**بسمه تعالی**


**دانشکده مهندسی کامپیوتر**

# یادگیری عمیق

**نام استاد: دکتر محمدی**

## پروژه نهایی

**آرمان حیدری – سید سینا ضیایی**

**شماره دانشجویی: ۹۷۵۲۱۲۵۲ – ۹۷۵۲۱۳۸۷**

**بهمن ۱۴۰۰**

# فهرست

## چکیده

در این پروژه می خواهیم با استفاده از تکنیک های ocr، که وظیفه تشخیص خودکار متون را دارد اعداد ۱۰ رقمی فارسی موجود روی کارت های ملی (قدیمی یا جدید) و همچنین اعداد انگلیسی ۱۶ رقمی روی کارت بانکی های مختلف را از تصاویر استخراج کنیم.

وظیفه تهیه دیتاست با خودمان بود و سعی کردیم دیتاستی تا حد امکان واقعی را تهیه کنیم و یک شبکه عصبی RCNN با تابع ضرر CTC را با آن آموزش دادیم تا این تسک را انجام دهد. طبیعتا داده آموزشی و ارزیابی جدا از هم به این منظور در نظر گرفته ایم.

## روند کلی کار

چون تجربه کار زمینه ocr را نداشتیم، به جستجو در اینترنت پرداختیم و با استفاده از این لینک مسئله را بهتر متوجه شدیم. و توضیحات و کد این لینک را هم بررسی کردیم. سپس برای تصمیم گیری بهتر در زمینه مدل انتخابی در چنین مسائلی این لینک که کد خوبی برای تشخیص reCAPTCHA با استفاده از کراس بود را استفاده کردیم.

سپس وارد کد شدیم و در بستر گوگل کولب در section های مختلفی (برای واضح بودن مراحل) این مسئله را پیاده سازی کردیم. به طور کلی اول دیتاستی را ساختیم که برای ذخیره آن، گوگل درایو را mount کردیم تا هر بار مجبور به ایجاد دوباره دیتاست نشویم. در ساختن این دیتاست هم از این منبع استفاده کردیم.

بعد از preprocessing روی دیتاها، مدلی را ساختیم و آن را روی دیتای آموزشی fit کردیم. در تمام مراحل دیتا های مربوط به فارسی و انگلیسی shuffle بودند و یک مدل آن ها را یاد میگیرد. البته تفاوت طول خروجی را با قرار دادن ۶ کاراکتر space در انتهای کدهای ملی حل کردیم. و در آخر دیتای ارزیابی را بررسی کردیم و دقت مدل و روند کاهش loss آن را بررسی کردیم. این مراحل به تفصیل در ادامه توضیح داده شده اند.

سپس یک پیاده سازی از این مدل با استفاده از فریم ورک فلاتر داشتیم. و اپلیکیشن اندرویدی ساده شده ای از آن را ایجاد کردیم.

## ایجاد dataset

```
print("current_path:", current_path)

font_arial_url = 'https://github.com/sinaziaee/optical_digit_recognizer/raw/main/fonts/arial.ttf'
font_ocr_url = 'https://github.com/sinaziaee/optical_digit_recognizer/raw/main/fonts/ocr_a.ttf'
font_yekan_url = 'https://github.com/sinaziaee/optical_digit_recognizer/raw/main/fonts/yekan.ttf'
font_nazanin_url = 'https://github.com/sinaziaee/optical_digit_recognizer/raw/main/fonts/nazanin.ttf'
bank_base_url = 'https://github.com/sinaziaee/optical_digit_recognizer/raw/main/bank_background_images'
id_base_url = 'https://github.com/sinaziaee/optical_digit_recognizer/raw/main/id_background_images/'

!wget -P /content/fonts $font_arial_url -N -q
!wget -P /content/fonts $font_ocr_url -N -q
!wget -P /content/fonts $font_yekan_url -N -q
!wget -P /content/fonts $font_nazanin_url -N -q

for i in range(1, 34):
  img_path = f'{bank_base_url}/{i}.jpg'
  !wget -P /content/bank $img_path -N -q

for i in range(1, 3):
  img_path = f'{id_base_url}/id{i}.jpg'
  !wget -P /content/id $img_path -N -q
```

