به نام خدا

واحد تهران جنوب

دانشکده فنی و مهندسی

پروژه کارشناسی

مهندسی کامپیوتر - نرم‌افزار

عنوان:

تشخیص اعداد دست‌خط فارسی به کمک یادگیری ماشین در اپلیکیشن iOS

استاد راهنما:

استاد کوروش صمیمی داریوش

نام و نام خانوادگی دانشجو:

طاده الکسانی خداویردیان

شماره دانشجویی:

۹۵۱۱۲۱۱۸۵۳

زمستان ۹۹

فهرست

چکیده

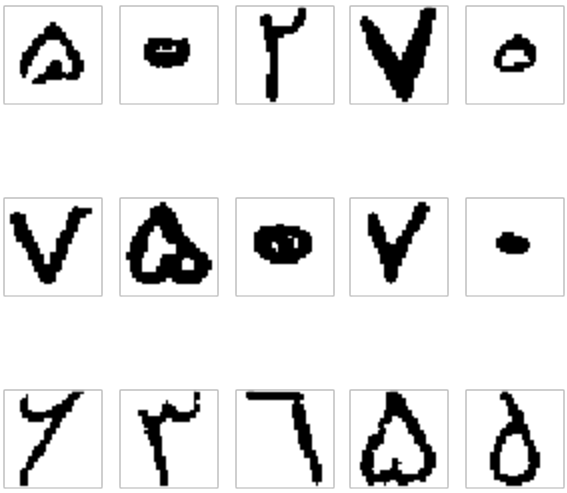
ابتدا به کمک مجموعه ارقام دستنویس هدی، که اولین مجموعه‌ی بزرگ ارقام دستنویس فارسی است، یک مدل به کمک پایتون و کتاب‌خانه‌های مربوط به یادگیری ماشین نوشته شود که بتواند داده‌های شامل عدد دستنویس را با دقت بالا ۹۷٪ پیش‌بینی کند.
سپس این مدل به مدل قابل استفاده در اپلیکیشن‌های iOS تبدیل شود و با استفاده از آن اپلیکیشنی دارای رابط گرافیکی نوشته شود که بتواند با استفاده از گالری یا دوربین فرد اعداد دست‌خط فارسی را تشخیص و روی صفحه به همراه احتمال هر عدد چاپ کند.

مقدمه

تکنولوژی‌های مورد استفاده

- Python
- Machine Learning
- Swift
- CoreML

توضیح مجموعه داده استفاده شده



کد مربوط به Visualization مجوعه داده هدی در بخشی با همین نام در فایل ... موجود است.

مجموعه ارقام دستنویس هدی که اولین مجموعه‌ی بزرگ ارقام دستنویس فارسی است، مشتمل بر ۱۰۲۳۵۲ نمونه دستنوشته سیاه سفید است. این مجموعه طی انجام یک پروژه کارشناسی ارشد درباره بازشناسی فرم‌های دست‌نویس تهیه شده است. داده های این مجموعه از حدود ۱۲۰۰۰ فرم ثبت نام آزمون سراسری کارشناسی ارشد سال ۱۳۸۴ و آزمون کاردانی پیوسته دانشگاه جامع علمی کاربردی سال ۱۳۸۳ استخراج شده است. خصوصیات این مجموعه داده:

درجه تفکیک نمونه‌ها: ۲۰۰ نقطه بر اینچ
تعداد کل نمونه‌ها: ۱۰۲۳۵۲ نمونه
تعداد نمونه‌های آموزش: ۶۰۰۰ نمونه از هر کلاس
تعداد نمونه‌های آزمایش: ۲۰۰۰ نمونه از هر کلاس
سایر نمونه‌ها: ۲۲۳۵۲ نمونه

فاز اول: یادگیری ماشین

در فاز اول، سعی شد تا با تحلیل مجموعه داده هدی و پیاده‌سازی الگوریتم‌های یادگیری ماشین مدل ما ساخته شود تا بتوان از آن در اپلیکیشن iOS استفاده کرد.
به این منظور از کتاب‌خانه Scikit-learn که کامل‌ترین کتاب‌خانه یادگیری ماشین حال حاضر در پایتون است استفاده شد.
این کتابخانه شامل پیاده سازی الگوریتم های یادگیری نظارت شده (Supervised Learning) و یادگیری نظارت نشده (Unsupervised Learning) است.

به طور کلی به مسئله‌هایی که ستون طبقه یا class را داشته باشند، مسائل طبقه‌بندی یا classification گفته می‌شود. این دست از مسائل به یاگیریِ با ناظر (supervised learning) نیز معروف هستند، چون در واقع یک ناظر وجود دارد که ستون آخر را برای ما برچسب‌زنی کند (مثلاً در این‌جا حدود ۱۰۰ هزار عدد وارد شده در فرم ثبت نام آزمون سراسری کارشناسی ارشد برای ما برچسب زده شده است).

در ابتدا یکی از چالش‌ها خواندن این مجموعه داده در زبان پایتون بود چون فایل‌های مجموعه داده در نظر گرفته شده با فرمت cdb یا همان constant database بودند و بطور طبیعی و یا توسط کتاب‌خانه‌هایی مثل pandas قابل خواندن نبودند. در پایتون معمولا فایل‌های مجموعه داده با فرمت csv هستند که به سادگی قابل خواندن هستند. به این منظور از کد آقای امیر سانیان که در گیت‌هاب موجود است استفاده شد و به کمک آن مجموعه‌داده هدی به راحتی در پروژه وارد شد. برای دیدن کد این بخش به بخش Load Data در فایل ... مراجعه کنید.

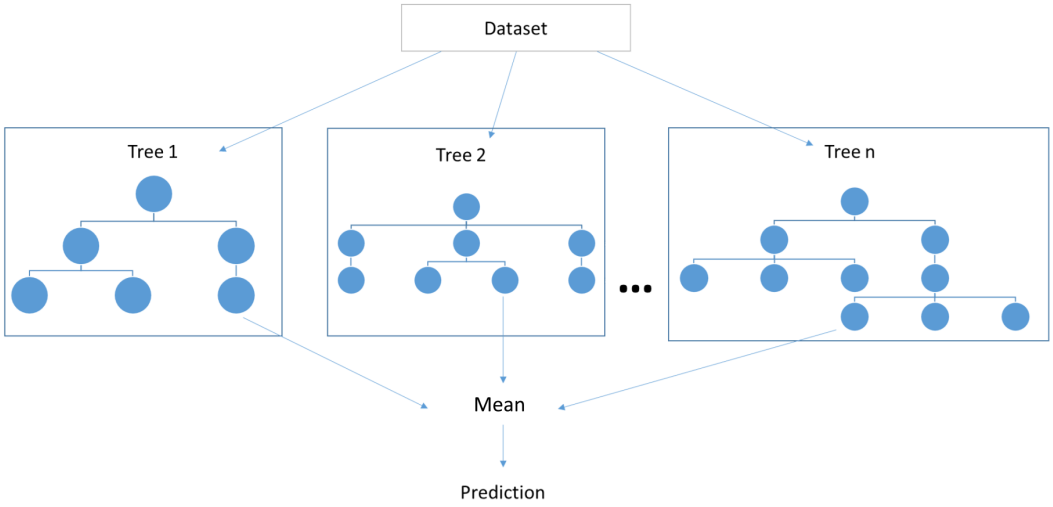
```
type(X_train):  <class 'numpy.ndarray'>
X_train.dtype: float32
X_train.shape: (reshape=True),  (60000, 1024)

type(y_train):  <class 'numpy.ndarray'>
y_train.dtype: float32
y_train.shape: (one_hot=False),  (60000,)

type(X_test):  <class 'numpy.ndarray'>
X_test.dtype: float32
X_test.shape: (reshape=False),  (20000, 1024)

type(y_test):  <class 'numpy.ndarray'>
y_test.dtype: float32
y_test.shape: (one_hot=True),  (20000,)
```

همانطور که در خروجی بالا مشاهده می‌کنید، داده کلی ما (حدود ۸۰ هزار عدد) به ۲ مجموعه داده train و test شکسته شده است. هر مجموعه دارای ۱۰۲۴ ستون است. این ۱۰۲۴ ستون در واقع عکس ۳۲ پیکسل در ۳۲ پیکسلی هستند که با توجه به سیاه یا سفید بودن پیکسل حاوی اعداد ۰ یا ۱ هستند که ۰ به معنی پیکسل سیاه و ۱ سفید است. هدف از تقسیم مجموعه داده ها به زیرمجموعه train و test جلوگیری از پدیده overfitting است.
در دنیای الگوریتم‌ها Overfit شدن به معنای این است که الگوریتم فقط داده‌هایی که در مجموعه آموزشی (train set) یاد گرفته است را می‌تواند به درستی پیش‌بینی کند ولی اگر داده‌ای کمی از مجموعه‌ی آموزشی فاصله داشته باشد، الگوریتمی که Overfit شده باشد، نمی‌تواند به درستی پاسخی برای این داده‌های جدید پیدا کند و آن‌ها را با اشتباهِ زیادی طبقه‌بندی می کند.
با انجام این تقسیم مدل ما ابتدا مجموعه داده آموزشی را یاد می‌گیرد و سپس بر روی مجموعه آزمایشی پیش‌بینی را انجام می‌دهد و در آخر ما نتیجه پیش‌بینی را با برچسب‌های واقعی مقایسه می‌کنیم و میزان دقت آن الگوریتم در پیش‌بینی اعداد دست‌خط به دست می‌آید.

