# عنوان گزارش: خواندن کاراکتر‌های تصاویر CAPTCHA

نگارش

نسرین علیپور

رشته تحصیلی:

دانشجوی ارشد مهندسی اطلاعات پزشکی(دانشگاه صنعتی امیرکبیر)

شهریور ۱۴۰۱

به نام هستی­بخش

# چكيده

در این گزارش پیاده‌سازی برنامه‌ی تشخیص تصاویر CAPTCHA شرح داده شده است. در فصل اول صورت و خواسته‌ی مسئله شرح داده شده است و در فصل دوم نحوه پیاده‌سازی و عملیات‌های انجام شده توضیح داده شده است و در انتها فصل سوم به ارائه‌ی نتایج اختصاص داده شده است.

واژه‌های کلیدی:

CAPTCHA، شبکه‌های عصبی کانولوشن، شبکه‌های عصبی بازگشتی

|  |  |
| --- | --- |
| فهرست مطالب | صفحه |

[چكيده ‌أ](#_Toc113624887)

[فصل اول مقدمه 1](#_Toc113624888)

[مقدمه 2](#_Toc113624889)

[فصل دوم روش انجام پژوهش 3](#_Toc113624890)

[روش انجام پژوهش 4](#_Toc113624891)

[1-2- مراحل اجرا 4](#_Toc113624892)

[2-2- پیاده‌سازی 4](#_Toc113624893)

[فصل سوم نتایج 9](#_Toc113624894)

[نتایج 10](#_Toc113624895)

# فصل اول مقدمه

# مقدمه

**صورت مسئله**

به عنوان بخشی از این پروژه، شما یک مدل OCR ساده با استفاده از TF2 یا Keras Functional API ایجاد خواهید کرد. در این پروژه می توانید از CNN، RNN و هر لایه دیگری که ممکن است به عنوان بخشی از پروژه خود ایجاد کنید استفاده کنید.

**دیتاست**



**ساختار گزارش**

در مقدمه­ی این گزارش صورت مسئله‌ی پروژه و دادگان شرح داده شد در فصل بعدی روش انجام پروژه و مراحل آن توضیح داده می‌شود (که شامل آماده‌سازی دادگان و پیش‌پردازش داده‌ها و استفاده از لایه‌های کانولوشنی و بازگشتی برای ساخت مدل و حل این مساله‌ است) و در آخر نتایج پیشبینی تصاویر CAPTCHA در دادگان validation بررسی می‌شود.

# فصل دوم روش انجام پژوهش

# روش انجام پژوهش

در این فصل ابتدا مراحل طی شده به طور کلی ذکر می‌شود و سپس با جزئیات و پیاده‌سازی در ادامه آورده می‌شود.

## مراحل اجرا

* خواندن پکیج‌ها(مانند tensorflow ، numpy و matplotlib و....)
* خواندن Data\_Utils.py (در این فایل توابعی نوشته شده‌اند که ابزارهایی برای کار با دیتاهای این مسئله هستند و به طور کافی در فایل مربوطه توضیح داده شده‌اند)
* خواندن دیتای مسئله
* تقسیم دیتا به دیتای تعلیم و اعتبارسنجی
* ساخت دیتاست تعلیم و اعتبارسنجی(همراه با اعمال پیش پردازش‌ها و بچ بندی)
* ساخت لایه‌های CTC و OCR
* ساخت مدل تعلیم با استفاده از لایه‌های CTC و OCR و کامپایل آن و فیت کردن آن به دیتاست تعلیم
* ساخت مدل inference برای پیش‌بینی با استفاده از لایه‌ی OCR
* پیشبینی کاراکتر‌های تصاویر CAPTCHA در دیتاست اعتبارسنجی

## پیاده‌سازی

درData\_Utils.py توابع زیر نوشته شده‌اند که برای کار با دیتاهای این مسئله هستند و توضیحات مربوط به هر کدام به طور کامل در فایل مربوطه آمده است.

