



## ارزشیابی پایان نامه رشته کامپیوتر (نرم افزار)

### ۱- مشخصات دانشجو

نام و نام خانوادگی:	شماره دانشجویی:	ورودی: بهمن / مهر (شماره آروژانه) سال: .....
استاد راهنما:	زمان اخذ پروژه:	تاریخ دفاع:
عنوان پایان نامه:	مقطع تحصیلی: کارشناسی پیوسته <input type="checkbox"/> ناپیوسته <input type="checkbox"/> کاردانی <input type="checkbox"/>	

### ۲- شاخص های ارزشیابی

طبقه بندی	موضوعات ارزشیابی	بارم	استاد راهنما	استاد مدعو
کیفیت علمی، رعایت محدوده زمانی و استمرار در تحقیق	<ul style="list-style-type: none"> <li>کیفیت علمی</li> <li>بررسی تاریخچه و جمع آوری اطلاعات</li> <li>تجزیه و تحلیل مطالب</li> <li>موثر بودن و جدید بودن پروژه</li> <li>میزان تحقق اهداف پایان نامه</li> <li>بحث در نتایج و پیشنهادهای آتی</li> <li>رعایت محدوده زمانی و استمرار در تحقیق</li> <li>پیگیری، نظم در تحقیقات و حضور فیزیکی دانشجو در طول پروژه</li> <li>ارائه گزارش پیشرفت منظم دو ماهه</li> </ul>	۱۰  ۳		
کیفیت نگارش	<ul style="list-style-type: none"> <li>شیوه تنظیم و ترتیب مطالب و فصل ها در پایان نامه</li> <li>میزان رعایت نکات دستور زبان</li> <li>میزان رعایت قوانین نگارش</li> <li>کیفیت جدول ها و شکل ها</li> </ul>	۵		
کیفیت دفاع شفاهی	<ul style="list-style-type: none"> <li>شیوه تنظیم، فن بیان و ارائه سمینار</li> <li>رعایت زمان بندی</li> <li>کیفیت و زیبایی اسلایدها در تفهیم مطالب</li> <li>توانایی در پاسخ گویی به سؤالات</li> </ul>	۳		
مجموع		۲۰		

نمره پایان نامه (۶۰٪ استاد راهنما + ۴۰٪ استاد مدعو)	به عدد: .....	به حروف: .....
---	---------------	----------------

اصلاحات لازم

نام و نام خانوادگی و امضاء:	استاد راهنما:	استاد مدعو:
-----------------------------	---------------	-------------

قابل ارائه در:

سمینارهای داخلی دانشکده	مجلات داخلی	سمینارهای علمی	جشنواره خوارزمی
-------------------------	-------------	----------------	-----------------

کسر نمره: .....	نمره نهایی پایان نامه: به عدد: ..... به حروف: .....
-----------------	---

نام و نام خانوادگی و امضاء مدیر گروه



دانشگاه فنی و حرفه‌ای

دانشکده فنی و حرفه‌ای دختران تهران دکتر شریعتی

# نرم افزار تشخیص چهره با استفاده از شبکه عصبی MTCNN

مستندات پروژه برای دریافت مدرک کارشناسی

در رشته مهندسی تکنولوژی نرم افزار

نام دانشجو

**زهرا اکبری**

استاد راهنما:

**سرکار خانم مهندس معینی**

نیم سال دوم ۱۳۹۸

## بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

وَقُلْ رَبِّ ادْخِلْنِيْ مُدْخَلَ صِدْقٍ وَّاَخْرِجْنِيْ مُخْرَجَ صِدْقٍ وَّاجْعَلْ لِّيْ مِنْ لَّدُنْكَ سُلْطٰنًا نَّصِيْرًا

و بگو: پروردگارا مرا (در هر کاری) به نیکی وارد کن و به نیکی بیرون آور و برایم از نزد خود نیروی یاری دهنده قرار ده.

اسراء ۸۰

## سپاس از

خانواده ی خوب و مهربانم که حامی همیشگی من در تمامی احوال بوده اند.  
سرکار خانم مهندس معینی استاد راهنمای من در این پروژه که صبورانه مرا یاری کرده اند  
و نیز با تشکر از تمامی اساتید محترمی که در رسیدن من به این جایگاه نقش داشته اند.

## چکیده

روش احراز هویت از طریق شناسایی چهره، یکی از روش های مرسوم می باشد که می تواند به افزایش امنیت کمک کند. در این پروژه با بهره گیری از شبکه عصبی عمیق کانولوشنال، با نام MTCNN از طریق مدل از پیش آموزش دیده و وزن های آماده، عمل جستجو و تشخیص چهره ی افراد از تصویر ورودی از وب کم انجام می شود. سپس چهره های شناسایی شده برش داده شده و با دیتابیس از افراد مجاز بررسی می شود. در صورت تطابق چهره ی موجود در تصویر با دیتابیس، یک log از عبور افراد از مقابل دوربین در فایل مربوطه با تاریخ و ساعت جاری سیستم ذخیره می شود. تصویر برش داده شده نیز با تاریخ و ساعت جاری سیستم برای بررسی های آتی در مسیری مشخص ذخیره می شود.

