به نام خدا

گزارش کار درس پردازش گفتار

تمرین پنجم

شامل پاسخ سوالات اول و دوم(الف تا ت)است

ریحانه سراج

شماره دانشجویی :

830598030

سوال اول(پژوهشی)

توضیح در مورد بسترهای keras، tensorflow، torch(pytorch) ومعرفی الگوریتم های یادگیری در آن ها همچنین مقایسه ی آن ها با یکدیگر :

تنسورفلو یک کتابخانه ی رایگان و منبع باز (open source) برای محاسبات عددی و یادگیری ماشین در مقیاس بزرگ است. این کتابخانه توسط تیم Google Brain برای مصارف داخلی گوگل توسعه داده شده بود، ولی در ماه نوامبر سال ۲۰۱۵ در دسترس عموم قرار گرفت. این کتابخانه ریاضی در یادگیری ماشین بسیار تاثیر گذار است .همان گونه که می دانید یادگیری ماشین از مباحث پیچیده ی هوش مصنوعی به شمار می اید ،تنسورفلو با داشتن چهارچوب های قدرت مند برای پیاده سازی مدل های یادگیری ماشین روند دسترسی به داده ها ، مدل های آموزشی ، پیش بینی ها (Prediction) و ارزیابی نتایج را آسان تر کرده است.
تنسورفلو می‌تواند شبکه‌های عصبی عمیق را برای طبقه‌بندی ارقام دست‌نویس شده، تشخیص تصویر، Classification ،شبکه‌های عصبی بازگرداننده(Recurrent Neural Networks) ، شبکه عصبی ماشین بولتزمن محدود شده(restricted Boltzmann machine) ، شبکه عصبی باور عمیق (Deep Belief Network) ، شبکه های عصبی پیچشی (Convolutional Neural Network) ، word embedding، پردازش زبان طبیعی(NLP) و غیره را آموزش داده و اجرا کند .

مؤلفه های تنسورفلو

تنسور(Tensor) که مسئولیت کلیه محاسبات در TensorFlow را بر عهده دارد، یک بردار یا ماتریس با ابعاد n است که انواع داده ها را نشان می دهد. نمودار، تمام عملیات موجود در TensorFlow ، که مجموعه ای از محاسبات است وبه صورت متوالی انجام می شود را، نشان می دهد. نمودار مسئولیت اتصالات بین گره ها را بر عهده دارد ، اما مقادیر را نشان نمی دهد. TensorFlow از فریم ورک گراف (Graph) استفاده می کند. گراف در حین آموزش ، توصیفات تمام سری های محاسباتی را جمع آوری می کند.

محاسبات موجود در نمودار با اتصال تنسور ، به کمک گره و لبه انجام می شود. گره وظیفه انجام عملیات ریاضی را بر عهده دارد ، و گره خروجی نقاط پایانی را تولید می کند در حالی که لبه ها روابط ورودی / خروجی بین گره ها را توضیح می دهند. مهمتر از همه این که می توان نمودار را به چند تکه شکست و آنها را به صورت موازی بر روی چند CPU یا GPU اجرا کرد. تنسورفلو از محاسبات توزیع شده نیز پشتیبانی میکند به طوری که شما می توانید با تقسیم محاسبات در صدها سرور، شبکه های عصبی عظیم را روی مجموعه های آموزش بسیار بزرگ در مدت زمانی معقول آموزش دهید.

این کتابخانه قابل اجرا روی چندین CPU و GPU با افزونه‌های اختیاری CUDA و SYCL برای انجام پردازش‌های همه منظوره روی واحد پردازنده گرافیکی است. کتابخانه تنسورفلو برای سیستم‌عامل‌های ۶۴ بیتی لینوکس(Linux)، ویندوز(Windows)، مک‌اواس (macOS) و پلتفرم‌های موبایل مانند اندروید(Android) و iOS موجود است.

این نرم افزار با زبان CUDA ،Python ،C++ نوشته شده است و می توان برای استفاده و توسعه ی آن از رابط های C / C++ ،پایتون(Python) ، جاوا(Java) ، جاوا اسکریپت (JavaScript) ،R ،Swift ،Go ، جولیا(Julia) بهره برد به همین دلیل است که انعطاف زیادی در توسعه دارد و توسعه دهنده می تواند امکانات و سرویس های مورد نظر خود را در آن پیاده کند و مدل مورد نظرش را طراحی کند.

تورچ یک کتابخانه ی یادگیری ماشین و یادگیری عمیق منبع باز (open source) است که توسط Ronan Collobert, Samy Bengio , Johnny Mariethoz نوشته شده است و در سال 2002 منتشر شد. دارای چارچوب محاسبات علمی و یک زبان اسکریپت‌نویسی بر پایه زبان برنامه‌نویسی لوا (Lua) است که گسترهی وسیعی از الگوریتم‌های یادگیری عمیق را فراهم آورده و از زبان اسکریپت‌نویسی LuaJITو یک پیاده‌سازی Cاستفاده می‌کند.

این نرم افزار با زبان C و Lua نوشته شده است و می توان برای استفاده و توسعه ی آن از رابط های C / C++ ،Lua ، LuaJIT بهره برد همچنین قابل ذکر است که از Cuda نیز پشتیبانی می کند و همانند تنسرفلو قادر است شبکه های عصبی مثل شبکه عصبی بازگرداننده(Recurrent Neural Networks) ، شبکه های عصبی پیچشی (Convolutional Neural Network) را اموزش دهد و اجرا کند ولی دیگر از سال 2018 ، Torch توسعه نداشته است در حالی که تنسرفلو همچنان در حال توسعه است. با این حال ، PyTorch به طور فعال از ژوئن سال 2020 توسعه یافته است.

پای‌تورچ(PyTorch) ، یک کتابخانه یادگیری عمیق متن‌باز (Open Source) بر پایه کتابخانه تورچ (Torch) است که در زمینه ی بینایی ماشین و پردازش زبان طبیعی مناسب است.

