

به نام خدا

پروژه پایانی درس یادگیری عمیق

استاد: آقای دکتر محمدی

اعضای گروه:

غزل زمانی نژاد 97522166

محمدعلی فراهت 97521423

مقدمه:

این پروژه مربوط به تشخیص دنباله اعداد موجود در کارت ملی و کارت بانکی است. این نوع مسئله از دسته مسائل optical character recognition است که در دو فاز کلی پیاده سازی شده است:

- فاز اول جمع آوری دیتاست که در نوت بوک deep_dataset_generation پیاده سازی شده است.
- فاز دوم پیاده سازی مدل و آموزش آن که در نوت بوک deep_final_model پیاده سازی شده است.

فاز اول:

تولید دیتاست

برای تولید دیتا ما با استفاده از کتابخانه PIL متن های تولید شده را روی عکس های آماده می گذاشتیم. برای این کار ابتدا باید validation کارت های بانکی و کد ملی را یاد می گرفتیم، که هر رقم چه جایگاهی دارد و چگونه میتوان یک شماره کارت یا کد ملی معتبر تولید کرد.

الگوریتم رقم کنترلی کارت بانکی:

ارقام موقعیت فرد را در ۲ و موقعیت زوج را در ۱ ضرب میکنیم. اگر حاصل ضرب هر مرحله، عددی بیش از ۹ شد، ۹ واحد از آن کم میکنیم تا تبدیل به عددی تک رقمی شود. سپس اعداد حاصل را با هم جمع میکنیم. اگر حاصل جمع بر ۱۰ بخش پذیر بود یعنی شماره کارت صحیح است و در غیر این صورت شماره کارت صحیح نیست. مثال:

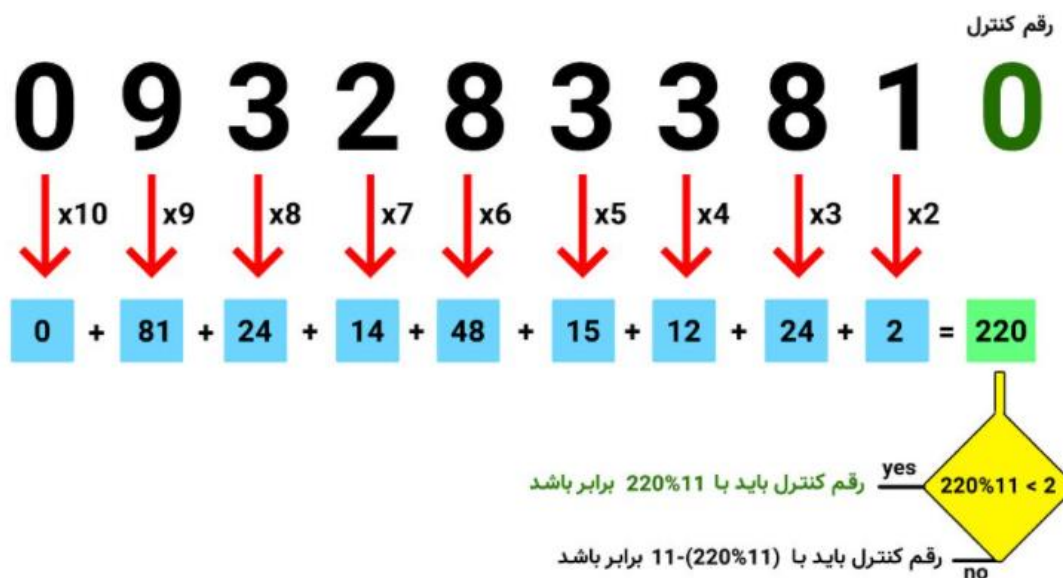
ساختار کد ۱۶ رقمی کارت های بانکی و اعتباری															
ارقام	رقم	شماره منحصر به فرد در مرکز صادر کننده کارت برای دارنده کارت													
م	کنترل	شناسه صادر کننده													
کد	ل	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴
موقعیت	۱۶	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	۱۴
عبارت															
ارقام	۲	۴	۷	۳	۷	۴	۵	۰	۰	۹	۲	۱	۴	۷	۶
م															
کارت															
ت															
الگو															
ی	۱	۲	۱	۲	۱	۲	۱	۲	۱	۲	۱	۲	۱	۲	۱
ضرب															
ب															
محاسبه															
حاصل	۸	۷	۶	۷	۸	۵	۰	۰	۹	۲	۲	۴	۵	۲	۳
ل															
ضرب															
ب															

اگر اعداد ردیف آخر را جمع کنیم، حاصل به 10 بخش پذیر است. پس رقم کنترلی به درستی محاسبه شده است.

