Amirkabir University of Technology
(Tehran Polytechnic)

Department of
Computer Engineering

Artificial Neural Networks

Homework 4

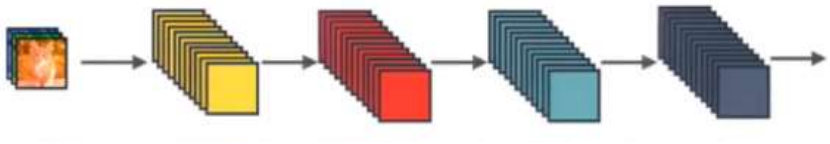
Najmeh Mohammadbagheri

99131009

گزارش تمرین

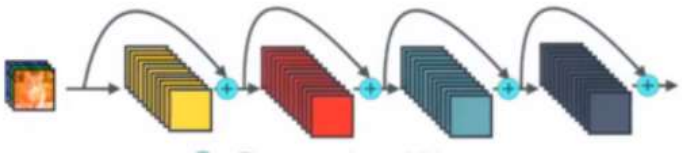
سوال اول

هر دو شبکه‌ی رزنت و دنس‌نت، نسخه‌های شبکه‌ی کانولوشنی هستند. معماری شبکه‌ی کانولوشنی ساده به صورت شکل۱ است.



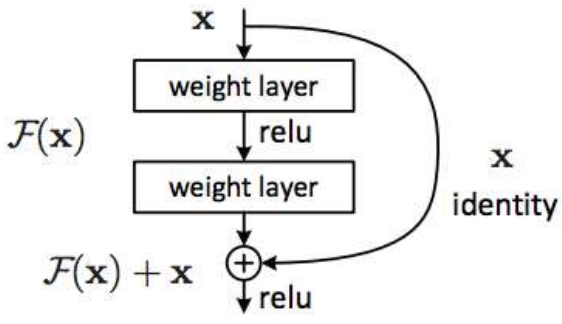
شکل ۱ شبکه‌ی کانولوشنی

در این شبکه‌های کانولوشنی هرچقدر عمق بیشتر می‌شود ویژگی‌های بیشتر و دقیق‌تری از ورودی استخراج می‌شود. اما یک مشکل وجود دارد: همانطور که می‌دانیم قانون یادگیری وزن‌ها در این شبکه انتشار به عقب است و در این یادگیری هر لایه که عقب می‌رویم یک بار گرادیان محاسبه می‌کنیم، بنابراین در لایه‌های ابتدایی گرادیان بسیار کوچک می‌شود و عملا یادگیری لایه-های ابتدایی صورت نمی‌گیرد. برای جلوگیری از این مشکل(یعنی مشکل ناپدیدشدن گرادیان)، شبکه‌های **رزنت** معرفی شدند. معماری این شبکه در شکل۲ قابل مشاهده است.



شکل ۲ مفهوم شبکه‌ی resnet

همانطور که از تصویر ۲ مشاهده می‌شود ورودی هر لایه، خروجی لایه‌ی قبل + ورودی لایه‌ی قبل است. برای درک بهتر معماری، اگر شبکه را متشکل از بلوک‌های کوچک در نظر بگیریم هر بلوک به صورت شکل ۳ است. همانطور که مشاهده می‌شود ورودی هر لایه از یک لایه‌ی کانولوشنی و ورودی همان لایه تشکیل شده است؛ یعنی زمان محاسبه‌ی گرادیان یک واحد جمع داریم که مانع از بین رفتن اثر گرادیان در لایه‌های ابتدایی می‌شود.



شکل ۳ یک بلوک resnet

در نهایت ورودی هر لایه به صورت رابطه‌ی زیر نوشته می‌شود که W_l وزن‌های لایه‌ی کانولوشنی مربوطه است.

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)$$

با استفاده از رابطه‌ی بالا و فرض همانی بودن توابع استفاده شده، می‌توان خروجی لایه‌ی L(مثلا ۱۵۰) را به صورت تابعی از ورودی لایه‌ی l(مثلا ۱) به شکل زیر نوشت:

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)$$

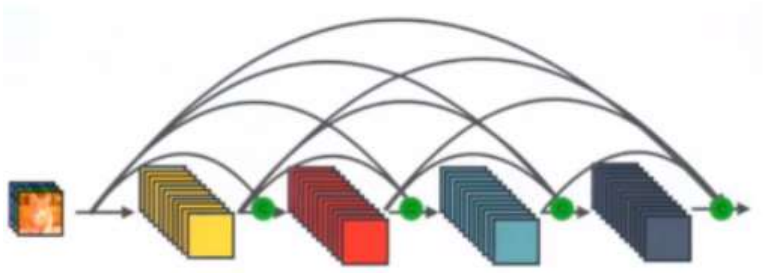
و در نهایت محاسبه‌ی وزن‌ها به صورت رابطه‌ی زیر انجام می‌شود:

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right)$$

همانطور که از رابطه‌ی بالا مشهود است به دلیل وجود عدد یک در گرادیان محاسبه شده، مشکل ناپدیدشدن گرادیان در لایه‌های ابتدایی پیش نمی‌آید.