فیس‌بوک(Facebook) این کتابخانه را سال 2016 راه‌اندازی کرد. بر روی پلتفرم های لینوکس(Linux)، ویندوز(Windows)، مک‌اواس (macOS) قابل اجرا است و با زبان CUDA ،Python ،C++ / C نوشته شده است همچنین از رابط های C++ و پایتون پشتیبانی می کند. شبکه های عصبی همچون عصبی بازگرداننده، شبکه های عصبی پیچشی را آموزش می دهد و اجرا می کند . PyTorchبه دلیل دو قابلیت سطح بالای منحصربه فرد خود معروف شده که شامل محاسبات تنسور با بهره‌گیری از توان شتاب‌دهنده پردازنده گرافیکی و ساخت شبکه‌های عصبی عمیق است.

یکی از عوامل کلیدی موفقیت PyTorch این است که کاملا بر پایه پایتون آماده‌سازی شده و برای توسعه یادگیری عمیق انعطاف پذیری بسیار بالایی دارد .هر کسی می‌تواند به سادگی مدل‌هایی از شبکه عصبی را با آن تولید کند. این کتابخانه در مقایسه با سایر رقبای خود قدمت کمتری دارد، اما به‌سرعت در حال پیشرفت است.

مقایسه ی PyTorch و TensorFlow

- ✓ برخلاف تنسورفلو که در آن کاربر باید کل گراف کامپیوتری را پیش از اجرای مدل تعریف کند، پای‌تورچ امکان تعریف گراف را به صورت «پویا» نیز فراهم می‌کند که این امکان اجازه می دهد مدل را در زمان اجرا بهینه کند و امکان تعریف / دستکاری گراف(نمودار) را در حال اجرا دارد.
- ✓ PyTorchیک API ساده است که وزن تمام مدل ها را ذخیره می کند اما TensorFlow مزیت های قابل توجه دیگری را نیز ارائه می دهد که می توان کل نمودار(گراف) را به عنوان یک بافر پروتکل ، از جمله پارامترها و عملکرد ها را ذخیره کرد. همچنین سایر زبان های پشتیبانی شده مانند C++ و Java ،می توانند گراف را بارگیری کنند که این برای توسعه پشته ها ضروری می باشد ولی پایتون آن را ارائه نمی دهد. همچنین هنگامی که کاربر کد اصلی مدل را تغییر می دهد اما می خواهد مدل قدیمی را اجرا کند بسیار مفید است.
- ✓ در قسمت مصور سازی فرآیند آموزش TensorFlow از کتابخانه ایی به نام TensorBoard وPyTorch از Visdom استفاده می کند .مصور سازی کمک می کند تا توسعه دهنده روند آموزش و پیگیری اشکالزدایی(developer track) را به روشی راحت تر دنبال کنند. ویژگی های ارائه شده توسط Visdom بسیار حداقلی و محدود است و TensorBoard در مصور سازی فرایند آموزش عملکرد بهتری نشان می دهد.

ویژگی های TensorBoard:

- ردیابی و مصور سازی معیارهایی مانند از دست دادن(loss) و دقت(accuracy)
- مصور سازی نمودار محاسباتی (لایه ها)
- مشاهده هیستوگرام وزن ها ، بایاس ها یا دیگر تنسورهای هایی که در طول زمان تغییر می کنند
- نمایش تصاویر ، متن و داده های صوتی.

تنسورفلو از سطح بالاتری از عملکرد پشتیبانی می کند مانند بررسی Tensor به صورت بی نهایت و NaN ،ارائه پشتیبانی برای تبدیل سری فوریه ولی PyTorch در این زمینه ویژگی های کمتری دارد.

- ✓ اگرچه می توان معماری یک شبکه عصبی را در هر یک از این چهارچوب ها پیاده سازی کرد ، اما نتیجه یکسان نخواهد بود. روند آموزش پارامترهای زیادی دارد که به چارچوب وابسته هستند. به عنوان مثال ، اگر شما در حال آموزش دیتابیس در PyTorch هستید می توانید فرآیند آموزش را با استفاده از GPU همانطور که در CUDA اجرا می شود (با پس زمینه C ++)تقویت کنید. در TensorFlow می توانید به GPU دسترسی پیدا کنید اما از شتاب GPU داخلی خود استفاده می کند ، بنابراین زمان آموزش این مدل ها همیشه بر اساس چارچوبی که انتخاب می کنید متفاوت خواهد بود.

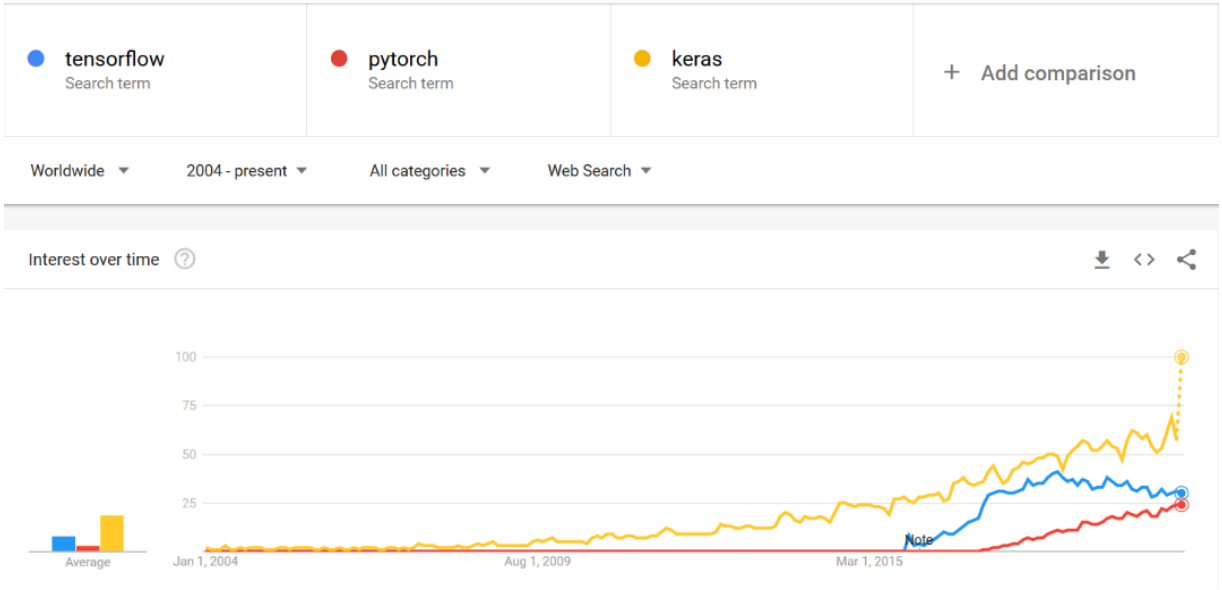
- ✓ هر دو فریم ورک مفید هستند و جامعه عظیمی از هردو استفاده می کنند. هر دو کتابخانه های یادگیری ماشینی را برای دستیابی به کارهای مختلف و انجام کار متفاوت فراهم می کنند. به طور خلاصه ، TensorFlowبرای سریعتر کردن کارها و ساختن محصولات مرتبط با هوش مصنوعی استفاده می شود ، در حالی که توسعه دهندگان تحقیق محور ، PyTorchرا ترجیح می دهند زیرا که پای تورچ، یک ابزار خوب برای پژوهش‌های یادگیری عمیق است و انعطاف‌پذیری و سرعت بالا را تامین می‌کند .

