اصول پردازش تصویر

نیم‌سال دوم ۱۳۹۹-۱۴۰۰

مدرس: دکتر مصطفی کمالی تبریزی

دانشکده علوم ریاضی

تمرین سری دوم

نام و نام‌خانوادگی: امین کشیری

شماره دانشجویی: ۹۷۱۰۱۰۲۶

توضیحات کلی

- عکس‌های ورودی باید در پوشه‌ی inputs/images قرار بگیرند.
- در پوشه‌ی inputs/images یک پوشه‌ی دیگر به نام giraffe وجود دارد، که عکس‌های ورودی سوال ۳ را در آن قرار داده‌ام.
- خروجی تمام کدها، با همان اسم گفته شده در صورت سوال، در پوشه‌ی outputs/images قرار خواهد گرفت (اگر عکس دیگری نیز در این پوشه باشد، در توضیحات سوال مربوطه نوشته‌ام).
- به جز دو پوشه‌ی بالا، یک پوشه‌ی دیگر نیز باید ساخته شود با نام outputs/temps وجود دارد. این پوشه باید حتما از قبل وجود داشته باشند. برای همین کافی است از ساختار پوشه‌بندی این که آپلود کرده‌ام استفاده کنید و تنها ورودی‌ها را در جای خود قرار دهید.
- بعضی از خط‌ها در کد، دارای comment هستند، که با uncomment کردن آن‌ها معمولاً می‌توانید عکس را در مراحل میانی ببینید (توضیحات بیشتر را در صورت لزوم، در توضیحات هر سوال داده‌ام).
- همراه فایل‌های یک فایل requirements.txt قرار دارد که محیط اجرای کدهای من است (در صورت نیاز).

سوال ۱

بخش ۱

ابتدا باید سه دسته خط موازی در دنیای واقعی به دست آوریم. برای این کار از hough transform استفاده می‌کنیم که در درس image processing آموخته بودیم. این تابع دو ورودی به نام‌های min_theta و max_theta می‌گیرد، که حداکثر و حداقل زاویه خط عمودی بر خط‌های تشخیص داده‌شده توسط الگوریتم hough است. با توجه به شکل داده شده، سه بازه‌ی حدودی به این تابع می‌دهیم، و در خروجی ۳ دسته خط به دست می‌آیند. برای دیدن این سه دسته خط، می‌توانید توابع draw_lines که کامنت کرده‌ام را آنکامنت کنید. این تابع نیز تنها کاری که انجام می‌دهد این است که یک عکس و تعدادی خط می‌گیرد و آن‌ها را رسم می‌کند.

سپس در چند خط بعدی، محل تقاطع هر دسته خط رو به دست می‌آوریم. ابتدا با استفاده از تابع transform_to_cartesian پارامترهای خط‌ها را که در مختصات قطبی هستند به مختصات کارتزین تبدیل می‌کنیم. سپس با استفاده از تابع get_vanishing_point محل تلاقی یک دسته از خط‌ها رو به دست می‌آوریم، که همان vanishing point های ما هستند.

این تابع به این صورت کار می‌کند که یک دسته خط دریافت می‌کند و بهترین محل تلاقی را برای آن‌ها تشخیص می‌دهد و خروجی می‌دهد. برای این کار، از این ایده استفاده کرده‌ام که در شرایط ایده‌آل، نقطه‌ی تلاقی باید در تمام معادلات خط‌ها صدق کند. برای مثال باید در معادله‌ی خط i م صدق کند یعنی:

$$y - (m_i \cdot x + y_0) = 0$$

که مجموع این معادلات دستگاه زیر را تشکیل می‌دهند:

$$AX = b$$

که

$$A = \begin{bmatrix} . & . & . \\ -m_i & 1 & -y_{0i} \\ . & . & . \end{bmatrix}$$

$$X = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

و $b = 0$. حال با این که این دستگاه لزوما جواب ندارد (به خاطر وجود نویز و...)، اما به کمک تجزیه SVD می‌توانیم بهترین جواب ممکن برای این دستگاه را به دست آوریم. با استفاده از این روش، بهترین x و y ممکن به دست می‌آیند.
به کمک روش بالا، نتیجه‌ی محاسبات من به صورت زیر بود:

$$V_x = \begin{bmatrix} 9364 \\ 2596 \\ 1 \end{bmatrix}$$

$$V_y = \begin{bmatrix} -26748 \\ 4061 \\ 1 \end{bmatrix}$$

$$V_z = \begin{bmatrix} -2996 \\ -133753 \\ 1 \end{bmatrix}$$

دو خط بعدی که کامنت شده‌اند در صورت آنکامنت شدن یک تصویر به شما نشان می‌دهند که خطوط موازی آنقدر ادامه داده شده‌اند که با هم تلاقی پیدا کنند.

