بسمه تعالی

دانشکده مهندسی کامپیوتر

یادگیری عمیق

نام استاد: دکتر محمدی

تمرین چهاردهم

آرمان حیدری

شماره دانشجویی: ۹۷۵۲۱۲۵۲

دی ۱۴۰۰

فهرست

پاسخ سوال اول

سوال ۱)

در مجموع عملکرد قابل قبولی دارد و از ۱۰ کلمه ای که پیشنهاد میدهد، حداقل ۸ مورد واقعا از لحاظ معنا به کلمه اصلی نزدیک هستند.

برای دو کلمه اول، شبیه ترین کلمه، جمع آن ها بوده (فقط s در پایان کلمه آمده) که زیاد جالب نیست.

میزان شباهت کلمه ها و همچنین ارتباط آن ها با کلمه ’hardy‘ که انتظار میرفت بازیگر مشهور یعنی tom hardy را بیابد بسیار کم است و زیاد نتایج جالبی نیست. البته در سایر موارد و به خصوص فعل ’breaking‘ خوب بوده است.

برخی خطاها مانند شبیه تر بودن spain به italy، نسبت به شبیه بودن intalians به Italy به نظر اشتباه می آید. یا شبیه بودن ’she‘ که یک ضمیر است به کلمه ’mother‘ احتمالا چون معمولا در کنار هم آمده اند هم از لحاظ معنایی جالب نیست. هرچند بقیه کلمات نزدیک به mother خیلی خوب پیشنهاد شده اند.

سوال ۲)

درواقع اعدادی که خروجی میدهد میزان فاصله بین کلمه اصلی با این کلمات است. که میبینیم به درستی فاصله بین کلمه دور و کلمه اصلی همواره بیشتر از فاصله بین کلمه نزدیک و کلمه اصلی بوده است. البته بعضی موارد مانند Zidane و real، فاصله بیشتر از حدی که باید به دست آمده است. اما در سایر موارد به طور فاحشی اختلاف دارند که این اتفاق خوبی است.

سوال ۳)

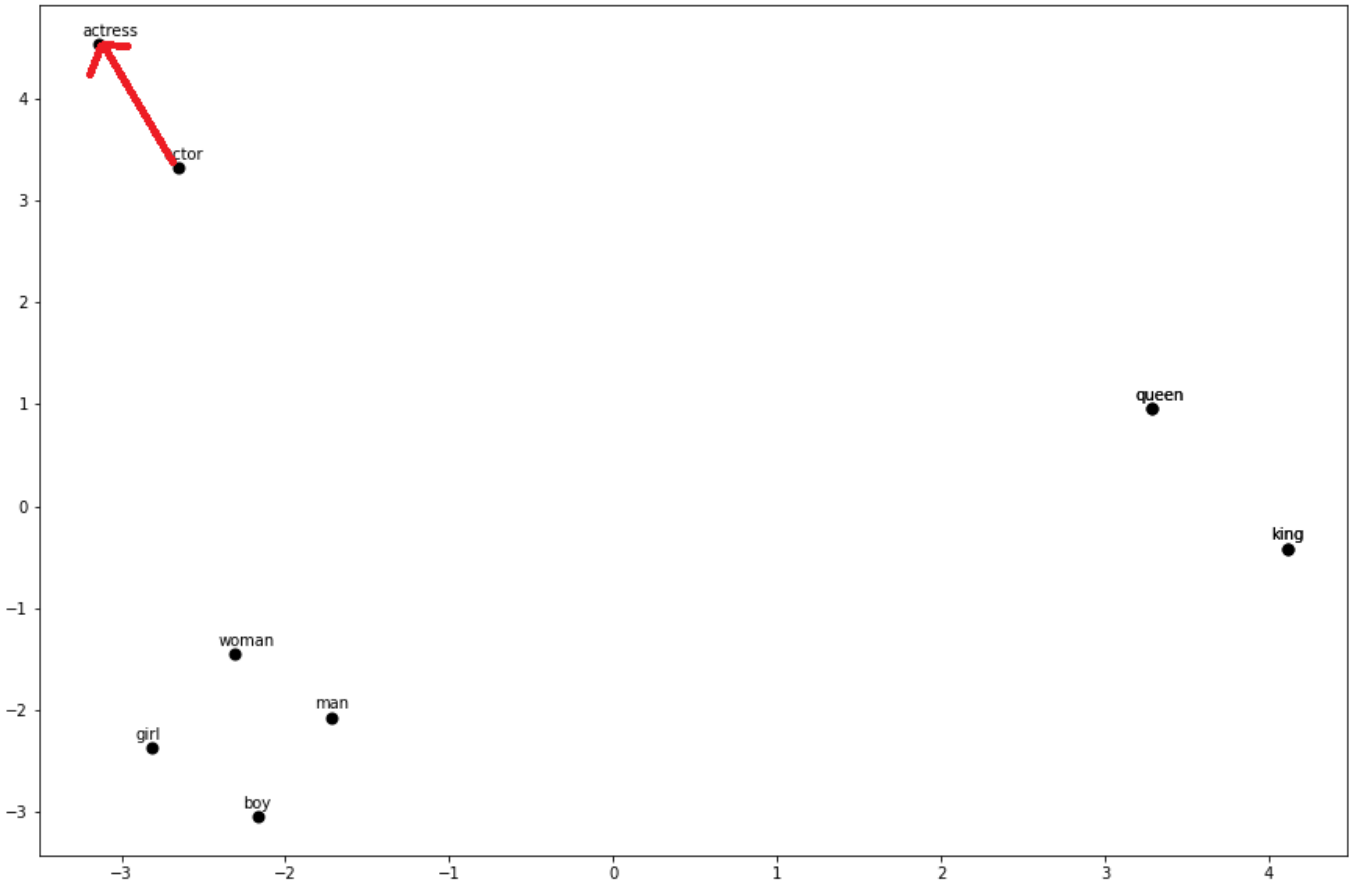
روش اول مناسبتر است. تفاوت این دو روش درواقع در برداری است که با w1 جمع میشود تا کلمه حاصل را بسازد. طبق صورت سوال، ما میخواهیم w2-w3 تقریبا معادل w4-w1 باشد. و در این قسمت الگوریتم:

```
most_similar(positive=[w1, w2], negative=[w3]),
.most_similar(positive=[w1, w3], negative=[w2]),
```

میبینیم که خط اول فرمول w4=w1+w2-w3 که ما میخواهیم را implement کرده است. البته از نتایج هم مشخص است که روش اول بهتر بوده و به جواب های عالی queen،actress، housewife رسیده است.