کراس یک کتابخانه منبع باز نوشته شده برای شبکه عصبی است. این کتابخانه توسط François Chollet نوشته شد و در سال 2015 منتشر شد. بر روی پلتفرم های لینوکس(Linux)، ویندوز(Windows)، مک‌اواس (macOS) قابل اجرا است .در محیط پایتون نوشته شده است ، همچنین از رابط های R و پایتون پشتیبانی می کند.

این کتابخانه برای سرعت بخشیدن به شبکه های یادگیری طراحی شده است .محیط آن بسیار user freindly است کار با آن برای کاربران بسیار راحت است و یادگیری آن اسان است.طراح این نرم افزار می گوید که کراس بیشتر به عنوان واسط کاربری است تا سطح انتزاع خوبی را برای استفاده از ابزار های یادگیری ماشین ارایه دهد .در واقع کراس بر روی بستر تنسورفلو پیاده سازی می شود که به استفاده ی سریع و راحت از تنسورفلو کمک می کند.

این کتابخانه برای شبکه هایی مانند شبکه عصبی پیچشی ،شبکه عصبی بازگرداننده طراحی شده است.قابل ذکر است که کراس بر خلاف تنسورفلو عملیات back end را برعهده نمی گیرد و برای آن به موتور هایback end نیاز دارد.کراس از بسیاری از ماژول هایی از جمله بهینه ساز ها تابع های فعال سازی بعضی از دیتا ست ها تابع های خطا ،لایه های عصبی ،تابع هزینه و ...پشتیبانی میکند.

این کتابخانه همچنان ورژن های جدیدی از آن در حال انتشار است.



این نمودار که بر اساس داده‌های گوگل ترندز (Google Trends) است، نشان دهنده ی استقبال کاربران از این سه بستر در سرتاسر دنیا از سال 2004 تا2020 است

سوال دوم

قسمت الف) کد این بخش به نام mymodel.py است و مدل به نام mymodel.hd5 با دقت 53.11 بر روی دیسک ذخیره شده است.

در این بخش برای ساختن مدل مورد نظر با استفاده از کتابخانه ی کراس مجموعه داده ی cifar10 را وارد میکنیم وداده های آموزش و برچسب های آن را به train_images وtrain_labels اختصاص می دهیم برای داده های تست و آزمایش هم test_images وtest_labels را اختصاص می دهیم. لازم به ذکر است که در ابتدا دیگر کتابخانه های مورد نیاز برای مدل درج می شوند.

```
from keras.datasets import cifar10

(train_images, train_labels), (test_images, test_labels) =cifar10.load_data()
```

همان گونه که مشاهده میکنید تعداد داده های اموزش 5000 تا است و ابعاد آن ها 32 در 32 است و به دلیل رنگی بودن تصاویر دارای 3 کانال (RGB) هستند و از نوع uint8 است. داده های آزمایش نیز دارای 1000 تا تصویر هستند .

این دیتا ها دارای 10 کلاس هستند.

```
train_images shape: (50000, 32, 32, 3)
train_images type: uint8
test_images shape: (10000, 32, 32, 3)
```

قبل از اینکه داده ها به شبکه برای آموزش و ارزیابی داده شوند نیاز به یکسری تغییرات بر روی داده هاست از جمله تبدیل نوع ان ها به `float` و نرمال سازی آن ها بین `0` تا `1` و همچنین تبدیل برچسب های داده ها به `0` و `1` به گونه ایی که مقدار `true` با `1` نمایش داده شود و بقیه ی ستون ها با مقدار `false`، `0` را نمایش دهند.

```
 #convert from integers to floats
train = train_images.astype('float32')
test = test_images.astype('float32')
 #normalize to range 0-1
X_train = train/255.0
X_test = test/255.0

 #one hot encode target values
Y_train = to_categorical(train_labels)
Y_test = to_categorical(test_labels)
```

شبکه را طبق گفته ی سوال می سازیم:

لایه dropout به عنوان regurarization برای کاهش مسئله overfit شدن تعریف می‌شود.

```
model = Sequential()
model.add(Conv2D(7,(3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(9,(3,3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(10,activation='softmax'))
```

این دستور در کنسول معماری شبکه و تعداد پارامتر های اموزش داده شده را نمایش می دهد:

```
model.summary()
```

```
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d_1 (Conv2D) (None, 30, 30, 7) 196
_________________________________________________________________
conv2d_2 (Conv2D) (None, 28, 28, 9) 576
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 9) 0
_________________________________________________________________
dropout_1 (Dropout) (None, 14, 14, 9) 0
_________________________________________________________________
flatten_1 (Flatten) (None, 1764) 0
_________________________________________________________________
dense_1 (Dense) (None, 10) 17650
=================================================================
Total params: 18,422
Trainable params: 18,422
Non-trainable params: 0
```

بعد از تعریف مدل، مدل باید کامپایل شود پس شبکه را با تابع خطای categorical_crossentropy و بهینع ساز adam و نرخ یادگیری 0.01 کامپایل میکنیم. وبا early stoping میزان خطای validation را معیاری برای توقف آموزش شبکه قرار می دهیم و در ان patience را 5 قرار می دهیم که از خطای احتمالی در زمان توقف جلوگیری شود و کمی با مکث توقف داشته باشد

```
# compile model
model.compile(optimizer=Adam(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
#================================================
# simple early stopping
es = EarlyStopping(monitor='val_loss',min_delta=1e-3,patience=5,verbose=1,mode='auto'
        ,restore_best_weights=True )
```