الگوریتم رقم کنترلی کارت ملی:

از سمت راست هر رقم (به غیر از رقم کنترل) را در جایگاهش ضرب و حاصل را با هم جمع میکنیم. سپس حاصل جمع را بر عدد ۱۱ تقسیم کرده و باقیمانده را بدست می آوریم.

اگر باقیمانده بزرگتر یا مساوی ۲ بود از عدد ۱۱ کم کرده در غیر این صورت همان عدد را قرار می دهیم. اگر عدد بدست آمده با رقم کنترل (رقم اول) مساوی نباشد شماره ی ملی نادرست است.



تولید کارت بانکی :

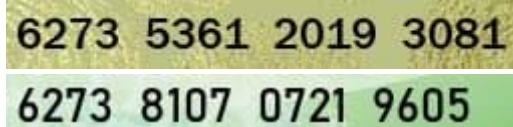
برای تولید شماره کارت ابتدا ۶ رقم اول آن را از بین حالت های مختلف انتخاب کردیم :

```
bank_type = [603799, 589210, 627648, 627961, 603770, 628023, 627760, 502908,
627412, 622106, 502229, 629599, 627488, 621986, 639346, 639607,
504706, 502806, 502938, 603769, 610433, 627353, 585983, 589463,
627381, 639370, 507677, 628157, 505801, 606256, 606373]
```

سپس ۹ رقم بعدی را رندوم انتخاب میکنیم و رقم آخر را هم با فرمول کنترلی شماره کارت حساب میکنیم و به آخر بقیه اضافه میکنیم:

```
def bank_num_generator():
    idx = random.randint(0, len(bank_type) -1)
    # append first 6 digits to card numbers
    num = num_to_list(bank_type[idx])
    # generate 9 next random digits
    for i in range(9):
        num.append(random.randint(0, 9))
    # calculate control digit
    control = bank_card_control_digit(num)
    num.append(control)
    return ''.join(map(str, num))
```

سپس این عدد تولید شده را با فونت های مختلف و تصاویر پس زمینه مختلف میسازیم، چند نمونه را در زیر مشاهده میکنید :



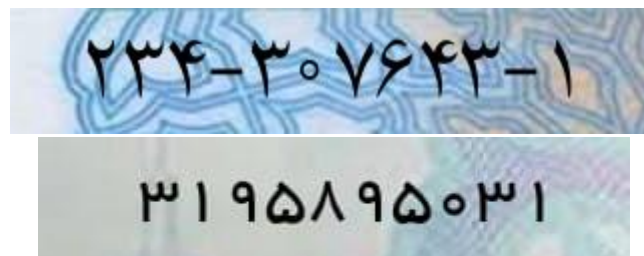
سپس تعداد زیادی فونت و تصویر پس زمینه پیدا کردیم و به صورت رندوم از بین آن ها انتخاب کردیم و دیتاست را در تعداد بالا تولید کردیم.

تولید کد ملی:

تولید کردن کد ملی هم بسیار شبیه کارت بانکی است ولی با این تفاوت که یک تصویر پس زمینه و یک نوع فونت بیشتر نداریم:

```
def id_num_generator():  
    # generate 9 first random digits  
    num = []  
    for i in range(9):  
        num.append(random.randint(0, 9))  
  
    # calculate control digit  
    control = id_card_control_digit(num)  
    num.append(control)  
    return ''.join(map(str, num))
```

نمونه تصاویر تولید شده در زیر مشاهده می شود:



ابتدا ۲۰۰ هزار تصویر تولید کردیم که آن ها را در فرمت h5 ذخیره کردیم ولی متاسفانه این فرمت قابلیت خواندن به صورت batch را نداشت و مجبور بودیم کل آن را روی RAM بیاوریم و با این کار colab کرش کرد و مجبور شدیم از اول کل دیتاست را با فرمت دیگری ذخیره کنیم و لود کنیم.