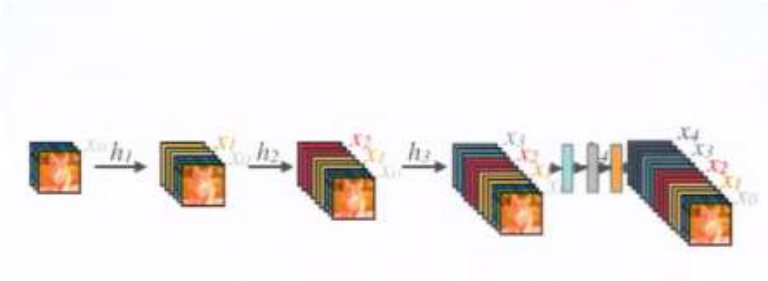
*نکته: در شبکه‌های جلورو ساده این مشکل گرادیان به این شدت وجود نداشت زیرا تعداد لایه‌های شبکه خیلی کم بودند. مشکل گرادیان در تعداد زیاد لایه‌ها خودش را نشان می‌دهد. از طرفی هم نمیخواهیم تعداد لایه‌های کانولوشنی را کاهش دهیم تا این اثر از بین برود، زیرا با کاهش تعداد لایه‌ها، ویژگی‌های کمتری استخراج می‌شوند.

شبکه‌ی **دنس‌نت** نسخه‌ی بعد از رزنت است که معماری آن در شکل ۴ مشاهده می‌شود.



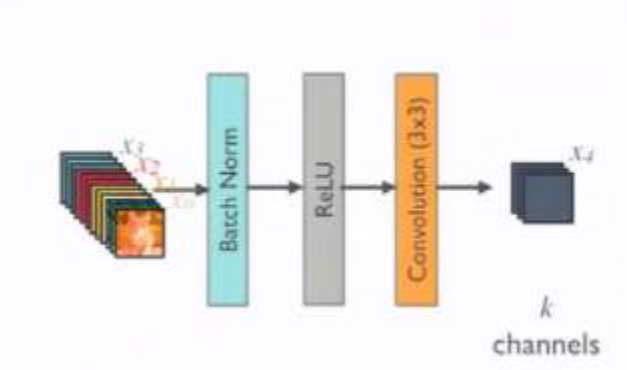
شکل ٤ مفهوم شبکه‌ی densenet

در این معماری هر لایه تنها ورودی لایه‌ی قبل را دریافت نمی‌کند بلکه ورودی تمام لایه‌های قبل از خودش را می‌گیرد. با اینکار تمامی اطلاعات لایه‌های قبلی در هر لایه حضور دارند. در شبکه‌های دنس‌نت بلوک‌های دنس به صورتی که گفته شد پیاده‌سازی می‌شوند و تعداد کانال‌ها در این بلوک‌ها افزایش می‌یابد(زیرا حاصل الحاق کانال‌های قبلی نیز هست.) در شکل ۵ داخل یک بلوک دنس را مشاهده می‌کنیم.



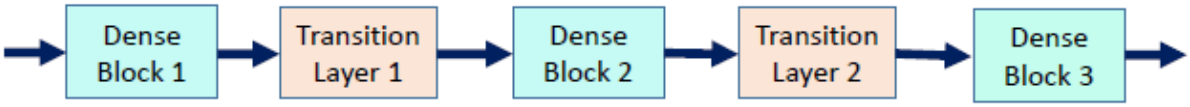
شکل ۵ داخل یک بلوک دنس

اما ما نمی‌خواهیم تعداد کانال‌ها به همین صورت افزایش یابد بنابراین بعد از هر بلوک دنس یک لایه‌ی انتقال قرار می‌دهیم که خروجی لایه‌ی دنس را در تعداد مشخصی کانال(growth rate = k) تجمیع می‌کند. عملیات این لایه به صورت شکل ۶ است.



شکل ۶ لایه ی انتقال

به صورت خلاصه معماری شبکه‌ی دنس‌نت به صورت شکل ۷ است.



شکل ۷ معماری شبکه‌ی دنس‌نت

مقایسه: تعداد پارامترهایی که دنس‌نت باید آموزش ببیند کمتر از پارامترهای رزنت است.

سوال دوم

برای پیش‌پردازش داده‌ها، سایز تمام داده‌ها ۱۲۸ در ۱۲۸ شد. همچنین شافلینگ برروی داده‌ها صورت گرفت. به تصاویر داخلی برچسب صفر و به تصاویر خارجی برچسب یک زده شد. داده‌های شافل شده به داده‌های اموزش(۷۰٪)و ارزیابی(۲۰٪) و تست (۱۰٪) تقسیم شدند.

سوال سوم

ابتدا شبکه‌ی از پیش آموزش داده‌شده را بارگذاری می‌کنیم. و سپس تلاش می‌کنیم از این شبکه برای رسیدن به هدف مساله‌ی خود استفاده کنیم.