* Split data
* Preprocess
* Create dataset
* Decode batch prediction

اینگونه این ماژول را می‌خوانیم و از توابع نوشته شده استفاده می‌کنیم:

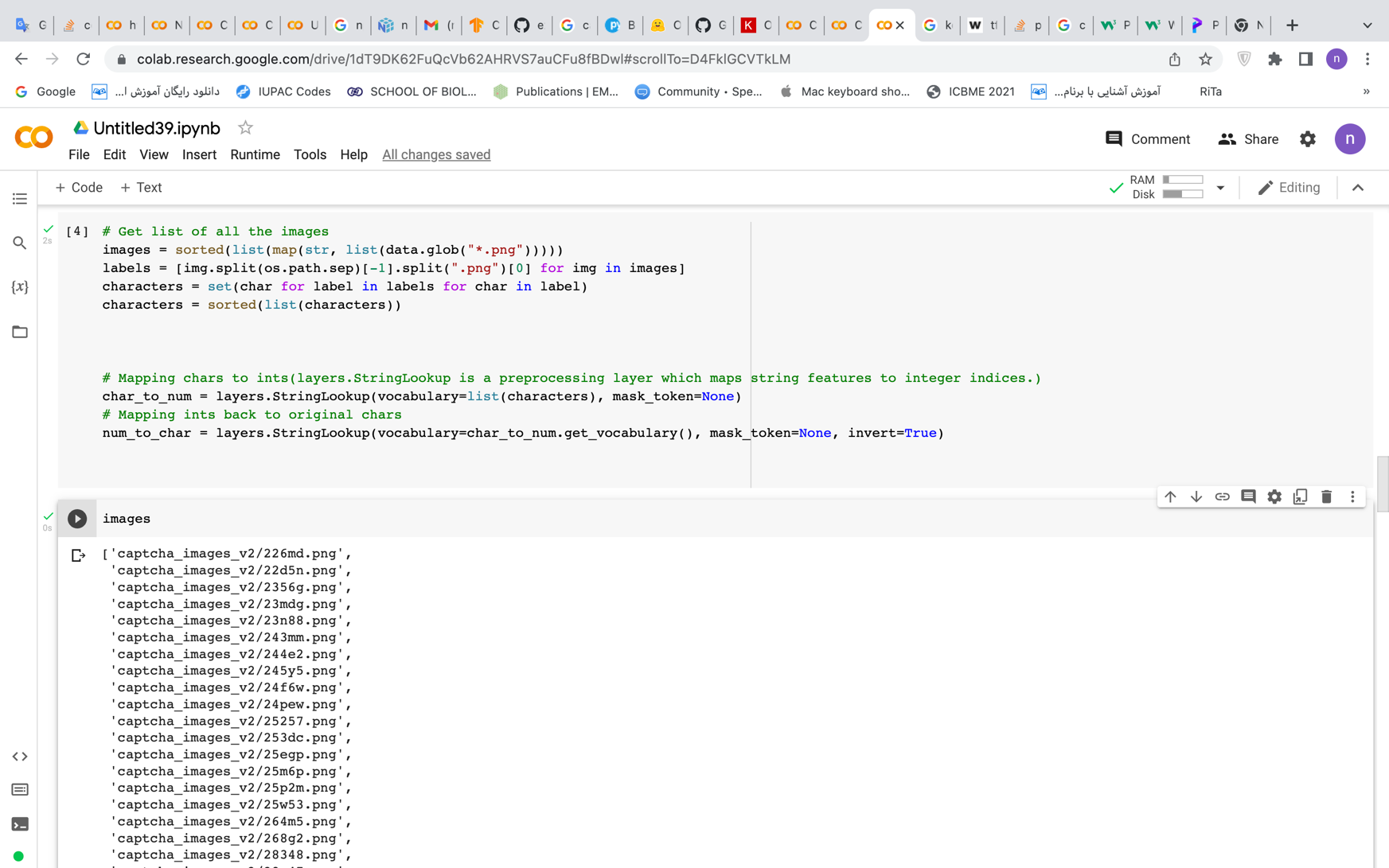
from data\_utilsss import \*

خواندن داده:

دیتا را میخوانیم:

data = Path("/content/drive/MyDrive/samples/")

images = sorted(list(map(str, list(data.glob("\*.png")))))

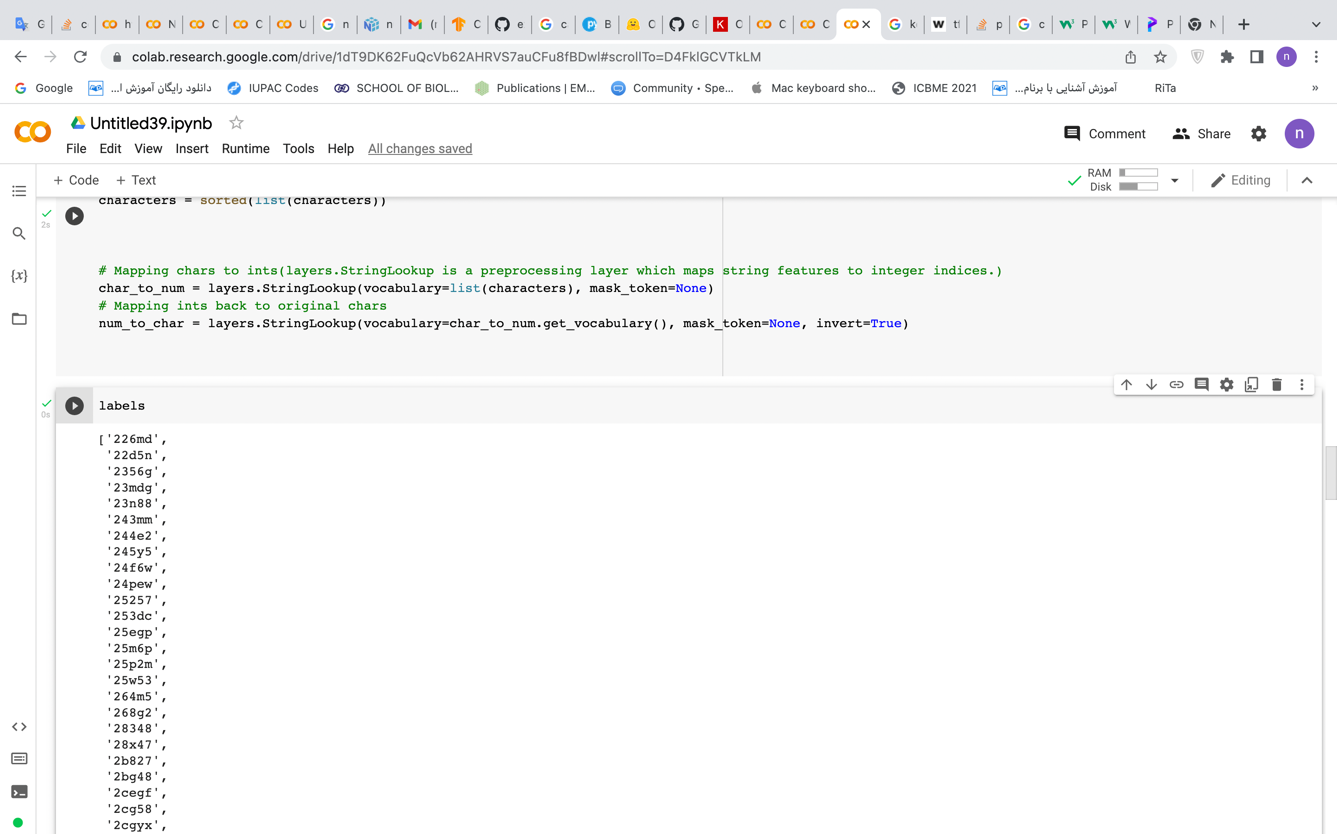


حاوی اطلاعات تصاویر

برچسب‌های تصاویر همان اسم تصویر(به جز .png) است پس از روی لیست images برچسب‌ها را استخراج می‌کنیم و لیستی از کاراکتر‌های استفاده شده در این برچسب‌ها پیدا می‌کنیم(که نتیجه نشان می‌دهد شامل ۱۹ کاراکتر هستند)

labels = [img.split(os.path.sep)[-1].split(".png")[0] for img in images]

characters = sorted(list(set(char for label in labels for char in label)))



حاوی کاراکترهای تصاویر

برای تبدیل این کاراکتر‌ها به integer یک لایه از keras (با استفاده از layers.StringLookup) تعریف می‌شود.(برای عکس این عملیات هم به کار می‌رود.)

