به نام خدا

پروژه پایان‌ترم درس جبر خطی کاربردی

استاد محترم درس: دکتر شجاعی

دانشجو: الهام رازی

شماره دانشجویی:۹۷۳۱۰۱۹

نیمسال دوم ۹۸-۹۹

بخش اول: چکیده

در این پروژه سعی شده تا از مباحث آموخته شده جبر خطی به صورت کاربردی استفاده شود. در این پروژه با کار با عکس آشنا شدیم و همچنین از تکنیکی به نام principal component analysis برای آنالیز داده‌ها استفاده کردیم که در بخش‌های بعد به معرفی آن خواهیم پرداخت.

با استفاده از این تکنیک و مفاهیم جبر خطی به انجام یک سری عملیات روی عکس ورودی برنامه پرداختیم و در نهایت از عکس ورودی، یک عکس دیگر در خروجی تولید کردیم که در بخش‌های بعدی به ویژگی‌های آن می پردازیم.

بخش دوم: مقدمه

در این بخش به معرفی ابزارهای لازم برای انجام این پروژه می‌پردازیم. برای نوشتن کد برنامه از زبان برنامه نویسی پایتون استفاده شده که با توجه به کتابخانه‌های متنوع و گسترده‌ای که دارد، ابزار بسیار مناسبی برای پردازش تصویر و کار بر روی آن به شمار می‌رود. همچنین برای پیاده کردن تکنیک pca و آشنایی با مباحث آن از بخش ۷,۵ کتاب مرجع درس(lay linear algebra) استفاده کردیم. در بخش برنامه نویسی هم از کتابخانه‌های numpy ، که ابزار قدرتمندی برای کار با ماتریس‌ها و انجام عملیات ریاضی بر روی آن‌هاست، کتابخانه‌های pillow و matplotlib برای کار با عکس در پایتون استفاده کردیم.

پردازش و پیاده سازی تکنیک pca را روی عکس های ۲۰۰۰*۲۰۰۰ پیکسل انجام دادیم، هرچند که برنامه برای سایزهای متفاوت عکس هم قابل استفاده است. همچنین ترجیحا بهتر است که از فرمت jpeg برای عکس ورودی استفاده شود تا بتوان در خروجی ابرداده را هم داشت.(در ادامه در مورد آن توضیح خواهیم داد.)

بخش سوم: بدنه اصلی گزارش

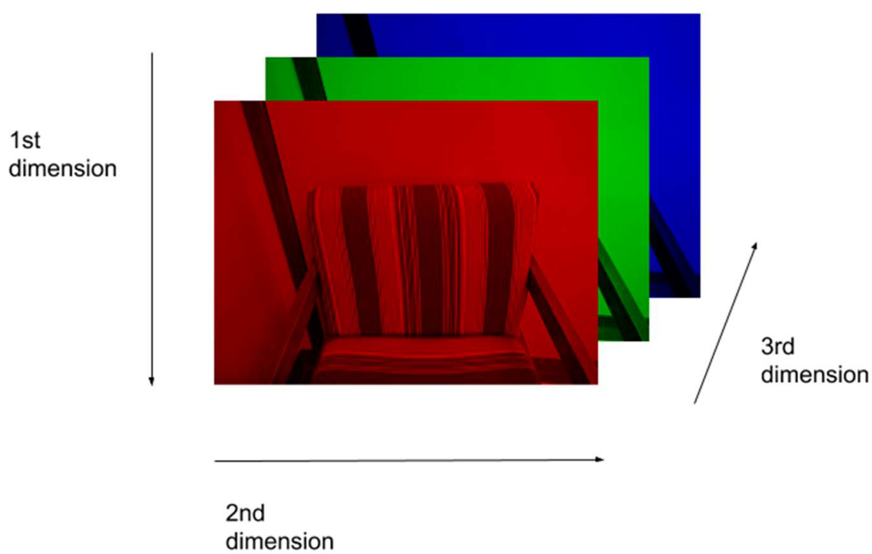
در این بخش به ترتیب گفته شده در پروژه به توضیح فرایندها می‌پردازیم و قطعه کد ها و سایر عملیات را توضیح می‌دهیم.