در این تمرین مساله‌ی ما دسته‌بندی دوکلاسه است و شبکه‌ی از پیش آموزش داده‌شده برای مساله‌ی دسته‌بندی ۱۰۰۰ کلاس است. یعنی ما باید لایه‌ی آخر این شبکه را کنار گذاشته و یک لایه‌ی دسته‌بند مناسب با داده‌های خود آموزش دهیم. در حقیقت در این مرحله از استخراج ویژگی این شبکه استفاده می‌کنیم و دسته‌بند را به صورت اختصاصی تنظیم می‌کنیم.

برای اینکه این شبکه بیشتر مطابق با داده‌های خودمان باشد می‌توانیم وزن لایه‌های آخر را بیشتر امورش دهیم و این آموزش بیشتر بر روی داده‌های مساله‌ی خودمان باشد.

بنابراین می‌توانیم یادگیری انتقالی را در دوسطح ذکر شده انجام دهیم. (البته حالت دیگری نیز وجود دارد که هیچ تغییری در شبکه‌ی از پیش آموزش داده‌شده نمی‌دهیم و کاملا از همان استفاده می‌کنیم. ولی در این تمرین نمی‌خواهیم این کار را انجام دهیم.)

این فرایند زمانی استفاده می‌شود که تسک ما بسیار شبیه به تسکی باشد که قبلا بر روی مجموعه داده‌ای بزرگ انجام شده است. و برای مساله‌مان داده‌های بسیار زیادی نداریم که بخواهیم خودمان از اول یک شبکه‌ی خوب آموزش دهیم. در این حالت از یادگیری انتقالی استفاده می‌کنیم و با داده‌های کمی که داریم شبکه‌ای که وجود داشته است را به سمت تسک خود نزدیک می‌کنیم.(با تنظیم دقیق.)

برای تنظیم دقیق در این تمرین اول بهترین معماری برای لایه‌ی دسته‌بند را پیدا می‌کنیم و سپس با استفاده از آن معماری بهینه تعدادی از لایه‌های آخر را بیشتر آموزش می‌دهیم در حالی که وزن لایه‌ی اول تغییر نمی‌کند. (در این تمرین ما تا ۵۰ لایه‌ی آخر را بررسی می‌کنیم.)

نکته: در مرحله‌ی تنظیم دقیق نرخ یادگیری باید خیلی کم باشد تا مدل بتواند وزن‌های بهینه را پیدا کند. زیرا در حالت فعلی نزدیک به نقطه‌ی بهینه است و با اندکی گام کوچک به نقطه‌ی هدف می‌رسد.

سوال چهارم

استفاده از شبکه‌ی دنس‌نت ۲۰۱:

قسمت اول آزمایش‌ها: پیدا کردن معماری بهینه برای دسته‌بندی (لایه‌ی آخر)

تعداد ایپاک: ۱۰۰

تعداد ویژگی‌ها در لایه‌ی آخر دنس‌نت : ۳۰۷۲۰

در صورتی که از لایه‌ی دسته‌بند شبکه‌ی دنس‌نت ۲۰۱ استفاده نکنیم و از معماری‌های زیر استفاده کنیم به دقت‌های ذکرشده می‌رسیم.

```
flat1 = Flatten()(densenet201)
dense1 = Dense(units=256, use_bias=True)(flat1)
out = Dense(units=1, activation='sigmoid')(dense1)
```

دقت در داده‌های آموزش: ۱ دقت در داده‌های ارزیابی : ۰.۹۴ دقت در داده‌های آزمون: ۰.۹۰

```
flat1 = Flatten()(densenet201)
dense1 = Dense(units=1024, use_bias=True)(flat1)
dense2 = Dense(units=256,activation='relu')(dense1)
out = Dense(units=1, activation='sigmoid')(dense2)
```

دقت در داده‌های آموزش:۰.۹۹ دقت در داده‌های ارزیابی : ۰.۸۹ دقت در داده‌های آزمون: ۰.۸۳

```
flat1 = Flatten()(densenet201)
dense1 = Dense(units=512, use_bias=True)(flat1)
dense2 = Dense(units=128,activation='sigmoid')(dense1)
out = Dense(units=1, activation='sigmoid')(dense2)
```

دقت در داده‌های آموزش:۱ دقت در داده‌های ارزیابی: ۰.۹۶ دقت در داده‌های آزمون: ۰.۹۳

```
flat1 = Flatten()(densenet201)
dense1 = Dense(units=512, use_bias=True)(flat1)
out = Dense(units=1, activation='sigmoid')(dense1)
```

دقت در داده‌های آموزش: ۱ دقت در داده‌های ارزیابی : ۰.۸۹ دقت در داده‌های آزمون: ۰.۸۶

در ادامه میخواهیم یک لایه‌ی bachNormalization به لایه‌ی اول اضافه کنیم.

```
flat1 = Flatten()(densenet201)
dense1 = Dense(units=256, use_bias=True)(flat1)
batchnorm1 = BatchNormalization()(dense1)
```

```
out = Dense(units=1, activation='sigmoid')(batchnorm1)
```

دقت در داده‌های آموزش: ۱ دقت در داده‌های ارزیابی : ۰.۹۷ دقت در داده‌های آزمون: ۰.۹۶

این آزمایش آخر دقت بهتری نسبت به آزمایش‌های قبلی داشت. همین معماری برای لایه‌ی دسته‌بندی درنظر گرفته می‌شود. در ادامه می‌خواهییم وزن‌های لایه‌های آخر شبکه را بهبود دهیم.