سوال ۴)

به خروجی کد، بردار قرمزرنگ را دستی اضافه کرده ام:

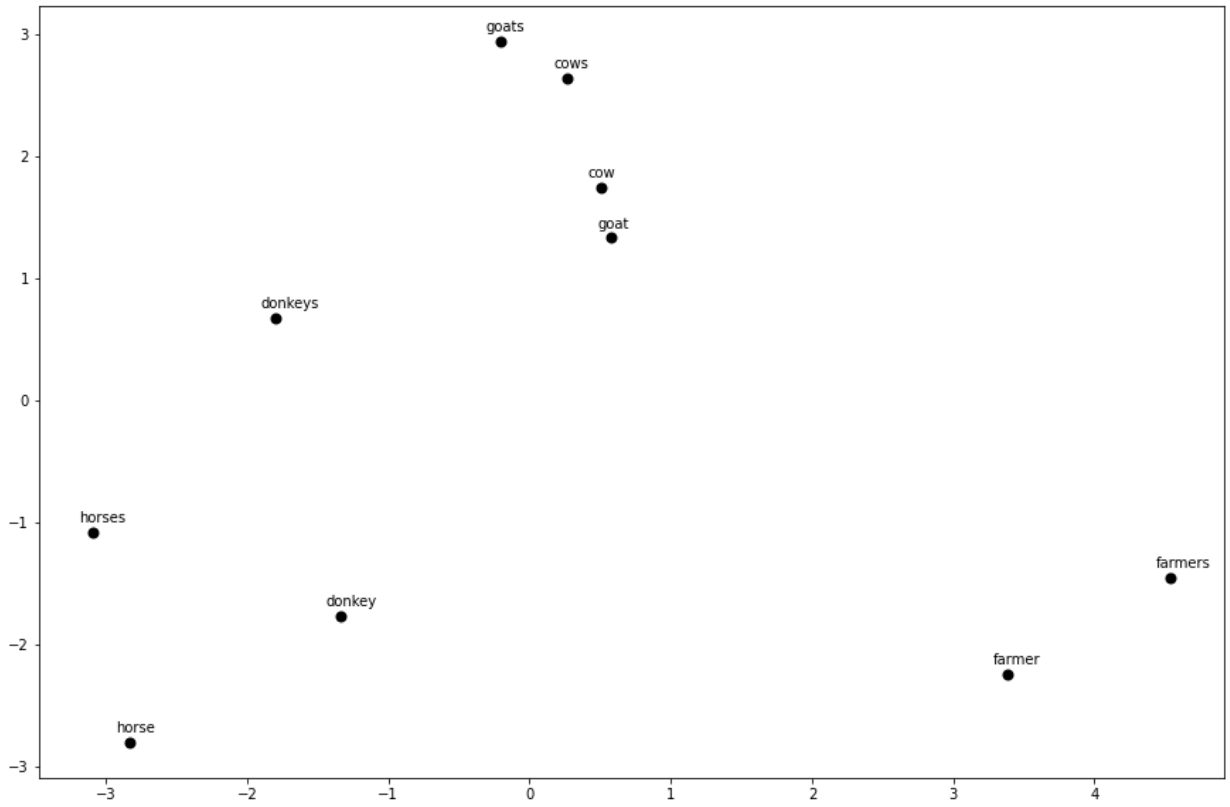


این بردار که فاصله بین دو کلمه actress – actor را نشان می دهد، به نوعی برای مونث کردن هر کلمه ای میتواند استفاده شود. تقریبا با برداری موازی همین میتوانیم از king به queen و از man به woman و حتی از boy به girl برسیم. این نشاندهنده خوب بودن فضای word embedding ماست که ویژگی های کلمات به خوبی با معنای آن ها مرتبط هستند.

حتی با برداری که از man به boy میرویم، میتوانیم از woman به girl برسیم و این عالی است.

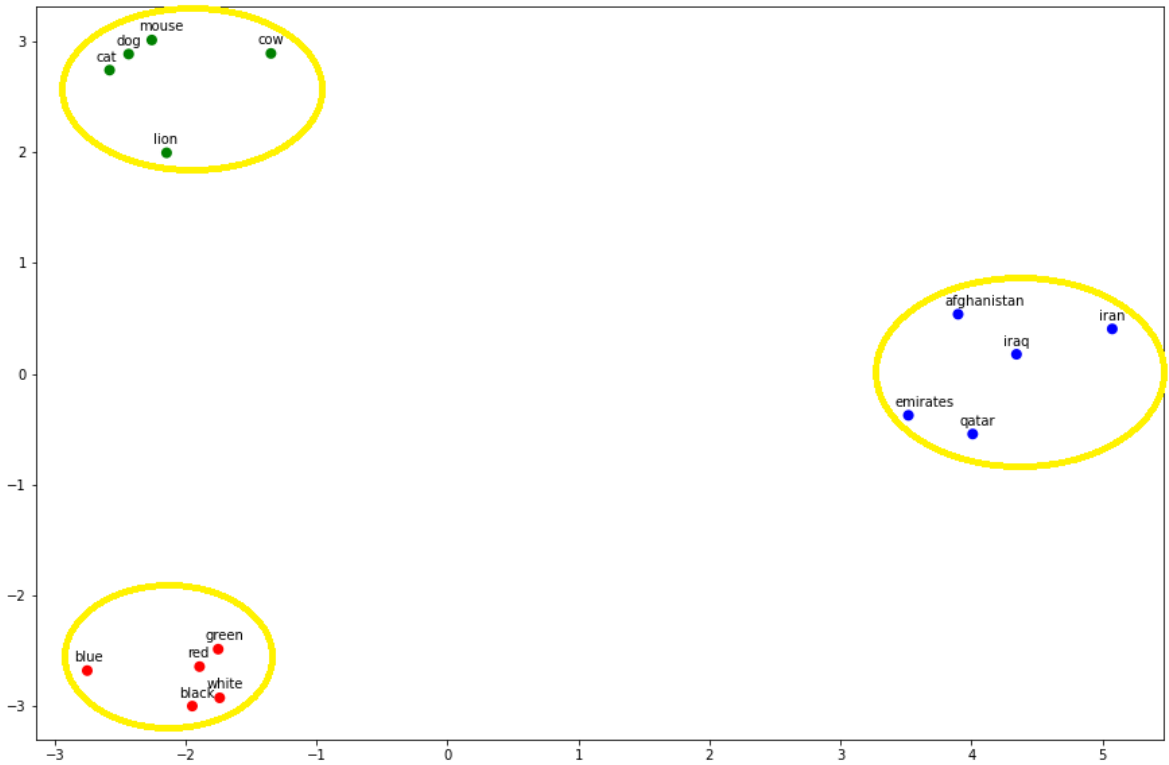
سوال ۵)

بله دقیقا الگویی مشابه قبل میبینیم. این بار به جای مونث کردن، درواقع بردار جمع کردن کلمات را میبینیم که بین ۴ جفت این موارد مشابه است. در واقع نشان میدهد که این embedding، ویژگی مفرد و جمع بودن لغات را هم تقریبا خوب یاد گرفته است. فقط خروجی برای farm که کمی متفاوت از سایر کلمات است، فرق جدی میکند :



سوال ۶)