معرفی الگوریتم‌های یادگیری ماشین استفاده شده

حال نوبت به پیاده‌سازی الگوریتم‌های یادگیری ماشین نظارت شده می‌رسد که از میان آن‌ها ۴ الگوریتم برای پیاده‌سازی این کار انتخاب شده‌اند که به توضیح مختصر آن‌ها می‌پردازیم:

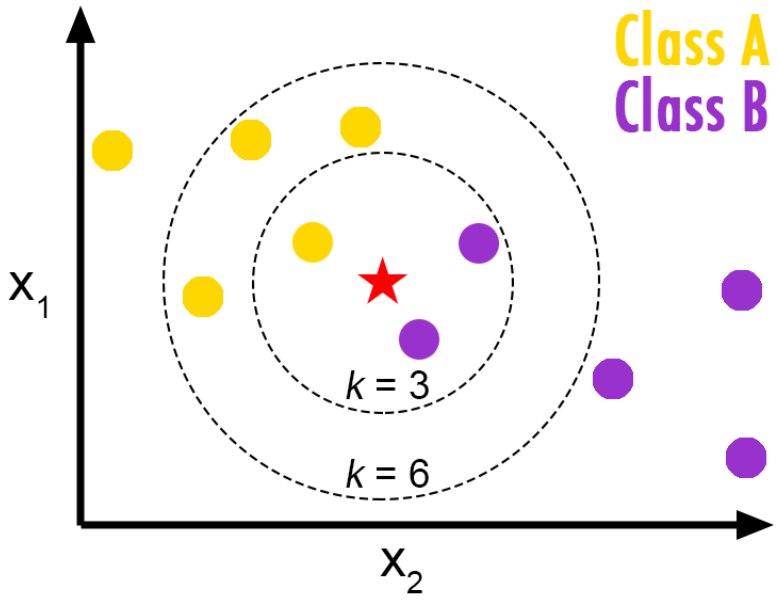
الگوریتم جنگل تصادفی (Random Forest)



در الگوریتمِ جنگل تصادفی از چندین درختِ تصمیم (برای مثال ۱۰۰ درخت تصمیم) استفاده می‌شود. در واقع مجموعه‌ای از درخت‌های تصمیم، با هم یک جنگل را تولید می‌کنند و این جنگل می‌تواند تصمیم‌های بهتری را (نسبت به یک درخت) اتخاذ نماید.

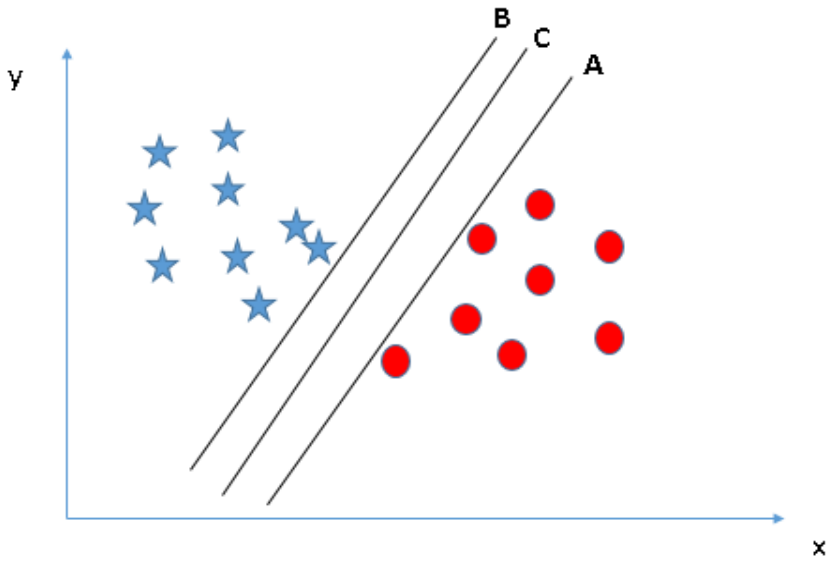
در الگوریتم جنگل تصادفی به هر کدام از درخت‌ها، یک زیرمجموعه‌ای از داده‌ها تزریق می‌شود. برای مثال اگر مجموعه داده‌ی شما دارای ۱۰۰۰ سطر (یعنی ۱۰۰۰ نمونه) و ۵۰ ستون (یعنی ۵۰ ویژگی) بود، الگوریتمِ جنگلِ تصادفی به هر کدام از درخت‌ها، ۱۰۰ سطر و ۲۰ ستون، که به صورت تصادفی انتخاب شده‌اند و زیر مجموعه‌ای از مجموعه‌ی داده‌ها هست، می‌دهد.

الگوریتم K نزدیک‌ترین همسایه (KNN)



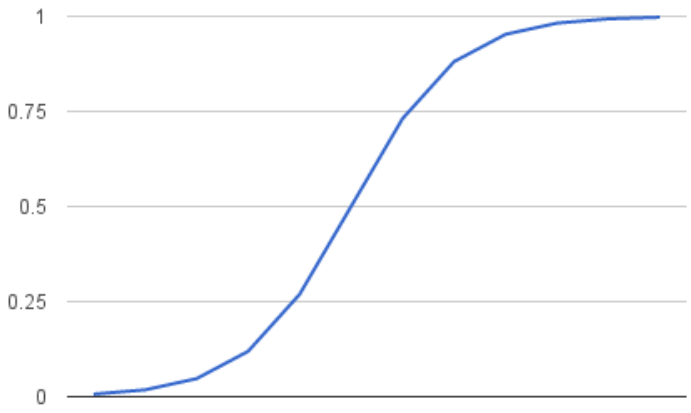
این الگوریتم یک متد آمار ناپارامتری است که برای طبقه‌بندی آماری و رگرسیون استفاده می شود. در هر دو حالت کی شامل نزدیک ترین مثال آموزشی در فضای داده ای می باشد و خروجی آن بسته به نوع مورد استفاده در طبقه بندی و رگرسیون متغیر است. در حالت طبقه بندی با توجه به مقدار مشخص شده برای کی، به محاسبه فاصله نقطه ای که میخواهیم برچسب آن را مشخص کنیم با نزدیک ترین نقاط میپردازد و با توجه به تعداد رای حداکثری این نقاط همسایه، در رابطه با برچسب نقطه مورد نظر تصمیم گیری می‌کنیم. برای محاسبه این فاصله میتوان از روش های مختلفی استفاده کرد که یکی از مطرح ترین این روش ها، فاصله اقلیدسی است. در حالت رگرسیون نیز میانگین مقادیر بدست آمده از کی خروجی آن می باشد.

بند ماشین بردار پشتیبان (Support Vector Machine) یا به اختصار SVM



این روش از جملهٔ روش‌های نسبتاً جدیدی است که در سال‌های اخیر کارایی خوبی نسبت به روش‌های قدیمی‌تر برای طبقه‌بندی نشان داده‌است. مبنای کاری دسته‌بندی کنندهٔ SVM دسته‌بندی خطی داده‌ها است و در تقسیم خطی داده‌ها سعی می‌کنیم خطی را انتخاب کنیم که حاشیه اطمینان بیشتری داشته باشد. حل معادله پیدا کردن خط بهینه برای داده‌ها به وسیله روش‌های QP که روش‌های شناخته شده‌ای در حل مسائل محدودیت‌دار هستند صورت می‌گیرد. قبل از تقسیمِ خطی برای اینکه ماشین بتواند داده‌های با پیچیدگی بالا را دسته‌بندی کند داده‌ها را به وسیلهٔ تابعِ phi به فضای با ابعاد خیلی بالاتر می‌بریم. برای اینکه بتوانیم مسئله ابعاد خیلی بالا را با استفاده از این روش‌ها حل کنیم از قضیه دوگانی لاگرانژ برای تبدیلِ مسئلهٔ مینیمم‌سازی مورد نظر به فرم دوگانی آن که در آن به جای تابع پیچیدهٔ phi که ما را به فضایی با ابعاد بالا می‌برد، تابعِ ساده‌تری به نامِ تابع هسته که ضرب برداری تابع phi است ظاهر می‌شود استفاده می‌کنیم. از توابع هسته مختلفی از جمله هسته‌های نمایی، چندجمله‌ای و سیگموید می‌توان استفاده نمود.

رگرسیون لجستیک (Logistic Regression)



رگرسیون لجستیک یک مدل آماری رگرسیون برای متغیرهای وابسته دوسویی مانند بیماری یا سلامت، مرگ یا زندگی است. این مدل را می‌توان به عنوان مدل خطی تعمیم‌یافته‌ای که از تابع لوجیت به عنوان تابع پیوند استفاده می‌کند و خطایش از توزیع چندجمله‌ای پیروی می‌کند، به‌حساب‌آورد. منظور از دو سویی بودن، رخ داد یک واقعه تصادفی در دو موقعیت ممکنه است. به عنوان مثال خرید یا عدم خرید، ثبت نام یا عدم ثبت نام، ورشکسته شدن یا ورشکسته نشدن و ... متغیرهایی هستند که فقط دارای دو موقعیت هستند و مجموع احتمال هر یک آن‌ها در نهایت یک خواهد شد. کاربرد این روش عمدتاً در ابتدای ظهور در مورد کاربردهای پزشکی برای احتمال وقوع یک بیماری مورد استفاده قرار می‌گرفت. لیکن امروزه در تمام زمینه‌های علمی کاربرد وسیعی یافته‌است. به عنوان مثال مدیر سازمانی می‌خواهد بداند در مشارکت یا عدم مشارکت کارمندان کدام متغیرها نقش پیش‌بینی دارند؟ مدیر تبلیغاتی می‌خواهد بداند در خرید یا عدم خرید یک محصول یا برند چه متغیرهایی مهم هستند؟ یک مرکز تحقیقات پزشکی می‌خواهد بداند در مبتلا شدن به بیماری عروق کرنری قلب چه متغیرهایی نقش پیش‌بینی‌کننده دارند؟ تا با اطلاع‌رسانی از احتمال وقوع کاسته شود.