قسمت دوم آزمایش‌ها: fine tune

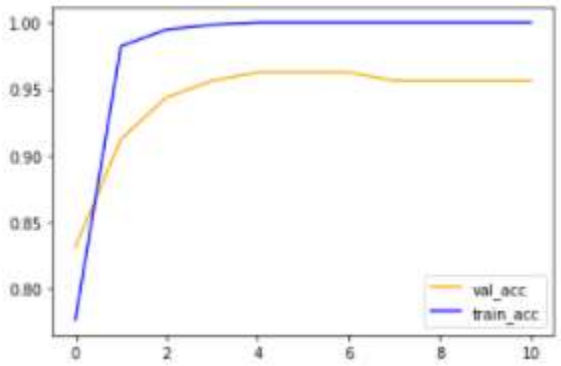
مشخصات آزمایش:

استفاده از کال‌بک early_stopping با patience: 5

تعداد ایپاک: ۱۰۰ نرخ یادگیری: ۰.۰۰۰۰۱

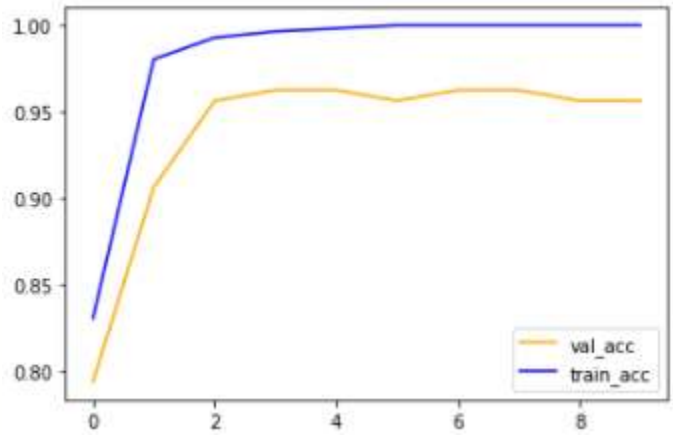
```
for layer in model.layers[0:190]:
   layer.trainable = False

for layer in model.layers[190:-1]:
    layer.trainable = True
```

```
train_acc: 1          val_acc: 0.9625     test_acc: 0.9375
```



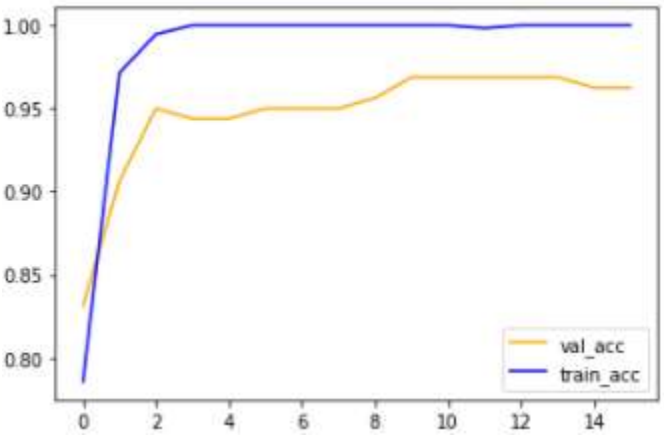
```
for layer in model.layers[0:180]:
   layer.trainable = False

for layer in model.layers[180:]:
    layer.trainable = True
learning_rate: 0.00001
```

```
train_acc: 1          val_acc: 0.9625     test_acc: 0.8875
```



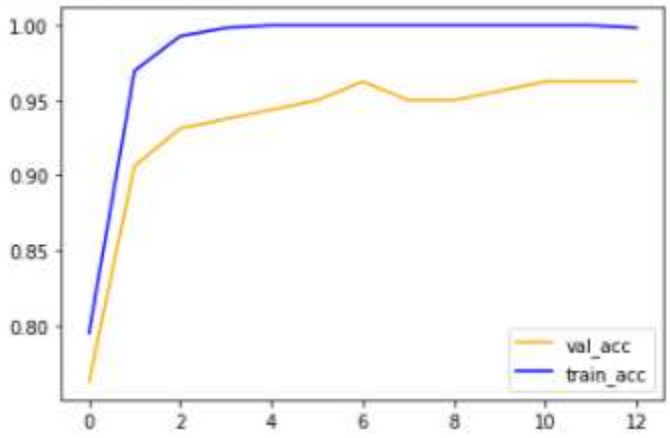
```
for layer in model.layers[0:170]:
   layer.trainable = False

for layer in model.layers[170:]:
    layer.trainable = True
learning_rate: 0.00001
```

```
train_acc: 1        val_acc: 0.9688     test_acc: 0.8875
```