خواسته‌ی بعدی این است که معادله‌ی خط افق را به دست آوریم. این کار در تابع print_horizon انجام می‌شود. اول V_x و V_y در هم کراس می‌شوند، سپس نرمالایز می‌شوند تا $a^2 + b^2 = 1$ شود. رابطه‌ی خط نیز چاپ می‌شود که به صورت زیر است:

$ax + by + c = 0$
$a = -4.05348901 \cdot 10^{-2}$
$b = -9.99178124 \cdot 10^{-1}$
$c = 2.97343512 \cdot 10^{3}$

در خط بعدی نیز در صورتی که تابع draw_horizon را آنکامنت کنید، که با وصل کردن V_x و V_y ، خط افق را روی تصویر (زیر تصویر) می‌کشد و نمایش داده می‌دهد (برای کشیدن خط، با داشتن معادله‌ی خط کافی است دو نقطه از آن را به دست بیاوریم. خب بدون اینکار از قبل دو نقطه V_x و V_y را داریم و برای کشیدن از همین دو استفاده می‌کنم).

تابع بعدی نیز سه نقطه‌ی محو شدن و خط افق و محدوده‌ی تصویر را رسم می‌کند. کد خود تابع که بسیار واضح است. با داشتن نقاط، ۳ نقطه را می‌کشیم، و خط گذرنده از دو تا از آن‌ها را نیز می‌کشیم. محدود عکس نیز کافی است منتقل شود. اما عکس نهایی بزرگ می‌شود. برای همین، این تابع تمام اعداد را بر scale تقسیم می‌کند، که در کد من برابر با ۱۰ است.

بخش ۲

ابتدا سه نقطه‌ی محو شدن را به صورت دستی از کد ۱ به کد ۲ منتقل کرده‌ام. سپس با استفاده از تابع get_camera_parameters مشخصات دوربین را به دست می‌آوریم. با استفاده معادلات جلسه‌ی ۱۳، به ازای هر جفت از نقطه‌های محو شدن، یک شرط روی f و P_x و P_y به دست می‌آیند. حالا چون ۳ جفت داریم، پس یک دستگاه سه معادله سه مجهول برای این سه متغییر به دست می‌آید. دو رابطه‌ی اول به سادگی یک دستگاه تشکیل می‌دهند که آن دستگاه را ساخته‌ام و حل کرده‌ام. یک فرمول نیز برای f برحسب دو مجهول دیگر در کلاس به دست آوردیم (منظور در اسلایدها) و f نیز بر اساس آن فرمول به دست می‌آید. این سه مقدار را نیز چاپ می‌کنم. با محاسبات من نتیجه به صورت زیر شد:

$f = 14067$
$P_x = 2487$
$P_y = 1410$

سپس تابع draw_principal_point عکس ساخته شده را می‌سازد و ذخیره می‌کند.

حال ماتریس دوربین را می‌سازیم، و با back projection راستای محورهای مختصات دنیا را به دست می‌آورم.

برای به دست آوردن میزان چرخش حول محور z دوربین، به این دقت کنید که اگر محور x دوربین قرار باید با افق موازی شود، باید با V_z زاویه ۹۰ درجه بسازد. از طرفی چون فقط در صفحه xy می‌توانیم دوران کنیم، باید تصویر V_z را روی صفحه‌ی xy به دست آوریم و سپس زاویه x را با این بردار به دست آوریم، و هرچقد تا ۹۰ درجه فاصله دارد را به دست آوریم. دقت کنید که فقط راستای مهم است، برای همین کافی است متغییر سوم V_z را صفر کنیم تا راستای تصویر آن روی xy به دست بیاید. برای به دست آوردن زاویه نیز از ضرب داخلی استفاده می‌کنیم.

برای قسمت دوم نیز دقیقا مانند این قسمت باید عمل کنیم. برای این که سنسور دوربین عمود بر زمین باشد، باید بردار نرمال آن یا همان محور z دوربین موازی با آن باشد. پس باید محور z بر V_z عمود شود. پس هرچقد تا ۹۰ کم دارد باید به آن اضافه کنیم (و اگر نزدیک تر از ۹۰ درجه است، دور کنیم). برای به دست آوردن زاویه هم دقت کنید که محور z و V_z در دوران قبلی تکان نخورده‌اند. پس با همان مقادیر قبلی آن‌ها می‌توانیم زاویه بین آن‌ها را به دست آوریم.