در اینجای کار سرغ منابع اولیه مانند فونت‌های مورد نیاز و عکس‌های بک گراندی که بعداً قرار هست برای تولید عکس به کار روند استفاده میکنیم.

```
# creating folders if they don't exist
if not os.path.isdir(f'/content/drive/MyDrive/{drive_path}/'):
  os.mkdir(f'/content/drive/MyDrive/{drive_path}/')

if not os.path.isdir(f'/content/drive/MyDrive/{drive_path}/bank/'):
  os.mkdir(f'/content/drive/MyDrive/{drive_path}/bank/')

if not os.path.isdir(f'/content/drive/MyDrive/{drive_path}/id/'):
  os.mkdir(f'/content/drive/MyDrive/{drive_path}/id/')

if not os.path.isdir(f'{bank_bg_base_path}/'):
  os.mkdir(f'{bank_bg_base_path}/')

if not os.path.isdir(id_bg_base_path):
  os.mkdir(id_bg_base_path)

if not os.path.isdir(id_base_path):
  os.mkdir(id_base_path)

if not os.path.isdir(bank_base_path):
  os.mkdir(bank_base_path)
```

در اینجای کار ما به دنبال ساخت پوشه هایی در google drive هستیم که بعداً قرار است عکس‌ها در آن‌ها ذخیره شوند.

```
# cropping images with the help of opencv and cutting
for i, file in enumerate(files):
  img = cv.imread(files[i])
  h, w, c = img.shape
  new = cv.cvtColor(img, cv.COLOR_BGR2RGB)
  if w > 200:
    h_sec = int(h/7)
    w_sec = int(w/6)
    new = img[3*h_sec:6*h_sec, w_sec:5*w_sec,]
  else:
    h_sec = int(h/14)
    w_sec = int(w/12)
    new = img[3*h_sec:12*h_sec, w_sec:12*w_sec,]
  try:
    cv.imwrite(f'{bank_bg_base_path}/{i}.jpg', new)
  except Exception as e:
    pass
print(len(files))
```

در این بخش نیز به کمک کتابخانه ی opencv عکس‌ها را لود کردیم و متناسب با اینکه عرض عکس از 200 پیکسل کوچکتر هست یا خیر عکس را کراپ کرده‌ایم تا به بخشی که معمولاً شماره های کارت های بانکی قرار میگیرند برسیم.

```
random_bank_num = 10
random_id_num = 100
```

این دو متغیر رندم نیز تعداد دفعاتی هستند که قرار است در حلقه ی loop مربوط به هر دیتاست عدد random تولید کنیم(التبه تعدادشان در اندازه ی loop های دیگر نیز ضرب می شود).

حال می‌رویم سراغ تولید دیتاست های مورد نیاز. مراحل کد های تولید عکس‌ها و دیتاست در کد کامنت گذاری نیز شده است.

**کارت های بانکی)**

از ترکیبی از دو فونت انگلیسی و ۳۳ عکس بک گراند واقعی کارت های بانکی با استفاده از یک عدد رندم استفاده شده. همچنین اعداد روی کارت ها با دو رنگ سفید و سیاه نوشته شده‌اند که هر دو حالت را ساپورت کنند. پس از این مرحله با یک حلقه ی for loop روی داده‌های هر بخش یک عدد رندم ایجاد بین ۱ تا ۱۰۰ ایجاد میکنیم که اگر از ۴۰ این عدد کمتر باشد عکس را به صورت رندم rotate و اگر بین ۲۰ و ۶۰ باشد blur میکنیم و این داده‌های اضافی شده را به دیتاست داده‌های اولیه میدهیم.

در آخر نیز حدود ۱۹ عکس از کارت های بانکی واقعی کراپ شده را به دیتاست اضافی میکنیم.