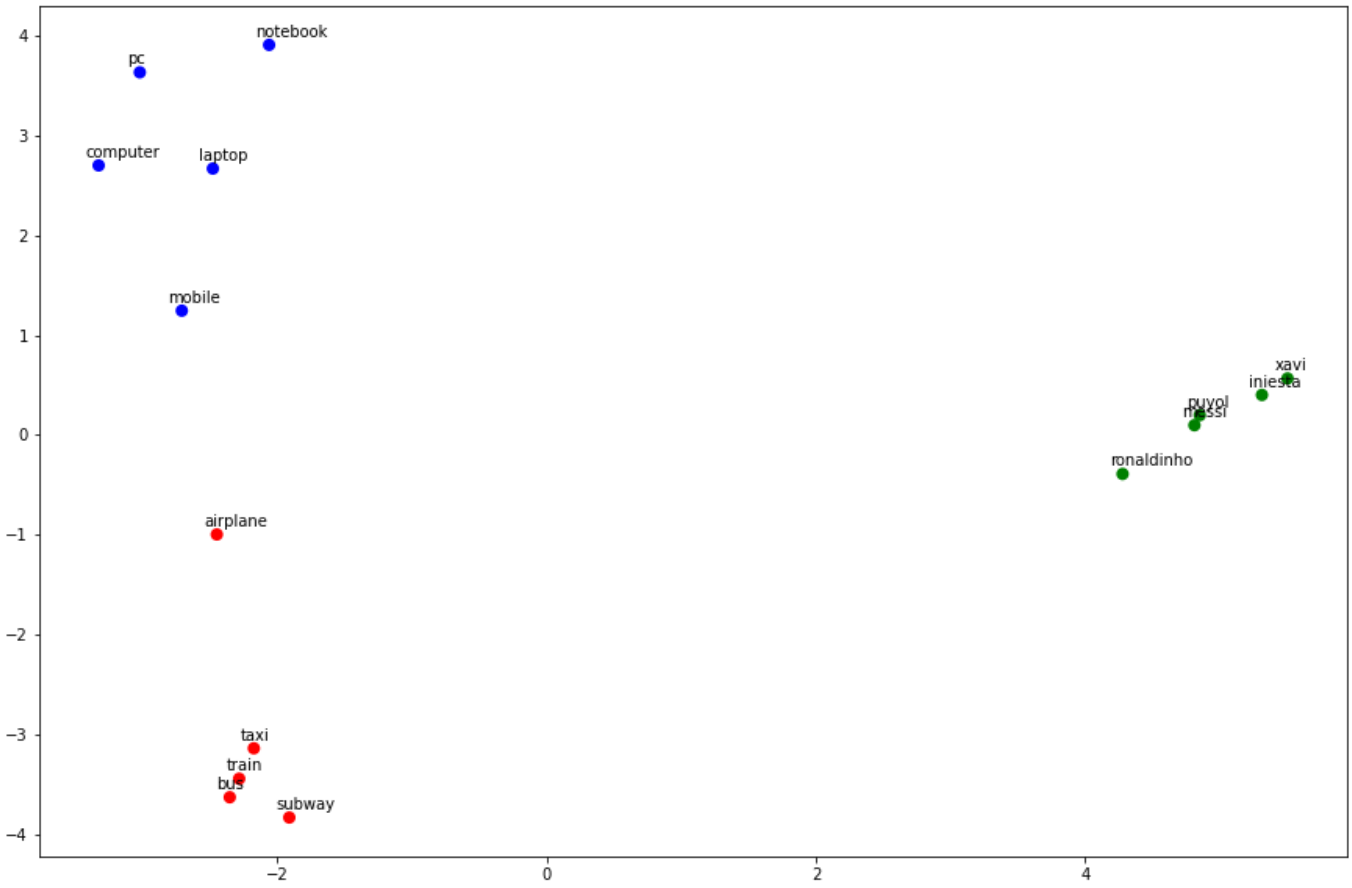
بله میبینیم که به راحتی خروجی، کلمات را دسته بندی کرده است: (دایره ها را دستی رسم کرده ام)



نشان میدهد که در ای embedding، کلماتی که از یک دسته بندی هستند نزدیک هم قرار میگیرند و ویژگی ها نشاندهنده حدود معنایی کلمات هستند.

سوال ۷)

بله دقیقا مانند حالت قبل، به خوبی گروه های مختلف را جدا کرده است:

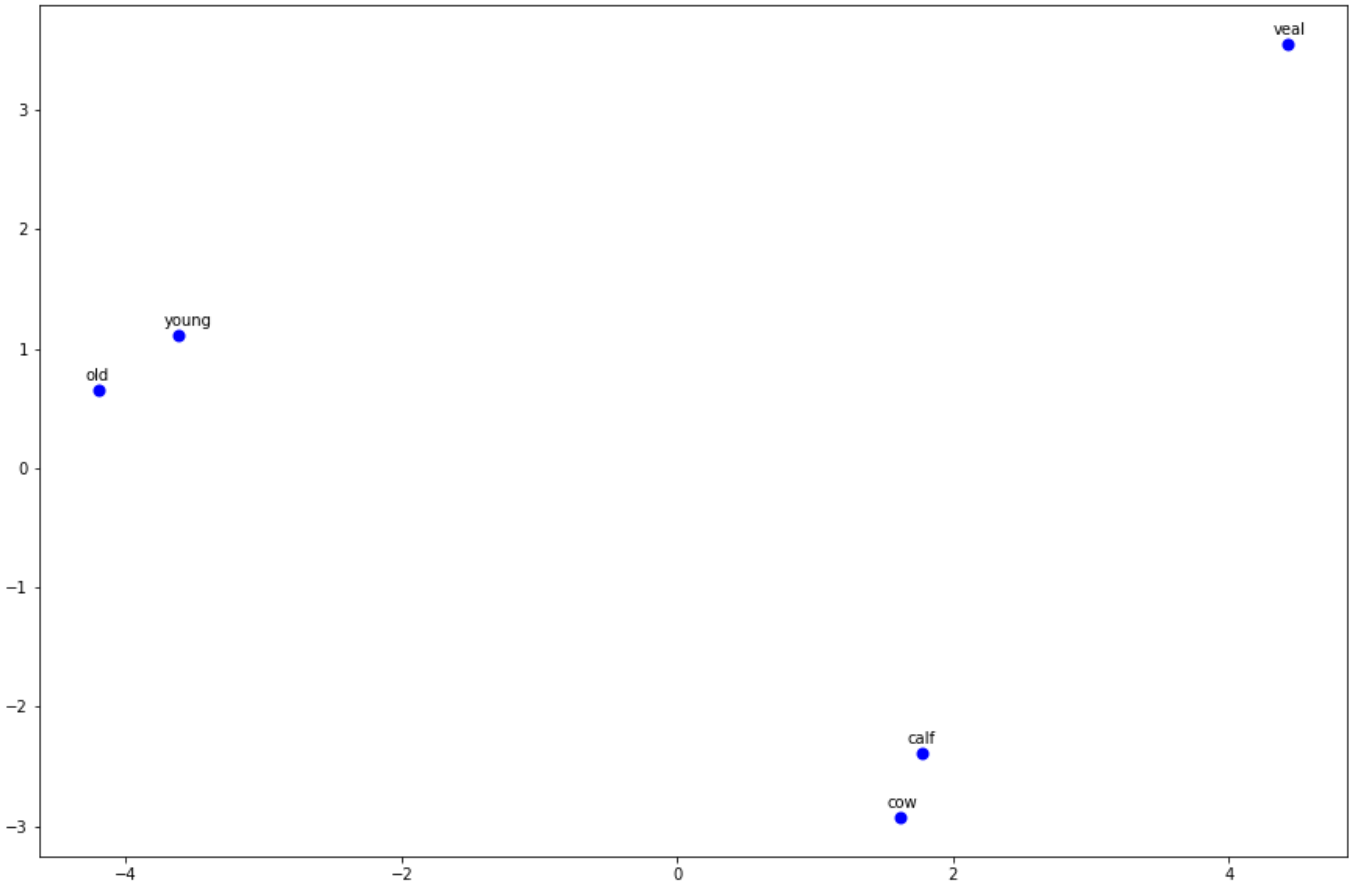


سوال ۸)

احتمالا باید کلمه ای معادل گاو جوان، مثلا calf یا veal را بدهد. اما خروجی عجیب است:

	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7
('young', 'cow', 'old')	(mad, 0.6)	(cows, 0.58)	(sheep, 0.52)	(pigs, 0.52)	(herd, 0.51)	(bovine, 0.5)	(cattle, 0.49)

برای تحلیل این موضوع، کلمات مربوطه را در این embedding با استفاده از دستورات قسمت های قبلی رسم میکنیم:



میبینیم که در این embedding، مفهوم گوساله به خوبی درک نشده است و بردار ویژگی خوبی برای آن ساخته نشده است. احتمالا در دیتاستی که این embedding را با توجه به آن pretrain کرده اند، (هر چقدر هم بزرگ) این کلمه خیلی کم تکرار شده است و به خوبی درک نشده است. احتمال دیگر هم این است که چون کلمه ی calf معانی مربوط به عضلات ماهیچه دارد، و کلمه veal هم معنی گوشت گوساله (خوراکی) دارد که کمی متفاوت از صرف گوساله است. میبینیم که بردار بین young و old کاملا از بردار بین cow و veal و تقریبا از بردار بین cow و calf به دور است. به همین خاطر است که آن ها را نمیابد و کلمه بی ربطی مثل mad را پیشنهاد میدهد.

پاسخ سوال دوم

ابتدا یک سلول برای normalization کردن جواب می افزاییم تا به نتایج بهتری برسیم. (تقسیم بر ۲۵۵ کردن ورودی)

تفاوت دیگری هم که در بخش preprocess انجام داده ام، تغییر نوع ورودی X_train از یک آرایه به لیست بود. چون مطابق توضیحات تایع triplet loss، برای این که بتواند به عنوان تابع ضرر در model.fit موجود در کراس استفاده شود باید نوع ورودی به این صورت میشد. همچنین از حالت flat هم آن ها در آورده ام.

الف)

درواقع این تابع را به شکل یک لایه، در تابع triplet_loss محاسبه کرده ایم. و تابع ضرری که در مدل استفاده کرده ایم mean_triplet_loss است که میانگین ضرر ها را به عنوان loss در نظر میگیرد. (منبع)

منطق این تابع که خیلی ساده است و دقیقا مطابق فرمول زیر از اسلایدهای درس پیاده سازی شده است:

$$\mathcal{L}(A,P,N) = \max(d(A,P) - d(A,N) + \alpha, 0)$$

نکته مهم این است که تمام مراحل آن را با استفاده از توابع آماده ی keras.backend پیاده سازی کرده ایم، چون گرادیان آن ها را آماده دارد و وقتی این تابع را به عنوان loss در model.fit قرار دهیم، به راحتی با استفاده از optimizer که در اینجا adam است، میتواند وزن ها را طبق گرادیان آپدیت کند.

Tripplet Loss

```
def triplet_loss(y_pred, alpha=0.4):
    anch, pos, neg = y_pred
    return K.maximum(K.sum(K.square(anch-pos),axis=1) - K.sum(K.square(anch-neg),axis=1) + alpha, 0)
```

```
#added cell
def mean_triplet_loss(y_true, y_pred):
    return K.mean(y_pred)
```

ب)

شبکه نیاز نیست زیاد سنگین باشد. در ابتدا از mobilenet-v2 استفاده کردم اما زمان آموزش طولانی بود و دیتاست mnist هم ساده است و نیاز به چنین شبکه عمیقی ندارد. صرفا شبکه ای با سه لایه کانولوشنی برای

استخراج ۳۲ ویژگی، به همراه یک لایه maxpooling و دو لایه dense که یکی با ۱۲۸ واحد با تابع فعالسازی relu به عنوان لایه مخفی و یکی با ۱۰ نورون به عنوان لایه خروجی در نظر میگیریم. که البته بین conv2d و dense ها باید از flatten استفاده کنیم.

```
def embedding_pred_net(dim = (28,28,1)):
    return Sequential([
                        Input(dim),
                        Conv2D(8, (3,3), activation='relu', padding='same'),
                        Conv2D(16, (3,3), activation='relu'),
                        MaxPooling2D(),
                        Conv2D(32, (3,3), activation='relu'),
                        Flatten(),
                        Dense(128, activation='relu'),
                        Dense(10)
    ])
```

پ)