# Mapping chars to ints(layers.StringLookup is a preprocessing layer which maps string features to integer indices.)

char\_to\_num = layers.StringLookup(vocabulary=list(characters), mask\_token=None)

# Mapping ints back to original chars

num\_to\_char = layers.StringLookup(vocabulary=char\_to\_num.get\_vocabulary(), mask\_token=None, invert=True)

همچنین می‌توان با اندازه‌گیری len (label) (با ایتریت کردن روی تمام لیبل‌ها) طول لیبل‌ها را به‌دست آورد. که نتیجه نشان می‌دهد تعداد کاراکترهای درون همه‌ی لیبل‌ها مساوی و برابر ۵ است.

**ساخت دیتاست:**

با استفاده از تابع نوشته‌شده در utils (به نام split\_data) دیتا(شامل تصاویر و برچسب‌هایشان) را به بخش تعلیم و اعتبارسنجی تقسیم می‌کنیم.

اکنون می‌توان با استفاده از تابع create\_dataset (نوشته‌شده در utils) که دیتاست از نوع slice\_dataset می‌سازد و با .map تابع پیش پردازش را روی دیتاست صدا بزنیم و...(توضیحات در فایل utils)

# Create train and validation dataset

train\_dataset = create\_dataset(x\_train, y\_train, batch\_size)

validation\_dataset = create\_dataset(x\_valid, y\_valid, batch\_size)

حال نوبت به ساخت مدل برای training می‌رسد. برای تعلیم از لایه‌ی CTC معرفی شده در رفرنس استفاده می‌شود اما در inference که بعد از تعلیم برای پیش‌بینی استفاده می‌شود از این لایه استفاده نمی‌شود لذا این لایه‌ را در کلاس(خواسته شده بود با استفاده از کلاس شبکه را تعریف کنیم) نوشته شده وارد نمی‌کنیم که کلاس برای هر دو شبکه قابل استفاده باشد.(پس مستقیما مدل را در کلاس نمی‌سازیم و صرفا باقی لایه‌ها را در یک کلاس می‌نویسیم البته می‌شود به شکل یک تابع نیز نوشته شود اما هدف استفاده از شی گراییست) لذا کلاس را تعریف می‌کنیم:

class ocr\_model(layers.Layer):

def \_\_init\_\_(self, name=None):

super().\_\_init\_\_(name=name)

سپس لایه‌ها را به شکل بلوک‌هایی به عنوان بلوک‌های لایه‌های کانولوشنی و بازگشتی و full-connected ذخیره می‌کنیم. به طور مثال:

self.cnn\_block1 = Sequential([

layers.Conv2D(32, (3, 3), activation="relu", kernel\_initializer="he\_normal", padding="same", name="Conv1"),

layers.MaxPooling2D((2, 2), name="pool1")

])

و با استفاده از call می‌گوییم وقتی یک ورودی به این لایه (کلاس ocr نوشته شده) داده شد چه عملیاتی انجام شود و خروجی را برگرداند:

def call(self, x):

x = self.cnn\_block1(x)

x = self.cnn\_block2(x)

x = self.dense\_block1(x)

x = self.rnn\_block(x)

x = self.dense\_block2(x)

return x

این به این معناست که ورودی را می‌گیرد و لایه‌های مربوط به این بلوک‌ها را روی ورودی به صورت متوالی انجام می‌دهد و در آخر خروجی dense را برمی‌گرداند.