شروع پیش‌پردازش داده

حال که با الگوریتم‌های مورد نظر آشنایی پیدا کردیم نوبت به پیاده‌سازی آن می‌باشد. در پروژه‌های یادگیری ماشین پیش از پیاده‌سازی مستقیم الگوریتم روی داده سعی می‌شود با داده بیشتر آشنا شد و حتی تغییراتی در نحوه ارائه آن به الگوریتم داده می‌شود تا سرعت و دقت یادگیری افزایش یابد. یکی از این کارها کاهش بعد می‌باشد. در بعضی از موارد تعدادِ ویژگی‌های مجموعه‌ی داده بسیار زیاد است و الگوریتم‌های داده‌کاوی (مانند طبقه‌بندی یا خوشه‌بندی) در ابعادِ زیاد دچار خطا می‌شوند و یا سرعت انجامِ عملیات در آن‌ها کاهش پیدا می‌کند. همچنین در بعضی از موارد می‌خواهیم با کاهشِ تعداد ابعاد، آن‌ها را در یک نمودار یا چارت رسم کنیم. برای همین بایستی داده‌ها را به تعداد ۲ یا ۳ بُعد تبدیل کرده تا قابل نمایش باشند.

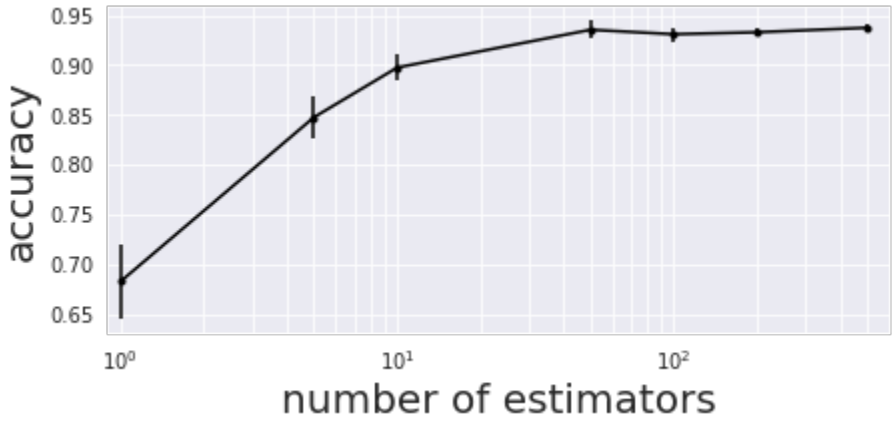
الگوریتم‌های مختلفی برای کاهش ابعاد وجود دارد. الگوریتمی که ما در این پروژه استفاده کردیم آنالیز مولفه اصلی (Principal Component Analysis) یا همان PCA است. این الگوریتم می‌تواند داده‌ها به ابعاد کوچک‌تر تبدیل کند. همچنین الگوریتم‌هایی مانند 2chi نیز می‌توانند با شناساییِ ویژگی‌ها ارزشمند، آن‌ها را از ویژگی‌های غیرِ ارزشمند جدا کرده و به نوعی کاهش ابعاد انجام دهند.

PCA همان‌طور که از نامش پیداست می‌تواند مولفه‌های اصلی را شناسایی کند و به ما کمک می‌کند تا به جای اینکه تمامیِ ویژگی‌ها را مورد بررسی قرار دهیم، یک سری ویژگی‌هایی را ارزشِ بیشتری دارند، تحلیل کنیم. در واقع PCA آن ویژگی‌هایی را که ارزش بیشتری فراهم می‌کنند برای ما استخراج می‌کند.

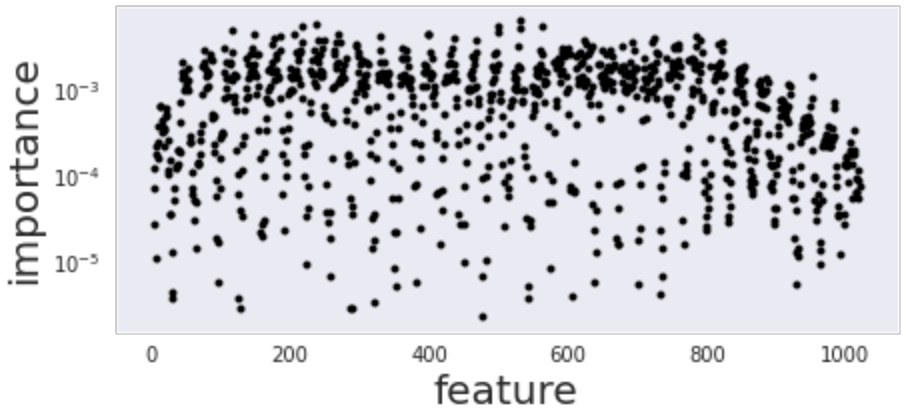
قبل از پیاده‌سازی PCA روی مجموع داده ما، ابتدا عملکرد الگوریتم جنگل تصادفی را بدون PCA بر روی ۱۰۰۰ نمونه از داده‌مان امتحان می‌کنیم.

محاسبه تعداد درخت مناسب الگوریتم جنگل تصادفی

همانطور که می‌دانید یکی از پارامترهای موجود در الگوریتم جنگل تصادفی n_estimators می‌باشد که همان تعداد درخت‌های مورد نظر جنگل ما است. ما با تعریف یک آرایه حاوی اعداد 1,5,10,50,100,200,500 دقت این الگوریتم را با تعداد درخت متفاوت می‌سنجیم.



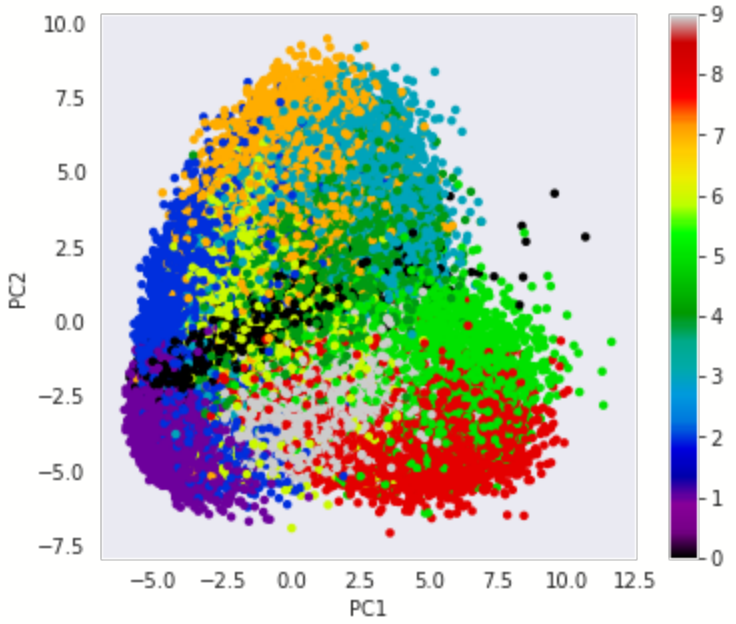
همانطور که در نمودار بالا مشاهده می‌کنید، دقت الگوریتم حول ۵۰ تا ۱۰۰ درخت اشباع می‌شود و دیگر بالا نمی‌رود.
آیا ویژگی‌هایی وجود دارد که از اهمیت ویژه ای برخوردار هستند؟ ما می توانیم این مورد را با استفاده از clf.feature_importances که کتاب‌خانه Sklearn برای ما فراهم کرده است بررسی کنیم:



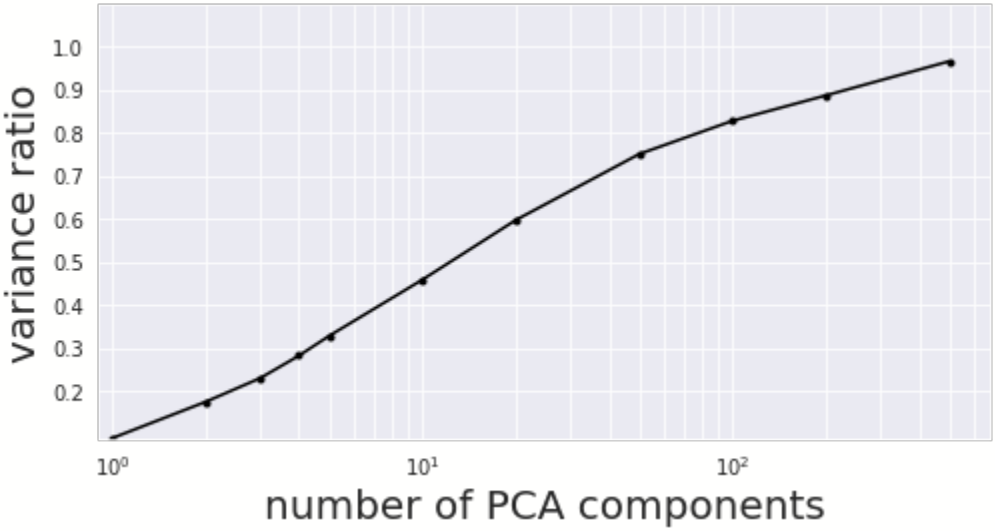
همانطور که در نتیجه می‌بینید تقریبا هیچ ویژگی (بُعد) خاصی وجود ندارد که از بقیه درجه اهمیت آن با شدت زیاد متفاوت باشد و تقریبا اکثر ویژگی‌ها در یک سطح می‌باشند.