## واژه های کلیدی:

تشخیص چهره، یادگیری عمیق، شبکه عصبی، پردازش تصویر، تایید و بازشناسی چهره

# فهرست مطالب

فصل اول: مقدمه.....	۱
۱-۱-انگیزه .....	۲
۱-۲-هدف .....	۲
فصل دوم: معرفی شبکه عصبی تمام متصل و کانولوشنال.....	۴
۲-۱-شبکه عصبی مصنوعی .....	۵
۲-۱-۱-ساختار یک پرسپترون .....	۵
۲-۱-۲-مفاهیم رایج در شبکه عصبی .....	۶
۲-۲-شبکه عصبی کانولوشنال .....	۹
۲-۲-۱-padding در کانولوشنال .....	۱۲
۲-۲-۲-stride در کانولوشنال .....	۱۳
۲-۲-۳-نحوه ی عملکرد کانولوشنال روی تصویر رنگی .....	۱۳
۲-۲-۴-لایه ی pooling در کانولوشن .....	۱۴
۲-۲-۵-انتقال یادگیری در شبکه عصبی .....	۱۶
فصل سوم: مبانی تشخیص چهره و شبکه عصبی MTCNN .....	۱۸
۳-۱-one shot learning .....	۱۹
۳-۱-۱-تابع فعالیت شباهت .....	۲۰
۳-۲-Siamese Network .....	۲۰
۳-۲-۱-یادگیری متریک .....	۲۱
۳-۲-۲-Triplet loss هزینه .....	۲۲

۲۳ ..... Center loss تابع هزینه ۳-۲-۳

۲۵ ..... Center loss با Triplet loss مقایسه ی ۳-۲-۴

۲۵ ..... Voila-Jones الگوریتم ۳-۳

۲۶ ..... MTCNN شبکه عصبی ۳-۴

۲۷ ..... P-Net یا شبکه پیشنهاد دهنده ۳-۴-۱

۲۷ ..... R-Net یا شبکه محدود کننده ۳-۴-۲

۲۸ ..... O-Net یا شبکه خروجی ۳-۴-۳

## ۲۹..... فصل چهارم: پیاده سازی

۳۰ ..... ۴-۱ پیش نیازها

۳۱ ..... ۴-۲ نصب کتابخانه های مورد نیاز

۳۱ ..... NumPy-۴-۲-۱

۳۲ ..... Matplotlib-۴-۲-۲

۳۲ ..... SciPy-۴-۲-۳

۳۳ ..... OpenCV-۴-۲-۴

۳۳ ..... Keras-۴-۲-۵

۳۴ ..... ۴-۲-۶ تفاوت چارچوب با کتابخانه

۳۴ ..... Tensorflow-۴-۲-۷

۳۶ ..... GPU نصب تنسورفلو روی ۴-۲-۸

۴۰ ..... ۴-۳ پیاده سازی

۴۱ ..... FaceRecognition.py-۴-۳-۱

## ۵۳..... فصل پنجم: نتیجه گیری

۵-۱- نتیجه گیری ..... ۵۴

۵-۲- پیشنهادات برای کارهای آتی ..... ۵۴

**مراجع ..... ۵۵**

**پیوست ها ..... ۵۶**

پیوست الف: مقاله ی MTCNN ..... ۵۷



# فصل اول

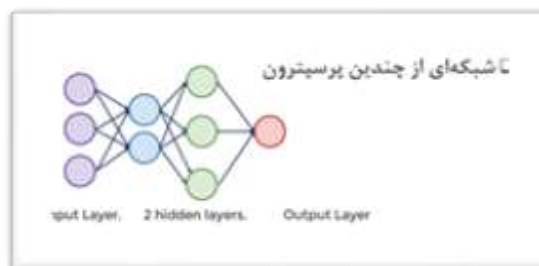
## مقدمه

## ۱-۱-انگیزه

گسترش محصولات هوشمند در عصر ارتباطات، ما را بر آن میدارد تا از جدیدترین روش ها در تمامی جنبه های زندگی ماشینی بهره مند شویم. روش احراز هویت با استفاده از چهره افراد، یکی از جدیدترین روش های جدید و مرسوم می باشد که می تواند به افزایش امنیت، شناسایی مجرمین در هنگام ارتکاب جرم، ثبت تردد افراد در سازمان ها و... کمک کند. روش های مختلفی برای تشخیص چهره موجود می باشد. از رایج ترین آنها روش های پردازش تصویری<sup>۱</sup> با استفاده از کتابخانه های مخصوص این زمینه مثل `matplotlib`، `numpy`، `opencv` می باشد. این روش ها که به روش های سنتی<sup>۲</sup> نیز شهرت دارند، با استفاده از فیلترهای مختلفی که مهندسی و محاسبه شده اند می توانند در شناسایی اعضای مختلف چهره (Classification) سودمند باشند. روش های کلاسیک با وجود سخت افزارهای قوی امروزی سرعت مناسب و محاسبات و پردازش های کمی دارند اما دقت لازم را ندارند. الگوریتم های سنتی برای عمل بازشناسی چهره (Verification) دقت بسیار کمی دارند بنابراین لازم است تا از روش های جدیدتر و حرفه ای تر استفاده کنیم. بدین منظور می توانیم از شبکه های عصبی عمیق<sup>۳</sup> و یادگیری ماشینی<sup>۴</sup> استفاده کنیم. از این تکنولوژی در تشخیص بیماری ها، پیش بینی وضعیت اقتصادی، پردازش سیگنال و پردازش زبان طبیعی و... می توان استفاده نمود.