**کارت های ملی)**

در اینجا نیز از ترکیبی از دو فونت فارسی و ۵ عکس پس زمینه ی واقعی کارت های ملی استفاده میکنیم ( همه در for loop ها قرار دارند) و به صورت رندم کارت های ملی را تولید میکنیم. سپس مرحله ی چرخش و blur را مانند بالا روی این عکس‌ها هم به صورت رندم اجرا میکنیم.

**تصاویر واقعی)**

تعدادی تصویر واقعی هم در درایو ذخیره کردیم و از آن ها هم برای آموزش مدل استفاده کردیم تا به نتایج بهتری برسیم.

## پیش پردازش داده ها

### هایپرپارامترها)

```
# Path to the data directory
bank_dir = Path('drive/MyDrive/project_ocr/bank')
bank_df = pd.read_csv('drive/MyDrive/project_ocr/bank_labels.csv')
id_dir = Path('drive/MyDrive/project_ocr/id')
id_df = pd.read_csv('drive/MyDrive/project_ocr/id_labels.csv')

# To show that we want to use a model that we had before, or a new model
is_load_model = True
# Batch size for training and validation
batch_size = 16

# Desired image dimensions
img_width = 200
img_height = 50
```

موارد بالا ابرپارامترهای مربوط به پیش پردازش و مدل هستند. چون تمام تصاویر ورودی به مدل باید هم سایز باشند، عددی که به نظر بین دیتاهای ما و در کل عکس های ورودی به این مدل معقول باشد را انتخاب کردیم. متغیر سوم عکس ها همواره ۳ است و به صورت رنگی RGB تصاویر آموزشی و ارزیابی را به مدل خواهیم داد و به همین خاطر داخل هایپرپارامترها نیامده است.

Is_load_model: اگر برابر false باشد مدل جدیدی ساخته می شود (مطابق توضیحات بخش مدل) و اگر true باشد، مدلی که قبلا تمرین داده شده است load می شود.

دایرکتوری ها هم برای عکس های بانکی و ملی، به همراه فایل های csv آن ها که در بخش قبل ساخته بودیم هستند.

### خواندن داده ها)

در این بخش ابتدا داده های بانکی و سپس داده های کارت ملی را میخوانیم. و سپس آن ها را در کنار هم قرار میدهیم. به این صورت داخل images، تمام آدرس های عکس ها را خواهیم داشت و داخل labels، برای هر index متناظر پاسخ شبکه را داریم. برای عکس های فارسی، label های فارسی با ۶ کاراکتر space در انتها قرار داده ایم. به این صورت همواره از مدل انتظار خروجی ۱۶ کاراکتری خواهیم داشت.

```
# Get list of all the images
bank_images = list(path for path in os.listdir(bank_dir) if path.endswith('.jpg'))
# Create list of labels with respect to bank_labels.csv
bank_labels = []
for i,img in enumerate(bank_images):
  bank_labels.append(str(bank_df[bank_df.image_name==int(img.split('.jpg')[0])].iloc[0]['label']))
  bank_images[i] = os.path.join(bank_dir, img)
# Do the same for id cards
id_images = list(path for path in os.listdir(id_dir) if path.endswith('.jpg'))
id_labels = []
for i,img in enumerate(id_images):
  label = str(id_df[id_df.image_name==int(img.split('.jpg')[0])].iloc[0]['label'])
  temp = ''
  if '[' in label:
    inx1 = label.index(',')
    inx2 = label.index(']')
    label = label[inx1+2: inx2]
  for c in label:
    temp+=to_persian(int(c))
  id_labels.append(temp + 6*' ')
  id_images[i] = os.path.join(id_dir, img)

# Concat all of the dataset
images = bank_images + id_images
labels = bank_labels + id_labels
max_length = max([len(label) for label in labels])
```

متغیر max_length که حداکثر طول label است (در اینجا ۱۶ به دست می آید) را در ادامه نیاز خواهیم داشت.