حال نوبت آن رسیده با استفاده از لایه‌ی ocr نوشته شده و ctc، مدل‌های تعلیم و پیش‌بینی را بسازیم.

در مدل تعلیم image ها به عنوان ورودی به لایه‌ی ocr داده می‌شوند و خروجی این لایه و label ها (دو ورودی) به لایه‌ي CTC داده می‌شوند و خروجی این لایه‌، خروجی مدل است. پس لایه‌های ورودی این مدل را تعریف می‌کنیم:

input\_image = layers.Input(shape=(200, 50, 1), name="image", dtype="float32")

labels = layers.Input(name="label", shape=(None,), dtype="float32")

ورودی‌ هارا به این لایه‌ها می‌دهیم:

ocr\_layer = ocr\_model()(input\_image)

output = CTCLayer(name="ctc\_loss")(labels, ocr\_layer)

مدل تعلیم را می‌سازیم و کامپایل می‌کنیم:

model = tfk.models.Model(inputs=[input\_image, labels], outputs=output, name="ocr\_model\_v1")

model.compile(optimizer=tfk.optimizers.Adam())

و به دادگان تعلیم فیت می‌کنیم:

history = model.fit(train\_dataset, validation\_data=validation\_dataset, epochs=epochs, callbacks=[early\_stopping])

در مدل پیش‌بینی فقط لایه‌ی ocr(این لایه‌ها را از مدل تعلیم داده شده بر میداریم تا پیشبینی را انجام دهد) استفاده می‌شود:

prediction\_model = tfk.models.Model(model.get\_layer(name="image").input, model.get\_layer(name="ocr").output)

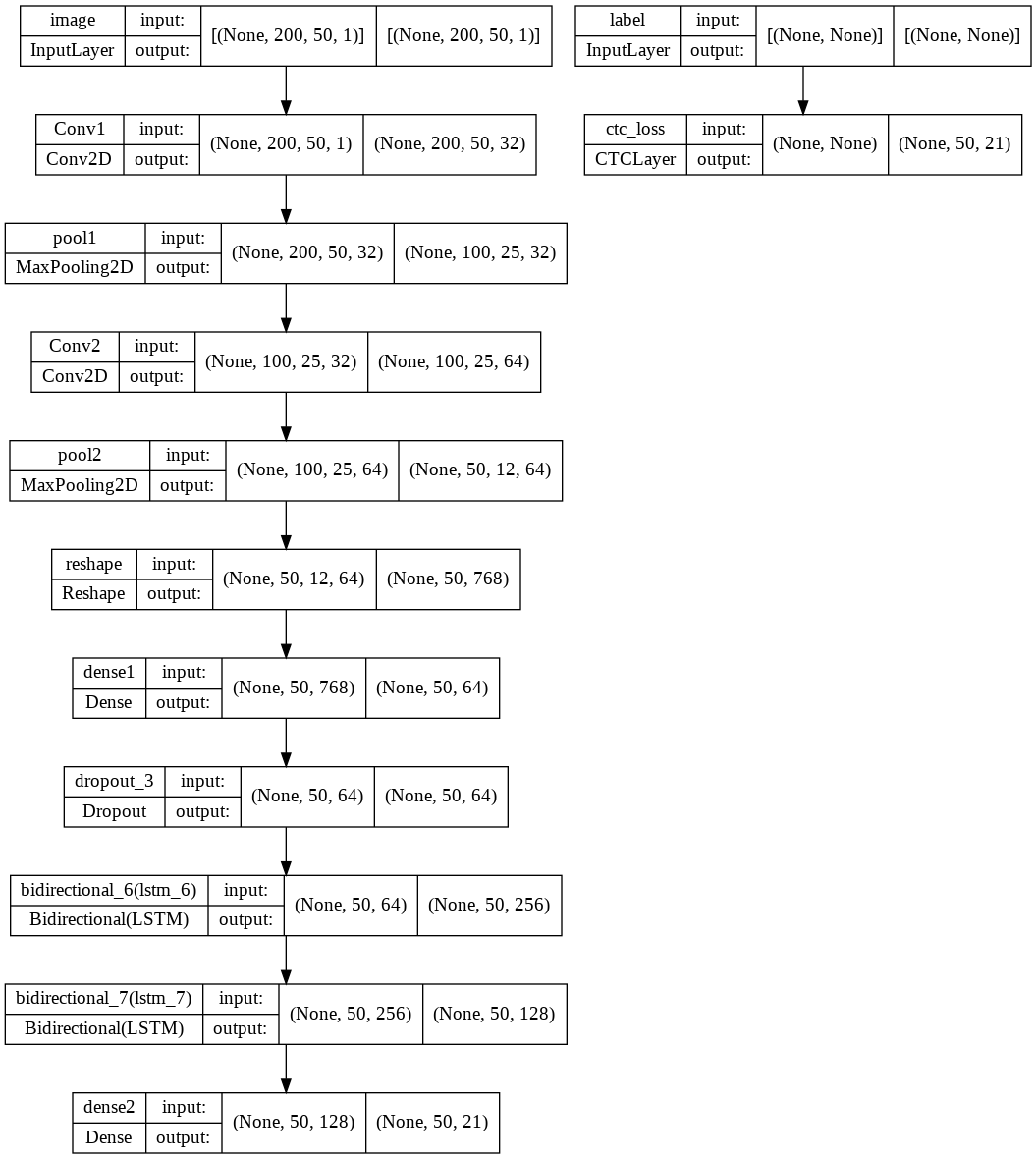
\*\*این نکته بسیار مهم است که ما این لایه‌ها از مدل تعلیم داده شده را برمی‌داریم(وزن‌های آن اصلاح شده است) تا پیشبینی را بر این أساس انجام دهیم و اگر مجدد دستور کامپایل این مدل را بنویسیم کار بیهوده‌ای کردیم چون مجدد وزن‌‌های یادگیری شده از دست می‌روند.