سپس آن ها را به صورت فرمت png ذخیره کردیم و label ها هم در یک فایل csv به همراه نام تصویر ها در همان مسیر قرار دادیم، این بار هم سائز دیتاست را 200 هزار در نظر گرفتیم و متاسفانه این بار هم در خواندن فایل CSV به مشکل خوردیم و به دلیل تعداد زیاد دیتاست باز هم موفق به لود کردن آن نشدیم.

مجبور شدیم سائز دیتاست را کاهش دهیم و این بار با ۵ هزار دیتا ادامه دادیم.

جمع آوری تصاویر واقعی کارت بانکی:

در این بخش از آشنایان خود خواهش کردیم تا برای ما دیتاست واقعی همراه با برچسب آن بفرستند اما متاسفانه به دلیل کمبود زمان نتوانستیم از آنها برای آموزش مدل استفاده کنیم. این داده ها در کانال زیر موجود هستند:

<https://t.me/deepDataSet>

فاز دوم:

گام اول: خواندن دیتاست ساخته شده و پیش پردازش

در این بخش ابتدا داده ای که از قبل روی درایو ذخیره کرده بودیم را میخوانیم. همچنین نیاز داریم طول برچسب تمامی داده ها یکسان باشد. بدین منظور آن برچسب داده های کارت ملی را pad میکنیم تا هم اندازه برچسب داده کارت بانکی شود.

در بخش بعدی تابع `split_data` را برای بر زدن و تقسیم دیتاست به دو مجموعه آموزش و ارزیابی پیاده سازی میکنیم.

```
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 تابع `encode_single_map`، پیش پردازش هایی روی تصاویر اعمال می کنیم. بعد از خواندن تصویر آن را به یک آرایه در بازه 0 تا 1 (به جهت اینکه داده نرمالایز باشد) تبدیل میکنیم، آنها را به اندازه یکسان `resize` می کنیم. و در نهایت برچسب داده ها را به آیدی مشخص آنها مپ می کنیم.

```
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=1)  
    # 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"))  
  
    return {"image": img, "label": label}
```

در پایان دیتاست آزمون و ارزیابی را با استفاده از توابع آماده موجود در `keras` می سازیم.

