دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه‌ی هوش سایبری

Captcha_Recognition_Project_Final

پدیدآورندگان:

محمد امین کیانی ٤٠٠٣٦١٣٠٥٢

سید محمد سید حسینی ٤٠٠٣٦٥٣٠٠٤

ارمان خلیلی ٤٠٠٣٦٢٣٠١٦

دانشجوی کارشناسی، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،
Aminkianiworkeng@gmail.com

استاد راهنما: جناب اقای دکتر مهدوی

نیمسال دوم تحصیلی ۰۳-۱٤۰۲

فهرست مطالب

مستندات

۱-مسئله و تحلیل کلی آن:

پروژه تشخیص کپچا (Captcha Recognition Project) یک پروژه گرافیک محاسباتی است که به منظور تشخیص و حل کدهای Captcha کاربرد دارد. Captcha یک ابزار امنیتی است که بعضی وبسایت‌ها از آن برای جلوگیری از حملات رباتیک و اسپم استفاده می‌کنند.

در این پروژه، ابتدا باید یک مجموعه داده از تصاویر Captcha جمع‌آوری کرد. سپس، الگوریتم‌های یادگیری ماشین یا شبکه‌های عصبی برای آموزش به تشخیص کدهای Captcha بر روی این داده‌ها استفاده می‌شود.

مراحل کلی پروژه به شرح زیر است:

۱. جمع‌آوری داده‌های Captcha

۲. پیش‌پردازش تصاویر (تبدیل به سیاه و سفید، اندازه‌بندی، حذف نویز و ...)

۳. آموزش الگوریتم‌های یادگیری ماشین یا شبکه‌های عصبی

۴. ارزیابی عملکرد سیستم بر روی داده‌های تست

۵. بهینه‌سازی و افزایش دقت تشخیص

در نهایت، پس از آموزش موفق سیستم، می‌توان از آن برای تشخیص کدهای Captcha در وبسایت‌ها و جلوگیری از حملات اسپم و رباتیک استفاده کرد.

۲-تاثیر نرمالایز کردن:

```
#preprocesss image
def preprocess():
  X = np.zeros((n,50,200,1)) #1070*50*200 array with all entries 0
  y = np.zeros((5,n,nchar)) #5*1070*36(5 letters in captcha) with all entries 0

  for i, pic in enumerate(os.listdir("/content/drive/My
Drive/captcha_dataset/samples")):
  #i represents index no. of image in directory
  #pic contains the file name of the particular image to be preprocessed at a
time

    img = cv2.imread(os.path.join("/content/drive/My
Drive/captcha_dataset/samples", pic), cv2.IMREAD_GRAYSCALE) #Read image in
grayscale format
    pic_target = pic[:-4]#this drops the .png extension from file name and
contains only the captcha for training

    if len(pic_target) < 6: #captcha is not more than 5 letters
      img = img / 255.0 #scales the
image between 0 and 1
      img = np.reshape(img, (50, 200, 1)) #reshapes image to width 200 , height
50 ,channel 1

      target=np.zeros((5,nchar)) #creates an array of size 5*36 with all entries
0

      for j, k in enumerate(pic_target):
      #j iterates from 0 to 4(5 letters in captcha)
      #k denotes the letter in captcha which is to be scanned
         index = character.find(k) #index stores the position of letter k of
captcha in the character string
         target[j, index] = 1 #replaces 0 with 1 in the target array at the
position of the letter in captcha

      X[i] = img #stores all the images
      y[:,i] = target #stores all the info about the letters in captcha of all
images

  return X,y
```

با **نرمالایز کردن** داده‌ها در کد و تقسیم آنها بر ۲۵۵، داده‌ها را از مقادیر پیکسل اولیه (بین ۰ تا ۲۵۵) به مقادیر واحد (بین ۰ و ۱) تبدیل می‌کنیم. این مرحله موارد زیر را کمک می‌کند:

۱ .استقرار سریع‌تر – اگر برای همگرایی بهتر الگوریتم‌های بهینه‌سازی مانند Stochastic Gradient Descent (SGD)وجود داشته باشد .

۲ .جلوگیری از اشباع شدن – اگر از توابع فعال‌سازی مانند Sigmoid یا Tanh استفاده کنیم، این تحول می‌تواند از شبکه‌ی عصبی از اینکه در لایه‌های عمیق به اشباع شود جلوگیری کند.