حال از مدل inference برای پیش‌بینی برچسب‌های تصاویر دادگان تست استفاده می‌کنیم.(نتایج در فصل بعد آمده است)

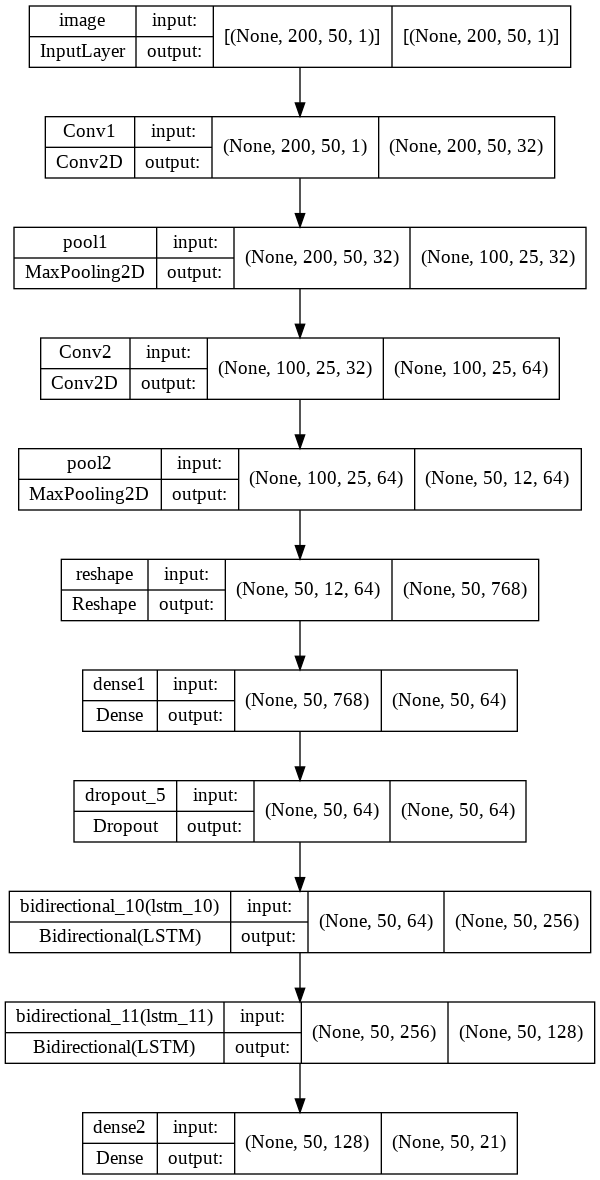
# فصل سوم نتایج

# نتایج

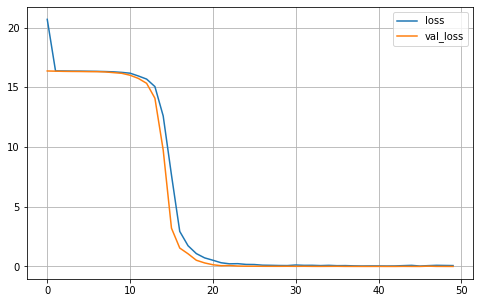
**مدل train** نسبت به مدل inference لایه‌ی CTC را به علاوه‌ی لایه‌های آن دارد که این لایه یک input دیگر(برچسب) می‌گیرد. این