بعد توابع زیر را تعریف کرده ایم که توضیح هر کدام در زیر تصویر کد آمده است:

```
# Mapping characters to integers
char_to_num = layers.StringLookup(
    vocabulary=characters, mask_token=None
)

# Mapping integers back to original characters
num_to_char = layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), mask_token=None, invert=True
)
```

این توابع با استفاده از لایه StringLookup در کراس، برای عددی کردن کاراکتر ها هستند. متغیر characters که به آن پاس داده شده است لیستی شامل تمام کاراکتر های موجود در فضای مسئله است. که در اینجا ۲۱ متغیر شامل اعداد ۱ تا ۹ و 1 تا 9 و ‘ ’ است. برای تبدیل برچسب های رشته ای ما به اعدادی برای دادن به مدل و محاسبه loss حساب می شود.

```python
def split_data(images, labels, train_size=0.9, shuffle=True):
    # 1. Get the total size of the dataset
    size = len(images)
    # 2. Make an indices array and shuffle it, if required
    indices = np.arange(size)
    if shuffle:
        np.random.shuffle(indices)
    # 3. Get the size of training samples
    train_samples = int(size * train_size)
    # 4. Split data into training and validation sets
    x_train, y_train = images[indices[:train_samples]], labels[indices[:train_samples]]
    x_valid, y_valid = images[indices[train_samples:]], labels[indices[train_samples:]]
    return x_train, x_valid, y_train, y_valid
```

با استفاده از این تابع، ترتیب عکس ها را به همراه label مربوط به هر عکس به هم میزنیم. سپس به مقدار train_size یعنی ۹۰ درصد آن را برای داده آموزشی و ۱۰ درصد داده ها را به عنوان داده validation برمیداریم تا عملکرد مدل را با آن بسنجیم.

```python
def encode_single_sample(img_path, label):
    # 1. Read image
    img = tf.io.read_file(img_path)
    # 2. Decode and convert to grayscale
    img = tf.io.decode_png(img, channels=3)
    # 3. Convert to float32 in [0, 1] range
    img = tf.image.convert_image_dtype(img, tf.float32)
    # 4. Resize to the desired size
    img = tf.image.resize(img, [img_height, img_width])
    # 5. Transpose the image because we want the time
    # dimension to correspond to the width of the image.
    img = tf.transpose(img, perm=[1, 0, 2])
    # 6. Map the characters in label to numbers
    label = char_to_num(tf.strings.unicode_split(label, input_encoding="UTF-8"))
    # 7. Return a dict as our model is expecting two inputs
    return {"image": img, "label": label}
```

این تابع مهمی است که مرحله به مرحله آن با کامنت توضیح داده شده است. از این تابع برای خواندن عکس ها و دادن آن به مدل استفاده میکنیم. چون تا به اینجا فقط مسیر عکس ها را داشتیم. این تابع با گرفتن آدرس عکس و برچسب آن، بر چسب را با استفاده از تابع char_to_num که بالاتر دیدیم به مقادیر عددی تبدیل میکند. و عکس را با استفاده از توابع tensorflow به صورت سه کاناله RGB

میخواند، آن را نرمال سازی بین ۰ و ۱ میکند و سپس سایز آن را به مقادیری که در هایپرپارامتر مشخص کردیم map میکند.

## Create `Dataset` objects

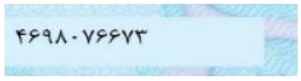
```
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = (
    train_dataset.map(
        encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE
    )
    .batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)

validation_dataset = tf.data.Dataset.from_tensor_slices((x_valid, y_valid))
validation_dataset = (
    validation_dataset.map(
        encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE
    )
    .batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)
```

در این سلول، تابع encode_single_sample بالا را هر بار روی یک batch از داده ها اعمال میکنیم و object هایی برای دیتاست train و validation ایجاد میکنیم. تعداد پردازش های موازی در این قسمت توسط api مخصوص tensorflow برای خواندن دیتاها استفاده میشود که زمان خواندن را به صورت پویا autotune میکند.