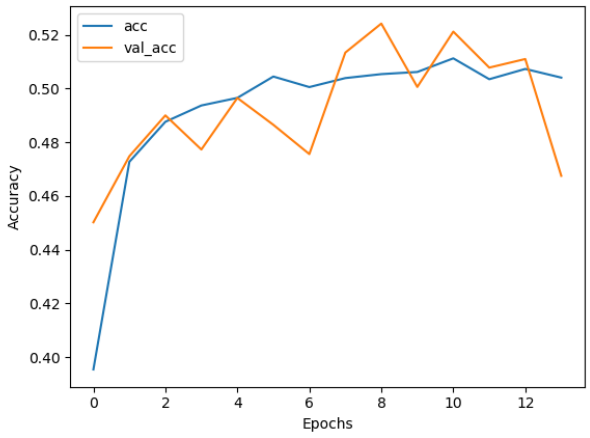
مدل را با x_train و y_train فیت می‌کنیم و مدل را آموزش می دهیم :

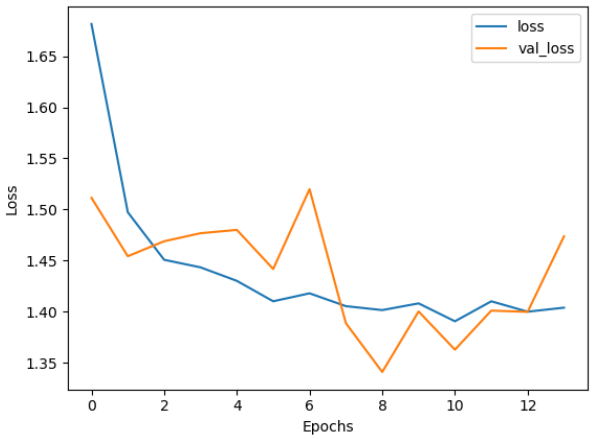
```
# Train our model
network_history = model.fit(X_train, Y_train, batch_size=32,#verbose=2
            callbacks=[es] ,epochs=1000, validation_split= .2 ,shuffle=True)
```

از 5000 داده ی آموزش 0.2 آن ها یعنی 1000 تای آن هارا به عنوان داده های validation (اعتبار سنجی) در نظر گرفتم. بعد از 14 epoch، early stoping صورت گرفت. هرepoch به معنی یک حرکت forward و backward به ازای تمام نمونه‌های آموزشی است.

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/1000
40000/40000 [==============================] - 10s 240us/step - loss: 1.6815 - acc:
0.3954 - val_loss: 1.5114 - val_acc: 0.4502
Epoch 2/1000
40000/40000 [==============================] - 5s 134us/step - loss: 1.4972 - acc:
0.4727 - val_loss: 1.4542 - val_acc: 0.4748
Epoch 3/1000
40000/40000 [==============================] - 5s 132us/step - loss: 1.4507 - acc:
0.4876 - val_loss: 1.4689 - val_acc: 0.4900
Epoch 4/1000
40000/40000 [==============================] - 5s 133us/step - loss: 1.4433 - acc:
0.4937 - val_loss: 1.4767 - val_acc: 0.4773
Epoch 5/1000
40000/40000 [==============================] - 5s 133us/step - loss: 1.4301 - acc:
0.4965 - val_loss: 1.4800 - val_acc: 0.4965
Epoch 6/1000
40000/40000 [==============================] - 5s 133us/step - loss: 1.4101 - acc:
0.5045 - val_loss: 1.4417 - val_acc: 0.4865
Epoch 7/1000
40000/40000 [==============================] - 5s 132us/step - loss: 1.4179 - acc:
0.5005 - val_loss: 1.5199 - val_acc: 0.4756
Epoch 8/1000
```

```
40000/40000 [==============================] - 5s 132us/step - loss: 1.4053 - acc:
0.5039 - val_loss: 1.3886 - val_acc: 0.5134
Epoch 9/1000
40000/40000 [==============================] - 5s 133us/step - loss: 1.4015 - acc:
0.5054 - val_loss: 1.3409 - val_acc: 0.5242
Epoch 10/1000
40000/40000 [==============================] - 5s 133us/step - loss: 1.4080 - acc:
0.5062 - val_loss: 1.4001 - val_acc: 0.5006
Epoch 11/1000
40000/40000 [==============================] - 5s 132us/step - loss: 1.3905 - acc:
0.5112 - val_loss: 1.3627 - val_acc: 0.5212
Epoch 12/1000
40000/40000 [==============================] - 5s 132us/step - loss: 1.4101 - acc:
0.5035 - val_loss: 1.4010 - val_acc: 0.5078
Epoch 13/1000
40000/40000 [==============================] - 5s 133us/step - loss: 1.3999 - acc:
0.5073 - val_loss: 1.3997 - val_acc: 0.5110
Epoch 14/1000
40000/40000 [==============================] - 5s 133us/step - loss: 1.4038 - acc:
0.5041 - val_loss: 1.4737 - val_acc: 0.4675
Restoring model weights from the end of the best epoch
Epoch 00014: early stopping
```





✓ **نمودار روند تغییرات خطا و دقت بر روی داده های train,validation**

```
def plot_history(net_history):

    history = net_history.history

    import matplotlib.pyplot as plt

    losses = history['loss']

    val_losses = history['val_loss']
```

```
    accuracies = history['acc']
    val_accuracies = history['val_acc']
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(losses)
    plt.plot(val_losses)
    plt.legend(['loss', 'val_loss']
    plt.figure()
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.plot(accuracies)
    plt.plot(val_accuracies)
    plt.legend(['acc', 'val_acc'])
```

✓ **نمایش دقت بر روی داده های آزمایش:**

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test accuracy:', score[1]*100)
```

```
Test accuracy: 53.11
```

✓ **میانگین مدت زمان اجرای اپوک ها:** 5.35 ثانیه

سوال دوم

قسمت ب) این بخش در پوشه ی ب با دقت 79 ذخیره شده:

ابتدا تمام داده های صوتی پوشه های TrainSetو TestSet رابا استفاده از کد زیر به اسپکتروگرام تبدیل کرده و حین تبدیل حاشیه های تصاویر را نیز حذف می کنیم و در اخر به صورت پوشه بندی شده از 0 تا 9 داده ها در پوشه هایی با برچسب مربوط به خود ذخیره می شوند در TrainImgو TestImg . این کد به نام spectrogramTrain.py و spectrogramTest.py در پوشه ی ب با دقت 79 ذخیره شده.

```
counter=0
path = os.path.dirname(os.path.realpath(__file__))+"/TrainSet/"
listpicpath=os.listdir(path)
for folder in listpicpath:
   piclist=os.listdir(path+folder)
   for file in piclist:
      counter+=1
      sample_rate, samples = wavfile.read(path + "/" + folder + "/" + file)
      fig,ax=plt.subplots(1)
      fig.subplots_adjust(left=0,right=1,bottom=0,top=1)
      ax.axis('off')
      pxx,freqs,bins,im=plt.specgram(samples, Fs=sample_rate)
      fig.savefig("TrainImg/" + folder + "/" +file +str(counter)+ ".png")
```

با این دستور سایز و نوع تصاویر تولیدی را متوجه می شویم :

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread('E:/tam5/TrainImg/0/FAC_ZA.08.wav.png')
imgplot = plt.imshow(img)
plt.show()
print("img shape: ", img.shape)
print("img type: ", img.dtype)
```

با انتخاب یکی از تصاویر اسپکترام و با اجرای این کد متوجه می شویم که اسپکترام های تولیدی دارای سایز 480در 640 و از نوع float 32 هستند.

```
img shape:  (480, 640, 4)
img type:  float32
```

حال باید تغییر سایز بدهیم تا سایز آن ها همانند دیتا ست قسمت الف 32 در 32 بشوند زیرا که شبکه تولیدی با این ابعاد آموزش دیده است.برای این کار تصاویر را از پوشه ی TrainImg یکی یکی با دستور cv2.imread می خوانیم و با دستور cv2.resize سایز آن ها را به سااایز 32 در 32 تغییر می دهیم. در انتها با دستور cv2.imwrite تصاویر آموزش را در TrainSpecResize ذخیره می کنیم.داده های تست نیز همین گونه در TestSpecResize ذخیره می شود.

```
path = os.path.dirname(os.path.realpath(__file__))+"/TrainImg/"
listpicpath = os.listdir(path)
for folder in listpicpath:
   piclist=os.listdir(path+folder)
   for file in piclist
      SpecTrain = cv2.imread(path + "/" + folder + "/" + file)
      new_width  = 32
      new_height = 32
      SpecTrainResize=cv2.resize(SpecTrain, (new_width,new_height))
      train_images=cv2.imwrite("TrainSpecResize/"+ folder + "/" +file ,SpecTrainResize)
```

در مرحله ی آخر که کد درفایل mymodelOnSpecImg.py قرار دارد در ابتدا داده ها که همان تصاویر ریسایز شده هستند را اماده می کنیم تا به شبکه قسمت الف بدهیم.

```
def read_data(path):
 images=[]
 labels = []
 listpath = os.listdir(path)
 for folder in listpath:
    filelist = os.listdir(os.path.join(path, folder))
    for file in filelist:
       samples = cv2.imread(path + "/" + folder + "/" + file)
       images.append(samples)
       labels.append(int(folder))
 images = np.array(images)
 return images, labels
```

در این تابع read_data با توجه به ادرسی که به عنوان ورودی می گیرد اسپکتوروگرام‌ها راخوانده و تصویر و برچسب آن ها را در دو ارایه مجزا قرار می‌دهد.

```
path_train="E:/tam5/TrainSpecResize"
path_test="E:/tam5/TestSpecResize"

train_images, train_labels = read_data(path_train)
test_images, test_labels = read_data(path_train)
```

بقیه مراحل که مشاهده می کنید همانند قسمت الف است.

```
# convert from integers to floats
train = train_images.astype('float32')
test = test_images.astype('float32')
# normalize to range 0-1
X_train = train/255.0
```

```
X_test = test/255.0


# one hot encode target values

Y_train = to_categorical(train_labels)

Y_test = to_categorical(test_labels)
```

همان گونه که در سوال از ما خواسته شده بود بعد از بارگذاری مدل برای از بین بردن وزن های لایه ی اخر با دستور پاپ لا یه ی اخر مدل را حذف می کنیم

```
model = load_model('mymodel.hdf5')

model.pop()

model.summary()
```

```
Layer (type) Output Shape Param #
=================================================================
conv2d_1 (Conv2D) (None, 30, 30, 7) 196
_________________________________________________________________
conv2d_2 (Conv2D) (None, 28, 28, 9) 576
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 9) 0
_________________________________________________________________
dropout_1 (Dropout) (None, 14, 14, 9) 0
_________________________________________________________________
flatten_1 (Flatten) (None, 1764) 0
```

سپس لایه های آخر را دوباره اضافه می کنیم.

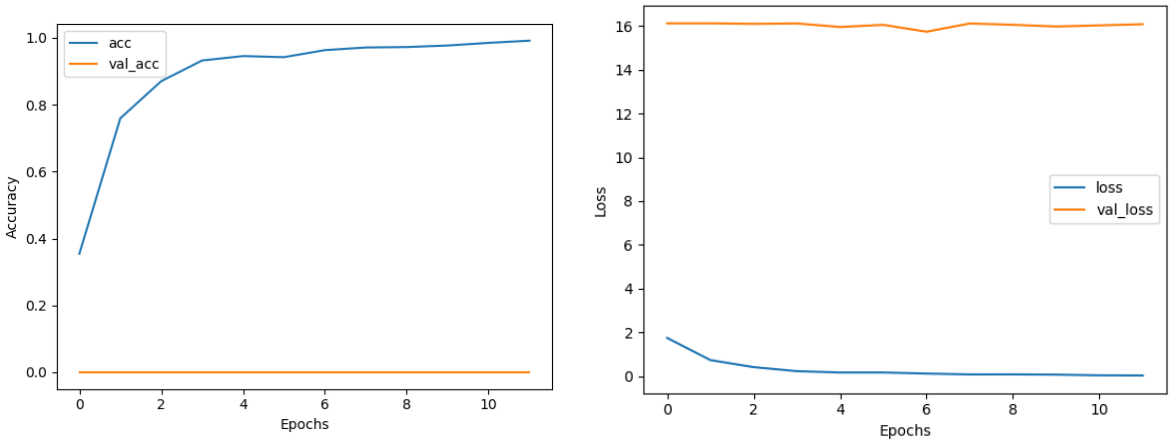
```
model.add(Dense(10,activation='softmax'))