## ۱-۲-هدف

هسته ی تشکیل دهنده ی یادگیری ماشین شبکه عصبی می باشد. شبکه های عصبی متشکل از چند لایه نورون می باشد که عملکرد آنان مشابه عملکرد مغز انسان در زمینه ی پردازش تصویر می باشد. بدین معنا که لایه های اولیه در تشخیص خطوط و لبه ها عملکرد خوبی دارند و هر جقدر به سمت لایه های آخر نزدیک می شویم اشکال شناسایی شده کاملتر و قابل فهم تر توسط انسان هستند. به شبکه های عصبی ای که بیشتر از سه لایه نورون داشته باشند شبکه عصبی عمیق می گویند.



شکل ۱-۱ شبکه عصبی ساده

<sup>1</sup> Image Processing

<sup>2</sup> Classic

<sup>3</sup> Deep Neural Network

<sup>4</sup> Machine Learning

در این پروژه با بهره گیری از یکی از مدل های از پیش آموزش دیده<sup>۵</sup> یادگیری عمیق<sup>۶</sup>، نرم افزار ثبت تردد افراد سازمان را که با عبور افراد از مقابل دوربین هویت آنان را شناسایی می کند پیاده سازی خواهیم کرد.

در این پروژه از مدل از پیش آموزش دیده ای به نام MTCNN<sup>۷</sup> که متشکل از سه شبکه که به صورت آبشاری کنار هم قرار گرفته اند با نام های P-net ، R-net و O-net استفاده می شود. مدل آموزش دیده عمل شناسایی اعضای چهره را انجام داده و دور هر چهره موجود در تصویر دوربین را با کادر سبز رنگی مشخص می کند. سپس برنامه ما با بررسی دیتابیس تصاویر افراد، و مقایسه تصویر گرفته شده از دوربین با دیتابیس در فایلی ورود و خروج افراد را با روز و ساعت ثبت می کند. برای معرفی افراد جدید کافی است یک عکس از چهره فرد را داخل پوشه ی افراد سازمان قرار دهیم. برنامه در اجرای بعدی فرد جدید را شناسایی خواهد نمود.

---

<sup>۵</sup> Pretrained Model

<sup>۶</sup> Deep Learning

<sup>۷</sup> Multi-task Cascaded Convolutional Neural Networks

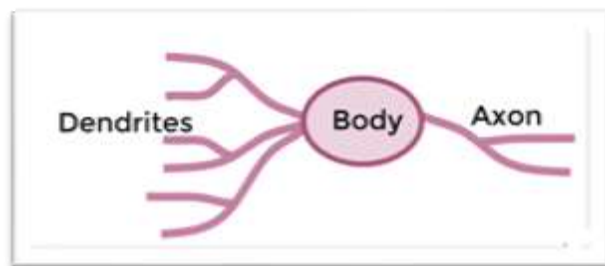
## **فصل دوم**

### **معرفی شبکه عصبی تمام متصل و کانولوشنال**

## ۱-۲- شبکه عصبی مصنوعی

### ۱-۲-۱- ساختار یک پرسپترون<sup>۸</sup>

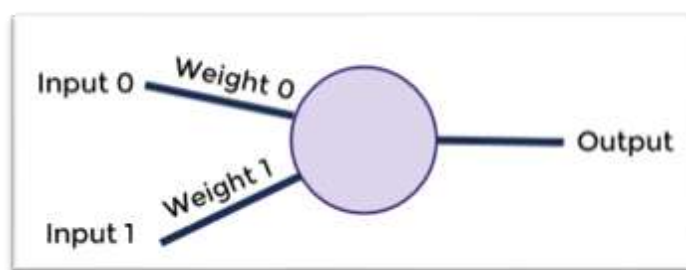
در طراحی و ساخت شبکه های عصبی مصنوعی از ساختار نورون های مغز انسان الهام گرفته شده است. ساختار شماتیک یک نورون مغزی مانند شکل زیر می باشد:



شکل ۲-۱ ساختار شماتیک نورون مغز انسان

هر نورون مغزی از تعدادی شاخه ورودی تشکیل شده است. اطلاعات از سایر نورون به بدنه ی نورون وارد شده و تغییراتی در اطلاعات از طریق واکنش های شیمیایی ایجاد می شود. سپس اطلاعات از طریق آکسون ها به نورون های بعدی منتقل می شود. از این ویژگی در طراحی شبکه های عصبی پرسپترون استفاده شده است. بدین معنی که در این نوع شبکه ها نیز هر لایه از تعدادی نورون با شاخه های ورودی<sup>۹</sup> (به این ورودی ها ویژگی<sup>۱۰</sup> نیز می گویند) و بدنه ای که یک واحد محاسبات ریاضی محسوب می شود با یک خروجی<sup>۱۱</sup> تشکیل شده است.

