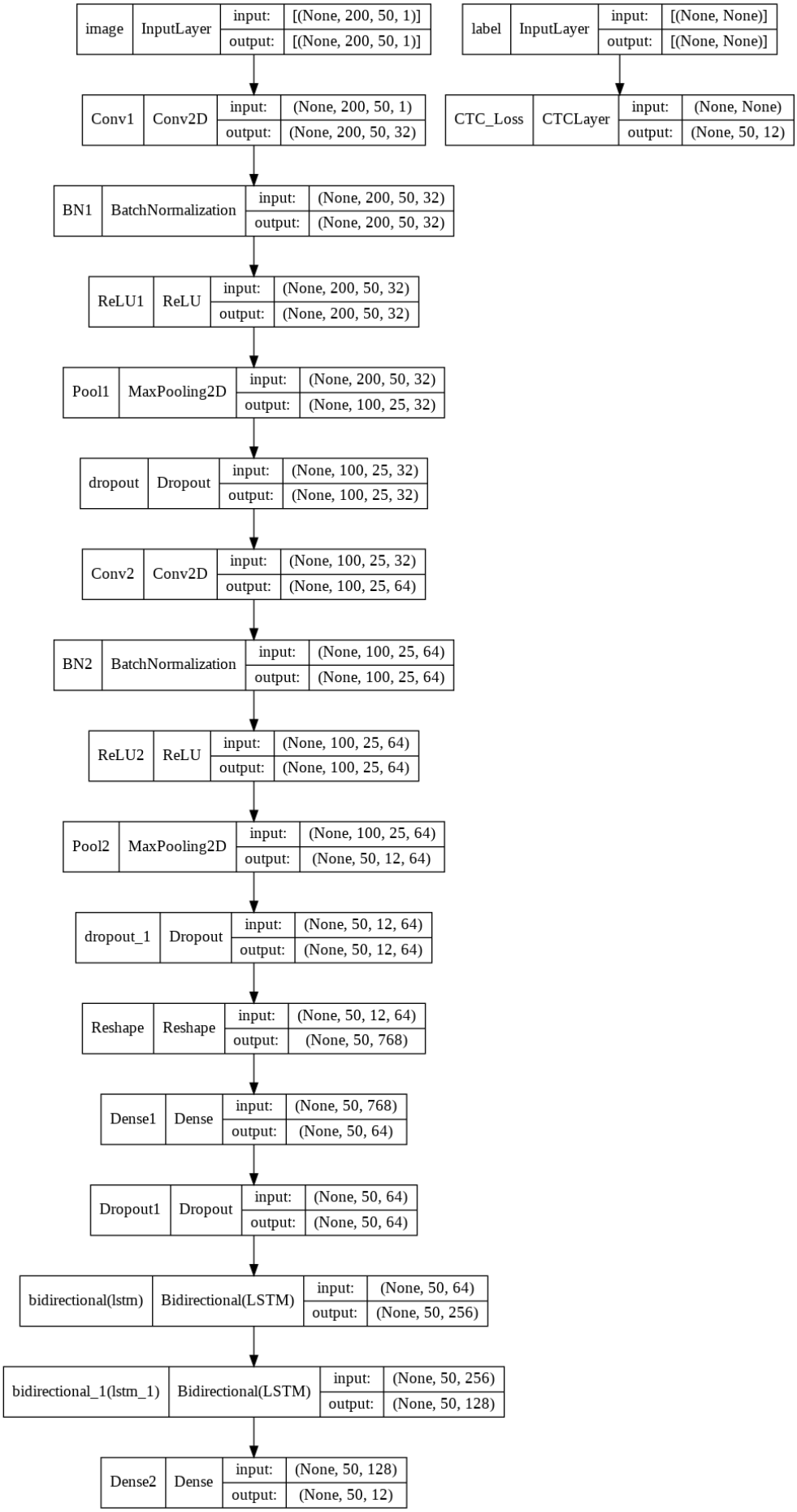


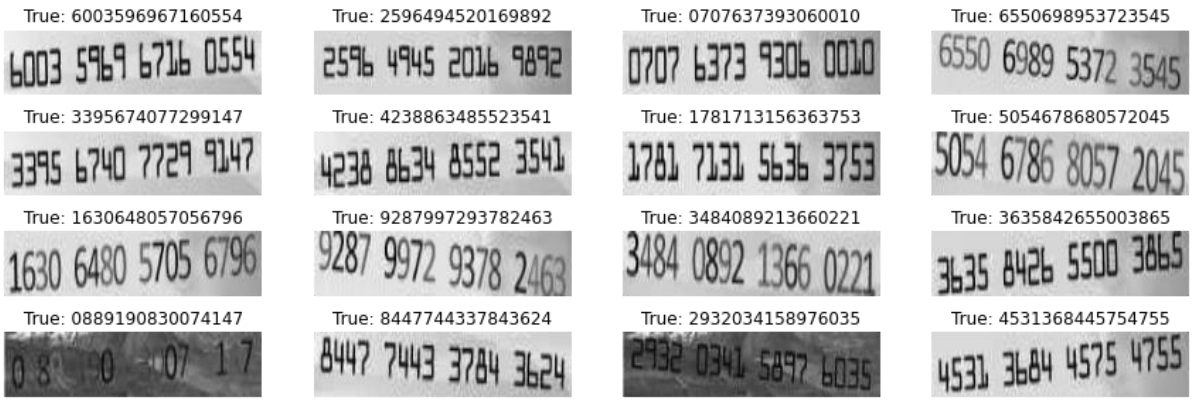
image InputLayer input: [(None, 200, 50, 1)] output: [(None, 200, 50, 1)]
label InputLayer input: [(None, None)] output: [(None, None)]
Conv1 Conv2D input: (None, 200, 50, 1) output: (None, 200, 50, 32)
CTC_Loss CTCLayer input: (None, None) output: (None, 50, 12)
BN1 BatchNormalization input: (None, 200, 50, 32) output: (None, 200, 50, 32)
ReLU1 ReLU input: (None, 200, 50, 32) output: (None, 200, 50, 32)
Pool1 MaxPooling2D input: (None, 200, 50, 32) output: (None, 100, 25, 32)
dropout Dropout input: (None, 100, 25, 32) output: (None, 100, 25, 32)
Conv2 Conv2D input: (None, 100, 25, 32) output: (None, 100, 25, 64)
BN2 BatchNormalization input: (None, 100, 25, 64) output: (None, 100, 25, 64)
ReLU2 ReLU input: (None, 100, 25, 64) output: (None, 100, 25, 64)
Pool2 MaxPooling2D input: (None, 100, 25, 64) output: (None, 50, 12, 64)
dropout_1 Dropout input: (None, 50, 12, 64) output: (None, 50, 12, 64)
Reshape Reshape input: (None, 50, 12, 64) output: (None, 50, 768)
Dense1 Dense input: (None, 50, 768) output: (None, 50, 64)
Dropout1 Dropout input: (None, 50, 64) output: (None, 50, 64)
bidirectional(lstm) Bidirectional(LSTM) input: (None, 50, 64) output: (None, 50, 256)
bidirectional_1(lstm_1) Bidirectional(LSTM) input: (None, 50, 256) output: (None, 50, 128)
Dense2 Dense input: (None, 50, 128) output: (None, 50, 12)

توسط لایه String Lookup که خارج از شبکه قرار دارد برای تبدیل الفبای ورودی به اعداد استفاده می‌کنیم. الفبای ما در این مسئله به صورت 0 تا 9 است. این لایه کاراکتر ناشناخته [UNK] را به این الفبا اضافه می‌کند. تبدیل این لایه در دو طرف صورت می‌گیرد، یعنی هم کاراکتر به رقم و هم رقم به کاراکتر ممکن است.