مقادیر دوران‌ها پس از این محاسبات به صورت زیر در می‌آید (θ_z و θ_x به ترتیب دوران اول و دوم هستند):

$\theta_z = -2.32°$

$\theta_x = -5.94°$

که این یعنی حول محور z باید حدود ۲ درجه ساعتگرد دوران کنیم و حول محور x در مرحله‌ی دوم، نزدیک به ۶ درجه ساعتگرد دوران کنیم (منفی به این معنی است که در خلاف جهت مثلثاتی حرکت می‌کنیم).

بخش ۳

ابتدا دقت کنید که من دوران (حول زوایای داخلی، یا intrinsic) را در مرحله‌ی قبلی به دست آوردم. در قسمت قبلی ابتدا یک دوران ساعتگرد دستگاه مختصات حول محور z و سپس یک دوران ساعتگرد دیگر حول محور x داشتیم. به این دلیل که علامت‌ها را الان بررسی می‌کنم، از قسمت قبل فقط از اندازه‌ی دوران استفاده می‌کنم. می‌خواهیم ماتریس دوران تبدیل مختصات نقاط در دستگاه اولیه به دستگاه ثانویه را به دست آوریم (همان R در معادلات زیر):

$$x_{new} = P_{new}X = K[R|t]X = KRX$$

که t برابر صفر بود. همچنین X مختصات دنیای واقعی یک نقطه، در دستگاه اولیه است (قبل از تبدیل). در نتیجه داریم:

$$x_0 = P_0X = K[I|0]X = KX$$

می‌خواهیم به دست آوریم:

$$x_{new} = Hx_0 \Rightarrow Hx_0 = KRX \Rightarrow HKX = KRX \Rightarrow H = KRK^{-1}$$

دقت کنید دوران θ حول یک محور خود دستگاه، معادل با دوران $-\theta$ مختصات نقاط است (حول همان محور). با تمام این تفاسیر، ما ابتدا R را به دست می‌آوریم. دوران اولیه حول محور z بود و ساعتگرد. درنتیجه نقطه‌ها در خلاف جهت عقربه‌های ساعتگرد (در جهت مثلثاتی) جابجا می‌شوند در نتیجه: $\theta_z = 2.32$. دقیقا همین اتفاق نیز برای دوران دوم می‌افتد و: $\theta_x = 5.94$. همچنین دقت کنید که ابتدا دوران حول z و سپس حول x رخ می‌دهد. پس ماتریس‌های دوران را می‌سازم، و ماتریس دوران حول x را از راست در ماتریس دوران حول z ضرب می‌کنم (دقت کنید که دلیل عوض شدن علامت دوران‌ها، این است که خود دستگاه چرخیده است) و R به دست می‌آید.

سپس از معادلات بالا به راحتی H به دست می‌آید، و سپس آن را روی عکس اعمال می‌کنم.

نکته: برای این که نتیجه دیده شود، کمی کادر را بزرگتر کردم، و همچنین نتیجه را به وسط تصویر منتقل کردم. با کامنت کردن کدی که H از راست در T ضرب شده است می‌توانید این انتقال را کنسل کنید.

هموگرافی به صورت زیر بود (بدون در نظر گرفتن T):

$$H = \begin{bmatrix} 1.00531127 \cdot 10^{0} & -2.23812856 \cdot 10^{-2} & 1.83484770 \cdot 10^{1} \\ 4.09549806 \cdot 10^{-2} & 1.00959293 \cdot 10^{0} & -1.57819171 \cdot 10^{3} \\ 2.99632225 \cdot 10^{-7} & 7.38631960 \cdot 10^{-6} & 9.88840104 \cdot 10^{-1} \end{bmatrix}$$

سوال ۲

در چند خط اول عکس‌های ورودی را می‌خوانم. سپس با استفاده از SIFT برای عکس‌ها تعدادی بردار ویژگی به دست می‌آورم و سپس با استفاده از تابع get_matches ، نقاط متناظر را به دست می‌آوریم. بعد هم با استفاده از تابع get_matched_keypoints_and_coords مختصات نقاط متناظر را جدا می‌کنیم. چون تمامی این مراحل را چند تا از تمرین‌های قبلی انجام داده بودم، دوباره توضیح نمی‌دهم (کد عیناً کپی شده از تمرین‌های قبلی خودم و هیچ تغییری ندادم).