```

1 train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
2 train_dataset = (
3     train_dataset.map(
4         encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE
5     )
6     .batch(batch_size)
7     .prefetch(buffer_size=tf.data.AUTOTUNE)
8 )
9
10 validation_dataset = tf.data.Dataset.from_tensor_slices((x_valid, y_valid))
11 validation_dataset = (
12     validation_dataset.map(
13         encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE
14     )
15     .batch(batch_size)
16     .prefetch(buffer_size=tf.data.AUTOTUNE)
17 )

```

گام دوم: طراحی و پیاده سازی مدل

در این بخش باید شبکه ای طراحی کنیم که وظیفه پیش بینی دنباله ارقام را داشته باشد، این کار را با یک شبکه CRNN مطابق تصویر زیر (برگرفته از یک مقاله) انجام میدهیم:

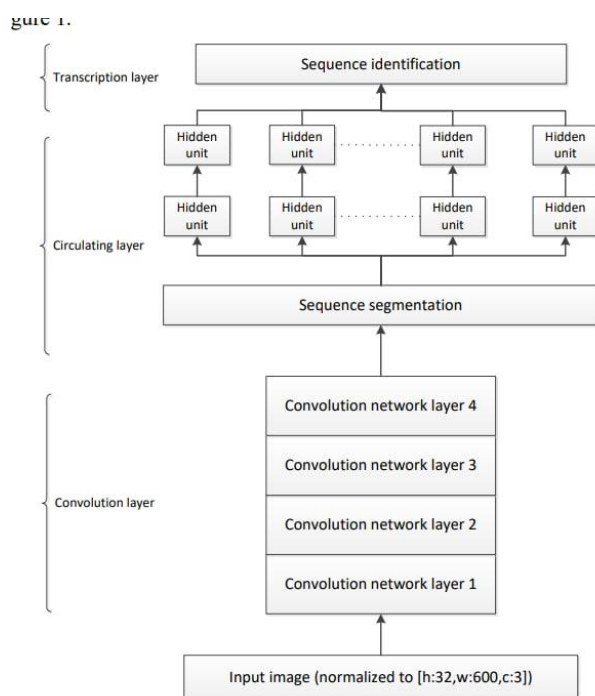


Figure 1. CRNN network structure.

در ابتدا از چندین لایه کانولوشنی برای استخراج ویژگی ها استفاده میکنیم و سپس از لایه های بازگشتی برای نشر دادن اطلاعات در دنباله عددی استفاده میکنیم.

توضیح مدل: مدل دارای دو ورودی، یکی از نوع دیتاست و دیگری فقط برچسب داده است. از دو بلوک کانولوشنی و دو لایه maxpooling استفاده می کنیم. همچنین برای جلوگیری از اورفیت شدن مدل، از لایه های dropout نیز استفاده می کنیم. سپس خروجی این لایه را تغییر شکل می دهیم و از یک لایه dense عبور می دهیم تا خروجی آن را به لایه بازگشتی دهیم. سپس از دو لایه lstm دو جهته استفاده می کنیم. در آخر آن را از لایه dense دارای 11 نرون خروجی عبور می دهیم تا مدل پیش بینی کند در هر گام زمانی کاراکتری که دیده از کدام یک از کاراکترهای موجود در الفبا بوده است. در آخر از لایه ctc استفاده کردیم که در ادامه به توضیح آن خواهیم پرداخت.