model.summary()
```

```
Layer (type) Output Shape Param #
=================================================================
conv2d_1 (Conv2D) (None, 30, 30, 7) 196
_________________________________________________________________
conv2d_2 (Conv2D) (None, 28, 28, 9) 576
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 9) 0
_________________________________________________________________
dropout_1 (Dropout) (None, 14, 14, 9) 0
_________________________________________________________________
flatten_1 (Flatten) (None, 1764) 0
_________________________________________________________________
dense_1 (Dense) (None, 10) 17650
```

حال دیگر وزن های لایه اخر حذف شده اند ومدل را کامپایل می کنیم وشبکه ی از پیش آموزش دیده شده را با دیتای جدید برای تشخیص اعداد 0 تا9 باز آموزش می دهیم.

1900 تا دیتای آموزش داریم که 0.2 آن را یعنی 380 تای آن را validation می گیریم. خروجی به صورت زیر است:

```
Train on 1520 samples, validate on 380 samples
Epoch 1/50
1520/1520 [==============================] - 2s 1ms/step - loss: 1.7463 - acc: 0.3546
- val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 2/50
1520/1520 [==============================] - 0s 123us/step - loss: 0.7328 - acc:
0.7592 - val_loss: 16.1181 - val_acc: 0.0000e+00
Epoch 3/50
1520/1520 [==============================] - 0s 121us/step - loss: 0.4124 - acc:
0.8704 - val_loss: 16.0960 - val_acc: 0.0000e+00
Epoch 4/50
1520/1520 [==============================] - 0s 123us/step - loss: 0.2282 - acc:
0.9322 - val_loss: 16.1128 - val_acc: 0.0000e+00
Epoch 5/50
1520/1520 [==============================] - 0s 123us/step - loss: 0.1672 - acc:
0.9454 - val_loss: 15.9509 - val_acc: 0.0000e+00
Epoch 6/50
1520/1520 [==============================] - 0s 124us/step - loss: 0.1671 - acc:
0.9421 - val_loss: 16.0494 - val_acc: 0.0000e+00
Epoch 7/50
1520/1520 [==============================] - 0s 123us/step - loss: 0.1180 - acc:
0.9632 - val_loss: 15.7347 - val_acc: 0.0000e+00
Epoch 8/50
1520/1520 [==============================] - 0s 123us/step - loss: 0.0790 - acc:
0.9711 - val_loss: 16.1117 - val_acc: 0.0000e+00
Epoch 9/50
1520/1520 [==============================] - 0s 123us/step - loss: 0.0808 - acc:
0.9724 - val_loss: 16.0525 - val_acc: 0.0000e+00
Epoch 10/50
1520/1520 [==============================] - 0s 126us/step - loss: 0.0696 - acc:
0.9770 - val_loss: 15.9732 - val_acc: 0.0000e+00
Epoch 11/50
1520/1520 [==============================] - 0s 133us/step - loss: 0.0374 - acc:
0.9849 - val_loss: 16.0253 - val_acc: 0.0000e+00
Epoch 12/50
1520/1520 [==============================] - 0s 119us/step - loss: 0.0313 - acc:
0.9914 - val_loss: 16.0781 - val_acc: 0.0000e+00
Restoring model weights from the end of the best epoch
Epoch 00012: early stopping
```



✓ **نمودار روند تغییرات خطا و دقت بر روی داده های `train,validation`**

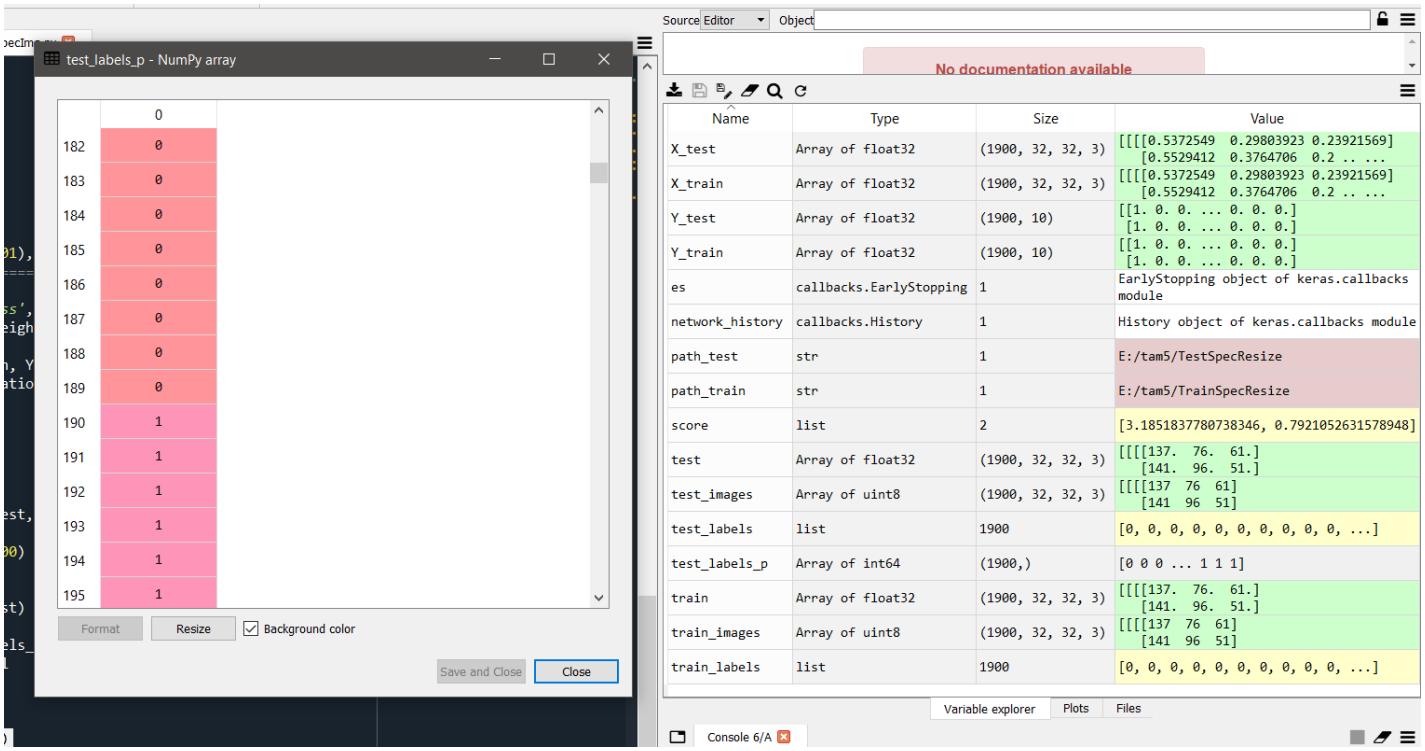
✓ **نمایش دقت بر روی داده های آزمایش:**