حال با استفاده از تابع findFundamentalMat ماتریس فاندامنتال (F) را به دست می‌آورم. این تابع با استفاده از RANSAC و نقاط متناظر F را محاسبه می‌کند، و همچنین یک mask به ما برمی‌گرداند که در آن نقاط متناظری که با این ماتریس همخوانی دارند را به ما نشان می‌دهد.

حال بسیار شبیه به ساختن عکس ۱۴ از تمرین ۱ سوال ۳، دو عکس می‌سازم. به کمک ماسک به دست آمده، یک بار نقاطی که inlier محسوب شده‌اند را می‌کشم، و بار دیگر روی همان، outlier ها اما با رنگی دیگر. سپس این دو عکس را به هم می‌چسبانم و ذخیره می‌کنم. در این قسمت، برای این که نتیجه بهتر مشخص شود، یک عکس دیگر با نام res05_2.jpg ذخیره کرده‌ام که همان عکس قبلی است، اما تناظرها را نیز با یک خط به هم وصل کرده‌ایم.

در ادامه با حل معادله‌ی $Fe = 0$ به کمک SVD ، e را به دست می‌آوریم. اما برای نشان دادن e ابعاد عکس کافی است. به همین دلیل مقداری همه چیز را انتقال می‌دهم. همچنین اندازه‌ی عکس نهایی خیلی بزرگ می‌شد. به همین خاطر همه چیز را scale کردم. برای e' نیز دقیقا به همین صورت عمل کردم. در نهایت:

$$e = \begin{bmatrix} -47873 \\ -6943 \\ 1 \end{bmatrix}$$

$$e' = \begin{bmatrix} 17046 \\ -1102 \\ 1 \end{bmatrix}$$

حال با استفاده از تابع draw_epilines می‌توانیم epiline ها را بکشیم. این تابع به این صورت کار می‌کند که برای هر عکس، ۱۰ نقطه از interest point ها را برمی‌دارد (۱۰ تا متناظر، از آن‌هایی که inlier هستند)، سپس با ضرب کردن ماتریس F از چپ در آن‌ها، مختصات خط متناظر آن‌ها در تصویر دیگر را به دست می‌آورد. سپس در هر معادله‌ی خط دو نقطه (چپ ترین و راست ترین نقطه) را قرار می‌دهد و مولفه‌ی دیگر را به دست می‌آورم، و سپس بین این دو نقطه یک خط می‌کشم. در نهایت نیز تصویر خواسته شده را ذخیره می‌کنم. این ۱۰ نقطه‌ی متناظر را نیز روی شکل مربوط به خودشان رسم می‌کنم.

سوال ۳

نتیجه‌ی این بخش با استفاده از openMVG انجام شده است. عکس‌ها را ورودی می‌دهید و کد را اجرا می‌کنید و به شما نتیجه را خروجی می‌دهد. نتایج را نیز می‌توانید با نرم افزاری شبیه به meshlab ببینید. خروجی‌های این بخش در پوشه‌ی outputs/giraffe_res قرار دارند. دو فایل cloud و colorized دو نتیجه‌ی خواسته شده‌ی این بخش هستند.
ورودی‌های این بخش نیز در آدرس زیر قرار دارند:
اینجا

سوال ۴

بخش ۱

پوشه‌ی Data دقیقا همانطوری که extract می‌شود در پوشه‌ی input قرار می‌گیرد. کد من از دو تابع اصلی ساخته شده است. در ابتدای کد، ثابت‌های اصلی برنامه مقداردهی می‌شوند. RESIZE_METHOD روشی است که با آن عکس‌ها را کوچک کرده‌ام. متغییرهایی به شکل DIR_ نیز آدرس‌هایی که برنامه استفاده می‌کند هستند. CLASS_NAMES نام تمام کلاس‌هایی است که برنامه قرار است تشخیص دهد، که با توجه به محتویات پوشه‌ی Data به دست می‌آید. بعضی از این متغییرها در قسمت‌های بعدی سوال نیز استفاده می‌شوند و آنجا دوباره توضیح نمی‌دهم.