```

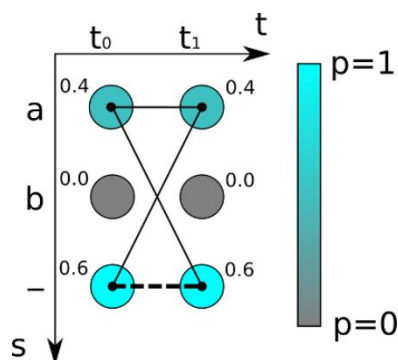
1 def build_model():
2     # Inputs to the model
3     input_img = layers.Input(
4         shape=(img_width, img_height, 1), name="image", dtype="float32"
5     )
6     labels = layers.Input(name="label", shape=(None,), dtype="float32")
7
8     # First conv block
9     x = layers.Conv2D(
10         64,
11         (3, 3),
12         activation="relu",
13         kernel_initializer="he_normal",
14         padding="same",
15         name="Conv1",
16     )(input_img)
17     x = layers.MaxPooling2D((2, 2), name="pool1")(x)
18     x = layers.Dropout(0.2)(x)
19
20     # Second conv block
21     x = layers.Conv2D(
22         128,
23         (3, 3),
24         activation="relu",
25         kernel_initializer="he_normal",
26         padding="same",
27         name="Conv2",
28     )(x)
29     x = layers.MaxPooling2D((2, 2), name="pool2")(x)
30     x = layers.Dropout(0.2)(x)
31
32     # We have used two max pool with pool size and strides 2.
33     # Hence, downsampled feature maps are 4x smaller.
34     # The number of filters in the last layer is 128.
35     new_shape = ((img_width // 4), (img_height // 4) * 128)
36     x = layers.Reshape(target_shape=new_shape, name="reshape")(x)
37     x = layers.Dense(64, activation="relu", name="dense1")(x)
38     x = layers.Dropout(0.2)(x)
39
40     # RNNs
41     x = layers.Bidirectional(layers.LSTM(128, return_sequences=True, dropout=0.25))(x)
42     x = layers.Bidirectional(layers.LSTM(64, return_sequences=True, dropout=0.25))(x)
43
44     # Output layer
45     x = layers.Dense(
46         len(char_to_num.get_vocabulary()) + 1, activation="softmax", name="dense2"
47     )(x)
48
49     # Add CTC layer for calculating CTC loss at each step
50     output = CTCLayer(name="ctc_loss")(labels, x)
51
52     # Define the model
53     model = keras.models.Model(
54         inputs=[input_img, labels], outputs=output, name="ocr_model"
55     )

```

تابع ضرر: از ctc loss استفاده می کنیم. Connectionist Temporal Classification نوعی از تابع ضرر است که در مسائل با ورودی از نوع دنباله (مثلا تشخیص صوت، تشخیص دنباله ای از ارقام) به کار می رود.

برای محاسبه آن باید از خروجی آخرین لایه شبکه عصبی به همراه برچسب داده ها استفاده کنیم. به کمک یک مثال نحوه محاسبه آن را توضیح می دهیم.

میخواهیم مقدار تابع ضرر را در دو گام زمانی محاسبه کنیم. در این مثال الفبای ورودی تنها شامل 3 کاراکتر (a,b,-) است. احتمال هر کاراکتر در هر گام زمانی در تصویر زیر درج شده است.



برای محاسبه امتیاز هر alignment، احتمال کاراکترهای متناظر را در هم ضرب می کنیم. مثلاً در مسیر aa، این امتیاز برابر است با: $0.4 * 0.4$

برای محاسبه امتیاز هر ground truth، مجموع امتیازات alignment های آن را بدست می آوریم. مثلاً برای محاسبه امتیاز برچسب a به صورت زیر عمل می کنیم:

$$\text{score}(aa) + \text{score}(a-) + \text{score}(-a) = 0.4 \cdot 0.4 + 0.4 \cdot 0.6 + 0.6 \cdot 0.4 = 0.64$$

در اینجا هدف آن است که شبکه عصبی به گونه ای آموزش ببیند که احتمال بالایی برای کاراکترهای درست پیش بینی کند، پس سعی می کنیم حاصل ضرب احتمالات برای برچسب صحیح را maximize کنیم. به بیان دیگر، کمینه کردن میزان ضرر که مقدار ضرر برابر است با:

negative sum of log-probabilities

پیاده سازی: برای این کار مطابق فرمول گفته شده عمل می کنیم.

```

class CTCLayer(layers.Layer):
    def __init__(self, name=None):
        super().__init__(name=name)
        self.loss_fn = keras.backend.ctc_batch_cost

    def call(self, y_true, y_pred):
        batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
        input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
        label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

        input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
        label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

        loss = self.loss_fn(y_true, y_pred, input_length, label_length)
        self.add_loss(loss)

        # At test time, just return the computed predictions
        return y_pred

```

آموزش مدل: پس از پیاده سازی مدل، آن را در 50 ایپوک با بهینه ساز adam آموزش می دهیم و مدل را ذخیره میکنیم. در اینجا از early stopping با مانیتور validation loss نیز استفاده می کنیم تا اگر طی چند ایپوک مقدار ضرر داده اعتبارسنجی کاهش چشمگیری نداشت، آموزش را متوقف کنیم.