اکنون به پیش‌پردازش داده ورودی خواهیم پرداخت. ابتدا 20 درصد تصاویر را برای Validation جدا می‌کنیم. با استفاده از tf.image.random_jpeg_quality کیفیت تصاویر را بین 40 تا 80 کاهش می‌دهیم که کمی از حالت مصنوعی بودن خارج شود. این کار دقت مدل در فاز تست روی داده واقعی را افزایش داد. سپس تصاویر RGB را به فضای Grayscale می‌بریم زیرا برای این Task وجود رنگ اهمیت زیادی نداشت. مقدار پیکسل‌ها را بین 0 تا 1 مقیاس می‌کنیم. ابعاد تصاویر را به ارتفاع 50 و عرض 200 می‌بریم. شکل هر تصویر را به صورت (Width, Height) تغییر می‌دهیم تا بتوانیم عرض تصویر را به عنوان سری زمانی (Sequence) در نظر بگیریم و در طی آن حرکت کنیم (به همین دلیل بود که return sequence لایه دوم Bi LSTM را True کردیم. بنابراین 64 کاراکتر به ازای هر تصویر ایجاد می‌شود.)

یک تابع به نام decode_batch_predictions ایجاد می‌کنیم که بر پایه تابع ctc_decode رشته خروجی داده شده توسط مدل را به خروجی مطلوب تبدیل می‌کند. در واقع مدل به ازای هر تصویر 64 کاراکتر خروجی می‌دهد اما تصاویر ما یا 10 رقمی یا 16 رقمی هستند پس تعدادی از این کاراکترهای خروجی داده شده باید حذف شوند. با توجه به اینکه رشته اعداد معنا ندارد از حالت Greedy تابع ctc_decode استفاده می‌کنیم تا بهترین مسیر (رشته) را برای ما پیدا کند. اگر داده ما متن معنا دار بود استفاده از روش Dictionary معقول‌تر از Greedy بود. برای خوانایی بیشتر کاراکتر ویژه [UNK] را با یک x جاگذاری می‌کنیم.

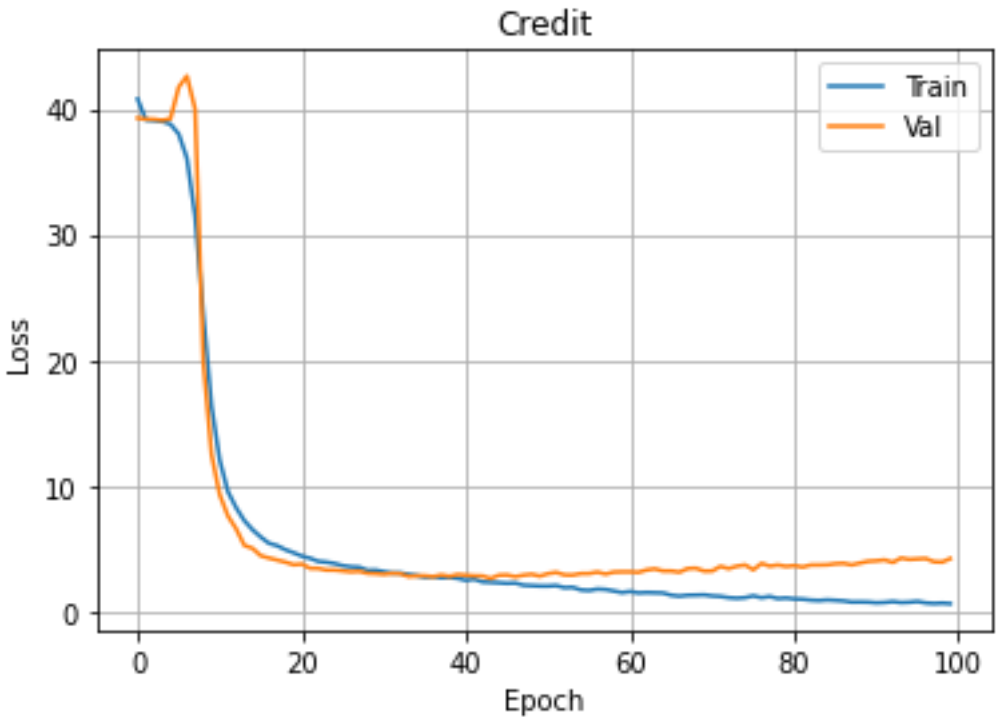
ابتدا یک مدل برای وظیفه کارت بانکی ایجاد می‌کنیم که باید خروجی 16 رقمی بدهد. ورودی این مدل به صورت زیر است:



مدل بالا را با هایپرپارامترهای زیر آموزش می‌دهیم:

- BATCH_SIZE = 16
- LEARNING_RATE = 0.001
- EPOCHS = 100

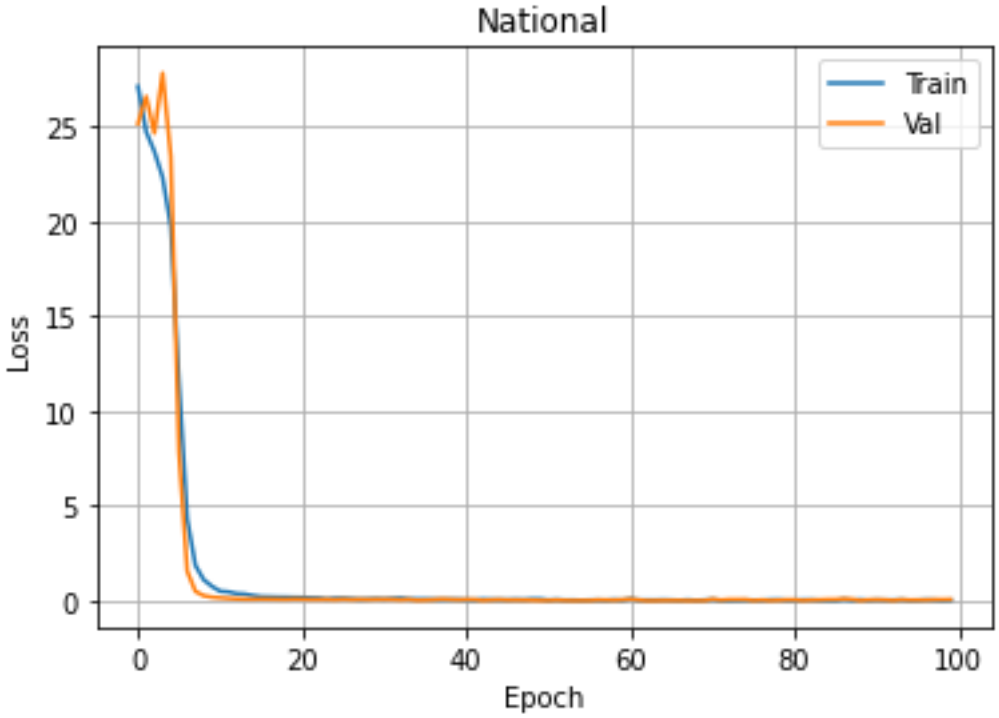
نتیجه آموزش به صورت زیر است:



همانطور که مشاهده می‌شود مدل از اپیاک 40 شروع به Overfit می‌کند. البته در ابتدا مشکل Overfit شدیدتر بود که با استفاده از روش‌های Augmentation، Dropout، BatchNormalization و ... این مشکل را کمتر کردیم. بهترین مدل از نظر Validation Loss را درون فایل credit_model.hdf5 ذخیره می‌کنیم. خروجی مدل روی داده ارزیابی به صورت زیر است:

همانطور که مشاهده می‌شود دقت مدل عالی بوده و توانسته داده‌های مصنوعی که تا کنون مشاهده نکرده بوده را به خوبی بخواند.