سپس یکی از batch های ساخته شده پس از این مراحل را به همراه برچسب آن ها را در صفحه بعد میبینیم.

۴۶۹۸۰۷۶۶۷۳




2033180185078506

۲۷۲۶۳۴۷۷۶۰


۳۰۴۰۶۵۰۷۹۲


9615524633883849

3770830181711571


3612979059386835

۳۶۰۴۰۴۳۰۲۱


۲۲۹۷۷۰۳۸۰۶


9056425591105545


2912967525617158

۸۶۷۹۸۷۱۶۱۴


2898341082117504

9846789912384761

1194909509665314


5558075762374714

## تعریف مدل و آموزش آن

ساختار مدلی که با مطالعه از اینترنت به دست آوردیم و کمی تغییرات روی آن دادیم به این صورت شد:

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| image (InputLayer) | [(None, 200, 50, 3)] | 0 | [] |
| Conv1 (Conv2D) | (None, 200, 50, 32) | 896 | ['image[0][0]'] |
| pool1 (MaxPooling2D) | (None, 100, 25, 32) | 0 | ['Conv1[0][0]'] |
| Conv2 (Conv2D) | (None, 100, 25, 64) | 18496 | ['pool1[0][0]'] |
| pool2 (MaxPooling2D) | (None, 50, 12, 64) | 0 | ['Conv2[0][0]'] |
| reshape (Reshape) | (None, 50, 768) | 0 | ['pool2[0][0]'] |
| dense1 (Dense) | (None, 50, 64) | 49216 | ['reshape[0][0]'] |
| dropout_10 (Dropout) | (None, 50, 64) | 0 | ['dense1[0][0]'] |
| bidirectional_20 (Bidirectional) | (None, 50, 128) | 66048 | ['dropout_10[0][0]'] |
| bidirectional_21 (Bidirectional) | (None, 50, 128) | 98816 | ['bidirectional_20[0][0]'] |
| label (InputLayer) | [(None, None)] | 0 | [] |
| dense2 (Dense) | (None, 50, 23) | 2967 | ['bidirectional_21[0][0]'] |
| ctc_layer (CTCLayer) | (None, 50, 23) | 0 | ['label[0][0]', |

این شبکه دارای دو کانال ورودی است که یکی برای تصاویر RGB و دیگری برای دریافت Label هر تصویر است. 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 ۳*۳ است. هم چنین این دو لایه از Padding در حالتSame نیز استفاده میکنند. در لایه اول تعداد ۳۲ فیلتر داریم و در لایه دوم ۶۴ فیلتر موجود است تا ویژگی های بیشتری استخراج شود. لایه های MaxPool نیز دارای Stride برابر ۲ هستند. بنابراین تصویر درون هر بلاک کانولوشنی نصف میشود بنابراین بعد از این دو بلاک اندازه تصویر، یک چهارم تصویر ورودی است. از Dropout و Batch Normalization برای حل مشکل احتمالی overfit استفاده شده است. چون دیتاست مصنوعی همواره خطر overfit شدن دارد.

پس از بلاکهای کانولوشنی از یک لایه Dense با ۶۴ نورون استفاده شده است. به دنبال آن یک ReLU و Dropout نیز داریم. در ادامه از ۲ لایه Bidirectional LSTM (که همان lstm دو طرفه است) با ۶۴ واحد برای هر کدام استفاده شده است. لایه اول تمامی ترتیب ها را بازمیگرداند و به لایه دوم میدهد که فقط آخری را برمیگرداند.

در انتهای شبکه یک دسته بند با تغداد کاراکترهای موجود در مسئله (۲۱) کلاسی با تابع فعالسازی Softmaxقرار گرفته است.