باز کردن عکس و ذخیره به صورت ماتریس سه بعدی:

در ابتدا عکس را با استفاده از PILLOW در فضای برنامه باز میکنیم و سپس آنرا به فرم ماتریسی تغییر میدهیم. یک عکس به صورت یک ماتریس سه بعدی است و در واقع از پیکسل تشکیل شده است؛ بعد اول و دوم مربوط به مکان آن در عکس است. (درواقع همان مختصات آن در فضای دو بعدی) و بعد سوم آن مربوط به رنگ پیکسل است(RGB). دو تصویر پایین چگونگی ابعاد آن را نشان میدهند:

این تصویر را درنظر بگیرید:

وقتی عکس به صورت ماتریس ۳ بعدی ذخیره میشود:



```python
# opening the image
image = Image.open("image.jpg")

# save the image in a numpy 3D array
matrix = np.array(image)
```

قطعه کد بالا ذخیره عکس به صورت ماتریس numpy سه بعدی در matrix در محیط پایتون را نشان می‌دهد.

گرفتن ابرداده(metadata):

در این قسمت ابرداده (metadata) مربوط به عکس را نمایش می‌دهیم. دوربین‌های دیجیتال و گوشی‌های هوشمند از استاندارد exif برای ذخیره‌ی عکس استفاده می‌کنند. این استاندارد شامل تگ(tag) های پرکاربردی است که می‌توان استخراج کرد و این تگ ها اطلاعاتی را در مورد عکس و دستگاه دوربین به ما می‌دهند؛ مانند مدل دوربین، تاریخ ایجاد عکس، و حتی اطلاعات gps دستگاه دوربین. برای گرفتن ابرداده از کتابخانه‌ی pillow استفاده میکنیم. قطعه کد پایین، کد مربوطه است:

```python
# getting the metadata, if any:
exifdata = image.getexif()
for tag_id in exifdata:
    # get the tag name, instead of human unreadable tag id
    tag = TAGS.get(tag_id, tag_id)
    data = exifdata.get(tag_id)
    # decode bytes
    if isinstance(data, bytes):
        data = data.decode()
    print(f"{tag:25}: {data}")
```

در خط دوم، ابرداده‌ی عکس را استخراج می‌کنیم، ولی این داده قابل فهم برای انسان نیست. به همین خاطر در ادامه تگ ها را به متن قابل خواندن برای انسان تبدیل می‌کنیم. در ادامه یک نمونه ابرداده برای یک عکس از دوربین شخصی را مشاهده می‌کنیم:

the metadata:

ExifVersion : 0220

ComponentsConfiguration : ▯ ▯

FlashPixVersion : 0100

DateTimeOriginal : 2015:08:25 12:08:32

DateTimeDigitized : 2015:08:25 12:08:32

ExposureBiasValue : 0.0

ColorSpace : 1

MeteringMode : 2

ExifImageWidth : 3264

Flash : 0

FocalLength : 3.2

ExifImageHeight : 1836

ExifInteroperabilityOffset: 412

Make : LG Electronics

Model : LG-D405

Orientation : 6

YCbCrPositioning : 1

XResolution : 72.0

YResolution : 72.0

FNumber : 2.4

GPSInfo : {5: b'\x00', 6: 0.0}

ResolutionUnit : 2

ExifOffset : 162

WhiteBalance : 0

مثلا می‌توان مشاهده کرد که عکس در سال ۲۰۱۵ گرفته شده و از یک گوشی هوشمند LG استفاده شده است. اطلاعات دیگر مربوط به عکس هم قابل مشاهده است.