کاهش بعد به کمک PCA

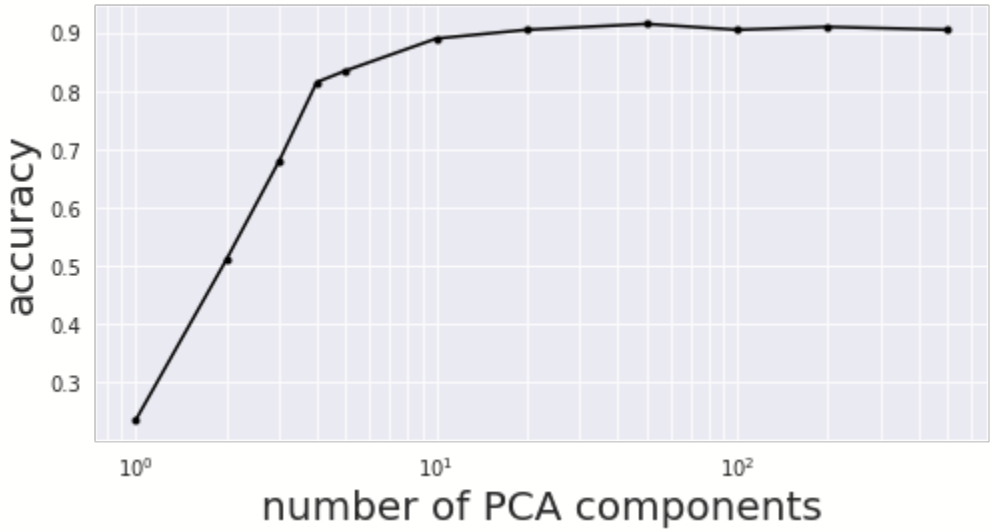
حال برای پیاده‌سازی PCA از ماژولی به همین نام که توسط Sklearn به صورت آماده پیاده‌سازی شده استفاده می‌کنیم.
(sklearn.decomposition.PCA)
ما داده‌مان را به ۲ مولفه کاهش می‌دهیم و توابع fit و transform را روی داده آموزشی اجرا می‌کنیم و نتیجه آن را بر روی نمودار رسم می‌کنیم:



همانطور که می‌بینید PCA به قدرت توانست فضای ویژگی‌ها را به خوشه‌های واضحی تقسیم کند و این تازه با ۲ مولفه صورت گرفت. اما اجرای PCA با چند متغیر کم توصیه نمی شود. در چنین مواردی، سهم هر متغیر در واریانس کل تقریباً کافی است و انتخاب بیشتر یا رد یک متغیر مشخص دشوار است. به این منظور در حرکت بعدی تعداد مولفه‌های PCA-مان را افزایش می‌دهیم تا ببینیم نتیجه چه تغییری می‌کند. همانطور که گفته شد هدف ما در آوردن مقدار مولفه‌ای است که حداکثر مقدار واریانس داده را فراهم کنید. برای دستیابی به این هدف ما از تابع pca.explained_variance_ratio استفاده می‌کنیم و نتیجه را روی نمودار رسم می‌کنیم:



همانطور که می‌بینید تقریبا ۱۰۰ مولفه PCA برای دریافت تقریبا ۹۰٪ واریانس بر روی داده کافی می‌باشد که تقریبا عدد بالای برای تعداد مولفه می‌باشد. حال سوال مهم‌تر این است که پیش‌بینی ما به عنوان تابعی از تعداد مولفه‌ها چقدر خوب است؟ بیایید این مورد بعدی را بررسی کنیم. ما یک طبقه بندی کننده kNN را در خروجی PCA آموزش خواهیم داد. نتیجه به صورت زیر می‌باشد:



به نظر می‌رسد که دقت برای تقریبا تعداد مولفه ۲۰ به بالا روی حدود ۹۰٪ ثابت می‌ماند (تقریبا مطابق با عملکرد طبقه بندی جنگل تصادفی). در حقیقت به نظر می رسد که دقت حتی برای تعداد بسیار زیاد مولفه‌ها کاهش می یابد، حتی اگر تعداد بیشتری از مولفه‌های PCA، بیشتر واریانس داده ها را دریافت کنند. کاهش دقت احتمالاً به دلیل رخداد overfitting است.

پیاده‌سازی نهایی PCA بر روی داده آموزشی و تست:

```
pca = PCA(n_components=50)
pca.fit(X_train)
transform_train = pca.transform(X_train)
transform_test = pca.transform(X_test)
```

همانطور که می‌بینید ما در نهایت ۵۰ را به عنوان تعداد مولفه در نظر گرفتیم و بر روی داده آموزشی fit و transform انجام دادیم حال آنکه بر روی داده تست فقط transform انجام دادیم.
برای توضیح تفاوت این ۲ تابع و اینکه چرا ما روی داده تست فقط transform انجام می‌دهید باید به پیش‌زمینه ریاضی آن برگردیم:
ما به منظور استانداردسازی داده خود (داشتن میانگین ۰ و ارور استاندارد واحد)، آن را از میانگین خود کم می‌کنیم و بر انحراف معیار تقسیم می‌کنیم:

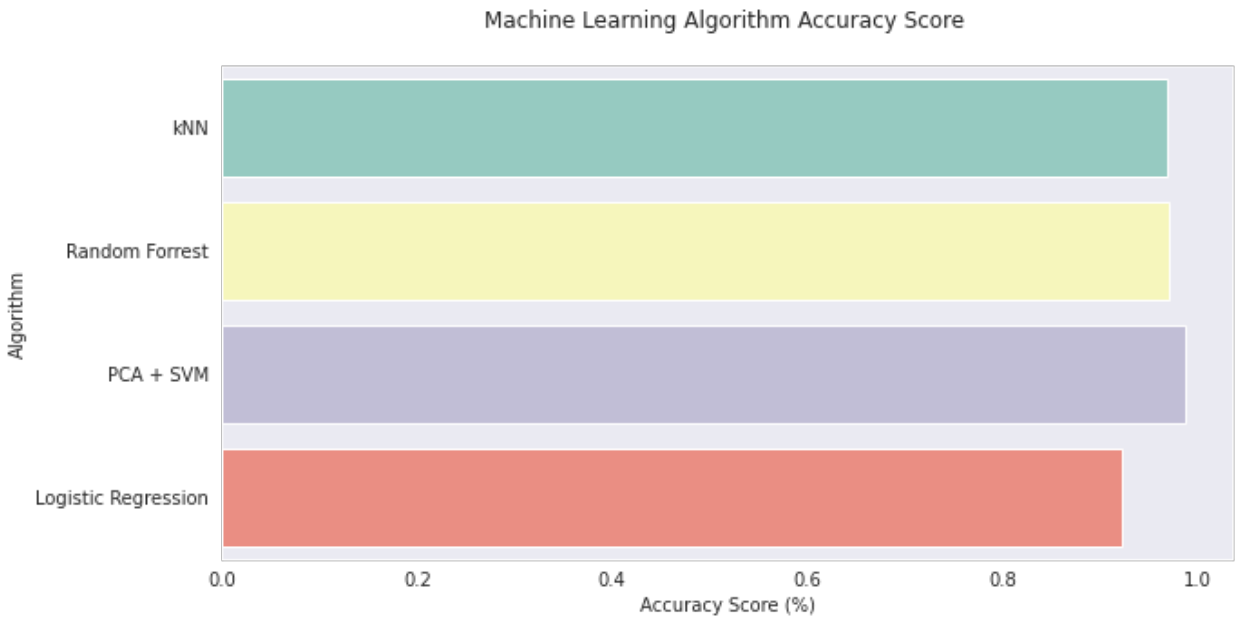
$$z' = \frac{x - \mu}{\sigma} \quad \text{یا} \quad z = ln(x)$$

شما این کار را در مجموعه آموزشی داده‌ها انجام می‌دهید. اما سپس شما باید همان تغییر را در مجموعه تست خود (مثلا در cross-validation) یا برای مثال‌های تازه بدست آمده قبل از پیش بینی اعمال کنید. اما شما باید از همان دو پارامتر μ و σ که برای استانداردسازی مجموعه آموزشی استفاده کردید، استفاده کنید.
از این رو، هر بار اجرای تابع ()fit بر روی داده این پارامترها (μ و σ) را محاسبه می‌کند و به عنوان متغییر درونی ذخیره می‌کند تا هر جا نیاز بود با دادن تابع ()transform بر روی داده مورد نطر تغییر را اعمال کنید.

پیاده‌سازی الگوریتم‌ها

حال که مرحله پیش‌پردازش داده‌ها نیز به اتمام رسید نوبت به اجرای الگوریتم‌های معرفی شده در بالا بر روی داده پردازش‌شده‌مان است.

پیاده‌سازی هر الگوریتم در source code برنامه موجود است و ما در اینجا فقط به نمایش دقت هر الگوریتم اکتفا می‌کنیم:



همانطور که می‌بینید ترکیب PCA و SVM بر روی داده ما بیشترین دقت در یادگیری و پیش‌بینی را با تقریبا ۹۸٪ داشته است.

تبدیل به مدل قابل استفاده در اپلیکیشن‌های iOS

برای این منظور از کتاب‌خانه coremltools که توسط Apple برای پایتون عرضه شده استفاده می‌کنیم. این کتاب‌خانه به راحتی امکان تبدیل مدل یادگیری شده در پایتون را به مدل قابل استفاده در دستگاه‌های iOS دارد. برای نصب آن دستور زیر را اجرا کنید:

```
pip install coremltools
```

تابع تبدیل نوشته شده برای این کار در source code برنامه و بعد از دستورات import وجود دارد.

فاز دوم: یادگیری عمیق

قبل از اینکه به فاز نهایی، یعنی فاز پیاده‌سازی اپلیکیشن، بپردازیم، نحوه پیاده‌سازی مدل یادگیری عمیق برای پیش‌بینی در لحظه اعداد دست‌خط فارسی را توضیح می‌دهیم.

یادگیری عمیق چیست؟

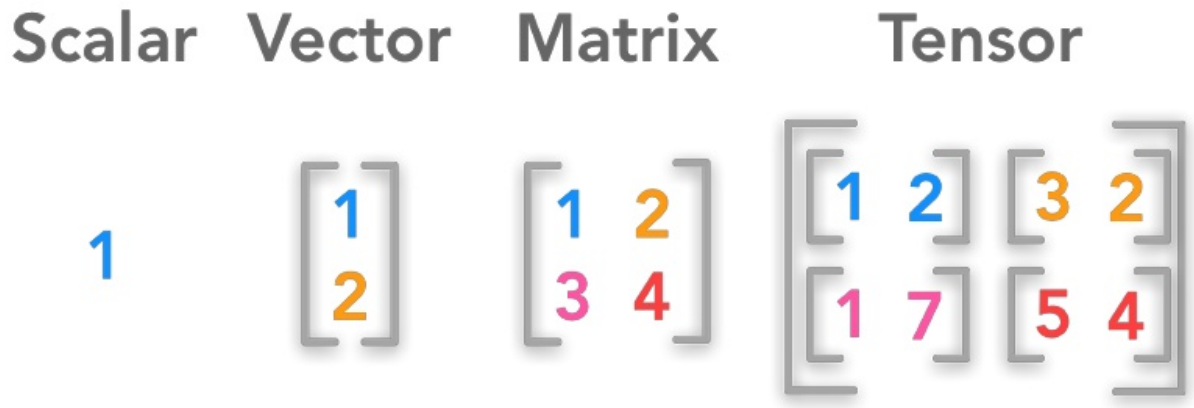
در یک تعریف کلی، یادگیری عمیق، همان یادگیری ماشین است، به طوری که در سطوح مختلف نمایش یا انتزاع (abstraction) یادگیری را برای ماشین انجام میدهد .با این کار، ماشین درک بهتری از واقعیت وجودی داده ها پیدا کرده و میتواند الگوهای مختلف را شناسایی کند.