کد من دو حالت اجرا دارد. درحالت اول، اگر متغییر FIND_BEST_HYPER_PARAMETERS برابر با False باشد، کد تنها یک بار اجرا می‌شود، با سه پارامتر SIZE ، که نشان‌دهنده‌ی اندازه‌ی عکس‌هاست، P_NORM ، که نشان دهنده‌ی نرم استفاده شده است، و K که متغییری است که در الگوریتم KNN استفاده می‌شود، و سپس اجرای برنامه پایان می‌یابد.

در حالت دوم که این متغییر برابر با True است، کد برای K ها و SIZE ها و P_NORM های مختلف اجرا می‌شود، و در نهایت بهترین نتیجه‌ی گرفته شده و پارامترهای آن را چاپ می‌کند. این حالت دوم را برای این استفاده کرده بودم که ببینم بهترین نتیجه را با چه هایپر پارامترهایی به دست می‌آورم. البته دقت کنید که در این حالت، جواب گزارش شده‌ی من به عنوان دقت الگوریتم، در واقع جواب درستی نیست. زیرا من پارامترهایی را استفاده کرده‌ام که روی دیتاست test من بهینه شده‌اند. در واقع راه درست‌تر استفاده از مقداری از عکس‌ها برای validation بود، و آن وقت عدد گزارش شده‌ی من به عنوان دقت الگوریتم عدد قابل اعتمادی می‌شد. به هر حال، چون این جزییات در صورت سوال خواسته نشده بود، این تکرار کردن برای اعداد مختلف فقط برای گزارش بهترین نتیجه استفاده شده است هرچند از نظر علمی اعتبار کافی ندارد.

حال کافی است توضیح دهم برای یک سری هایپرپارامتر ورودی، کد من چگونه کار می‌کند. تابع اصلی برنامه‌ی من تابع test_KNN است. اول این تابع، تابع دیگری به نام create_feature_vectors صدا زده می‌شود. این تابع به این صورت کار می‌کند که به ازای تمام عکس‌های train (و در مرحله‌ی بعدی test) در هر کلاس، آن‌ها را به اندازه‌ی $SIZE \times SIZE$ تبدیل می‌کند، و سپس سطرهای آن تصویر را پشت هم می‌گذارد و به عنوان یک feature_vector در نظر می‌گیرد. تمام این بردارهای ویژگی برای عکس‌های یک دسته خاص را زیر هم قرار می‌دهد، و سپس ماتریس نتیجه را ذخیره می‌کند. این ذخیره کردن به این خاطر است که مثلا اگر ده بار قرار است ما کد را با $SIZE = 20$ اجرا کنیم، هر ده بار تمام این مراحل را انجام ندهد. این اتفاق نیز به این صورت می‌افتد که در اول این تابع، چک می‌کند آیا این بردارهای ویژگی (با این SIZE مشخص) قبلا محاسبه شده‌اند یا خیر، و اگر جواب بله بود، دوباره محاسبه نمی‌کند. یک متغییر به نام RECOMPUTE_FEATURE_VECTORS نیز در اول برنامه وجود دارد، که در صورتی که آن را set کنید، حتی اگر بردارهای ویژگی با یک سایز خاص وجود داشته باشند، باز هم آن‌ها را از اول محاسبه می‌کند.

پس از این که تابع اولیه کار خود را تمام کرد، در ادامه‌ی کد، تمام بردارهای ویژگی همه‌ی کلاس‌ها پشت هم قرار می‌گیرند، و label هر کدام از آن‌ها نیز در آرایه‌ی train_labels قرار می‌گیرد. همین کار را نیز برای عکس‌های test انجام می‌دهیم.

سپس با استفاده از تابع KNN آماده در sklearn ، نزدیک‌ترین نقطه به هرکدام از feature_vector های تست را به دست می‌آوریم و سپس با مقایسه‌ی آن‌ها با label های اصلی، دقت الگوریتم را به دست می‌آوریم.

مقادیری که بهترین نتیجه را به من دادند، به شرح زیرند:

$SIZE = 22$
$K = 1$
$P\_NORM = 1(L_1)$
$Accuracy = 0.22 = 22\%$

بخش ۲

برای این بخش، من بردارهای ویژگی تصاویر، لغت نامه (dictionary) و هیستوگرام تصاویر را تنها یک بار محاسبه می‌کنم و بارهای بعدی از نتیجه ذخیره شده‌ی آن‌ها استفاده می‌کنم. برای خاموش کردن این رفتار، می‌توانید از ۳ متغییر RECOMPUTE_FEATURE_VECTORS و RECOMPUTE_DICTIONARY و RECOMPUTE_HISTOGRAMS استفاده کنید. متغییر K همان متغییر استفاده در KNN نهایی است (مثل بخش قبلی).