محاسبه میانگین داده ها:

حال میانگین داده‌های عکس ورودی را محاسبه می‌کنیم. ولی در ابتدا بایستی ماتریس سه بعدی عکس را به یک ماتریس دو بعدی تبدیل کنیم تا عملیات محاسبات بر روی آن ساده‌تر شود. یعنی این ماتریس 3*2000*2000 را تبدیل به یک ماتریس 3*4000000 کنیم. در این ماتریس دو بعدی هر سطر آن مربوط به داده های یک رنگ از پیکسل است؛ یعنی مثلا ستون اول مربوط به پیکسل (0,0) است و سطر اول آن مربوط به رنگ قرمز، سطر دوم مربوط به رنگ سبز و سطر سوم مربوط به رنگ آبی است. حال بردار میانگین را با استفاده از فرمول زیر محاسبه می‌کنیم. در ادامه کد مربوط نیز آورده شده است.

$$M = \frac{1}{N}(X_1 + X_2 + \cdots + X_N)$$

هر کدام از X_i ها یکی از ستون‌های ماتریس دوبعدی ماست.M نیز یک بردار با ۳ عضو و در فضای R^3 است.

```
# we need to save the 3d data as a 2d to work with it
reshaped_matrix = matrix.transpose(2, 0, 1).reshape(3, -1)
print(reshaped_matrix.shape)

# calculating the mean vector
mean_vector = np.sum(reshaped_matrix, axis=1)
mean_vector = mean_vector / (shapes[0] * shapes[1])
print("the mean vector is:", mean_vector)
```

در قطعه کد بالا هم ماتریس را به ماتریس دو بعدی تبدیل کردیم و هم بردار میانگین را محاسبه کردیم.

به دست آوردن ماتریس همبستگی(covariance matrix):

در این قسمت می‌خواهیم ماتریس کواریانس را به دست آوریم. ماتریس کواریانس ماتریسی است که مقادیر واریانس و کواریانس مجموعه‌ای از بردارها خواهد داشت و اطلاعات زیادی را درمورد ماتریس ما به ما خواهد داد.

واریانس یک متغیر ، میزان پراکندگی داده ها پیرامون مبانگین را نشان می‌دهد. کواریانس نیز نشان دهنده‌ی میزان همبستگی دو متغیر است. یا درواقع این کمیت، چگونگی تغییرات توام دو متغیر را نشان می‌دهد. مثبت بودن مقدار کواریانس نشان‌دهنده‌ی تغییرات هم‌جهت دو متغیر است و منفی بودن آن هم تغییرات غیر هم‌جهت دو متغیر را نشان می‌دهد. صفر بودن آن نیز نشان دهنده‌ی عدم وجود وابستگی خطی بین دو متغیر است.

حال با استفاده از بردار میانگین، ماتریس به فرم mean-deviation را به دست می‌آوریم. یعنی از هرکدام از ستون‌های ماتریس دو بعدی تصویر بردار میانگین را کم می‌کنیم و ماتریس B را به دست می‌آوریم:

$$B = [\hat{X}_1 \quad \hat{X}_2 \quad \cdots \quad \hat{X}_N]$$

و $\hat{X}_i = X_i - M$ ستون‌های ماتریس B می‌باشد.

حال ماتریس کواریانس را با استفاده از رابطه‌ی زیر محاسبه می‌کنیم:

$$S = \frac{1}{N-1} BB^T$$

ماتریس S یک ماتریس متقارن و ۳ * ۳ است. مقادیر روی قطر مقادیر واریانس متغیرها هستند. اولین مقدار روی قطر مربوط به رنگ قرمز است، دومی سبز و سومی رنگ آبی است.

واریانس همیشه مقداری نامنفی است. مقدار واریانس صفر نشان می‌دهد که مقدار تمام داده ها برابر است. واریانس کوچک نشان می‌دهد که مقدار داده ها به هم و به میزان میانگین نزدیک است، و واریانس بزرگ نشان می‌دهد که مقادیر داده ها از هم دور و پراکنده است.

هنگامی که ماتریس کواریانس را برای یکی از عکس‌های نمونه محاسبه کردیم، مقدار واریانس ها بزرگ بود؛ پس می‌توان نتیجه گرفت که مثلا برای داده های رنگ قرمز، میزان قرمز بودن پیکسل های عکس مثل هم نیست. همین نتیجه مشابه را می‌توان با استفاده از تحلیل میزان واریانس برای بقیه‌ی متغیرها گرفت.