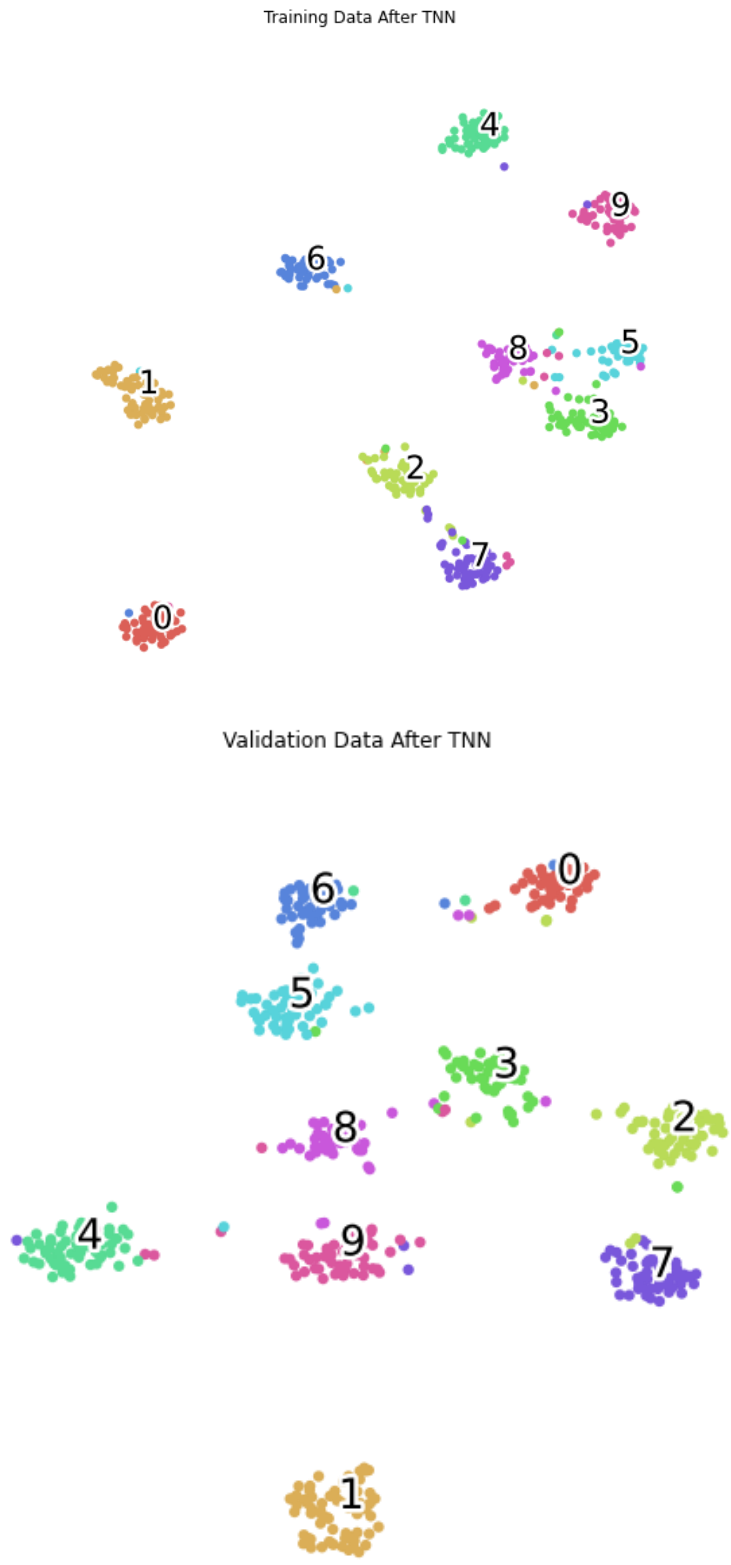
ابتدا تغییر کوچکی قبل از compile کردن model داده ایم، به این صورت که ابتدا ورودی ها را با استفاده از Lambda به تابع triplet_loss پاس داده ایم و سپس مراحل بعدی را انجام داده ایم. به این صورت مدل را compile و سپس fit کرده ایم:

```
model.compile(loss=mean_triplet_loss, optimizer=adam_optim)
model.fit(
    x= X_train,
    y= np.zeros(X_train[0].shape[0]),
    batch_size= 128,
    epochs= 5,
    validation_data = [X_test, np.zeros(X_test[0].shape[0])]
)
```

چون label ها هنگام استفاده از triplet loss بی استفاده هستند، اما در کراس باید حتما ورودی ای جایگزین آن ها شود، از آرایه ای تماما صفر به عنوان برچسب ها استفاده میکنیم. داده های تست هم به عنوان ارزیابی میدهیم تا نتایج را روی آن ها هم ببینیم و چک کنیم که overfit رخ نداده باشد. نتایج آموزش:

```
Epoch 1/5
1407/1407 [==============================] - 38s 21ms/step - loss: 0.0139 - val_loss: 0.0177
Epoch 2/5
1407/1407 [==============================] - 28s 20ms/step - loss: 4.3228e-04 - val_loss: 0.0240
Epoch 3/5
1407/1407 [==============================] - 29s 20ms/step - loss: 3.6115e-04 - val_loss: 0.0191
Epoch 4/5
1407/1407 [==============================] - 28s 20ms/step - loss: 2.2195e-04 - val_loss: 0.0172
Epoch 5/5
1407/1407 [==============================] - 28s 20ms/step - loss: 1.7126e-04 - val_loss: 0.0230
```

میبینیم که عدد loss خیلی کم است و هم روی داده آموزشی و هم تست کاهش خوبی داشته است. پس نیازی به آموزش بیشتر مدل نداریم. معیار accuracy در این نوع مدل جایی ندارد و با روند کاهشی و مقدار loss میتوان خوب عمل کردن مدل را سنجید. و البته در آخر با دیدن نمای گرافیکی تفکیک داده ها با استفاده از t-SNE از خوب کار کردن مدل اطمینان پیدا میکنیم:



بهینه انتخاب شدن triplet ها؟

خیر در این سوال محدودیت خاصی روی انتخاب کردن آن ها نگذاشته ایم و صرفا با ساختن تمام آن ها و در هر بار آموزش انتخاب 128 تا(batch_size) که به صورت stochastic انتخاب میشوند، مدل را بهینه کرده ایم. و با این روش احتمالا تعداد خوبی از نمونه ها همواره loss=0 دارند و کار خاصی روی مدل انجام نمیدهند.

در این سوال برای انتخاب بهینه باید مقادیر loss را برای ۳ تایی های مختلف محاسبه کنیم. سپس برای هر کدام از anchor ها، جایی که d(anchor, negative) برای آن کمترین است و d(anchor,positive) برای آن بیشترین است را انتخاب کنیم. یعنی برای هر anchor، آن منفی ای که اشتباها بیش از همه شبیه به anchor به نظر میرسد را با آن مثبتی که اشتباها شبیه به anchor تشخیص داده نمیشود بیابیم. و سپس آن را به مدل بدهیم تا بهینه شود. به این صورت بیشترین loss ها را بهینه میکنیم و وقت مدل برای loss های صفر، که پس از کمی آموزش تعدادشان بسیار زیاد است، تلف نمیشود.

پاسخ سوال سوم

الف)

خیر دقت زیاد معیار مناسبی برای این سوال نیست. چون در این مسئله اشتباه نکردن بسیار مهم است. مثلا وقتی میخواهیم سگ و گربه را تفکیک کنیم، احتمالا اشتباهات به اندازه درستی پیشبینی های مدل مهم هستند و دقت ۷۰ درصد دقت بدی نیست. اما هنگامی که مدلی کار پزشکی به این مهمی انجام می دهد و ممکن است مثلا ضایعاتی که به دلیل سرطان پوست است را به عنوان جوش ساده ی نوجوانی طبقه بندی کند! که این اشتباه مهلک است و در نتیجه ۳۰درصد خطا برای این مدل اصلا جالب نیست.