۳ .ایجاد شرایطی بهتر برای یادگیری –این تغییرات معمولاً موجب بهبود همگرایی شبکه و بهبود عملکرد آن می‌شود.

پس با نرمالایز کردن داده‌ها، امکان افزایش سرعت و دقت آموزش شبکه عصبی MLP بر روی دیتاست وجود دارد. این مرحله تضمین می کند که مدل در طول آموزش سریعتر همگرا می شود و از مسائل مربوط به داده های ورودی در مقیاس های مختلف جلوگیری می کند.

۳-انتخاب مدل و لایه بندی همراه با Dropout :

```
#create model
def createmodel():
    img = layers.Input(shape=imgshape) # Get image as an input of size 50,200,1
    conv1 = layers.Conv2D(16, (3, 3), padding='same', activation='relu')(img)
#50*200
    mp1 = layers.MaxPooling2D(padding='same')(conv1)  # 25*100
    conv2 = layers.Conv2D(32, (3, 3), padding='same', activation='relu')(mp1)
    mp2 = layers.MaxPooling2D(padding='same')(conv2)  # 13*50
    conv3 = layers.Conv2D(32, (3, 3), padding='same', activation='relu')(mp2)
    bn = layers.BatchNormalization()(conv3) #to improve the stability of model
    mp3 = layers.MaxPooling2D(padding='same')(bn)  # 7*25

    flat = layers.Flatten()(mp3) #convert the layer into 1-D

    outs = []
    for _ in range(5): #for 5 letters of captcha
        dens1 = layers.Dense(64, activation='relu')(flat)
        drop = layers.Dropout(0.5)(dens1) #drops 0.5 fraction of nodes
        res = layers.Dense(nchar, activation='sigmoid')(drop)

        outs.append(res) #result of layers

    # Compile model and return it
    model = Model(img, outs) #create model
    model.compile(loss='categorical_crossentropy',
optimizer='adam',metrics=["accuracy"])
    return model
    ])
```

در اینجا، از TensorFlow برای تعریف این معماری استفاده شده است.

سپس نیاز به افزودن لایه‌های لایه‌ها و تنظیمات مدل، مانند تعداد نورون‌ها و توابع فعال‌سازی مربوطه، خواهیم داشت. این مدل می‌تواند در پیش‌بینی یا دسته‌بندی ارقام دست نوشته شده در تصاویر به کار رود.

به طور کلی، اضافه کردن لایه‌های عمیق‌تر می‌تواند به مدل قدرت بیشتری برای یادگیری اطلاعات پیچیده بدهد اما نیاز به مراقبت بیشتر در مورد تطبیق آن با داده‌ها و جلوگیری از overfitting وجود دارد. پس با تست گذاشتن چندین حالت مختلف از تعداد لایه

های متفاوت و تحقیق در کتاب هندز ان ماشین لرنینگ میتوان نتیجه گرفت که این میزان لایه برای مدل انتخاب شده توسط بنده کافی است و از اورفیت شدن هم جلو گیری می کند و لایه های بیشتر از این نیاز نبوده و تنها سرعت اجرا را پایین می برند و لایه ی کمتر نیز دقت و قدرت را پایین می برد.

Batch یک لایه‌ی TensorFlow در tf.keras.layers.BatchNormalization() Normalization اضافه می‌کند تا آموزش شبکه عصبی سریعتر شود، مشکل محوشوندگی را کاهش می‌دهد، باعث افزایش دقت مدل خود می‌شود و از مشکل برازش بیش‌افرازی جلوگیری می‌کند. این لایه به آموزش شبکه عصبی کمک می‌کند تا به سرعت به یک حالت تعادل مطلوب برسد.

یک لایه Dropout با نرخ ۰/۵ را به معماری شبکه عصبی اضافه می‌کند. این لایه Dropout به طور تصادفی بخشی از ورودی‌ها را با احتمالی حذف می‌کند. این کار باعث کاهش اورفیت مدل می‌شود و از بروز پدیده‌هایی مانند بیش‌برازش جلوگیری می‌کند.