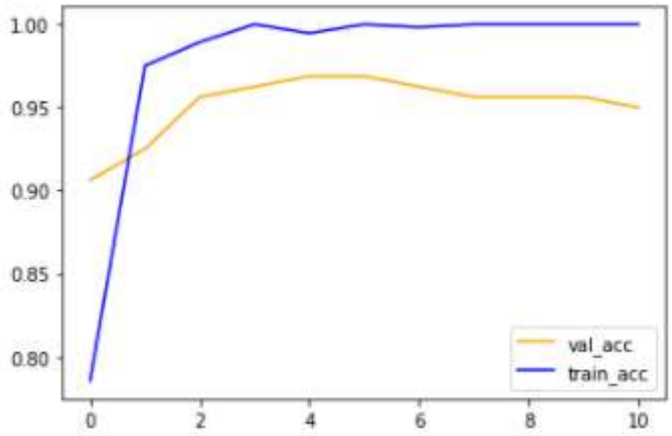


```
for layer in model.layers[0:160]:
   layer.trainable = False

for layer in model.layers[160:]:
    layer.trainable = True
learning_rate: 0.00001
```

```
train_acc: 1        val_acc: 0.9625     test_acc: 0.9000
```



```
for layer in model.layers[0:150]:
   layer.trainable = False

for layer in model.layers[150:]:
    layer.trainable = True
learning_rate: 0.00001

train_acc: 0.9998        val_acc: 0.9625     test_acc: 0.9000
```



نتیجه: همانطور که از آزمایش‌های بالا مشاهده شد در حالتی که بخواهیم تعدادی از لایه‌های شبکه را تنظیم دقیق کنیم دقت کمتر می‌شود. به همین دلیل معماری انتخابی : ثابت بودن وزن‌های شبکه‌ی پایه + آموزش لایه‌ی دسته‌بند به صورت زیر :

(در این حالت دقت داده‌های ارزیابی نسبت به دیگر حالات بیشتر شده است.)

```
flat1 = Flatten()(densenet201)
dense1 = Dense(units=256, use_bias=True)(flat1)
batchnorm1 = BatchNormalization()(dense1)
out = Dense(units=1, activation='sigmoid')(batchnorm1)
```

دقت در داده‌های آموزش: ۱ دقت در داده‌های ارزیابی : ۰.۹۷ دقت در داده‌های آزمون: ۰.۹۱

ماتریس درهم‌ریختگی : $\begin{bmatrix} 46 & 5 \\ 2 & 27 \end{bmatrix}$

استفاده از شبکه‌ی رزنت ۱۵۲:

قسمت اول آزمایش‌ها: پیدا کردن معماری بهینه برای دسته‌بندی (لایه‌ی آخر)

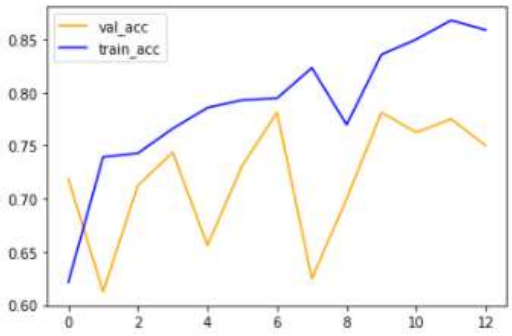
تعداد ویژگی‌ها در لایه‌ی آخر رزنت: ۳۲۷۶۸

استفاده از کال‌بک early_stopping با patience: 5

(در چند آزمایش متوالی مشاهده شد در حالتی که این لایه‌ها بدون توقف تا ۱۰۰ ایپاک یادگیری شوند نتیجه‌شان با حالتی که با توقف یاد بگیرند فرقی نمی‌کند. یعنی بعد از ۱۰۰ ایپاک هم به همین نتیجه میرسیدیم. بنابراین آزمایش‌های زیر را با توقف انجام می‌دهیم.)