همچنین نا متوازن بودن دیتاها باعث میشود که اگر کلاسی با داده های زیاد را درست بگوید (مثل کلاس ۵)، دقت خیلی زیاد شود در حالی که در سایر موارد اشتباه کرده است.

ب)

این معیار ها را با استفاده از توابع آماده ی scikit learn به دست آورده ایم. خروجی روی داده های تمرینی:

Classification Report

	precision	recall	f1-score	support
class 0	0.78	0.90	0.83	228
class 1	0.80	0.87	0.84	359
class 2	0.87	0.71	0.78	769
class 3	0.85	0.66	0.75	80
class 4	0.82	0.75	0.78	779
class 5	0.93	0.96	0.95	4693
class 6	0.97	0.93	0.95	99
accuracy			0.90	7007
macro avg	0.86	0.83	0.84	7007
weighted avg	0.90	0.90	0.90	7007

که نسبتا اعداد قابل قبولی است. اما روی داده های ارزیابی و تست اینطور نیست:

Classification Report

	precision	recall	f1-score	support
class 0	0.29	0.44	0.35	66
class 1	0.36	0.38	0.37	103
class 2	0.48	0.35	0.40	220
class 3	0.00	0.00	0.00	23
class 4	0.37	0.39	0.38	223
class 5	0.85	0.87	0.86	1341
class 6	0.81	0.59	0.68	29
accuracy			0.71	2005
macro avg	0.45	0.43	0.43	2005
weighted avg	0.70	0.71	0.70	2005

Classification Report

	precision	recall	f1-score	support
class 0	0.19	0.30	0.24	33
class 1	0.41	0.50	0.45	52
class 2	0.41	0.32	0.36	110
class 3	0.20	0.08	0.12	12
class 4	0.35	0.37	0.36	111
class 5	0.84	0.84	0.84	671
class 6	0.64	0.50	0.56	14
accuracy			0.68	1003
macro avg	0.43	0.42	0.42	1003
weighted avg	0.68	0.68	0.68	1003

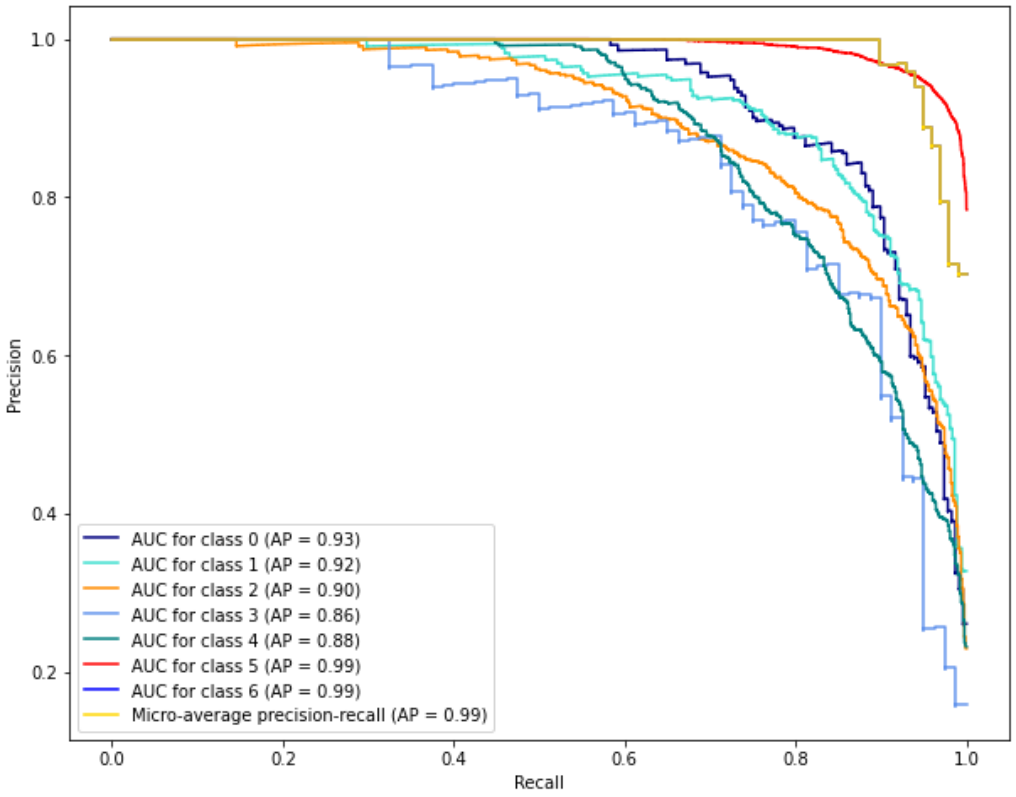
از این اعداد میفهمیم که دقتی که مدل به ما داده اصلا قابل اتکا نیست چون خیلی از آن به خاطر کلاس ۵ است که داده های زیادی از آن داریم. در حاللی که در کلاس های دیگر خطای زیادی صورت گرفته و برای این مسئله حساس جالب نیست.

پ)

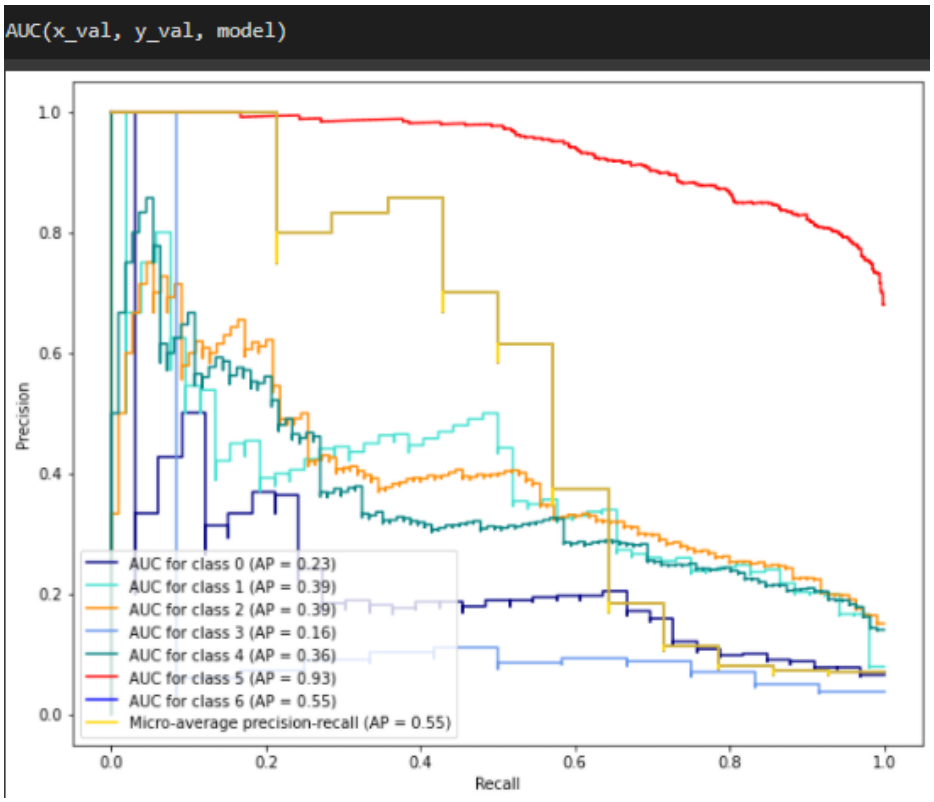
منحنی AUC - ROC یک اندازه گیری عملکرد برای مشکلات طبقه بندی در تنظیمات آستانه های مختلف است. ROC یک منحنی احتمال است و AUC نشان دهنده درجه یا معیار تفکیک پذیری است. این نشان می دهد که مدل چقدر می تواند بین کلاس ها تمایز قائل شود. هر چه AUC بالاتر باشد، مدل در پیش‌بینی کلاس‌های ۰ به‌عنوان ۰ و کلاس‌های ۱ به‌عنوان ۱ بهتر است. بر اساس قیاس، هرچه AUC بالاتر باشد، مدل در تشخیص کلاس ها بهتر عمل کرده است.