موضوع در شماتیک زیر دیده می‌شود:



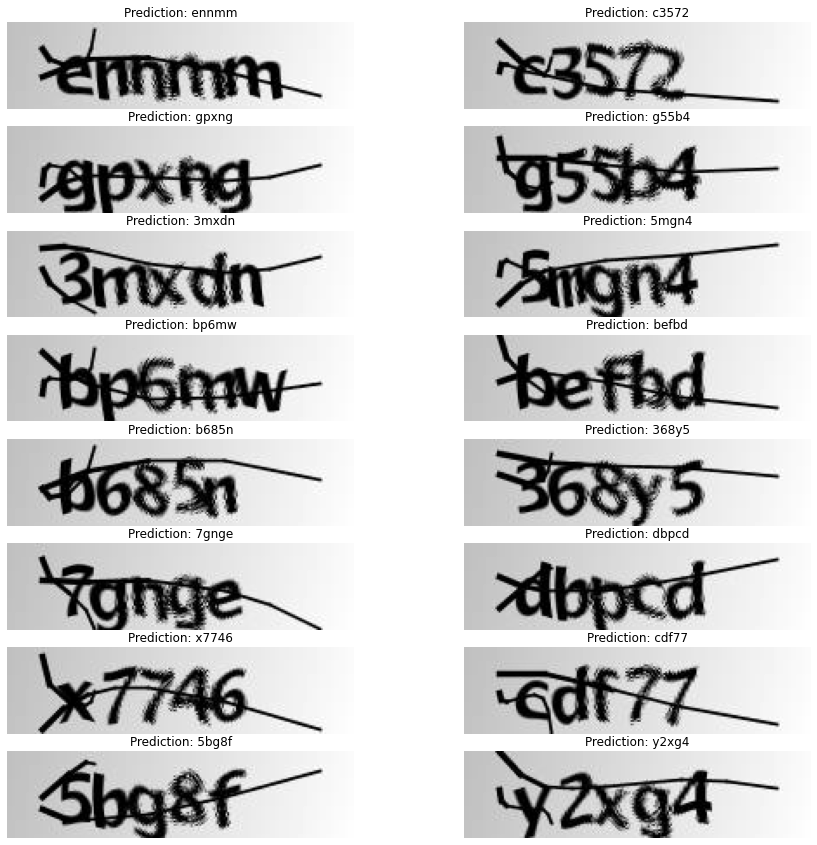
با توضیحات ارائه شده، **مدل infernce** از آنجا که لایه‌ی CTC را ندارد شکل Sequential خود را حفظ می‌کند:



تغییرات loss برای دادگان تعلیم و اعتبار سنجی طی ۵۰ اپوک به این صورت است:



نتیجه‌ی پیشبینی در یک subplot(8,2) نمایش داده می‌شود:



رفرنس: https://keras.io