بر اساس تعریف مشهور، یادگیری عمیق در واقع همان یادگیری به وسیله شبکه های عصبی ای هستند که دارای لایه های ی پنهانی (Hidden Layers) زیادی می باشند .هر چقدر در لایه های یک شبکه عصبی عمیق جلو تر میرویم، به مدلهای پیچیده تر) و کامل تری (میرسیم.

یادگیری عمیق (Deep Learning) در حوزه های مختلفی مانند، دسته بندی تصاویر (تشخیص تصاویر)، دسته بندی متون، تشخیص صدا و... کاربردهای فراوانی دارد.

آماده‌سازی داده

آماده‌سازی داده برای یادگیری در اینجا کمی با یادگیری ماشین متفاوت است. ما در اینجا با tensor ها سر و کار داریم. Tensor در واقع ماتریسی است که هر کدام از خانه‌های آن به جای این‌که یک عدد داشته باشند، می‌تواند چندین عدد را در خود جای دهد. ما در اینجا به جای استفاده از کتاب‌خانه Sklearn از Tensorflow و Keras استفاده می‌کنیم که به توابع Keras می‌توانید مستقیما از Tensorflow دسترسی داشته باشید.



Tensorflowیک کتابخانه متن باز برای زبان پایتون است که توسط گوگل در سال ۲۰۱۵ انتشار پیدا کرد .این کتابخانه در به برنامه نویسان کمک میکند تا بتواند طراحی و پیاده سازی شبکه های عصبی عمیق (Deep Learning Network)را به سادگی انجام دهند.

هدف از آماده‌سازی داده تبدیل داده خام به قالب مورد نیاز برای ساخت شبکه عصبی‌مان می‌باشد. (برای مشاهده تابع آماده‌سازی داده به source code مراجعه کنید). در تابع نوشته شده ابتدا ابعاد مورد نیاز عکس‌مان را مشخص می‌کنیم که ما در اینجا ۳۲ در ۳۲ در نظر گرفتیم (۲۸ در ۲۸ نیز زیاد مورد استفاده قرار می‌گیرد و جوری به استاندارد ابعاد عکس در یادگیری شبکه عصبی تبدیل شده است).

حال پارامتر image_data_format را از backend کتاب‌خانه Keras چک می‌کنیم. در واقع backend ها در Keras یکی از موارد زیر می‌تواند باشد:
TensorFlow, CNTK, Theano

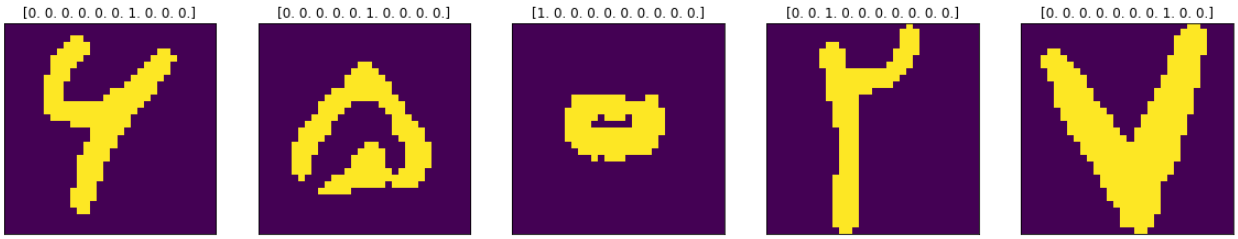
چرا این پارامتر image_data_format بسیار مهم است؟

ما برای تبدیل داده‌مان به مدل قابل استفاده توسط Tensorflow باید داده آموزشی و تست ورودی‌مان را به یک تنسور ۴ بعدی تبدیل کنیم. یکی از این ۴ بعد، بعد channel تصویر می‌باشد که جایگاه آن در backend های مختلف متفاوت می‌باشد.

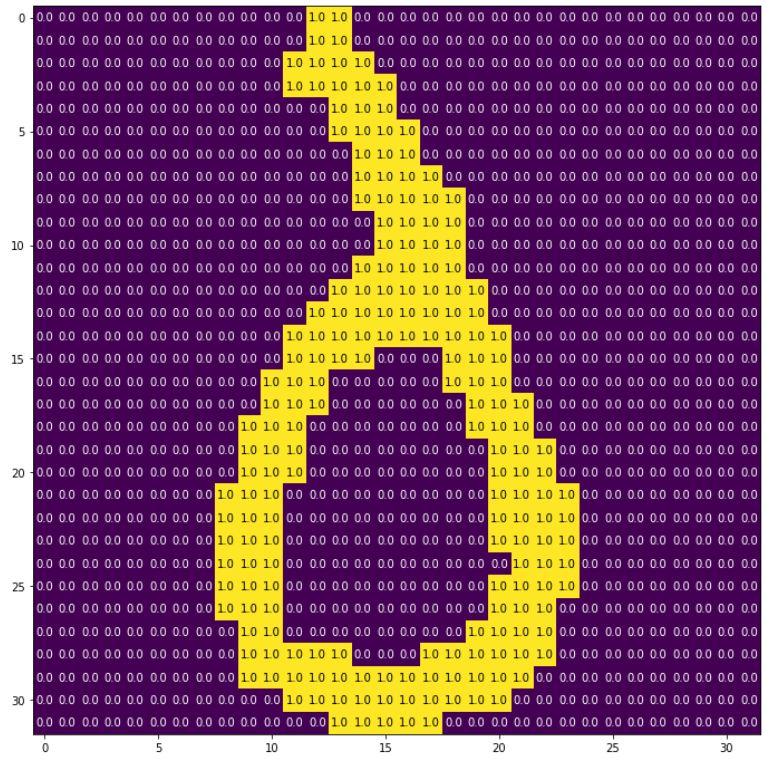
پارامتر "backend" باید یا "cntk" ، "tensorflow" یا "theano" باشد. هنگام تعویض backend، حتماً پارامتر image_data_format را نیز تغییر دهید. برای بک‌اندهای "tensorflow" یا "cntk" ، باید "channel_last" باشد. برای "theano" ، باید "channel_first" باشد.

بعد از اینکه تنسور خود را با توجه به نوع فرمت داده تصویر ساختیم حال نوبت به انکود کردن داده آموزشی و تست خروجی‌مان می‌رسد. ما به این منظور از categorical encoder استفاده کردیم که وکتور کلاس (اعداد صحیح) را به ماتریس کلاس باینری تبدیل می کند.

داده تبدیل شده‌مان به شکل زیر خواهد بود:



اگر بخواهیم به صورت جزیی تر به یکی از این تصاویر تبدیل شده نگاه کنیم می‌بینیم که پیکسل‌هایی که خاموش هستند ۰ و پیکسل‌هایی که حاوی عدد ما هستند ۱ است:

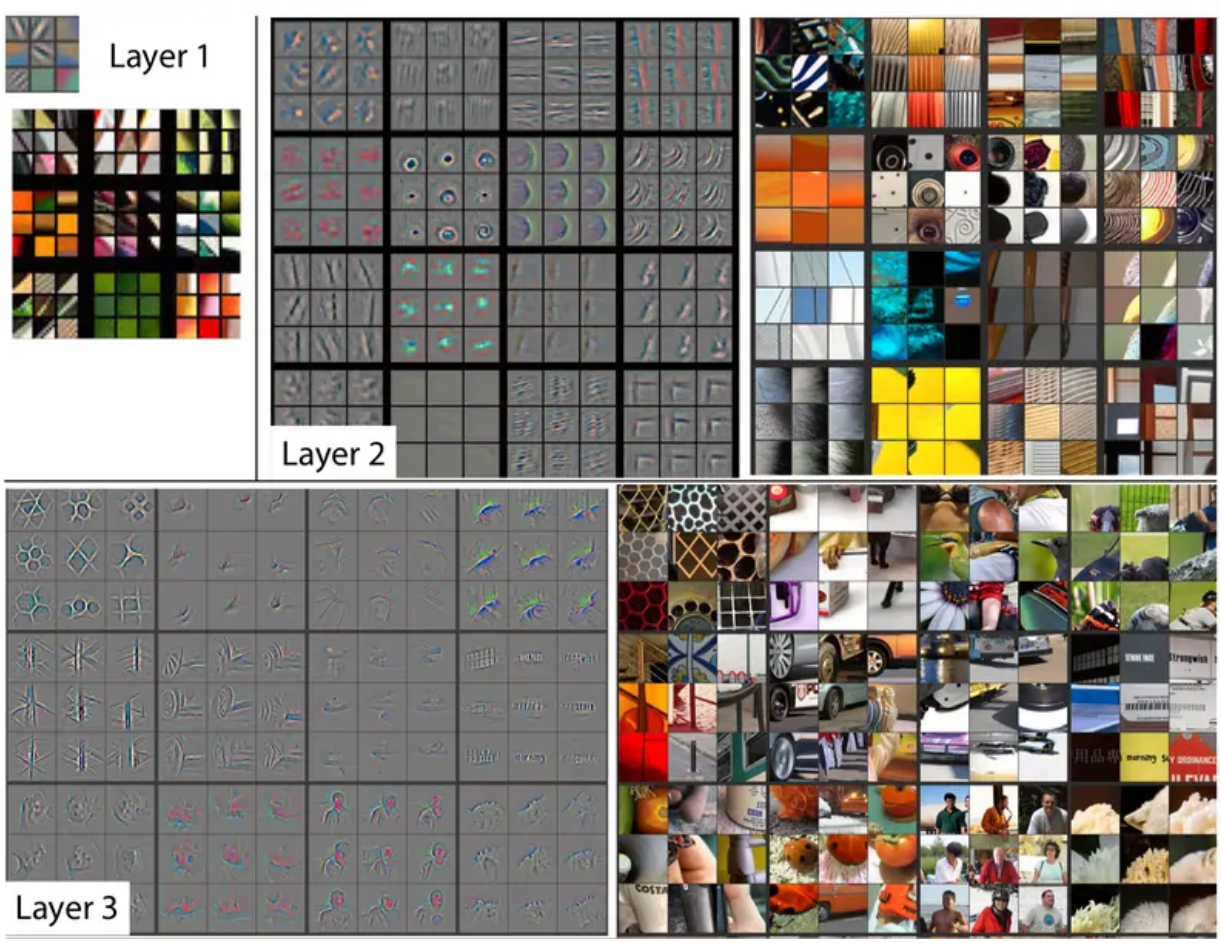


حال لایه‌های شبکه خود را می‌سازیم تا با استفاده از آن بتوانیم اعداد دستخط را تشخیص دهیم.
ابتدا ابعاد ورودی و تعداد کلاس‌ها را مشخص می‌کنیم. ورودی ما ۳ بعد دارد که به صورت ۳۲ در ۳۲ در ۱ تبدیل شده و تعداد کلاس‌ها (خروجی) هم ۱۰ عدد می‌باشد که اعداد ۰ تا ۹ را شامل می‌شود. قبل از اینکه نحوه چینش لایه‌ها را توضیح دهیم ابتدا به معرفی ۲ تکنیک Pooling و Convolution در یادگیری عمیق می‌پردازیم. این دو تکنیک از پراستفاده‌ترین‌ها در تشخیص عکس می‌باشند توسط یادگیری عمیق می‌باشند.

تکنیک Convolution

این تکنیک در واقع یک این انتقال (transformation) در تصویر است که تقریبا از معرفی آن 20 سال می‌گذرد و ما روزانه در نرم‌افزارهای ویرایش تصویر با آن سر و کار داریم. چیزهایی مانند تیز کردن (sharpness) تصویر یا تار شدن آن یا پیدا کردن لبه ها، اساسا به کمک Convolution است. در اجرا هم یک ماترسی کوچک، مثلا ماتریس 3x3، را بر روی هر پیکسل تصویر خود قرار داده و این مقدار را با پیکسل های همسایه ضرب کرده و سپس نتایج آن انتقال را در یک تصویر جدید جمع آوری می‌کند.

بنابراین در یادگیری عمیق به یک لایه که قادر به اعمال چنین فیلتر بر روی تصویر می‌باشد، یک لایه convolutional گفته می‌شود. اگر چند لایه این شکلی را روی هم قرار دهیم، هر یک از این لایه ها فیلترهای مخصوص به خود را پیدا می کنند که مفید می‌باشد. علاوه بر این، هر یک از این فیلترها پیچیده‌تر می‌شوند و قادر به شناسایی ویژگی های دقیق‌تراند.

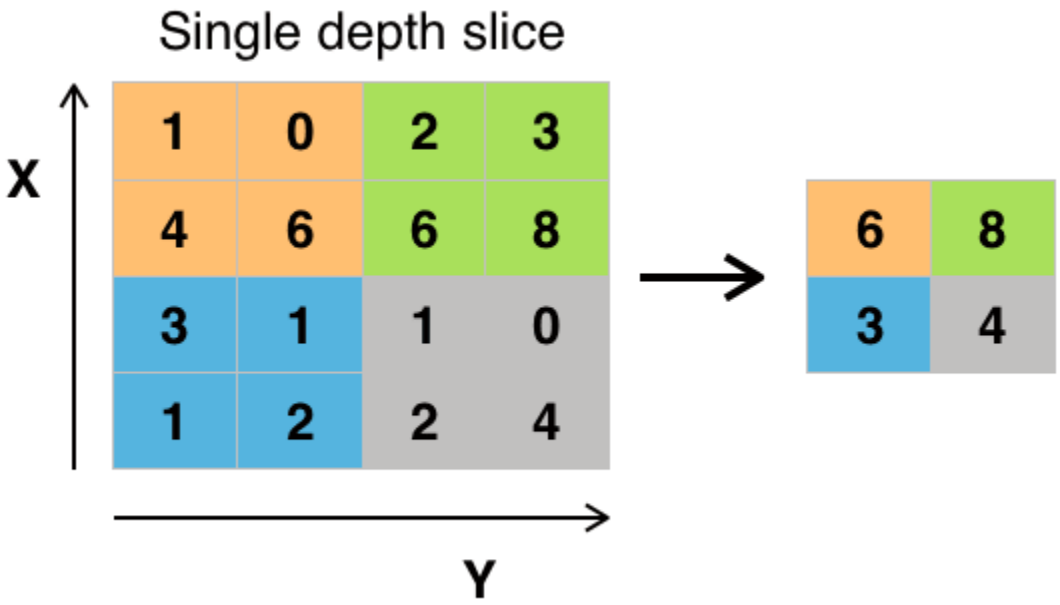


در تصویر بالا جعبه‌ها و تصاویر خاکستری را مشاهده می کنید. یک راه عالی برای نشان دادن این فیلترها نشان دادن فعال‌سازی‌ها یا پیچ‌های حلقوی این جعبه‌های خاکستری است. تصاویر نمونه‌هایی هستند که باعث ایجاد این فیلترها می شوند. و یا برعکس، این عکس‌هایی است که این فیلترها به خوبی تشخیص می دهند.

به عنوان مثال در لایه اول متوجه خواهید شد که شبکه لبه های عمودی، افقی و مورب را تشخیص می‌دهد. در لایه دوم آن کمی "باهوش تر" است و قادر به کشف موارد پیچیده‌تر است، به عنوان مثال چشم‌ها یا گوشه‌های فریم و غیره. در لایه سوم آن کمی باهوش‌تر می‌شود و قادر به تشخیص نه تنها وسایل گرد بلکه مواردی است که به عنوان مثال مانند لاستیک اتومبیل است. این لایه‌بندی اغلب برای بسیاری از لایه‌ها ادامه می یابد. برخی از شبکه ها بیش از 200 عدد از این لایه‌ها را دارند. به همین دلیل به آن‌ها عمیق گفته می شود. بنابراین معمولا افزودن بیشتر و بیشتر این لایه ها باعث می شود شبکه در تشخیص چیزها بهتر عمل کند اما همچنین کندتر می کند.

تکنیک Pooling

دومین کلمه ای که شاید درمعماری زیاد مشاهده کنید، کلمه جمع کردن (Pooling) است. در اینجا این ترفند بسیار ساده است:
چند پیکسل (مثلا ۲ در ۲) در کنار هم را مشاهده کنید و به سادگی بزرگترین مقدار را وردارید (که به آن مقدار max-pooling هم گفته می‌شود)
در تصویر زیر از این ترفند برای هر منطقه ۲ در ۲ رنگی استفاده شده است و خروجی یک تصویر بسیار کوچکتر است. حالا چرا این کار را می کنیم؟



پاسخ ساده است، به منظور متغیر بودن اندازه. ما سعی می کنیم تا چندین بار تصویر را به پایین و بالا بکشیم تا بتوانیم یک گورخر را که واقعاً نزدیک دوربین است، در مقابل یکی از آنها تشخیص دهیم که فقط در مسافت دور قابل مشاهده است.

یادگیری عمیق

حال به مدل خودمان بر می‌گردیم. شما می‌توانید کد استفاده شده به این منظور را در source code برنامه بخش یادگیری عمیق مشاهده کنید.

ما برای دستیابی به هدفمان از شبکه عصبی کانولوشنی (CNN) استفاده کردیم. CNN ها شبکه های عمیقی هستند که برای تشخیص تصویر، شی و حتی تشخیص گفتار استفاده می‌شوند. این شبکه‌ها که توسط یان لکون در دانشگاه نیویورک توسعه داده شده اند، در حال حاضر در صنعت فناوری، مانند فیسبوک برای تشخیص چهره استفاده می شود.

همانطور که در لایه‌های مختلف شبکه‌مان مشاهده می‌کنید ما از ۶ لایه Conv2d با تعداد فیلترهای مختلف استفاده کردیم. از ۶۴ تا ۱۲۸ فیلتر. همچنین از ۳ لایه Max Poolin دو در دو بهره بردیم. در وسط و انتهای هر سری یک LeakyReLU استفاده کرده‌ایم. در واحد یکسو شده‌ی بانشت یا LeakyReLU، واحدها یک شیب نا صفر را در هنگامی که واحد فعال نیست عبور می‌دهند:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

واحد یکسو شده‌ی پارامتردار این ایده را با در نظر گرفتن ضریب نشت به عنوان یک پارامتر قابل یادگیری همراه بقیه‌ی پارامترهای شبکه‌های عصبی کامل‌تر می‌کنند.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$

اما قبل از پیاده‌سازی LeakyReLU ما لایه نرمال‌سازی را اضافه کرده‌ایم که در مورد ما BatchNormalization استفاده شده است. نرمال سازی دسته ای روشی است که به طور خودکار ورودی های یک لایه در یک شبکه عصبی با یادگیری عمیق را استاندارد سازی می کند. همچنین از این لایه نرمال‌سازی به منظور تسریع روند آموزش شبکه مورد استفاده قرار می‌گیرد. در پایان نیز از لایه‌های Dense و Dropout استفاده کرده‌ایم. این ۲ لایه را با یک مثلا توضیح می‌دهیم:

فرض کنید که شما یک بردار ورودی n بعدی به نام u دارید، $u \in R^{n \times 1}$:

یک لایه متراکم (Dense) نشان دهنده ضرب بردار ماتریس است. (با فرض اینکه اندازه دسته شما ۱ باشد) مقادیر موجود در ماتریس پارامترهای قابل آموزش هستند که در طول پردازش مجدد به‌روز می شوند.