اکنون کارهای انجام شده برای کارت بانکی را بر روی کارت ملی انجام می‌دهیم. این Task ساده‌تر بوده و انتظار عملکرد بهتری داریم. نتایج به صورت زیر است:



همانطور که مشاهده می‌شود این بار دیگر مشکل Overfit پیش نیامده و مقدار خطا هم روی داده آموزش و هم ارزیابی تقریبا به صفر همگرا شده. در شکل زیر نیز مشاهده می‌شود که مدل هیچ خطایی نداشته است.

بهترین مدل این بخش درون فایل national_model.hdf5 موجود است.

۴ شبکه End-to-End

اکنون با داشتن موارد زیر به پیاده‌سازی یک شبکه E2E می‌پردازیم.

- مدل classifier_model.hdf5
- مدل credit_model.hdf5
- مدل national_model.hdf5

نحوه کار این شبکه به این صورت است که ابتدا یک تصویر (یا یک پوشه حاوی تصاویر) ورودی داده می‌شود، سپس کلاس هر تصویر مشخص می‌شود که مربوط به کارت ملی است یا کارت بانکی. بر اساس کلاس مشخص شده، تصویر وارد شبکه مروبطه خواهد شد و ارقام درون آن خوانده می‌شود.

✓ پیاده‌سازی این بخش درون نوتبوک E2E_Wrapper.ipynb موجود است.

برای Test کامل مدل از ۲ دیتاست استفاده خواهیم کرد:

- tests_fake که شامل 40 تصویر (هر کلاس 20) مصنوعی است که نه در داده Train بوده نه Validation

- tests_real که شامل 9 تصویر Wild و واقعی است.

نتایج Test روی داده مصنوعی بسیار عالی است و دقت برابر 97.69 درصد شده است.

```
Accuracy: 0.9769230769230769
```

اما نتایج Test روی داده واقعی خیلی جالب نبوده و برابر 44.73 درصد است. البته شایان ذکر است که تعداد داده واقعی بسیار کم بود (فقط ۹ عدد) و امکان بی کیفیت بودن این تصاویر وجود دارد. نتیجه این بخش در ادامه آورده شده است:

```
Pred Class: National
Pred Label: 0006165591
True Label: 0006165591
----------------------
Pred Class: National
Pred Label: 0006657184
True Label: 0006657184
----------------------
Pred Class: National
Pred Label: 9022995419
True Label: 0022785418
----------------------
Pred Class: National
Pred Label: 61666911xx
True Label: 0067062571
----------------------
Pred Class: National
Pred Label: 1244567890
True Label: 1234567890
----------------------
Pred Class: Credit
Pred Label: 5666122383321523
True Label: 6037991165316294
----------------------
Pred Class: Credit
Pred Label: 7612752276885627
True Label: 6037993249885617
----------------------
Pred Class: Credit
Pred Label: 51693379165666xx
True Label: 6104337018560571
----------------------
Pred Class: Credit
Pred Label: 1622186106984365
True Label: 6221061069843656
----------------------
Accuracy: 0.4473684210526316
```

شما به وسیله تابع predict_folder(folder_path) و ورودی دادن مسیر یک فولدر می‌توانید نتایج بالا را مشاهده کنید.

شما به وسیله تابع single_file_show(path, label) می‌توانید خروجی به ازای یک تصویر را به صورت زیر مشاهده کنید.

```
path = 'tests_fake/0380794529239760.jpg'
label = '0380794529239760'
single_file_show(path, label)
```


```
Pred Class: Credit
Pred Label: 08807945292397 60
True Label: 0380794529239760
```

۵ منابع

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

https://keras.io/examples/vision/captcha_ocr/

https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5

https://deepspeech.readthedocs.io/en/v0.8.2/Decoder.html

https://docs.w3cub.com/tensorflow~python/tf/keras/backend/ctc_batch_cost

https://docs.w3cub.com/tensorflow~python/tf/keras/backend/ctc_decode

https://www.cs.toronto.edu/~graves/icml_2006.pdf

در پیاده‌سازی از لینک دوم کمک زیادی گرفته شده است.