```

1 early_stopping_patience = 10
2 # Add early stopping
3 early_stopping = keras.callbacks.EarlyStopping(
4     monitor="val_loss", patience=early_stopping_patience, restore_best_weights=True
5 )
6
7 def train_and_save(epochs):
8     # Train the model
9     history = model.fit(
10         train_dataset,
11         validation_data=validation_dataset,
12         epochs=epochs,
13         callbacks=[early_stopping],
14     )
15     # save the model
16     model.save("/content/gdrive/My Drive/saved_model_final")
17
18
19 for i in range(10):
20     train_and_save(5)

```

مقدار ضرر در حین آموزش کاهش می یابد.

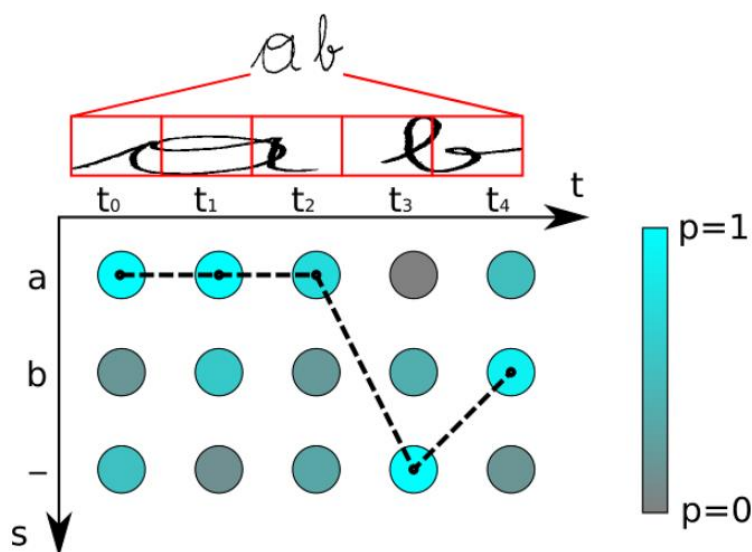
گام سوم: پیش بینی مدل

توضیح ctc decoding: می‌خواهیم دنباله ای که مدل پیش بینی می کند، نزدیکترین دنباله به نوشته موجود در تصویر باشد. برای رسیدن به این هدف نمیتوانیم تمامی احتمال ترکیب های ممکن را محاسبه کنیم. پس یک الگوریتم **best path decoding** استفاده می کنیم که شامل دو مرحله اصلی زیر است:

بهترین مسیر را با گرفتن محتمل ترین کاراکتر در هر مرحله زمانی محاسبه می کند.

با حذف کاراکترهای تکراری، **decoding** را انجام می دهد. آنچه باقی می ماند نشان دهنده متن شناخته شده است.

نمونه این کار در تصویر زیر مشاهده می شود:



در پیاده سازی مدل، برای استفاده از تابع ضرر ctc، از نمونه آماده آن موجود در keras استفاده کردیم.

```
1 # function to decode the output of the network
2 def decode_batch_predictions(pred):
3     input_len = np.ones(pred.shape[0]) * pred.shape[1]
4
5     results = keras.backend.ctc_decode(pred, input_length=input_len, greedy=True)[0][0][:, :max_length]
6     # Iterate over the results and get back the text
7     output_text = []
8     for res in results:
9         res = tf.strings.reduce_join(num_to_char(res)).numpy().decode("utf-8")
10         output_text.append(res)
11     return output_text
12
```

در پایان می توانیم از تابع prediction استفاده کنیم تا پیش بینی شبکه را از تصویر دلخواه ورودی ببینیم. برای این کار ابتدا دیتاست را آماده میکنیم، سپس مدل (که آن را ذخیره کرده بودیم) لود می کنیم. از تابع decode که قبلا تعریف کردیم استفاده میکنیم تا مدل نتیجه را پیش بینی کند.

```
def prediction(images):
    result = []
    test_dataset = tf.data.Dataset.from_tensor_slices((images, ['0' * 16] * len(images)))
    test_dataset = (
        test_dataset.map(
            encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE
        )
        .batch(batch_size)
        .prefetch(buffer_size=tf.data.AUTOTUNE)
    )

    # CHANGE
    # please put appropriate address to load the model
    prediction_model = keras.models.load_model(f"/content/gdrive/My Drive/saved_model_final")
    prediction_model = keras.models.Model(
        prediction_model.get_layer(name="image").input, prediction_model.get_layer(name="dense2").output
    )

    for batch in test_dataset.take(1):
        batch_images = batch["image"]
        batch_labels = batch["label"]

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

        result = result + pred_texts

    return result
```

از توجه شما سپاسگزاریم

<https://vrgl.ir/m5FOE>

<https://omigo.ir/r/3bce>

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

<https://voidful.medium.com/understanding-ctc-loss-for-speech-recognition-a16a3ef4da92>

<https://vijayabhaskar96.medium.com/tutorial-on-keras-flow-from-dataframe-1fd4493d237c>

<https://iopscience.iop.org/article/10.1088/1742-6596/1345/2/022049/pdf>

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