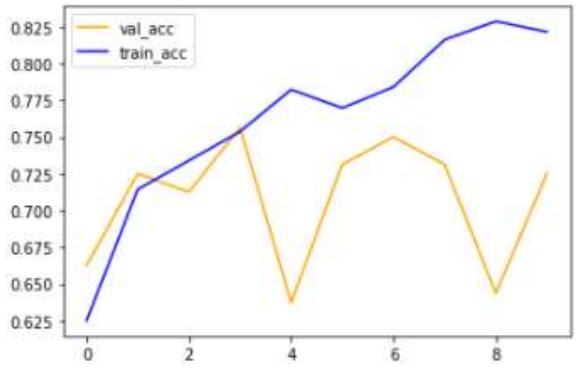
تعداد ایپاک: ۱۰۰ نرخ یادگیری: ۰.۰۰۰۱

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(5000,activation='relu'))
model.add(keras.layers.Dense(500,activation='relu'))
model.add(keras.layers.Dense(2))
```



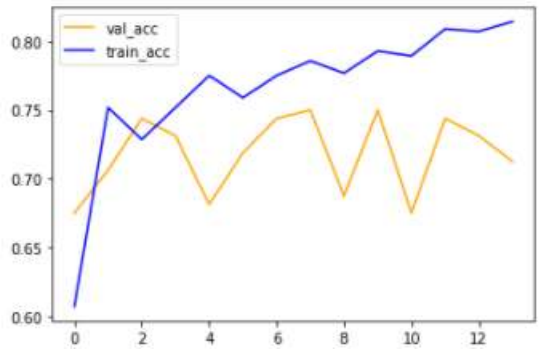
دقت در داده‌های آموزش: ۰.۸۵ دقت در داده‌های ارزیابی : ۰.۷۵

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(1000,activation='relu'))
model.add(keras.layers.Dense(256,activation='relu'))
model.add(keras.layers.Dense(2))
```



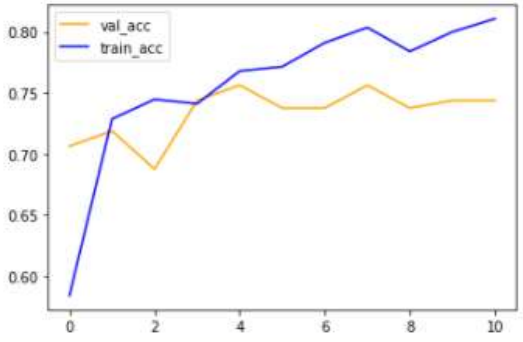
دقت در داده‌های آموزش: ۰.۸۲ دقت در داده‌های ارزیابی : ۰.۷۵

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(256,activation='relu'))
model.add(keras.layers.Dense(2))
```



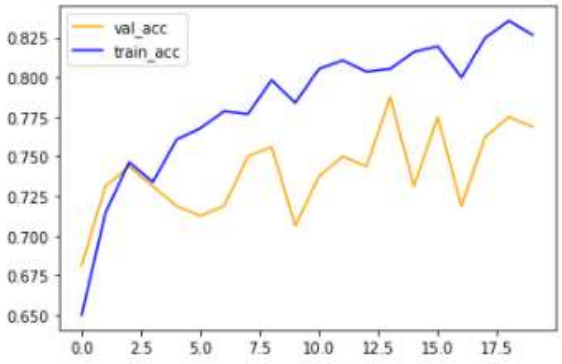
دقت در داده‌های آموزش: ۰.۷۸ دقت در داده‌های ارزیابی : ۰.۷۵

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512,activation='relu'))
model.add(keras.layers.Dense(2))
```



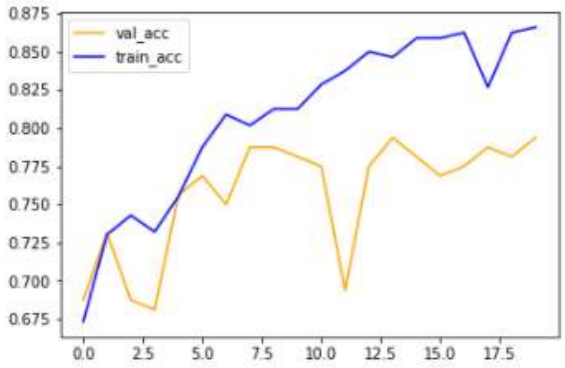
دقت در داده‌های آموزش: ۰.۷۶ دقت در داده‌های ارزیابی : ۰.۷۵

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512,activation='relu'))
model.add(keras.layers.Dense(64,activation='relu'))
model.add(keras.layers.Dense(2))
```



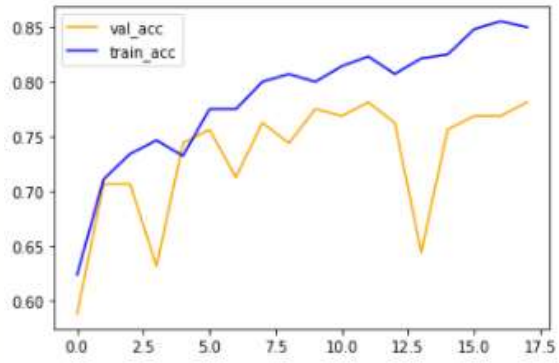
دقت در داده‌های آموزش:۰.۸۰ دقت در داده‌های ارزیابی : ۰.۷۸

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(1024,activation='relu'))
model.add(keras.layers.Dense(64,activation='relu'))
model.add(keras.layers.Dense(2))
```