```
Test accuracy: 79.21052631578948
```

✓ **میانگین مدت زمان اجرای اپوک ها:** 1.16 ثانیه

✓ **این شبکه به اسم mymodelOnSpecImg ذخیره شده همچنین فایلی در پوشه ی(ب با دقت 79)قرار دادم که نشان دهنده ی ارایه ای ازبرچسب های تشخیص کلاس ها ی 0 تا 9 بر روی داده های تست است.**

```
test_labels_p = model.predict(X_test)
test_labels_p = np.argmax(test_labels_p, axis=1)
```

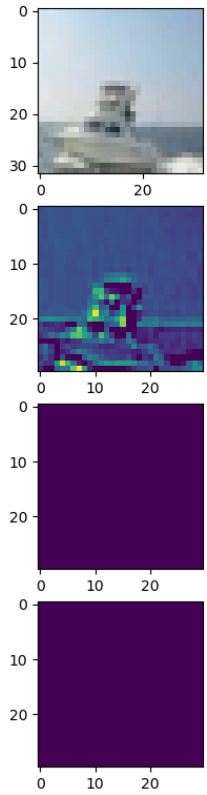

test_labels_p - NumPy array
0
182 0
183 0
184 0
185 0
186 0
187 0
188 0
189 0
190 1
191 1
192 1
193 1
194 1
195 1
Format
Resize
Background color
Save and Close
Close
Source Editor
Object
No documentation available
Name Type Size Value
X_test Array of float32 (1900, 32, 32, 3) [[[[0.5372549 0.29803923 0.23921569] [0.5529412 0.3764706 0.2 .. ...
X_train Array of float32 (1900, 32, 32, 3) [[[[0.5372549 0.29803923 0.23921569] [0.5529412 0.3764706 0.2 .. ...
Y_test Array of float32 (1900, 10) [[1. 0. 0. ... 0. 0. 0.] [1. 0. 0. ... 0. 0. 0.]
Y_train Array of float32 (1900, 10) [[1. 0. 0. ... 0. 0. 0.] [1. 0. 0. ... 0. 0. 0.]
es callbacks.EarlyStopping 1 EarlyStopping object of keras.callbacks module
network_history callbacks.History 1 History object of keras.callbacks module
path_test str 1 E:/tam5/TestSpecResize
path_train str 1 E:/tam5/TrainSpecResize
score list 2 [3.1851837780738346, 0.7921052631578948]
test Array of float32 (1900, 32, 32, 3) [[[[137. 76. 61.] [141. 96. 51.]
test_images Array of uint8 (1900, 32, 32, 3) [[[[137 76 61] [141 96 51]
test_labels list 1900 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]
test_labels_p Array of int64 (1900,) [0 0 0 ... 1 1 1]
train Array of float32 (1900, 32, 32, 3) [[[[137. 76. 61.] [141. 96. 51.]
train_images Array of uint8 (1900, 32, 32, 3) [[[[137 76 61] [141 96 51]
train_labels list 1900 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]
Variable explorer
Plots
Files
Console 6/A

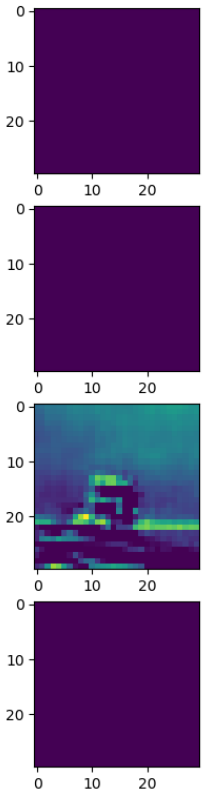
سوال دوم

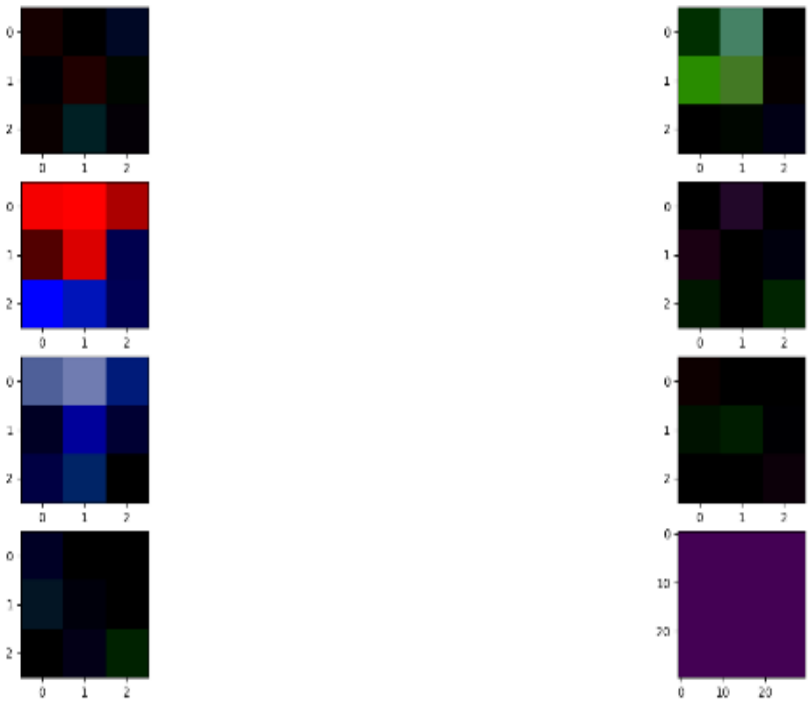
قسمت پ) تمام اطلاعات این کد در پوشه ی 3 ذخیره شده.در این سوال به دلیل وجود ابهام که کدام شبکه باید مورد استفاده قرار گیرد هم بر روی شبکه ی الف (Q3shabakeye A) وهم بر روی شبکه ی ب (Q3shabakeye B)وزن ها ی کانال اول و فیلتر های لایه پیچشی نمایش داده می شود که تا جایی که قابل مشاهده است تفاوتی در خروجی نداشتند. در زیر به عنوان نمونه تصاویر تولیدی از فیلتر ها و وزن های کانال اول لایه ی اول پیچشی درشبکه ی مدل قسمت ب را نمایش میدهم.