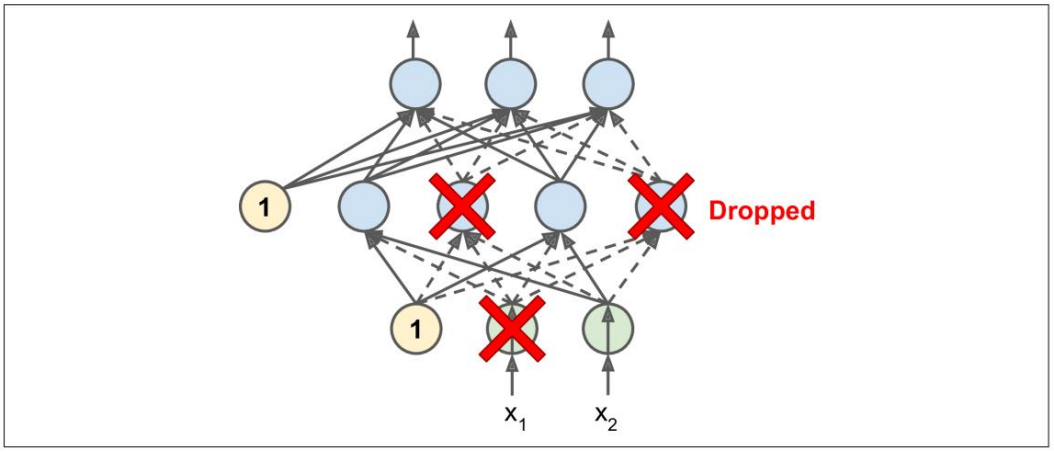


Figure 11-9. Dropout regularization

٤-تاثیر توابع فعالسازی:

ReLU (Rectified Linear Activation): این تابع غیرخطی است و در لایه‌های مخفی شبکه عصبی به طور گسترده استفاده می‌شود. این تابع اعداد منفی را به صفر تبدیل می‌کند.

استفاده از تابع فعال‌سازی Relu (Rectified Linear Unit) به طور معمول در شبکه‌های عصبی عمیق (Deep Neural Networks) توصیه شده است، زیرا این تابع بهبود مهمی در آموزش و عملکرد مدل‌ها می‌آورد. از مزایای استفاده از Relu می‌توان به سرعت آموزش، جلوگیری از مشکل مواجهه با مشکل مرگ نورون (Vanishing Gradient Problem) و افزایش قدرت انتقال سیگنال‌های غیرخطی اشاره کرد. به همین دلیل استفاده از Relu به جای Sigmoid در شبکه‌های عصبی رایج تر است. در واقع سیگموید را میتوان با سافت مکس مقایسه کرد که در ادامه شکست ان را میبینیم زیرا برای مثال دو کلاسه خوب است!

Softmax: این تابع بیشتر برای مسائل طبقه‌بندی استفاده می‌شود. این تابع ورودی‌های خروجی را به احتمالات مقابله‌ای تبدیل می‌کند که مجموع آن‌ها برابر با ۱ است، بنابراین می‌توان احتمال تعلق هر ورودی به هر کلاس را مشخص کرد.

تابع sigmoid به عنوان تابع فعال‌ساز در مسائل دسته‌بندی دو دسته‌ای (binary classification) معمولاً استفاده می‌شود، زیرا اعداد را به بازه ۰ تا ۱ محدود می‌کند که متناسب با خروجی‌های تنها یک برچسب است.

۵-نرخ یادگیری و الگوریتم های بهینه سازی:

در مدل‌های شبکه‌های عصبی، نرخ یادگیری (learning rate) میزانی است که مشخص می‌کند که چقدر وزن‌های شبکه در هر مرحله به سمت جواب بهینه تغییر کنند. این نرخ یادگیری می تواند بر اساس تجربه و تلاش های انجام شده توسط افراد و یا با استفاده از الگوریتم‌های بهینه‌سازی مشخص شود.

برای کد تایید نرخ یادگیری در مدل‌های شبکه‌های عصبی، معمولا از رویکردهای زیر استفاده می‌شود:

۱. Grid Search: با استفاده از روش Grid Search، می‌توان یک مجموعه از مقادیر نرخ یادگیری را تعیین کرده و سپس مدل را بر اساس هرکدام از این مقادیر آموزش داده و به دنبال بهترین عملکرد مدل با توجه به مقدار نرخ یادگیری باشیم.