همچنین از adam به عنوان optimizer و ctc loss به عنوان تابع ضرر استفاده کرده ایم. که لایه مربوط به ctc loss را به صورت custom تعریف میکنیم چون در keras پیاده سازی نشده است. در مسائل ocr معمولا از این تابع ضرر connection temporal classification استفاده میکنند. درواقع 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 به کلی از شبکه حذف می‌شود.

```
epochs = 1
early_stopping_patience = 10
# Add early stopping
early_stopping = keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=early_stopping_patience, restore_best_weights=True
)

# Train the model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs,
    callbacks=[early_stopping],
)

model.save('drive/MyDrive/project_ocr/our_model.h5')
```

```
131/131 [==============================] - 190s 1s/step - loss: 5.2494 - val_loss: 4.9654
```

حالا که مدل و دیتاست آماده شده است، آن را فیت میکنیم. از EarlyStopping برای متوقف کردن مدل در صورت افزایش loss برای داده های validation استفاده میکنیم، تا اگر مدل در حال overfit بود

روند متوقف شود. متاسفانه از ابتدای آموزش مدل اسکرین شاتی در دسترس نیست چون زمان آموزش طولانی بود و محدودیت GPU باعث میشد که با save کردن مدل در درایو و دوباره load کردن آن آموزش را ادامه دهیم. در مجموع از train_loss=58 به train_loss=5 در طول ۱۲۰ ایپاک رسیده است که کاملا روند آموزشی صحیح را نمایش میدهد. در طول آموزش مقدار تایع ضرر روی validation هم به طور مشابه کاهش می یابد و لذا overfit هم در کار نبوده و از کالبک هم استفاده نمی شود.

## ارزیابی مدل و نتایج

میخواهیم آزمایش های مختلفی روی مدل کنیم و عملکرد و دقت آن را بسنجیم.

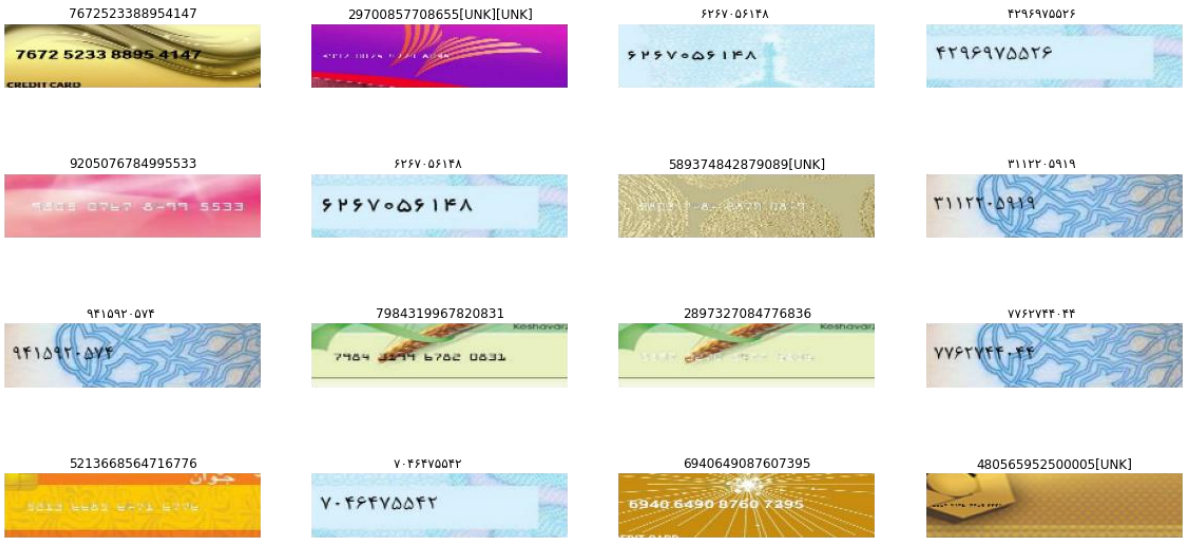
Test

```
# Get the prediction model by extracting layers till the output layer
prediction_model = keras.models.Model(
    model.get_layer(name="image").input, model.get_layer(name="dense2").output
)

# A utility function to decode the output of the network
def decode_batch_predictions(pred):
    input_len = np.ones(pred.shape[0]) * pred.shape[1]
    # Use greedy search. For complex tasks, you can use beam search
    results = keras.backend.ctc_decode(pred, input_length=input_len, greedy=True)[0][0][:, :max_length]
    # Iterate over the results and get back the text
    output_text = []
    for res in results:
        res = tf.strings.reduce_join(num_to_char(res)).numpy().decode("utf-8")
        output_text.append(res)
    return output_text
```