S[i][j] = S[j][i] برای حالت **i≠j** . این مقادیر برابر با کواریانس دو داده‌ی **i**ام و **j**ام است. با تحلیل

[[1901.91651165, 1968.76131126, 1515.14196136]

[1968.76131126, 2466.33207291, 1438.50467379]

[1515.14196136, 1438.50467379, 1431.28244387]]

ماتریس کواریانس در بالا آمده است، با بررسی کواریانس بین متغیرهای مختلف می‌توانیم متوجه شویم که میزان تغییرات دو به دوی آن‌ها هم جهت است و وابستگی بین آن‌ها وجود دارد.(correlation)

در پایین کد مربوطه را مشاهده می‌شود:

```
b_matrix = np.copy(reshaped_matrix)
e = b_matrix.transpose()
ee = e - mean_vector

# getting the covariance matrix
b = ee.transpose()
bt = ee
covariance_matrix = np.dot(b, bt)
covariance_matrix = (1/(shapes[0] * shapes[1] - 1)) * covariance_matrix
print("the covariance matrix is:")
print(covariance_matrix)
```

در اینجا بردار میانگین را از ماتریس اولیه کم می‌کنیم و ماتریس B را به دست می‌آوریم و ترانهاده‌ی آن را هم به دست می‌آوریم. و سپس از ضرب این دو ماتریس، در ادامه ماتریس کواریانس را به دست می‌آوریم.

کاهش بعد ماتریس:

در این قسمت می‌خواهیم با استفاده از تکنیک pca بررسی کنیم که آیا می‌توان بعد ماتریس تصویر را کاهش داد یا نه.

هدف principal component analysis این است که یک ماتریس مربعی متعامد P بیابیم که یک تغییر متغیر X = PY را برای ما ایجاد کند. این به این معنی است که هر ستون X ، یک ستون از Y به آن نظیر می‌شود. حال می‌خواهیم چک کنیم که آیا می‌توانیم از ماتریس Y سطری را کم کنیم یا نه. برای اینکار در ماتریس کواریانس مقادیر ویژه را محاسبه می‌کنیم. در numpy با استفاده از تابع‌های آماده این کار به راحتی قابل انجام است. با استفاده از آن eigenvector ها را هم به دست می‌آوریم. با استفاده از eigenvector ها و مرتب کردنشان بر اساس ترتیب eigenvalue به ترتیب نزولی، ماتریس P را هم به دست می‌آوریم.(باید دقت کنیم که eigenvector ها به صورت واحد باشند.)

حال واریانس کلی را محاسبه می‌کنیم(tr). مقدار برابر با جمع مقادیر ویژه‌ی ماتریس کواریانس است. مقادیر ویژه‌ی ما principal components نامیده می‌شوند. حال باید درصد دخالت هرکدام از مقادیر ویژه در tr را محاسبه کنیم. در کد مشخص کرده‌ایم که اگر درصد به دست آمده برای یک component کمتر از ۵ درصد بشود می‌توانیم آن را حذف کنیم؛ انگار که آن متغیر در مختصات جدید(Y) واریانسی ندارد. می‌توانیم به جای آن صفر بگذاریم. به این ترتیب خواهیم توانست که بعد داده‌ی سه بعدیمان را کم کنیم.

کد این بخش در ادامه آمده است.