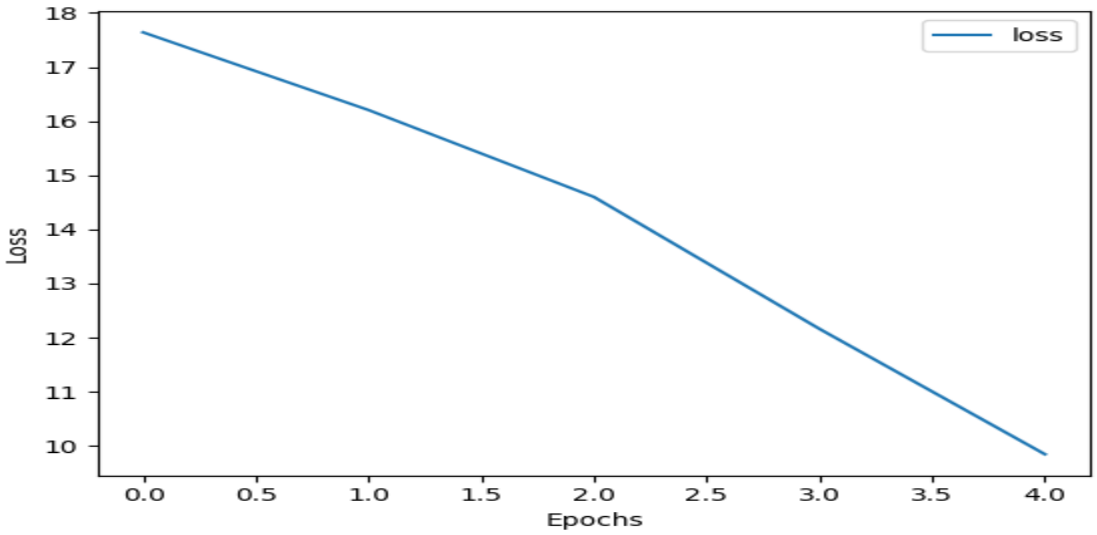
۲. Random Search: در این روش، مقادیر نرخ یادگیری به صورت تصادفی انتخاب می‌شوند و مدل بر اساس این مقادیر آموزش داده می‌شود. این روش می‌تواند به صورت موثرتری مقدار بهینه را پیدا کند به واسطه جستجو در فضای مقادیر به صورت تصادفی.

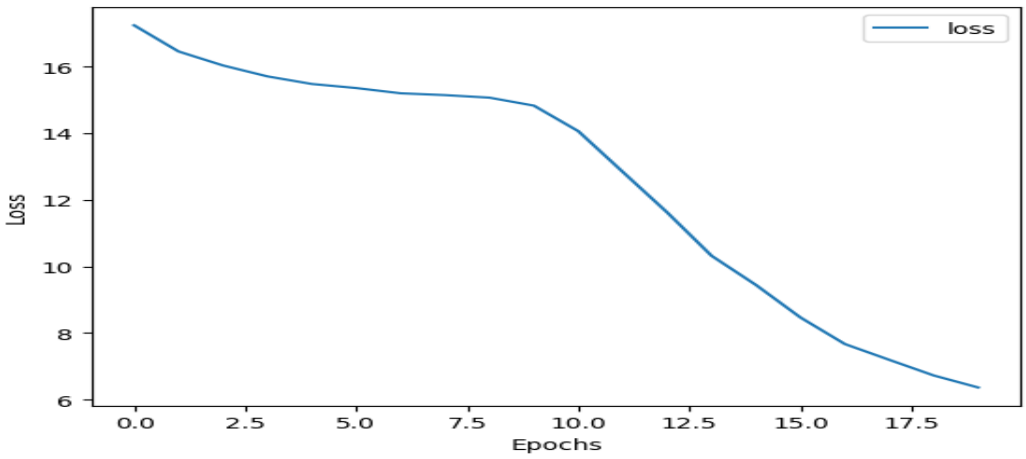
۳. Optimization Algorithms: الگوریتم‌های بهینه‌سازی مانند Adam, RMSprop و SGD می‌توانند کمک کنند تا نرخ یادگیری بهینه برای مدل شبکه عصبی شما پیدا شود. که در اینجا ما از ادام برای بهبود نرخ یادگیر استفاده کردیم. که البته ادام به صورت دیفالت اگر نرخ یادگیری برایش تعیین نکنیم دیفالت ۰.۰۰۱ میگذارد یعنی :