به مجموع ورودی ها و عملیات ریاضی و خروجی یک واحد پرسپترون می گویند. شماتیک یک پرسپترون مانند شکل زیر است:



شکل ۲-۲ شماتیک یک پرسپترون

<sup>8</sup> Perceptron

<sup>9</sup> Input

<sup>10</sup> feature

<sup>11</sup> output

" الگوریتم پرسپترون یک الگوریتم دسته‌بندی دودویی (نوعی از دسته‌بندی که می‌تواند با توجه به بردار ورودی تصمیم بگیرد که این ورودی متعلق به یک کلاس هست یا خیر) است. این الگوریتم یک دسته‌بند خطی است، به این معنا که پیش‌بینی‌هایش را با توجه به ترکیب خطی وزن دار ورودی الگوریتم انجام می‌دهد. همچنین این الگوریتم به دلیل اینکه ورودی‌هایش را به صورت تک تک در زمان بررسی می‌کند، یک الگوریتم برخط می‌باشد. الگوریتم پرسپترون در سال ۱۹۵۷ در لابراتوار کرنل آرونوتیکال به وسیله فرانک روزنبلت ابداع شد. در واقع این الگوریتم جزء اولین شبکه‌های عصبی مصنوعی است که به کار گرفته شده‌است." ویکی پدیا<sup>۱۲</sup>

در شکل ۲-۲ علاوه بر ورودی‌ها عبارتی با عنوان وزن ها (weight, weight0) نیز مشخص شده است. بسته به این که بخواهیم اولیتی را برای یک ورودی کمتر یا بیشتر در نظر بگیریم این مقادیر تعیین می‌شوند. البته در فرآیند آموزش<sup>۱۳</sup> شبکه این مقادیر تغییر پیدا می‌کند.

پرسپترون در قالب ریاضی به شرح زیر است:

$$\sum W_i * X_i + b$$

که در آن  $W$  مقادیر وزن ها،  $X$  مقادیر ورودی ها و  $b$  نماد مقدار بایاسی می باشد که برای حل مشکل صفر بودن ورودی ها استفاده می شود.

در بدنه ی هر پرسپترون یک عمل ریاضی انجام می شود. به این عمل ریاضی تابع فعالیت<sup>۱۴</sup> گفته می شود. انتخاب نوع تابع فعالیت بسته به نظر طراح شبکه عصبی و کاربرد شبکه، متغیر است. از جمله فرمول های رایج و اثبات شده در این زمینه سیگنویید و ReLU می باشد.

## ۲-۱-۲- مفاهیم رایج در شبکه عصبی

**تابع هزینه<sup>۱۵</sup>:** مقداری است که میزان تفاوت خروجی مدل تا مدل مورد نظر را نمایش می دهد. در فرآیند آموزش و ساخت مدل باید تلاش کنیم به حداقل برسد. در بخش بهینه سازی مدل مطرح می شود. این تابع هم مانند تابع فعالیت بنا به کاربرد شبکه متفاوت است.

<sup>12</sup> [www.wikipedia.org](http://www.wikipedia.org)

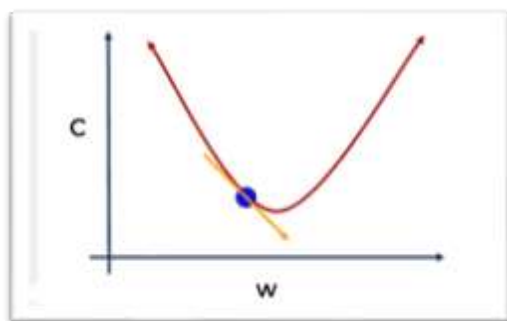
<sup>13</sup> Train

<sup>14</sup> Activation function

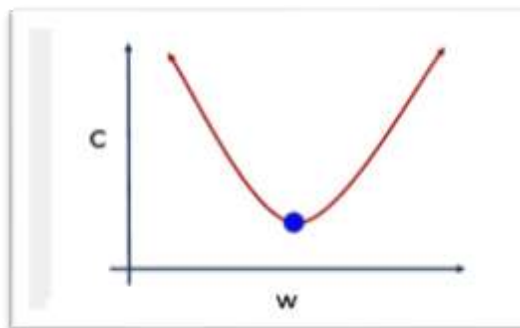
<sup>15</sup> Cost Function یا Loss Function

پس انتشار خطا<sup>۱۶</sup>: پس از مرحله ی آموزش مدل، تابع هزینه را فقط می توانیم در مورد لایه ی آخر شبکه حساب کنیم. زیرا از مقادیر لایه های مخفی خبر نداریم. به همین دلیل پس انتشار خطا مطرح می شود. پس انتشار خطا برای محاسبه سهم خطای هر نورون پس از پردازش یک دسته از داده ها استفاده می شود.