مثلا برای یک مسئله ی دو کلاسه، یک مدل عالی دارای AUC نزدیک به ۱ است که به این معنی است که معیار خوبی برای تفکیک پذیری دارد. یک مدل ضعیف دارای AUC نزدیک به ۰ است که به این معنی است که بدترین معیار تفکیک پذیری را دارد. در واقع به این معنی است که نتیجه را متقابل می کند. ۰ ها را به صورت ۱ و ۱ ها را به صورت ۰ پیش بینی می کند. و وقتی AUC 0.5 باشد، به این معنی است که مدل به هیچ وجه ظرفیت جداسازی کلاس را ندارد.

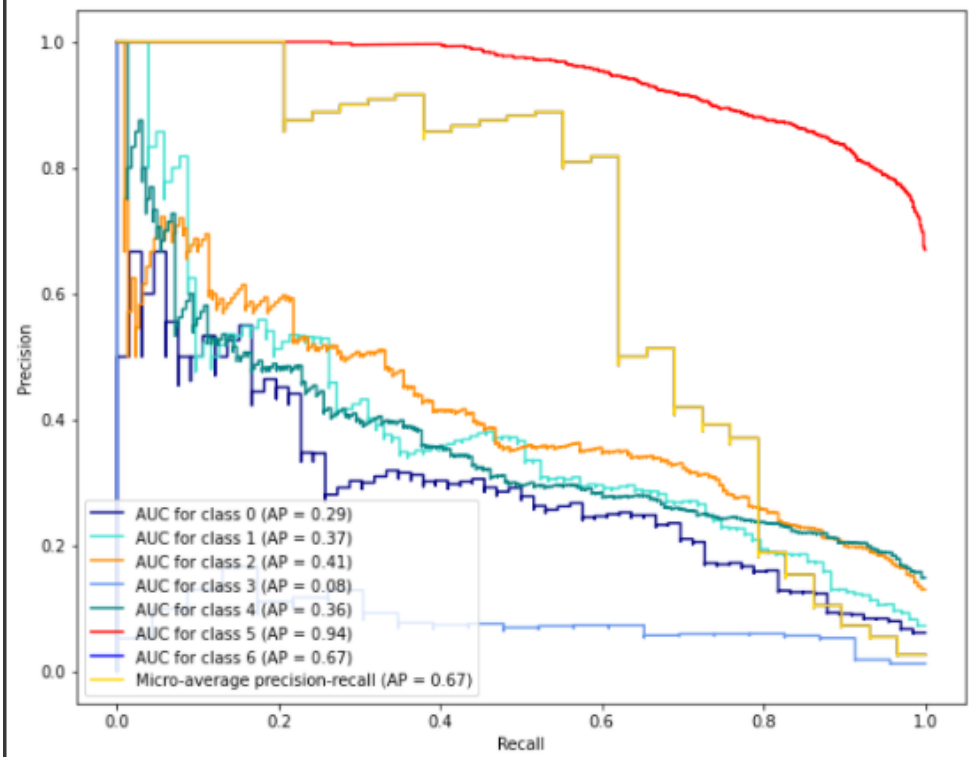
نمودارهای خواسته شده را برای داده های آموزشی به این صورت به دست آوردیم که طبق تعریف بالا میبینیم عملکرد مدل بد نبوده است:



ولی مطابق انتظار (از اعداد قسمت قبل) این عملکرد روی داده های ارزیابی و تست چندان خوب نبوده است:



```
AUC(x_test, y_test, model)
```



ت)

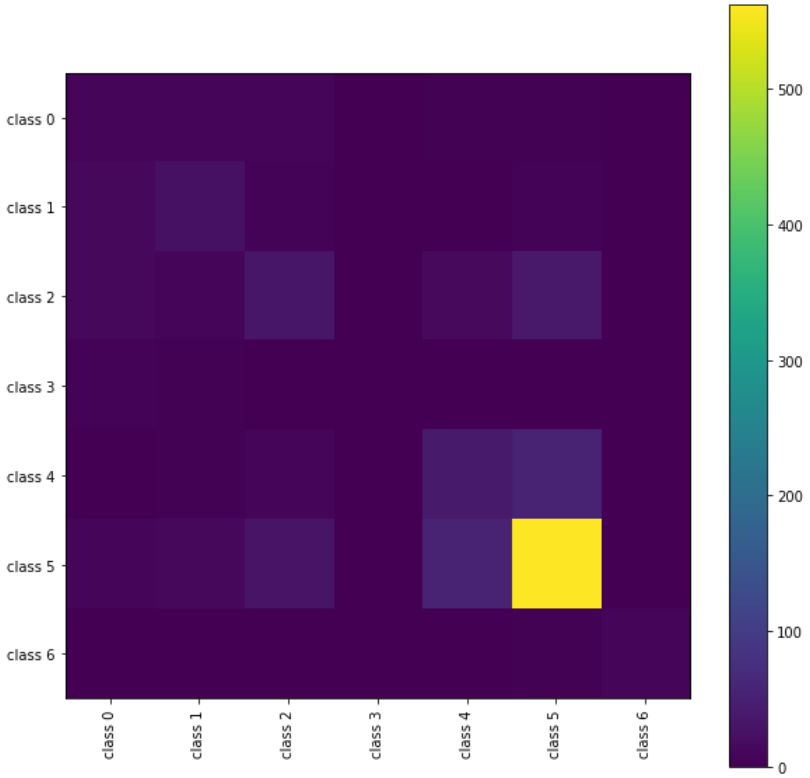
این جدول به خوبی توضیح این ۴ حالت را نشان می دهد:

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

در تجزیه و تحلیل پیش‌بینی‌کننده، confusion matrix جدولی مربعی شکل به طول ضلع تعداد کلاس هاست که تعداد مثبت‌های درست، منفی‌های کاذب، مثبت‌های کاذب و منفی‌های واقعی را گزارش می‌کند. این امکان تجزیه و تحلیل دقیق تری را نسبت به صرفاً مشاهده نسبت طبقه بندی صحیح (دقت) فراهم می کند. اگر مجموعه داده ها

نامتعادل باشد، دقت نتایج گمراه کننده ای به همراه خواهد داشت. یعنی زمانی که تعداد مشاهدات در طبقات مختلف بسیار متفاوت است. که در این سوال این چنین است و از نمودار های اولیه میتوانیم این موضوع را متوجه شویم.