```
(train_images, train_labels), (test_images, test_labels) =cifar10.load_data()

X=train_images[100]

Y=train_labels[100]

pyplot.subplot(421)

pyplot.imshow(X)
```

به دلخواه تصویر 100 را انتخاب می کنم که یک کشتی است.







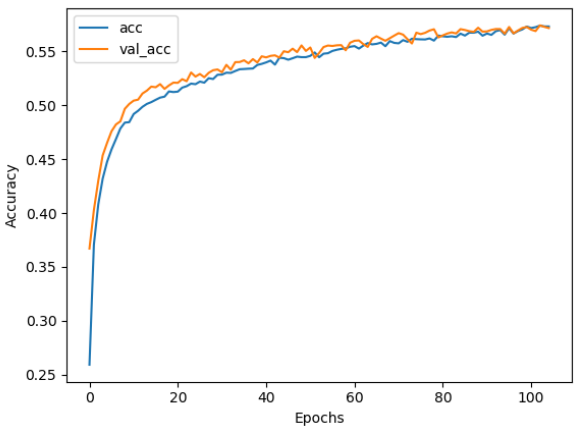
فیلتر های موجود در لایه ی پیچشی هر کدام ویژگی ایی از تصویر ورودی را نمایش می دهند به این صورت که یک فیلتر به تشخیص خطوط مستقیم در تصویر کمک می‌کند و دیگری در تشخیص دایره‌ها و غیره کمک می‌کند.در این جا بین فیلتر و وزن ها بحث global patterns و local patterns مطرح می شود. در واقع وزن ها ویژگی های محلی تر از تصویر را نمایش می دهند و فیلتر که تصویر کلی از دیتا را نمایش می دهند از کنار هم قرار دادن این وزن ها حاصل می شود که با آموزش شبکه می توان به مدلی دست یافت که بتواند با یادگیری نقاط محلی و به صورت کلی فیلتر ها به تشخیص خوبی نسبت به دیتای جدید برسد.

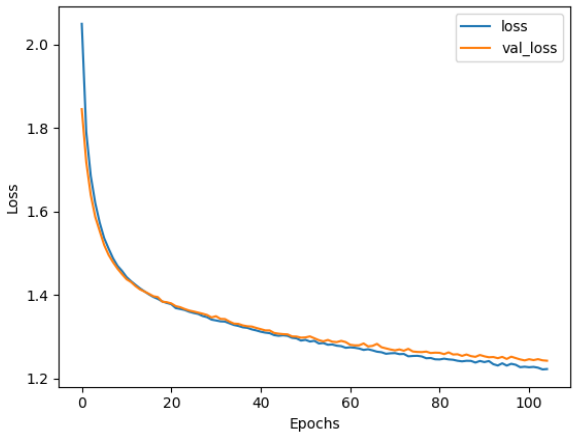
سوال دوم

قسمت ت) تمام اطلاعات در پوشه ی 4 ذخیره شده است.

تغییرات لازم بر روی شبکه ی الف طبق صورت سوال اعمال می شود:

کد آن به اسم ghesmate4_1 قرار گرفته است.





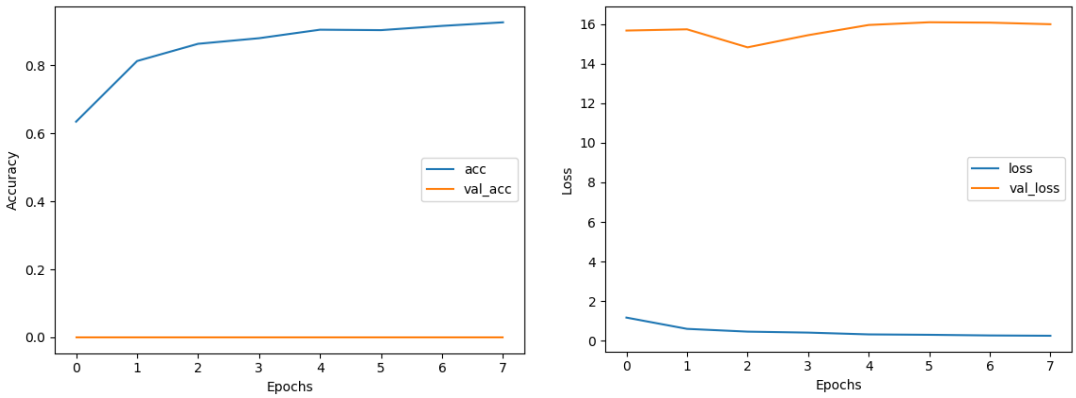
- ✓ **نمودار روند تغییرات خطا و دقت بر روی داده های `train,validation`**
- ✓ **مدت زمان آموزش این مدل به 105 اپوک رسید که بسیار بیشتر از قسمت الف است**
- ✓ **نمایش دقت بر روی داده های آزمایش:**
- ✓ **همان گونه که مشاهده می کنید دقت مدل نیز افزایش داشته است**

```
Epoch 00105: early stopping
Test accuracy: 57.07
```

میانگین مدت زمان اجرای اپوک ها:4.58 ثانیه

تغییرات لازم بر روی شبکه ی ب طبق صورت سوال اعمال می شود:

کد آن به اسم ghesmate4_2 قرار گرفته است.



- ✓ **نمودار روند تغییرات خطا و دقت بر روی داده های `train,validation`**
- ✓ **مدت زمان آموزش این مدل به 8 اپوک رسید.**

- ✓ **نمایش دقت بر روی داده های آزمایش:**

```
Test accuracy: 71.47368421052632
```

میانگین مدت زمان اجرای اپوک ها:0.25 ثانیه