گرادیان کاهشی<sup>۱۷</sup>: یک الگوریتم بهینه سازی برای یافتن حداقل یک تابع است. برای یافتن کمینه محلی یک تابع با استفاده از این الگوریتم، گام های متناسب با منفی گرادیان تابع در محل فعلی برداشته خواهد شد.



شکل ۳-۲ گام برای یافتن بهینه محلی



شکل ۴-۲ بهینه محلی یک تابع

مقادیر وزن های هر یال ورودی به نورون ها و لایه ها، قبل از فرآیند آموزش مدل، به صورت تصادفی انتخاب می شود. با تکرار فرآیند آموزش مقدار وزن ها به روز می شود و دستخوش تغییرات می شود. برای یافتن بهترین و مناسب ترین وزن ها و مقادیر یال ها، گرادیان کاهشی و پس انتشار خطا کمک زیادی به حداقل شدن مقدار تابع هزینه می کنند. بدین ترتیب خروجی مدل به خروجی مورد انتظار نزدیک تر می شود.

**Ephocs**: یکی از مهم ترین فاکتورهایی که برای فاز آموزش مدل حتما باید تعیین شود تعداد ephocs می باشد. بدین معنا که در فاز آموزش چند بار خطا پیمایش شود (چند بار داده ها را مشاهده کند- آموزش روی مدل چند بار تکرار شود)

**Fully connected layer**: نحوه ی اتصال لایه ها و نورون های هر شبکه عصبی، باعث به وجود آمدن انواع مختلفی از شبکه ها می شود. به لایه ای که تمام نورون های آن به نورون های لایه بعد متصل است **fully connected** می گویند.

<sup>16</sup> Backpropagation

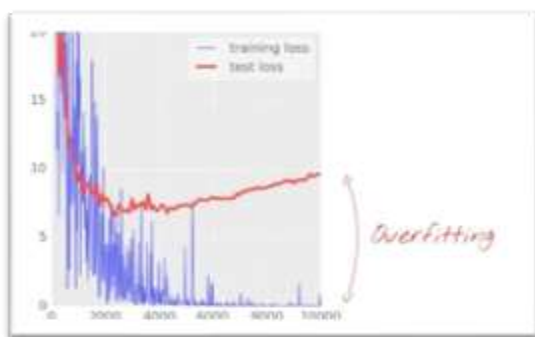
<sup>17</sup> Gradient Descent



شکل ۵-۲ نمونه ای از شبکه ی fully connect

**مجموعه داده<sup>۱۸</sup>:** به طور کلی تمامی روش ها و متدهای یادگیری ماشین بر روی داده ها اجرا می شوند و در واقع این داده ها هستند که نقطه شروع فرایند داده کاوی، علم داده ها و به طور کلی یادگیری ماشین می باشند. به داده هایی گفته می شود که با موضوع، خواص مشخص و یکسان جهت انجام تحقیقات و پروژه های مربوط به Data Science (علم داده) جهت کسب دانش از داده ها استفاده می شود. دیتاست ها را در فاز آموزش به سه دسته ی داده های آموزش<sup>۱۹</sup>، داده های تست<sup>۲۰</sup>، داده های اعتبارسنجی<sup>۲۱</sup> تقسیم می کنند. از داده های آموزش برای استخراج ویژگی های مدل، از داده های تست برای بررسی عملکرد مدل و از داده های اعتبارسنجی برای سنجش درستی تست استفاده می شود. بدین ترتیب بیش برآزش کمتری ایجاد می شود.

**بیش برآزش<sup>۲۲</sup>:** Overfit شدن به معنای این است که الگوریتم فقط داده هایی که در مجموعه آموزشی (train set) یاد گرفته است را می تواند به درستی پیش بینی کند ولی اگر داده ای کمی از مجموعه ی آموزشی فاصله داشته باشد، الگوریتمی که Overfit شده باشد، نمی تواند به درستی پاسخی برای این داده های جدید پیدا کند و آن ها را با اشتباه زیادی طبقه بندی می کند.



شکل ۶-۲ نمونه ای از نمودار بیش برآزش

<sup>18</sup> Dataset

<sup>19</sup> Train Dataset

<sup>20</sup> Test Dataset

<sup>21</sup> Validation Dataset

<sup>22</sup> Overfitting



**Dropout layer:** با نرخی معین، در هر پس انتشار خطا وزن یال ها به روز نمی شود. نوعی از لایه های شبکه عصبی می باشد. خاصیت fully connected یک لایه را نقص می کند. برای کاهش میزان بیش برآزش مطرح شد و منجر به ساخت شبکه های مختلف می شود. شبکه ی ایجاد شده در زمان تست و کارکرد واقعی مدل متفاوت می شود و بدین ترتیب از حفظ کردن داده ها توسط مدل و overfit زیاد جلوگیری می شود.