برای داده های آموزشی و ارزیابی و تست، محتوای این ماتریس و شکل آن را در نوتبوک رسم کرده ام. تقریبا شکل همه آن ها به این صورت است:



از این ماتریس و مقادیرش میفهمیم که مدل اشتباه زیادی در تفکیک کلاس ۰ و کلاس ۱، کلاس ۰ و کلاس ۲، کلاس ۵ و کلاس ۶ داشته است و خطاهای زیادی در آن قسمت رخ داده اند. شاید با مدل بهتر یا داده های بهتر یا حتی داده های متوازن بتوان این مشکل را تا حد خوبی حل کرد. عملکرد مدل فعلی چندان عالی نیست ولی افتضاح هم نبوده و تا حدی قابل اتکاست.

ث)

برای این سوال استفاده از recall را توصیه میکنم. چون هزینه ی اشتباه در این مسئله بالاست. و میبینیم که مقادیر recall به خوبی به ما میگویند که مدل در خیلی از کلاس ها، خطای زیادی دارد.

اگر از بتدا این معیار را ملاک قرار دهیم به خوبی هزینه ی اشتباه را درک میکنیم و رسیدن به recall بالای ۸۰ برای همه کلاس ها میتواند معیار بهتری باشد.

ج)

بهترین کاری که میتوانیم در این قسمت کنیم، ایجاد دیتاست در کلاس هایی که کمتر داده دارند است. مثلا با data augmentation با روش های مختلف به نحوی که به اصل داده آسیب نرسد میتوانیم این کار را انجام دهیم.

منابع:

https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

https://medium.com/@erika.dauria/accuracy-recall-precision-80a5b6cbd28d#:~:text=Recall%20calculates%20the%20percentage%20of,high%2C%20you%20should%20use%20recall.

https://en.wikipedia.org/wiki/Confusion_matrix

پاسخ سوال چهارم

با استفاده از امکانات keras tuner، یک tuner تعریف کرده ایم و مقادیر بین ۱۶ تا ۵۱۲ با step=16 را به عنوان تعداد نورون های لایه dense تعریف کردیم. همچنین مقدار learning rate هم بین ۶ تا مقدار داده شده میتواند انتخاب شود. با tuner.search و با حداکثر ۱۰ ایپوک برای هر حالت، بهترین جواب را پیدا کرده ایم. ملاک تشخیص بهترین را هم دقت validation قرار میدهیم چون مهمترین موضوع است. که بهترین جواب ها به صورت مرتب شده به این صورت بودند:

```
tuner.results_summary()
```

```
Results summary
Results in ./Q4/untitled_project
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
dense_neurons: 352
learning_rate: 0.001
tuner/epochs: 10
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.8915833234786987
Trial summary
Hyperparameters:
dense_neurons: 432
learning_rate: 0.001
tuner/epochs: 10
tuner/initial_epoch: 4
tuner/bracket: 2
tuner/round: 2
tuner/trial_id: eaf52b95623c107b745c4e3bc4c95a98
Score: 0.8865000009536743
```

حالا مدلی را از ابتدا با بهترین هایپرپارامتر ها، یعنی تعداد نورون 352 و نرخ یادگیری 0.001 که می شود چنین مدلی:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_2 (Dense)             (None, 352)               276320

 dense_3 (Dense)             (None, 10)                3530

=================================================================
Total params: 279,850
Trainable params: 279,850
Non-trainable params: 0
```

آموزش میدهیم. سایر موارد مانند batch size و epochs و optimizer و ... را مثل قبل در نظر گرفتیم تا تفاوت در نتیجه به دلیل سایر موارد نباشد. دیتای آموزشی و تست هم که تغییری نکرده اند. چنین نتایجی در ۱۰ ایپاک آخر به دست آمد:

```
Epoch 41/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0903 - accuracy: 0.9672 - val_loss: 0.3964 - val_accuracy: 0.8941
Epoch 42/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0853 - accuracy: 0.9696 - val_loss: 0.3910 - val_accuracy: 0.8969
Epoch 43/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0838 - accuracy: 0.9702 - val_loss: 0.4197 - val_accuracy: 0.8886
Epoch 44/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0802 - accuracy: 0.9710 - val_loss: 0.4026 - val_accuracy: 0.8935
Epoch 45/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0775 - accuracy: 0.9718 - val_loss: 0.4194 - val_accuracy: 0.8879
Epoch 46/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0797 - accuracy: 0.9711 - val_loss: 0.4076 - val_accuracy: 0.8951
Epoch 47/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0755 - accuracy: 0.9729 - val_loss: 0.4077 - val_accuracy: 0.8946
Epoch 48/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0738 - accuracy: 0.9725 - val_loss: 0.4193 - val_accuracy: 0.8956
Epoch 49/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0718 - accuracy: 0.9743 - val_loss: 0.4131 - val_accuracy: 0.8955
Epoch 50/50
375/375 [==============================] - 1s 4ms/step - loss: 0.0681 - accuracy: 0.9754 - val_loss: 0.4741 - val_accuracy: 0.8895
```

که درواقع دقت روی داده آموزشی ۹ درصد و روی تست ۴ درصد حدودا افزایش داشته است.

از آنجایی که یادگیری این دیتاست fashion_mnist چندان پیچیده نیست، در حالت قبلی که از ۵۱۲ نورون در لایه مخفی استفاده شده بود به دقت کمتری رسیدیم. چون با ۳۵۲ بهتر میتوان تفکیک کلاس ها را انجام داد و با مقادیر بیشتر از آن، فقط داریم باعث کند تر شدن یادگیری شبکه و شاید overfit (در اینجا هنوز اتفاق نیفتاده است) آن میشویم. همانطور که میبینیم دقت داده ارزیابی در حالت قبل از ایپاک ۱۰ به بعد تغییری نکرده است. و دلیل مهمتر برای این موضوع، زیاد بودن learning rate است. که باعث شده است مدل فقط حول نقطه بهینه حرکت کند و هرگز به آن نرسد. به همین دلیل هم دقتمان از ایپاک ۱۰ به بعد تقریبا تفاوتی نداشته است. اما در مدل جدید با کاهش learning rate، این مسئله تا حدی حل شده است. و روند افزایشی دقت ها و کاهش loss ها هم روی داده آموزشی و هم ارزیابی مشهود است.