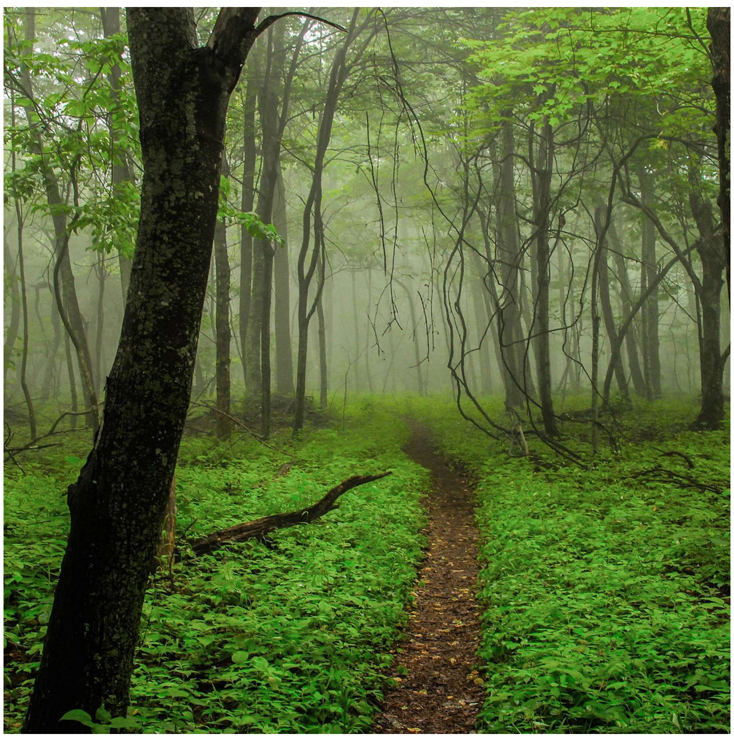
```python
# getting the eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
eigenvalues_sorted = np.sort(eigenvalues)
eigenvectors_sorted = eigenvectors[:, eigenvalues_sorted.argsort()]
print("the eigenvalues and the eigenvectors of the covariance matrix:")
print(eigenvalues_sorted)
print(eigenvectors_sorted)

eigvectors = np.asarray([[eigenvectors_sorted[0][2],
eigenvectors_sorted[0][1], eigenvectors_sorted[0][0]],
                        [eigenvectors_sorted[1][2],
eigenvectors_sorted[1][1], eigenvectors_sorted[1][0]],
                        [eigenvectors_sorted[2][2],
eigenvectors_sorted[2][1], eigenvectors_sorted[2][0]]])
print(eigvectors)
y = np.dot(eigvectors.transpose(), b)
print("the new matrix for the image matrix:")
print(y)


# calculating the total variance
tr = sum(eigenvalues_sorted)
print("the total variance is: ", tr)

components = [eigenvalues_sorted[2] / tr * 100, eigenvalues_sorted[1] / tr *
100, eigenvalues_sorted[0] / tr * 100]
print(*components)

pca = []
for i in range(3):
    if components[i] < 5:
        pca.append(i)
print(pca)

# reducing the dimension
for i in pca:
    y[i] = 0
```

می‌دانیم که $P^{-1} = P^{T}$، چرا که P یک ماتریس symmetric است. پس $Y = P^{T}X$.

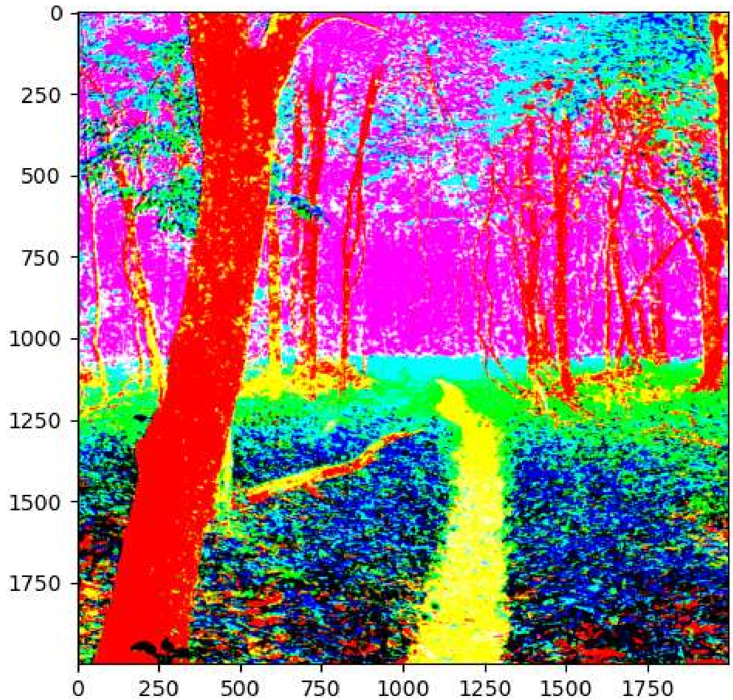
به دست آوردن عکس با استفاده از PCA:

تا به اینجا ماتریس کاهش بعد یافته‌ی Y را به دست آورده‌ایم. حال لازم است که ماتریس دوبعدی Y را به ماتریس سه بعدی 3*2000*2000 تبدیل کنیم تا بتوانیم از روی آن یک عکس را تولید کرده و در خروجی نمایش دهیم. می‌دانیم که می‌توانیم یک ماتریس سه بعدی 3*2000*2000 را به صورت سه ماتریس 1*2000*2000 در بیاوریم؛ هر ماتریس برای یک رنگ از RGB. با استفاده از کتابخانه‌ی matplotlib تابعی را ایجاد کرده ایم که ماتریس دو بعدی Y را به ۳ ماتریس سه بعدی 1*2000*2000 شکستیم.(تابع decompose_2d_array(matrix)). سپس با تابع دیگری(combine_matrix) این سه ماتریس را با هم ترکیب کردیم تا ماتریس سه بعدی نهایی برای عکس خروجی را به دست آوریم. در نهایت، تصویر را خروجی میدهیم:

عکس ورودی به صورت زیر است:



عکس خروجی هم به این ترتیب می‌باشد:



هر دو عکس به ابعاد ۲۰۰۰*۲۰۰۰ پیکسل می‌باشند.

قطعه کد مربوط به هر دو تابع و نمایش عکس در ادامه مشاهده می‌شود:

```
def decompose_2d_matrix(mat):
    new_r = [[0] * (shapes[1]) for i in range((shapes[0]))]
    new_g = [[0] * (shapes[1]) for i in range((shapes[0]))]
    new_b = [[0] * (shapes[1]) for i in range((shapes[0]))]
    for i in range(0, len(mat[0])):
        new_r[int(i / shapes[1])][i % shapes[1]] = mat[0][i]
    for i in range(0, len(mat[0])):
        new_g[int(i / shapes[1])][i % shapes[1]] = mat[1][i]
    for i in range(0, len(mat[0])):
        new_b[int(i / shapes[1])][i % shapes[1]] = mat[2][i]
    return new_r, new_g, new_b


def combine_matrix():
    compressed = (np.dstack((newr, newg, newb)))
    return compressed
#-------------------------------------------
y = np.dot(eigvectors.transpose(), b)
newr, newg, newb = decompose_2d_matrix(y)
plt.imshow(combine_matrix())
plt.show()
```

بخش چهارم: نتیجه گیری

Pca یک نکنیک ریاضیاتی است برای کاهش بعد و هدف آن این است که تعداد متغیرها را کاهش دهد، در حالی که متغیرهای مهم و تاثیرگذار باقی بمانند. در این پروژه هم ما سعی کردیم که در عکس خود ابعادی که اهمیت کمتری داشتند را حذف کنیم تا بتوانیم حجم داده و محاسبات را کاهش دهیم.

از مزایای روش pca می‌توان به کاهش زمان محاسبات، حذف داده‌های پرت و همچنین قابل فهم تر کردن مساله اشاره کرد(مورد آخر معمولا در مورد مسائلی است که تعداد متغیرهای ۳ و به بالا دارند. چرا که فهم ابعاد بالا برای انسان بسیار دشوار و درواقع غیرممکن است.)

همچنین در این پروژه از ابزارهای آماری مانند واریانس، میانگین، کواریانس و .. و همچنین مفاهیم جبر خطی که پیش‌تر در درس آموخته بودیم نیز استفاده کردیم تا بتوانیم یک عکس در خروجی تولید کنیم و همچنین داده های عکس اولیه را هم تحلیل کنیم.