TRAIN_FEATURE_VECTORS و TEST_FEATURE_VECTORS دو متغییر هستند که در آن‌ها تمام بردارهای ویژگی تصاویر آموزش و آزمون به ترتیب قرار دارند.

به دست آوردن بردارهای ویژگی، توسط تابع compute_feature_vectors انجام می‌شود. در صورتی که کاربر مجبور کرده باشد یا بردارهای ویژگی از قبل وجود نداشته باشند، بردارهای ویژگی از اول محاسبه می‌شوند. این تابع نیز تنها کاری که انجام می‌دهد این است که عکس‌ها را یکی یکی باز می‌کند، به کمک SIFT بردارهای ویژگی را به دست می‌آورد، آن‌ها را پشت هم می‌چیند و در نهایت نتیجه را ذخیره می‌کند.

مرحله‌ی بعدی، به دست آوردن dictionary است که آن را در متغییر DICTIONARY نگه داری می‌کنم. این کار با تابع compute_visual_words انجام می‌شود. تعداد لغات لغت نامه را، که در واقع همان K در الگوریتم K_means است، در متغییر DICTIONARY_SIZE قرار گرفته است.

این تابع نیز دقیقا مانند تابع قبلی قابلیت استفاده از نتایجی که قبلا ذخیره شده‌اند را دارد. در غیر این صورت، الگوریتم K_means را روی تمام بردارهای ویژگی آموزش که در TRAIN_FEATURE_VECTORS قرار داشتند با K برابر با DICTIONARY_SIZE اجرا می‌کنیم. این کار باعث به دست آمدن لغت نامه می‌شود.

تابع آخر ما نیز compute_histograms_and_test است. این تابع نیز به این صورت کار می‌کند که به کمک KNN ، به ازای هرکدام از بردارهای ویژگی، نزدیک‌ترین لغت لغت‌نامه را پیدا می‌کند. این KNN با KNN آخر کار تفاوت دارد، و در اینجا تنها می‌خواهیم نزدیک‌ترین نقطه به یک بردار را پیدا کنیم، پس به صورت دستی K آن را برابر با ۱ قرار می‌دهم. این‌کار را برای تمام بردارهای ویژگی هر تصویر انجام می‌دهد (که با تابع knn.findNearest انجام می‌دهد). سپس، به کمک این که هر عکس، بردارهای ویژگی اش به کدام لغات نزدیک تر بودند، برای هر عکس یک هیستوگرام می‌سازد. بقیه مراحل عیننا مانند قسمت قبلی است. برای هر تصویر، چه آموزش و چه آزمون، یک هیستوگرام به دست می‌آورم، سپس دقیقا مانند بخش قبلی، با کمک KNN ، دقت را اندازه گیری می‌کنم و چاپ می‌کنم.

کد این بخش نیز دقیقا مانند بخش قبلی یک متغییر با نام FIND_BEST_HYPER_PARAMETERS دارد، که توضیحات آن عیناً مثل بخش قبلی است. در صورتی که False باشد، کد تنها یک بار و با پارمترهای set شده اجرا می‌شود.

بهترین نتیجه با پارامترهای زیر گرفته شد:

$K = 20$ (in final KNN)
DICTIONARY_SIZE $= 110$ (K in Kmeans)
Accuracy $= 0.472 = 47.2\%$

بخش ۳

توضیحات این بخش، تا قسمت آخر که بعد از پیدا کردن هیستوگرام‌هاست، دقیقا یکسان با بخش ۲ است. فقط در مرحله‌ی آخر، به جای استفاده از KNN از SVD استفاده می‌کنیم.

بهترین نتیجه با پارامترهای زیر گرفته شد:

DICTIONARY_SIZE $= 110$ (K in Kmeans)
Accuracy $= 0.543 = 54.3\%$

در انتهای کد نیز، در صورتی که در حالت پیدا کردن بهترین پارامتر نباشیم، ماتریس confusion را به دست می‌آورم و ذخیره می‌کنم.

در هر دو بخش این سوال، من سعی کردم که هیستوگرام‌ها را نرمالایز کنم. اما نتیجه‌ام بدون نرمالایز کردن بهتر بود، به همین دلیل آن کدها را حذف کردم.