**Learning Rate Decay:** ضریب گرادیان کاهشی می تواند تغییر کند. به طور مثال در کوهنوردی در ابتدای رسیدن به قله ابتدا گام های بلند برمیداریم و در صورتی که به نزدیکی قله رسیدیم از قدم های کوچک استفاده می کنیم. از مواردی است که در هنگام آموزش تعیین می شود و به بهبود بیش برآزش کمک می کند.

**Batch size:** داده های ورودی از دیتاست به لایه ها، باید به صورت دسته ای وارد شوند. اندازه و تعداد داده های ورودی را batch size می گویند.

## ۲-۲- شبکه عصبی کانولوشنالی<sup>۲۳</sup>

در شبکه های عصبی کلاسیک، نسبت پیکسل های یک تصویر به یکدیگر در نظر گرفته نمی شد. به این معنا که فرقی نداشت با چه ترتیبی پیکسل ها وارد شبکه شوند. اشکال این روش این است که برای هر پیکسل جداگانه تصمیم گیری می شود.

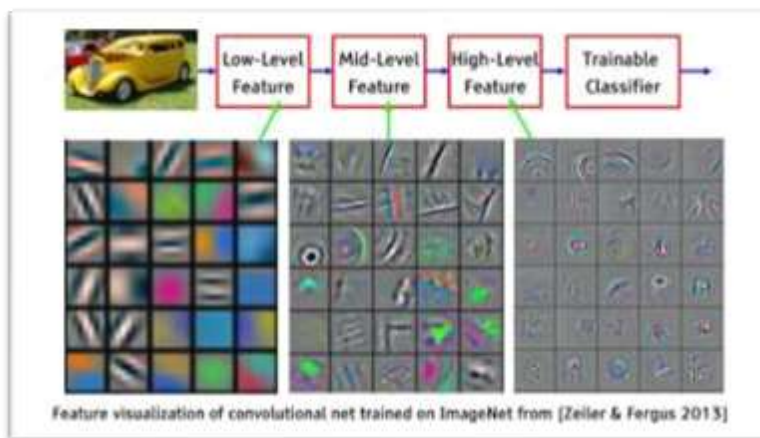
در شبکه های قبلی برای پردازش و محاسبات ناچار به حذف جزییاتی مثل تک کاناله کردن تصویر رنگی و کوچک کردن ابعاد را داشتیم. در صورتی که بخواهیم داده ای loss نشود و از روش قبلی (fully connected) جلو برویم تعداد زیادی نورون داریم که کامپیوترهای فعلی توان پردازش این نورون ها را ندارند.

پس شبکه cnn به دلایل زیر مطرح شد:

- کد کردن لوکالیتی پیکسل های تصویر
- جلوگیری از loss داده ها

<sup>23</sup> Convolutional Neural Network

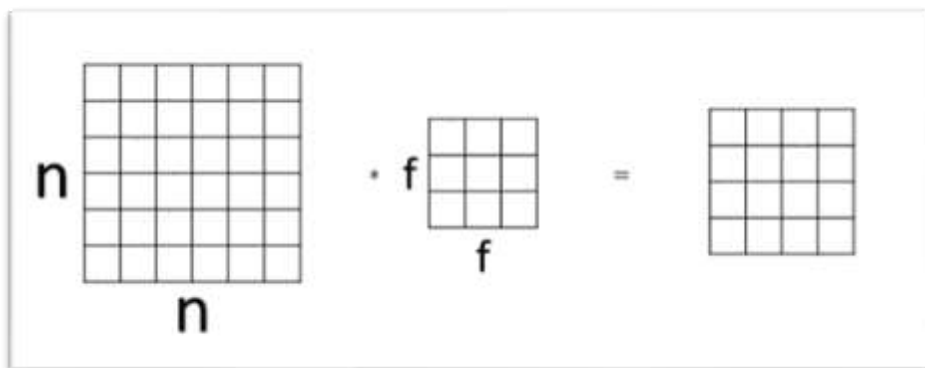
بدین ترتیب در این شبکه بر خلاف شبکه های قبلی، به یادگیری فیلتر روی آورده شد. این ایده توسط یان لکان<sup>۲۴</sup> مطرح شد.



شکل ۷-۲ کارکرد شبکه کانولوشنال

مانند آنچه در شکل ۷-۲ نشان داده شده است نتیجه ی استفاده از شبکه های کانولوشنال این است که لایه های اولیه در شناسایی خطوط و لبه ها کار می کنند. لایه های میانی اشکال واضح تری را شناسایی می کنند و لایه های آخر توانایی شناسایی قسمت های کامل تری از تصویر را دارند. این نوع عملکرد شبکه های عصبی مانند عملکرد مغز موجودات زنده در بحث بینایی می باشد.

بدین ترتیب به جای اتصال همه ی نورن ها به هم، مقادیر  $^{25}$  Activation map و سایر لایه ها در یک فیلتر ضرب می شوند.