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
```

```
            loss='categorical_crossentropy',
            metrics=['accuracy'])
```

حال با تغییر این عدد و تست مقادیر مختلف مثلا learning_rate=۰.۰۰۰۰۰۱ باعث می شد دقت پایین تر رود درواقع در اکسترمم های محلی گیر کند و گام برداشتن ان بسیار کوچک باشد. از طرفی هم بزرگ تر کردن ان سبب دور شدن ناگهانی از اکسترمم گلوبال بود و نمودار ها را از فیت دور تر می کرد:





پس در نهایت مقدار ۰.۰۰۱ را برگزیدم که به نسبت بر اساس تست و سرچ انتخاب مناسب تری به ازای تغییر اپوک ها و بچ نیز بود...!

بهینه‌سازی‌ها در اصل الگوریتم‌هایی هستند که هدف آن‌ها بهینه کردن پارامترهای مدل به نحوی است که تابع هدف کاهش یابد. بهینه‌سازها می‌توانند با تنظیم نرخ یادگیری، انجام بهینه‌سازی محلی یا جلوگیری از گیر کردن در نقاط مینیمم موجود، اهمیت بیشتری به انتقال سریع‌تر یا پایدار تر به مینیمم بدهند.

در مورد انتخاب الگوریتم بهینه‌سازی، تجربه، آزمون و خطا، و خصوصیات مدل و مساله می‌تواند به تصمیم نهایی کمک کند. بهینه‌سازی مناسب می‌تواند منجر به آموزش بهتر و سریع‌تر مدل شود، اما همیشه لازم است نکاتی همچون افزایش یا کاهش نرخ یادگیری را نیز در نظر گرفت.

برای آموزش مدل MLP بر روی دیتاست ، می‌توان از بهینه‌سازی‌های مختلفی مانند RMSprop یا SGD نیز استفاده کرد. انتخاب بهینه‌سازی مناسب بستگی به ویژگی‌های خاص مدل، مساله و دیتاست دارد.

- RMSPROP یک روش بهینه‌سازی است که از شبکه عصبی برای آموزش با داده‌های بزرگ استفاده می‌شود. این الگوریتم از نسبت تغییرات گرادیان را برای هر وزن استفاده می‌کند تا مقدار learning rate را تطبیق دهد. با استفاده از این متد، روشی موثر برای جلوگیری از شلیک زودرس هنگام یادگیری عمیق است. ولی طبق نتایج و مباحث یاد گرفته ادام از ان بهتر است زیرا یک روش ترکیبی از این و یک الگوریتم دیگر است پس قطعا این مورد از لیست انتخاب ها حذف می شود.

- SGD_M یک نوع از روش Stochastic Gradient Descent (SGD) است که از مفهوم momentum برای سرعت بخشیدن به فرایند یادگیری استفاده می‌کند. این الگوریتم از مفهوم گذشته گرادیان‌ها برای بهبود سرعت یادگیری استفاده می‌کند تا از مشکلات سرعت کوهیدن گرادیان در بهینه‌سازی SGD معمولی کاسته شود. این روش می‌تواند بهبود قابل توجهی در سرعت و کیفیت یادگیری شبکه‌های عصبی داشته باشد. پس این روش نیز از SGD بهتر است اما باز هم از ادام ضعیف تر زیرا طبق تئوریات ادام ترکیبی از انهاست و بهتر عمل می کند. حتی با جایگذاری انها به جای ادام از دقت بالای ۹۰ درصدی کاسته شد!

- Adam یک الگوریتم بهینه‌سازی است که ترکیبی از روش‌های RMSprop و Momentum است. Adam با استفاده از میانگین ریاضی و تجهیز شده‌ی گرادیان‌ها بهبودی بهینه‌سازی بخصوص در مسائل با مقیاس حداقلی دارد. همانطور که در درس ماشین لرنینگ خواندیم این الگوریتم به طور کلی می‌تواند به صورت موثری در مقایسه با الگوریتم‌های دیگر عمل کند و به سرعت و کارآیی مدل کمک کند.

نوع توابع هزینه و معیارها (Metrics) بستگی به نوع مسأله یادگیری ماشین دارد.

"لاس (loss)" میزان خطای تخمینی مدل در هر مرحله از آموزش است که به منظور بهبود عملکرد مدل کاهش داده می‌شود.

برخی از توابع هزینه معروف شامل:

۱. Binary Crossentropy (دسته‌بندی دودویی)

- استفاده معمولی برای مسائل تصمیم‌گیری دودویی است.

۲. Categorical Crossentropy (دسته‌بندی چند دسته‌ای)

- معمولا برای آموزش مدل‌هایی که باید داده‌ها را به یکی از چند دسته تقسیم کنند، مورد استفاده قرار می‌گیرد.

۳. Mean Squared Error (خطا میانگین مربعات)

- معمولا برای مسائل رگرسیون استفاده می‌شود.

۴. Kullback-Leibler Divergence (انحراف کولباک-لایبلر)

- برای مدل‌های توزیع احتمالاتی و یادگیری نظارت شده به‌خصوص مسائل تولید محتوا مانند مولدهای مقابله‌ای (GANs) استفاده می‌شود.

"متریک (metrics)" به معنای معیارهایی است که برای ارزیابی عملکرد مدل در هنگام آموزش یا آزمون استفاده می‌شود، مانند دقت، دقت خاصیتی و ...

معیارها نیز برای ارزیابی مدل استفاده می‌شوند. علاوه بر دقت، معیارهای دیگری نیز وجود دارند که می‌توان در مورد عملکرد مدل استفاده کرد مانند:

- فراخوانی (recall)

- دقت (precision)

- اف اسکور (F1-score)

- ماتریس درهم‌ریختگی (confusion matrix) و...

با توجه به نوع مسأله و نوع داده‌ها، انتخاب صحیح توابع هزینه و معیارهای مناسب بسیار حیاتی است. اما انتخاب categorical_crossentropy و accuracy در این مسئله ی خاص مناسب ترین حالت می تواند باشد.

- loss=categorical_crossentropy: این بخش مشخص می‌کند که برای اندازه‌گیری خطا یا هزینه در حین آموزش از تابع هزینه‌ی "categorical_crossentropy" استفاده شود. این تابع مخصوص کاربردی است که برای مسائل دسته‌بندی چند دسته‌ای مناسب است.

البته که یک نکته دیگر هم داریم:

اگر داده‌ها به صورت integer labels هستند، انتخاب sparse_categorical_crossentropy مناسب است. اما اگر داده‌ها تبدیل به one-hot encode شده‌اند، انتخاب categorical_crossentropy صحیح است. که در اینجا چون هات انکود کردیم پس کتگوریکال بهتر است!

- metrics=['accuracy']: این بخش به مدل مشخص می‌کند که در هر مرحله از آموزش، دقت (accuracy) را به عنوان معیار برای ارزیابی عملکرد مدل استفاده کند. که دقت نشان دهنده درصد داده‌هایی است که به درستی تشخیص داده شده‌اند.

٦-تعیین پارامتر و نرخ ها:

```
#Applying the model
hist = model.fit(X_train, [y_train[0], y_train[1], y_train[2], y_train[3],
y_train[4]], batch_size=32, epochs=60, validation_split=0.2)
#batch size- 32 defines no. of samples per gradient update
#Validation split=0.2 splits the training set in 80-20% for training nd testing
```

۱. num_epochs: این پارامتر تعداد دوره‌های آموزش را مشخص می‌کند، به این معنی که داده‌ها به مدل به مدتی آموزش داده می‌شوند. این مقدار بر اساس تست چندین عدد و تئوریات کتاب اصلی بر اساس پیچیدگی مسئله و حجم داده‌ها تنظیم کردم و نه اجرا انقدر کند و نفس گیر شد تا فیت شود و نه انقدر کم و

سریع بود که دقت پایین داشته باشد یعنی به یک نتیجه ی مطلوب در احتمالا مناسب ترین عدد رسیدیم. عداد نزدیک ۶۰ نیز یا دقت را پایین می اورد یا چندان سبب بهبود ان نبود و فقط سرعت را پایین می اورد.

۲. ۳۲ = batch_size: این پارامتر تعداد داده‌هایی که به صورت همزمان به مدل وارد می‌شود را مشخص می‌کند. استفاده از دسته‌های کوچک (batch) از داده‌ها بهینه‌سازی فرآیند آموزش را کمک می‌کند، زمانی که داده‌های زیادی داریم. به دلیل تعادل بین سرعت آموزش و حافظه ۳۲ استفاده شده است. انتخاب اندازه batch معمولاً یک **ترید اف** بین عملکرد و سرعت است. انتخاب اندازه batch بزرگتر از یک، می‌تواند کمک کند تا برای همه داده‌ها یک بار گرادیان‌ها محاسبه شود ولی از حافظه بیشتری استفاده کند. از سوی دیگر، انتخاب اندازه batch کوچکتر می‌تواند منجر به یک فرآیند آموزش ناپایدارتر شود ولی از حافظه کمتری استفاده کند. در اینجا با تنظیم اندازه batch به ۳۲، تلاش برای تعادل بین استفاده از حافظه و سرعت آموزش می‌شود. برای مجموعه داده که نسبتاً کوچک است، استفاده از یک اندازه batch بزرگتر مانند ۳۲ ممکن است به صورت عملی باشد و به سرعت آموزش کمک کند بدون اینکه به نحو چشمگیری از مقدار حافظه استفاده شود.

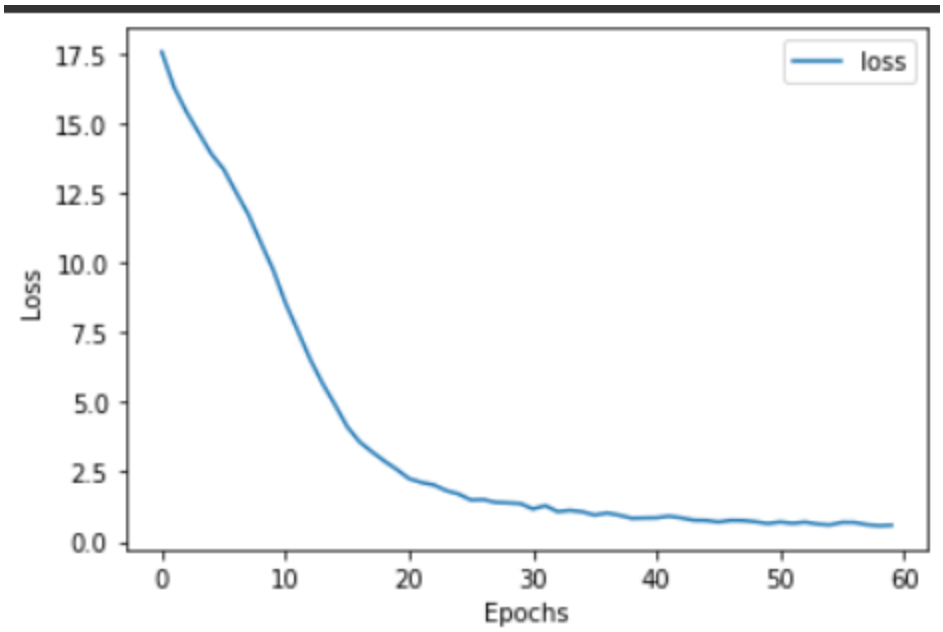
۷-رسم نمودار های مربوطه:

- برای تفسیر عملکرد کلی مدل از پلات های پایتون کمک گرفتم