ابتدا مدل را بدون تابع ضرر ctc میسازیم و از وزن های طول آموزش استفاده میکنیم. بعد تابعی برای تبدیل کد های خروجی مدل به رشته ای از کاراکتر های تعریف شده نوشته ایم که خروجی text میدهد و label ای که مدل میدهد را decode میکند. خروجی را به طول max_length که در ابتدا حساب کردیم و برابر ۱۶ است قرار میدهد.

برای یک batch از داده های Validation که مدل آن ها ندیده است، خروجی گرفته ایم و نتیجه پیشبینی های مدل را میبینیم:



که نتایج بسیار خوبی است و تنها خطای مدل در جایی بوده که رقم آخر عکس ردیف دوم و ستون سوم کمی ناواضح است و عکس پایین سمت راست هم نوشته بسیار ریز است. چون دیتاست ما کمی نویزی هم هست تا به دنیای واقعی نزدیک تر باشد.

در نهایت دقت را با محاسبه ی مقدار تشابه کاراکتر به کاراکتر به صورت زیر روی داده Validation و به طور مشابه روی داده تست محاسبه کردیم که عدد قابل قبولی است:

```
# Calculate accuracy on validation set
# We count the true predicted digits for each of the data and then divide it by all of them
results=[]
for batch in validation_dataset.take(500):
    batch_images = batch["image"]
    batch_labels = batch["label"]

    preds = prediction_model.predict(batch_images)
    pred_texts = decode_batch_predictions(preds)

    orig_texts = []
    for label in batch_labels:
        label = tf.strings.reduce_join(num_to_char(label)).numpy().decode("utf-8")
        orig_texts.append(label)

    true_count = 0
    total_count = 0
    for i in range(len(orig_texts)):
      for j,char in enumerate(orig_texts[i]):
        # Check if the charracters are the same, we have to do this for every char of the whole batch
        if char == pred_texts[i][j]:
          true_count+=1
        total_count+=1
    results.append(true_count/total_count)

print("accuracy on validation:", np.mean(results)*100, "%")
```

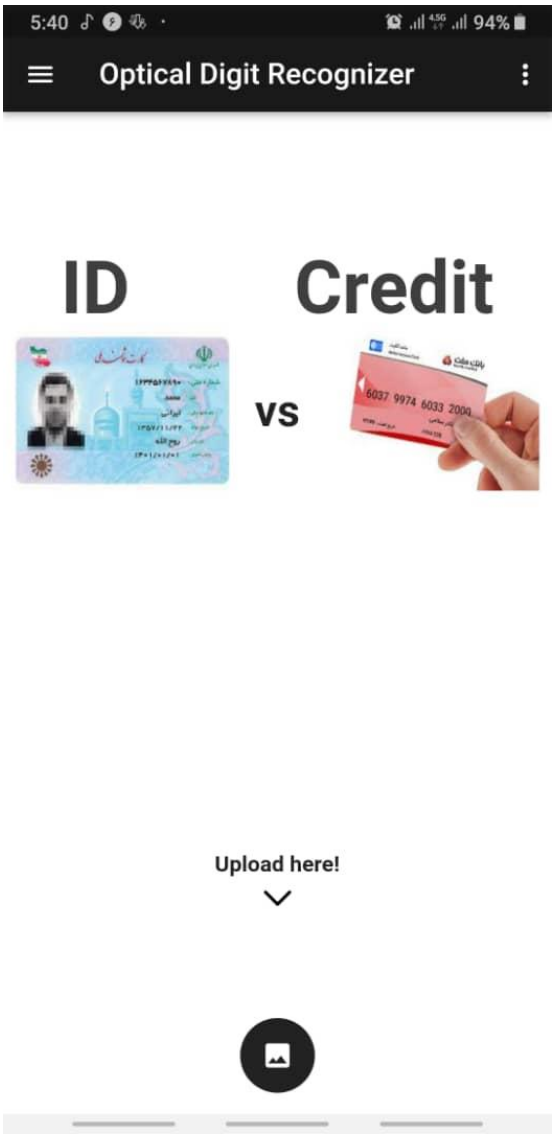
```
accuracy on validation: 84.69907407407408 %
```