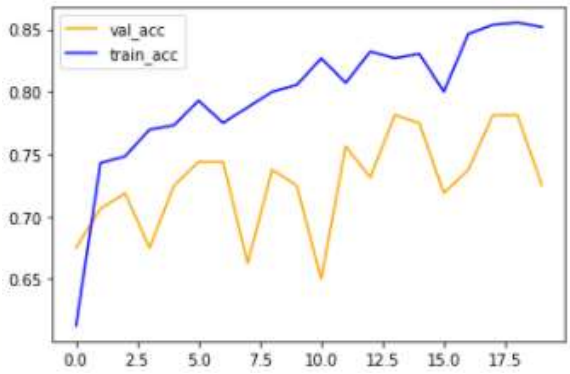
دقت در داده‌های آموزش: ۰.۸۴ دقت در داده‌های ارزیابی : ۰.۷۹

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(1024,activation='relu'))
model.add(keras.layers.Dense(256,activation='relu'))
model.add(keras.layers.Dense(32,activation='relu'))
model.add(keras.layers.Dense(2))
```



دقت در داده‌های آموزش: ۰.۸۲ دقت در داده‌های ارزیابی : ۰.۷۸

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(1024,activation='relu'))
model.add(keras.layers.Dense(2))
```



دقت در داده‌های آموزش: ۰.۸۲ دقت در داده‌های ارزیابی : ۰.۷۸

بهترین دقت بر روی داده‌های ارزیابی ۰.۷۹ بود که از معماری زیر بدست آمد :

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(1024,activation='relu'))
model.add(keras.layers.Dense(64,activation='relu'))
model.add(keras.layers.Dense(2))
```

در ادامه با ثابت نگه داشتن این معماری برای لایه‌ی دسته‌بند سعی می‌کنیم بر روی لایه‌های آخر شبکه‌ی رزنت تنظیم دقیق انجام دهیم.

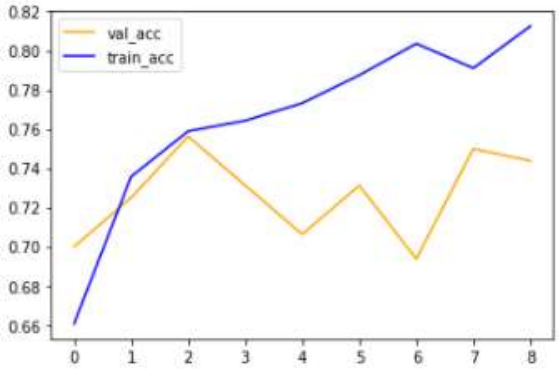
قسمت دوم آزمایش‌ها: fine tune

استفاده از کال‌بک early_stopping با patience: 5

تعداد ایپاک: ۱۰۰ نرخ یادگیری: ۰.۰۰۰۱

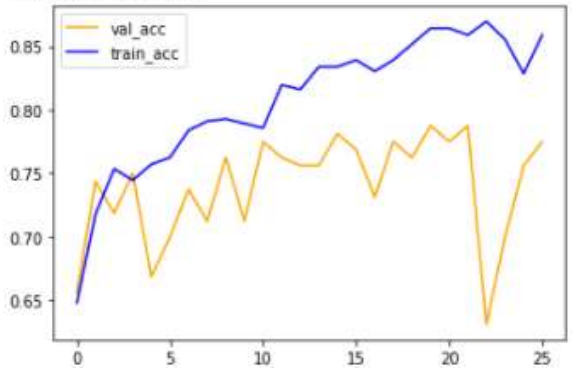
```
for layer in model1.layers[0:-10]:
   layer.trainable = False

for layer in model1.layers[-10:-1]:
    layer.trainable = True
```

train_acc: 0.75 val_acc: 0.75

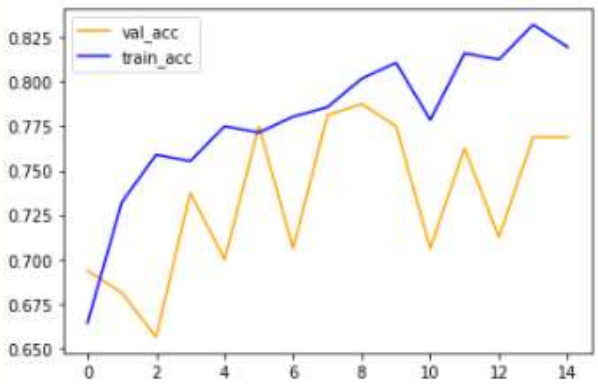