```
#graph of loss vs epochs
for label in ["loss"]:
  plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
```

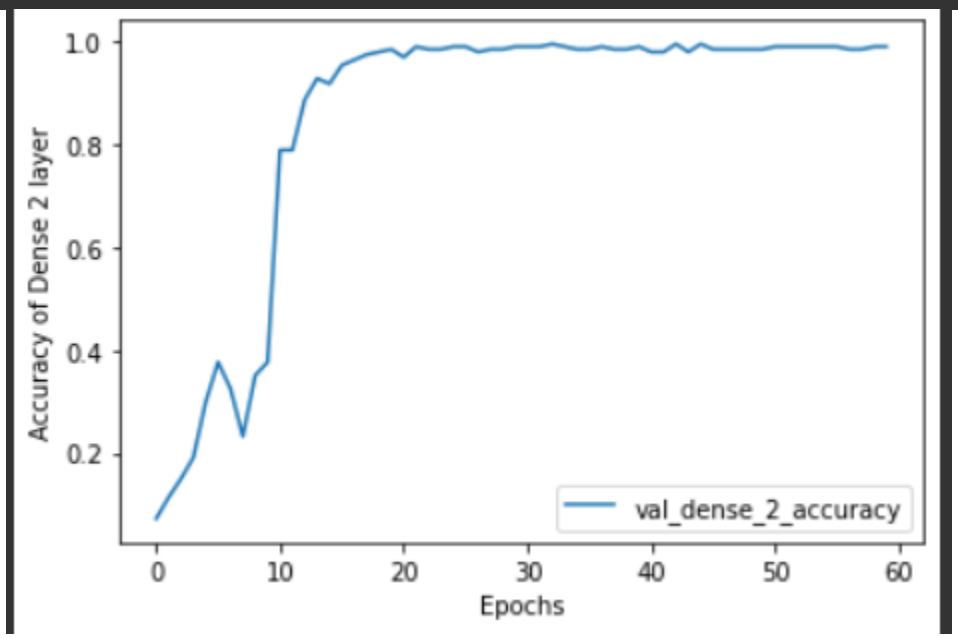
loss (که مربوط به loss در مرحله‌ی آموزش باشد) معمولا به میانگین خطای محاسبه شده بر روی داده‌های آموزش اشاره دارد، در حالی که val_loss مربوط به خطای محاسبه شده بر روی داده‌های اعتبارسنجی یا ارزیابی (validation set) است. پس درواقع علاوه بر ترین کردن یک تست روی داده های ترین نیز رخ داده که همان ولیدیت کردن می باشد.

اختلاف بین این دو معمولا نشان دهنده‌ی عملکرد مدل در داده‌های دیده نشده‌ی validation set نسبت به داده‌های آموزش است. اگر val_loss بیشتر از loss باشد، مدل ممکن است دچار overfitting (برازش بیش از حد) شده باشد، به این معنا که در حالتی شده که به داده‌های آموزش بسیار خوب عمل کند اما در داده‌های جدید (مانند داده‌های اعتبارسنجی) عملکرد بهتری نداشته باشد.



رسم می کنیم:

```
#graph of accuracy of dense_2 vs epochs
for label in ["val_dense_2_accuracy"]:
  plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("Accuracy of Dense 2 layer")
plt.show()
```



همانطور که از نمودار ها پیداست یعنی میزا اختلاف ارتفاع تست و ترین به میزان مناسب و معقولی است یعنی نه اورفیت رخ داده و نه اندرفیت و مدل به درستی فیت شده است!

۸-فیت شدن و پیشگویی هر داده با آن:

Captcha_Recognition_Project_Final.ipynb

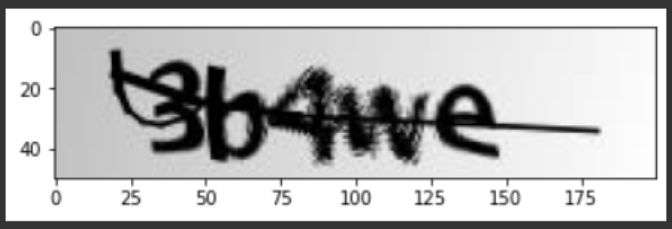
Captcha_Recognition_Project_Final.ipynb > ...

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...

```
#Check model on samples
img=cv2.imread('/content/drive/My Drive/mnop2.png',cv2.IMREAD_GRAYSCALE)
plt.imshow(img, cmap=plt.get_cmap('gray'))
```

[ ]

```
<matplotlib.image.AxesImage at 0x7f08a0788cf8>
```




```
print("Predicted Captcha =",predict('/content/drive/My Drive/mnop2.png'))
```

[ ]

```
Predicted Captcha = 3b4we
```

```
#Sample 2
```

۹-خروجی نهایی:

```
    #Loss on training set
    #Finding Loss on training set
    preds = model.evaluate(X_train, [y_train[0], y_train[1], y_train[2], y_train[3], y_train[4]])
    print ("Loss on training set= " + str(preds[0]))
[ ]
...  970/970 [==============================] - 3s 3ms/step
     Loss on training set= 0.23914067802816322


    #Finding loss on test set
    preds = model.evaluate(X_test, [y_test[0], y_test[1], y_test[2], y_test[3], y_test[4]])
    print ("Loss on testing set= " + str(preds[0]))
[ ]
...  100/100 [==============================] - 0s 3ms/step
     Loss on testing set= 2.1232204596325754


    #to predict captcha
    def predict(filepath):
        img = cv2.imread(filepath, cv2.IMREAD_GRAYSCALE)

        if img is not None: #image foud at file path
            img = img / 255.0 #Scale image
```

١٠-مراجع

[1] https://github.com
[2] https://stackoverflow.com/questions
[3] https://www.wikipedia.org/
[4] https://colab.research.google.com/
[5] https://www.tensorflow.org/guide/
[6] https://pandas.pydata.org/
[7] https://keras.io/
[8] https://github.com/topics/captcha-recognition
https://medium.com/@manvi./captcha-recognition-using-convolutional-neural-network-d191ef91330e