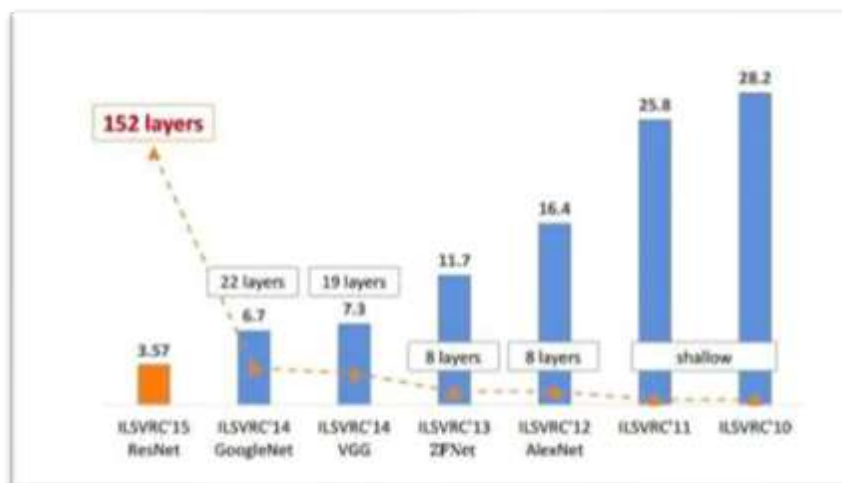


شکل ۸-۲ نحوه ی ایجاد یک لایه کانولوشنال

<sup>24</sup> Yonn LeCun

<sup>25</sup> به لایه ی اول شبکه با همان تصویر ورودی می گویند

پس از معرفی ایده ی شبکه های کانولوشنالی شبکه های عمیق گسترش پیدا کردند. این ایده در سال ۲۰۱۲ مطرح شد. در شکل زیر زیاد شدن لایه ها پس از معرفی کانولوشنال در برخی از شبکه های عصبی عمیق نشان داده شده است.

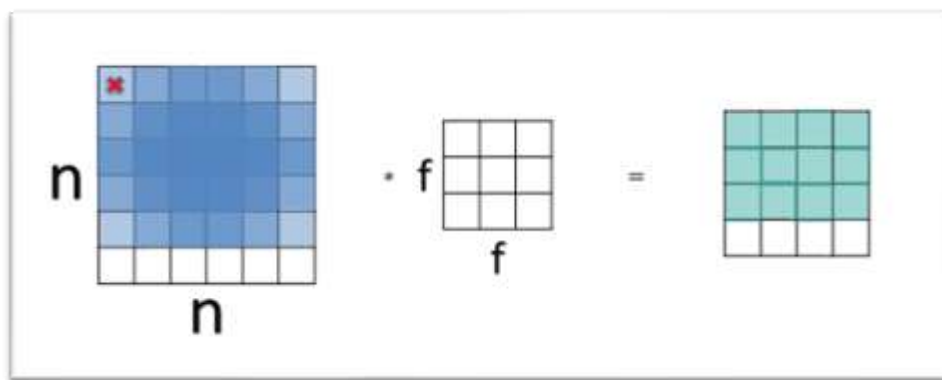


شکل ۹-۲ مقایسه شبکه ها پس از معرفی cnn

بدین ترتیب در این شبکه، به آموزش فیلتر روی آورده شد. تا قبل از این فیلترهای پردازش تصویری مهندسی شده و روی شبکه اعمال می شدند تا خروجی مناسب تولید شود اما پس از کانولوشنال عمل انتخاب فیلتر مناسب به عهده ی شبکه گذاشته شد. یک فیلتر از سوی طراح شبکه انتخاب میشود و در همه ی پیکسل های تصویر ضرب می شود.

اما مشکلاتی نیز در این شبکه ها وجود داشت:

- تصویر خروجی کوچک تر از تصویر ورودی به شبکه می باشد.
- پیکسل هایی که در لبه های بیرونی قرار دارند به اندازه ی پیکسل های داخلی در محاسبات شرکت داده نمی شوند.



مانند آنچه در شکل ۲-۱۰ نشان داده شده است، فیلتری با ابعاد  $f \times f$  در ماتریس تصویر با ابعاد  $n \times n$  از پیکسل  $(0,0)$  ضرب می شود. مقادیر جدید با هم جمع شده و عدد خانه ی  $(0,0)$  ماتریس خروجی را تشکیل می دهد. به همین ترتیب فیلتر روی ماتریس حرکت داده می شود تا ماتریس خروجی ایجاد شود. اما در شکل بالا، خانه ای که با علامت ضربدر مشخص شده است کمرنگ تر از خانه های داخلی و مرکزی ماتریس می باشد و این بدین معناست که خانه های داخلی دفعات بیشتری در عمل ضرب فیلتر نسبت به خانه ی ضربدر دار شرکت می کنند.