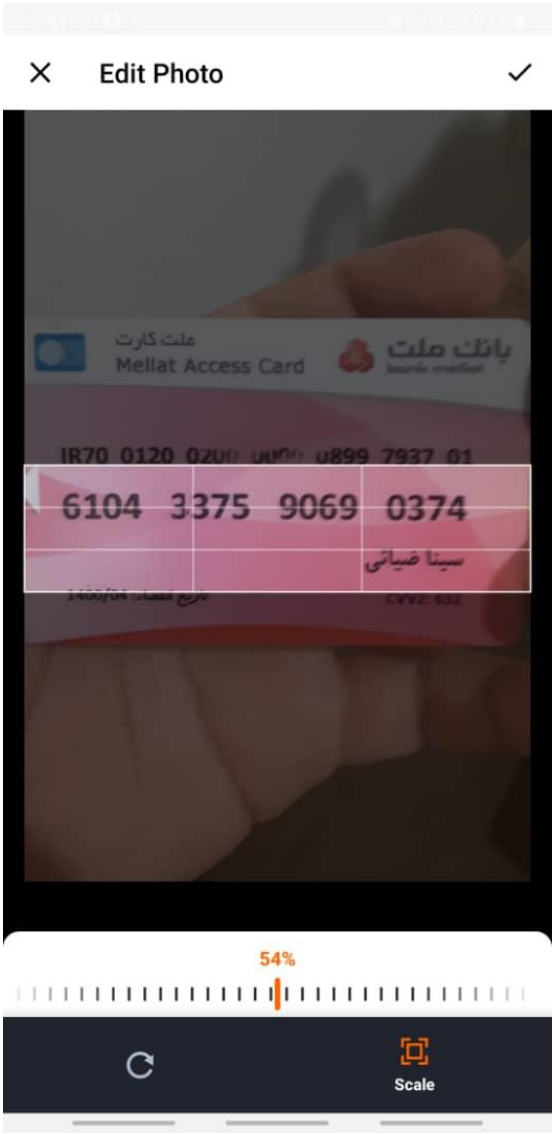
```
accuracy on train: 87.87869751908397 %
```

در نهایت تابع predict را تعریف کرده ایم که مسیر یک عکس را میگیرد، پیش پردازش روی آن را مشابه تابع encode_single_sample انجام میدهد و آن را به مدل میدهد و خروجی را Decode کرده و بازمیگرداند. به این ترتیب برای هر عکس دلخواه میتوان درستی مدل را آزمود.

## اپلیکیشن اندروید

نکته ی قابل توجه ای نیز وجود دارد که بعد از آموزش مدل و خروجی گرفتن از آن کدی مربوط به پیاده‌سازی مدل در برنامه ی اپلیکیشن اندرویدی نیز زده شده که با خروجی عکس گرفتن و یا انتخاب عکس از گالری میتوان به عنوان ورودی مدل داده و پیشبینی کرد که عدد ما کارت بانکی یا کارت شناسایی است و چه عددی داخلش وجود دارد.




که کد مربوط به اپلیکشن اندرویدی در odr_android ضمیمه می شود.