بنابراین شما یک بردار n بعدی به عنوان خروجی دریافت می کنید. بنابراین یک لایه متراکم برای تغییر ابعاد بردار شما استفاده می شود.

یک لایه Dropout برای منظم‌سازی استفاده می شود که به طور تصادفی برخی از ابعاد بردار ورودی خود را با احتمال $keepprob$ صفر می کنید. یک لایه Dropout هیچ پارامتر قابل آموزش ندارد، یعنی هیچ چیزی در طول بخش عقب‌گرد یادگیری پس انتشار خطا به روز نمی شود.
همان‌طور که گفتیم پایه‌ی یادگیری در شبکه‌های عصبی تکرار است .یکی از روش‌های بسیار پرکاربرد برای تکرار در شبکه‌های عصبی روشِ پسْ انتشارِ خطا (back propagation of error) است .در این روش، در هر دور)یعنی در هر تکرار (دو مرحله خواهیم داشت .مرحله‌ی اول حرکت رو به جلو (feed forward) است که با ضربِ داده‌های ورودی در وزن‌ها و سپس جمع آن با انحراف انجام می‌شود .سرانجام در همان مرحله‌ی اول به یک خروجی می‌رسیم که احتمالاً با خروجیِ واقعی تفاوت دارد .اینجاست که توسط تابع ضرر مشخص می‌کنیم که مرحله‌ی feed forward چه مقدار خطایی داشته است.
برای اطمینان از اینکه مقدار مورد نظر از جمع بردارها در صورت عدم اجرای Dropout با اجرای آن ثابت می‌ماند، ابعاد باقی‌مانده که مقدار آن ۰ تنظیم نشده با مقدار $\frac{1}{keepprob}$ گسترش داده می‌شود.

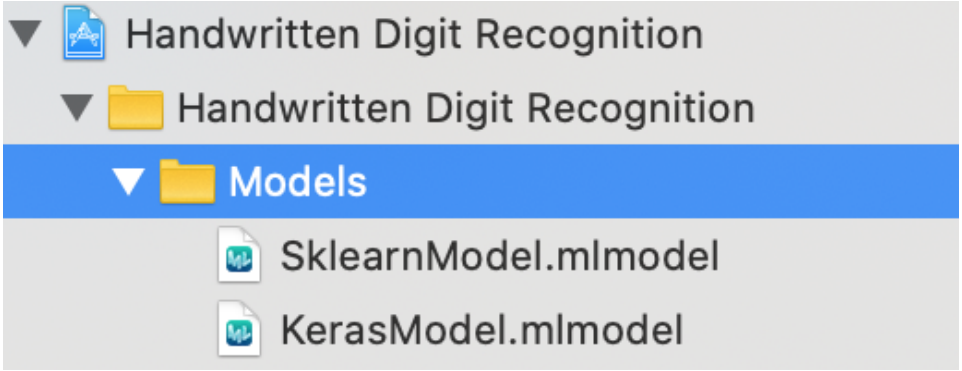
سپس، و بعد از ساخت لایه‌ها، ما با استفاده از یک تابع loss و بهینه‌ساز شبکه‌مان را کامپایل می‌کنیم: در مورد ما categorical_crossentropy را انتخاب می کنیم، زیرا چندین کلاس داریم (اعداد ۰ تا ۹). Keras بهینه‌سازهای مختلفی را ارائه می دهد، بنابراین سعی کنید چند مورد را امتحان کنید و موردی که برای پروژه شما بهتر است استفاده کنید. در مورد ما بهینه‌ساز Adam به خوبی کار می‌کند.

بعد از آموزش (train) شبکه‌مان، مدلی را بدست می‌آوریم که دقت آن ۹۹٪ است که با توجه به زیرساخت های شبکه ای بسیار ساده، بسیار عالی است. اکنون الگویی داریم که به خوبی می تواند اعداد ۰ تا ۹ را از نمایشگر پیکسلی ۳۲ در ۳۲ پیش‌بینی کند.

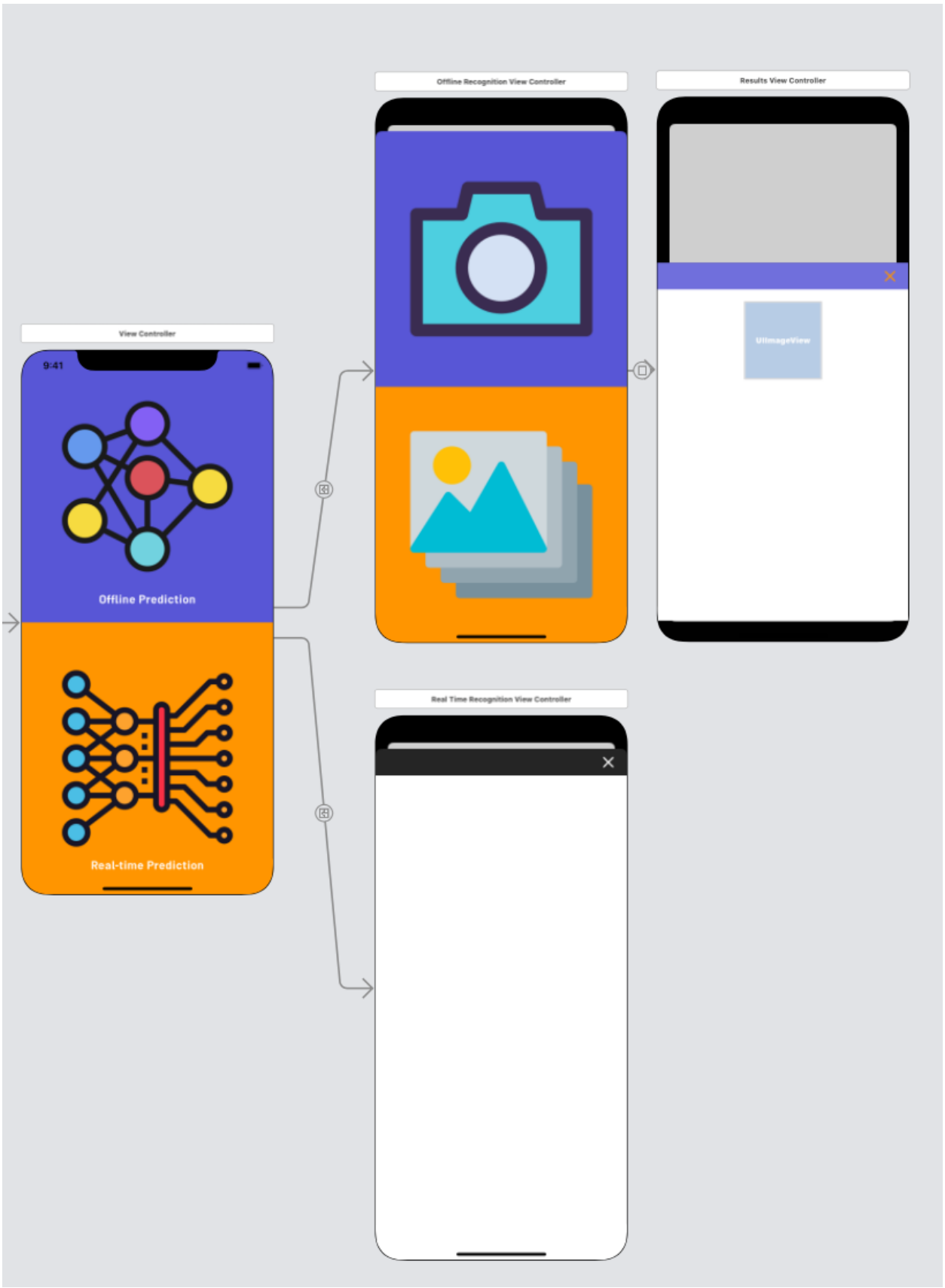
فاز سوم: اپلیکیشن iOS

حال که مدل‌ها یادگیری شدند و خروجی‌های مخصوص اپلیکیشن iOS نیز گرفته شد نوبت به پیاده‌سازی اپلیکیشن iOS ما است که مراحل آن به شکل زیر می‌باشد.

ابتدا Xcode را بر روی دستگاه Mac خود نصب کنید و یک پروژه Single View Application ایجاد کنید.
مدل‌های .mlmodel خود را به پروژه اضافه کنید (مطمئن شوید علاوه بر رفرنس، یک کپی از فایل نیز به پروژه اضافه شده).



حال با توجه به کوچکی پروژه و به کمک استوری‌بود UI مورد نظر خود را پیاده‌سازی کنید. در مورد ما، یک صفحه انتخاب نوع پیش‌بینی (آفلاین یا درلحظه) و صفحه جزییات هر کدام کافی می‌باشد. برای حالت آفلاین ما این حالت را برای کاربر قرار می‌دهیم تا از بین گالری یا دوربین خود یکی را انتخاب کند و عکسی را به ما بدهد تا بتوان پردازش مورد نیاز را انجام دهدیم و سپس در یک مدال به او نتیجه به دست آمده و درصد احتمال داده شده برای هر کلاس (در اینجا ۰ تا ۹) را به کاربر نشان می‌دهیم. برای حال درلحظه نیز ما یه مدال به همراه دوربین به کاربر نشان می‌دهیم و که با استفاده از فریموورک AVFoundation که یکی از امکانات آن ضبط در لحظه اتفاقات دوربین می‌باشد استفاده می‌کنیم تا بتوانیم هر لحظه از فریم موجود در دوربین استفاده کرده و پیش‌بینی را انجام دهیم. برای این حالت نیز یک جعبه متنی بر روی دوربین قرار دادیم که هر لحظه پس از گرفتن نتیجه پیش‌بینی بروز می‌شود.