```
for layer in model1.layers[0:-20]:
   layer.trainable = False

for layer in model1.layers[-20:-1]:
    layer.trainable = True
```

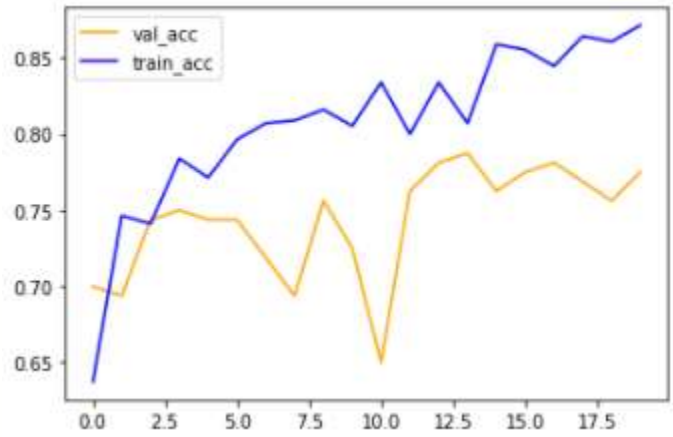
train_acc:0.86 val_acc: 0.78



```
for layer in model1.layers[0:-30]:
   layer.trainable = False

for layer in model1.layers[-30:-1]:
    layer.trainable = True
train_acc: 0.80  val_acc: 0.78
```

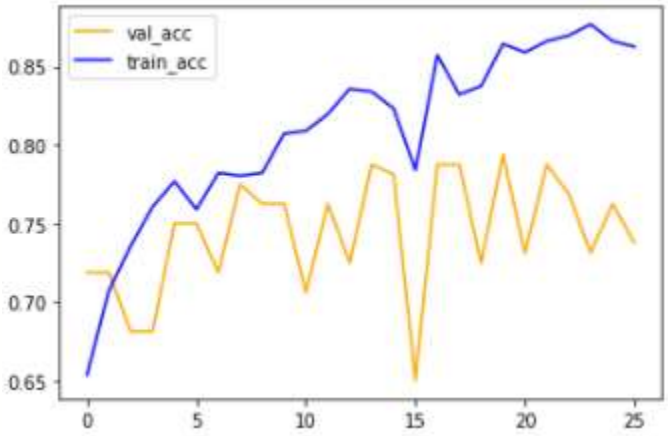


```
for layer in model1.layers[0:-40]:
   layer.trainable = False

for layer in model1.layers[-40:-1]:
    layer.trainable = True

train_acc: 0.80  val_acc: 0.78
```



```
for layer in model1.layers[0:-50]:
   layer.trainable = False

for layer in model1.layers[-50:-1]:
    layer.trainable = True
```

train_acc: 0.86 val_acc: 0.79



نتیجه: همانطور که از آزمایش‌های بالا مشاهده شد بهترین حالت برای لایه‌ی دسته‌بند و تنظیم دقیق لایه‌های آخر شبکه برابر با حالتی بود که تنظیم دقیق انجام نمی‌دادیم. بنابراین بهترین مدل به صورت زیر است: (در این حالت دقت داده‌های ارزیابی نسبت به دیگر حالات بیشتر شده است.)

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(1024,activation='relu'))
```

```
model.add(keras.layers.Dense(64,activation='relu'))
model.add(keras.layers.Dense(2))
```

دقت در داده‌های آموزش: ۰.۸۶ دقت در داده‌های ارزیابی : ۰.۷۹ دقت در داده‌های آزمون: ۰.۷۵

ماتریس درهم‌ریختگی : $\begin{bmatrix} 40 & 11 \\ 9 & 20 \end{bmatrix}$

تحلیل

از مقایسه‌ی نتایج دو شبکه می‌بینیم دقت در شبکه‌ی دنس‌نت بیشتر از رزنت است. دلیل این نتیجه ۲ مورد زیر است:

۱. همانطور که می‌دانستیم هر لایه در شبکه‌ی دنس‌نت از ویژگی‌ها و اطلاعات تمام لایه‌های قبل از خودش استفاده می‌کرد در حالی که در رزنت فقط از ورودی یک لایه‌ی قبل از خودش استفاده می‌کرد. این یعنی در دنس‌نت ویژگی‌های بیشتری برای تصمیم‌گیری و ساخت مدل وجود دارد --> دقت بیشتری در دسته‌بند نهایی خواهیم گرفت.

۲. در این تمرین از شبکه‌ی دنس‌نت با ۲۰۱ لایه و شبکه‌ی رزنت با ۱۵۲ لایه استفاده کردیم. همانطور که میدانیم هرچقدر تعداد لایه‌ها بیشتر باشد ویژگی‌های بیشتری استخراج می‌شود. بنابراین این نکته نیز در بهتر بودن جواب دنس‌نت نسبت به رزنت موثر بوده است.

یک مشاهده‌ی دیگر این آزمایش‌ها: دقت در حالتی که تنظیم دقیق انجام می‌دادیم یا تغییر نمی‌کرد یا کاهش می‌یافت. دلیل این امر این است که این دو شبکه خود بر روی مجموعه داده‌های بسیار بزرگ‌تر و تسک جامع‌تری آموزش دیده‌اند در نتیجه با تنظیم دقیق تنها بیش‌برازش رخ می‌دهد و این باعث می‌شود که یا دقت بیشتر نشود یا کاهش یابد.