Offline Recognition View Controller
Results View Controller
View Controller
9:41
Offline Prediction
UIImageView
Real-time Prediction
Real Time Recognition View Controller

پیش‌بینی حالت آفلاین

به منظور گرفتن نتیجه از حالت آفلاین و عکس وارد شده توسط کاربر ما ابتدا باید به جزییات مدل یادگیری ماشین آموزش دیده شده توجه کنیم:

Name	Type
▼ Inputs	
input	MultiArray (Double 1024)
▼ Outputs	
classLabel	Int64
classProbability	Dictionary (Int64 → Double)

همانطور که مشاهده می‌کنید مدل کا به عنوان ورودی یک آرایه می‌گیرد نه عکس پس اولین کار ما باید تبدیل عکس ورودی به یک آرایه قابل استفاده در مدلمان باشد.

```
func generateMultiArrayFrom(image: UIImage) -> MLMultiArray? {
  guard let data = try? MLMultiArray(shape: [32,32], dataType: .double) else {
    return nil
  }

  let pixelColors = image.getPixels()

  for (idx,color) in pixelColors.enumerated() {
    var grayscale: CGFloat = 0
    var alpha: CGFloat = 0

    color.getWhite(&grayscale, alpha: &alpha)

    if grayscale == 0.0 {
```

```
      data[idx] = 1.0
    } else {
      data[idx] = 0.0
    }

  }

  return data
}
```

با استفاده از تابع بالا ابتدا عکس را تبدیل به پیکسل‌های عددی می‌کنیم، سپس روی آن یک حلقه می‌زنیم و مقدار سفیدی هر پیکسل را گرفته و با ۰ یا ۱ جایگزین می‌کنیم تا بتواند مورد استفاده مدل ما که با آرایه‌ای از ۰ و ۱ ها کار می‌کند باشد.

حال یک تابع به شکل زیر می‌نویسیم که پس از تبدیل عکس ورودی به آرایه مورد نظر پیش‌بینی را انجام داده و نتیجه را به مدال نتایج منتقل می‌کند.

```
func recognizeWithLocalModel(_ image: UIImage) {
  if let data = generateMultiArrayFrom(image: image) {
    guard let modelOutput = try? SklearnModel().prediction(input: data) else {
      return
    }

    if let result = modelOutput.classLabel as Int64?, let proba = modelOutput.classProbability as [Int64 : Double]? {
      notificationGenerator.notificationOccurred(.success)
      self.proba = proba
      self.result = result
      performSegue(withIdentifier: showResultsSegue, sender: nil)
    } else {
      notificationGenerator.notificationOccurred(.error)
      print("no result available")
    }
  }
}
```

پیش‌بینی درلحظه

حال وقتی به مدل یادگیری داده شده یادگیری عمیق خود نگاه می‌کنیم می‌بینیم که به عنوان ورودی یک عکس سیاه و سفید ۳۲ در ۳۲ می‌گیرد (همانطور که موقع ساخت شبکه تعریف کرده بودیم).

Name	Type
▼ Inputs	
conv2d_input	Image (Grayscale 32 x 32)
▼ Outputs	
Identity	Dictionary (String → Double)
classLabel	String

```
func startCaptureSession() {

    var deviceInput: AVCaptureDeviceInput!

    let videoDevice = AVCaptureDevice.DiscoverySession(deviceTypes: [.builtInWideAngleCamera], mediaType:
.video, position: .back).devices.first
    do {
      deviceInput = try AVCaptureDeviceInput(device: videoDevice!)
    } catch {
      print("Could not create video device input: \(error)")
      return
    }

    session.beginConfiguration()
    session.sessionPreset = .vga640x480 // Model image size is smaller.

    guard session.canAddInput(deviceInput) else {
      print("Could not add video device input to the session")
```

```
    session.commitConfiguration()
    return
  }
  session.addInput(deviceInput)

  if session.canAddOutput(videoDataOutput) {
    session.addOutput(videoDataOutput)
    // Add a video data output
    videoDataOutput.alwaysDiscardsLateVideoFrames = true
    videoDataOutput.videoSettings = [kCVPixelBufferPixelFormatTypeKey as String:
Int(kCVPixelFormatType_420YpCbCr8BiPlanarFullRange)]
    videoDataOutput.setSampleBufferDelegate(self, queue: videoDataOutputQueue)
  } else {
    print("Could not add video data output to the session")
    session.commitConfiguration()
    return
  }

  let captureConnection = videoDataOutput.connection(with: .video)
  // Always process the frames
  captureConnection?.isEnabled = true
  do {
    try  videoDevice!.lockForConfiguration()
    let dimensions = CMVideoFormatDescriptionGetDimensions((videoDevice?.activeFormat.formatDescription)!)
    bufferSize.width = CGFloat(dimensions.width)
    bufferSize.height = CGFloat(dimensions.height)
    videoDevice!.unlockForConfiguration()
  } catch {
    print(error)
  }

  session.commitConfiguration()

  previewLayer = AVCaptureVideoPreviewLayer(session: session)
  previewLayer.frame = view.frame
  view.layer.addSublayer(previewLayer)

  session.startRunning()

 }
```

ما ابتدا در تابع بالا، که برای گرفتن تصویر در لحظه از دوربین کاربر می‌باشد، دوربین کاربر را تشخیص دادیم، سپس یک session شروع کردیم و اندازه فریم را ۶۴۰ در ۴۸۰ قرار دادیم. این کار به ۲ دلیل صورت گرفته:
۱) استفاده از کل ظرفیت صفحه دوربین کاربر باعث ایجاد لود سنگین و داغی سریع روی موبایل طرف می‌شود.
۲) همانطور که در ورودی مشاهده کردید، ورودی مورد نظر ما ۳۲ در ۳۲ است که خیلی کوچکتر از فریم ما می‌باشد پس نیازی به تمام ظرفیت نیست.
سپس دوربین را به session اضافه کردیم و شروع به ضبط کردیم. شما وقتی session را شروع می‌کنید به طورخودکار یک تابع با نام captureOutput در هر لحظه صدا زده می‌شود که به ما خروجی را در هر لحظه در صورت وجود می‌دهد. پیاده‌سازی body این تابع دست برنامه‌نویس می‌باشد که چطور از خروجی استفاده کند. ما بدین‌صورت پیاده‌سازی کرده‌ایم:

```
func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection: AVCaptureConnection) {

  // load our CoreML model
  guard let model = try? VNCoreMLModel(for: KerasModel().model) else {
   return }

  // run an inference with CoreML
  let request = VNCoreMLRequest(model: model) { (finishedRequest, error) in

   // grab the inference results
   guard let results = finishedRequest.results as? [VNClassificationObservation] else { return }

   // grab the highest confidence result
   guard let Observation = results.first else { return }

   // create the label text components
   let predclass = "\(Observation.identifier)"

   // set the label text
   DispatchQueue.main.async(execute: {
    self.label.text = "\(predclass) "
   })
  }
```

همانطور که مشاهده می‌کنید ما مدل یادگیری عمیق خود را لود کرده‌ایم و سپس با استفاده از خروجی در لحظه پیش‌بینی را انجام داده‌ایم و نتیجه بهترین مشاهده (بیشترین درصد برای یک کلاس از بین ۰ تا ۹) را بر روی label چاپ می‌کنیم.

جمع‌بندی

همانطور که مشاهده کردید یادگیری ماشین و یادگیری عمیق در کنار اپلیکیشن‌های موبایلی به راحتی کار می‌کنند. تبدیل مدل یادگیری شده در پایتون به مدل قابل استفاده در اپلیکیشن موبایل و استفاده از آن مزایایی مثل مزایای زیر دارد:
۱) در هنگام ایجاد پیش‌بینی و حتی مواردی که نیاز به یادگیری می‌باشد از امکانات کامل گوشی موبایل مثل CPU و GPU استفاده می‌شود که در گوشی‌های امروزی برای این کار بهینه شده اند.
۲) پردازش به سرعت انجام می‌گیرد و نتیجه در کسری از ثانیه اعلام می‌شود و دیگر نیاز به ارتباط با اینترنت و ارسال ریکوئست‌های سنگین نمی‌باشد.
۳) و در آخر امنیت و حریم شخصی کاربر حفظ می‌شود چون اطلاعات ارسالی توسط کاربر درهیچ‌جا جز گوشی او استفاده نمی‌شود و در جایی غیر از گوشی خود کاربر ذخیره نمی‌شود.
شما هم می‌توانید هر مدلی را با استفاده از یادگیری ماشین و یا یادگیری عمیق آموزش داده و نتیجه آن را در دستگاه‌های مختلف مشاهده کنید و از قابلیت‌های زیادی که دستگاه‌های مخصوصا موبایل در اختیار تحلیلگران داده و برنامه‌نویسان قرار می‌دهد، مانند دوربین یا میکروفون، که می‌تواند کمک شایانی به فرایند پردازش صدا و تصویر بکند، استفاده کنید.

تشکرات

در این‌جا جا دارد تشکر کنم از استاد گرامی که طی سالیان گذشته به جز در طی این پروژه در طی درس‌هایی که با ایشنان داشته‌ام من را یاری کرده‌اند. از دوستان و استادانم در رهنما کالج که یادگیری ماشین و یادگیری عمیق را به من یاد دادند و حمایتم کردند. از همکارانم در شرکت ایده‌ران که راه‌وروش برنامه‌نویسی و تفکر برنامه‌نویسی را به من یاد دادند و حمایتم کردند و در آخر از مادر عزیزم که تلاش زیادی برای به نحو احسن تمام کردن این ۴ سال دانشگاهی داشته‌اند.

منابع:

1. مجموعه ارقام دستنویس هدی | پردازش فارسی
2. https://github.com/amir-saniyan/HodaDatasetReader
3. الگوریتم جنگل تصادفی (Random Forest)
4. انتخاب ویژگی (Feature Section) و کاهش ابعاد
5. آنالیز مولفه اصلی (Principal Component Analysis) یا همان PCA چیست؟
6. الگوریتم کی-نزدیک‌ترین همسایه
7. رگرسیون لجستیک
8. ماشین بردار پشتیبانی
9. https://datascience.stackexchange.com/questions/51203/in-sklearn-pipeline-why-are-all-parameters-fit-transform-but-the-last-one-ca
10. یادگیری عمیق چیست؟