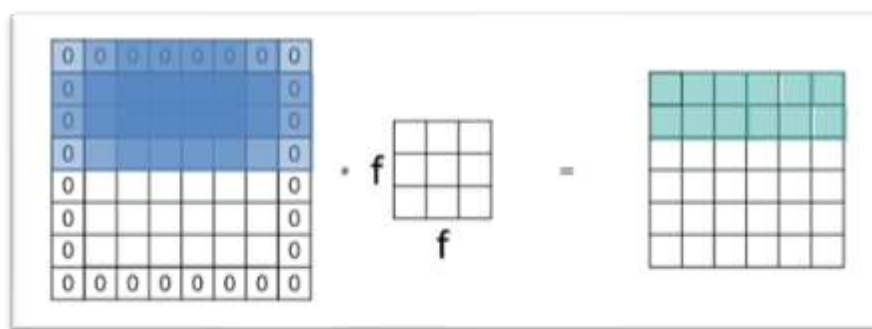
ابعاد ماتریس تصویر خروجی از طریق فرمول زیر محاسبه می شود:

$$n \times f - 1$$

### ۱-۲-۲ padding در کانولوشنال

برای حل مشکلات ذکر شده راه حلی نیز مطرح شد. پیشنهاد شد از padding برای ماتریس ورودی استفاده شود تا مشکل شرکت نکردن لبه ها در محاسبات حل شود. همچنین تصویر خروجی همان ابعاد تصویر ورودی را دارا خواهد بود.

Zero padding یک نوع padding یا حاشیه در تصویر است که مقادیر صفر هم به طول و هم به عرض تصویر اضافه می شود. تاثیری در خروجی ما نمی گذارد زیرا مقادیر خانه های آن صفر می باشد اما میتواند مشکلات ما را حل کند.



شکل ۲-۱۱ padding در کانولوشن

با مطرح شدن بحث padding نوع لایه های موجود در شبکه کانولوشنال شامل تغییراتی شد:

- لایه ی کانولوشنالی valid: در این لایه padding نداریم و تصویر خروجی از تصویر ورودی کوچک تر می شود.

ابعاد لایه ی خروجی از این لایه با فرمول زیر محاسبه می شود:

$$n-f+1 * n-f+1$$

- لایه ی کانولوشنالی same: در این لایه از padding استفاده می شود. پس ابعاد ورودی و خروجی برابر است و از طریق فرمول زیر محاسبه می شود:

$$n+2p-f+1 * n+2p-f+1$$

مقدار p یا padding نیز از طریق فرمول زیر محاسبه می شود:

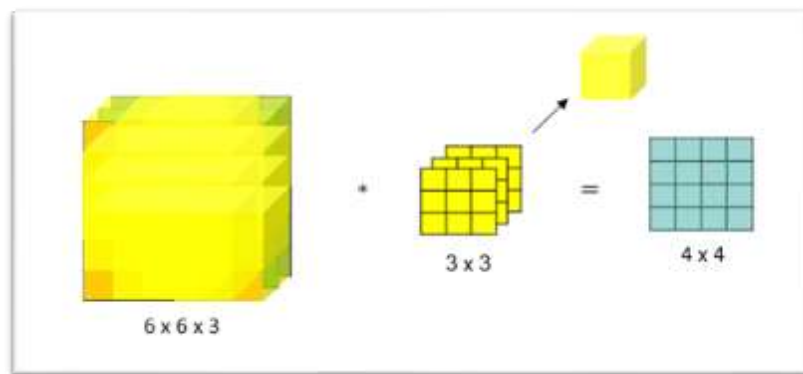
$$p=(f-1)/2$$

## ۲-۲-۲ stride در کانولوشنال

گفتیم که فیلتر با ابعاد  $f*f$  روی تصویر با ابعاد  $n*n$  حرکت داده می شود تا تصویر خروجی تولید شود. حرکت فیلتر روی تصویر در حالت عادی به صورت خانه به خانه است، اما با تعیین stride، می توانیم به صورت پرشی و چندتا چندتا از روی پیکسل ها عبور کنیم تا فیلتر را در خانه های تصویر ضرب کنیم. به صورت خلاصه تعداد پرش های فیلتر برای اعمال روی تصویر را stride می گویند. با این کار تصویر خروجی کوچک خواهد شد.

## ۲-۲-۳ نحوه ی عملکرد کانولوشن روی تصویر رنگی

یکی از ویژگی های مهم کانولوشن جلوگیری از حذف جزئیات داده ها می باشد. تصاویر در دنیای واقعی شامل سه کانال رنگ RGB می باشند. کانال R برای رنگ قرمز، کانال G برای رنگ سبز و کانال B برای رنگ آبی می باشد. ترکیب این سه رنگ در دنیای واقعی رنگ های ثانویه را ایجاد می کند. با تک کاناله کردن تصویر (سیاه و سفید کردن تصویر) جزئیات مهمی از تصویر از دست می رود. با وجود کانولوشن از این کار جلوگیری می کنیم. اما چگونه؟ فیلتری که پیش از این بیان کردیم فقط بر روی یک بعد از تصویر سه بعدی ما با سه کانال رنگی اعمال می شود. پاسخ این است که از فیلترهای سه بعدی برای اعمال بر همه ی بعدهای تصویر استفاده می کنیم. مانند آنچه در شکل زیر نشان داده شده است:



شکل ۱۲-۲ فیلتر سه بعدی بر روی تصویر رنگی

با بیان مطالب فوق ابعاد هر لایه از یک شبکه ی کانولوشنال طبق فرمول زیر محاسبه می شود:

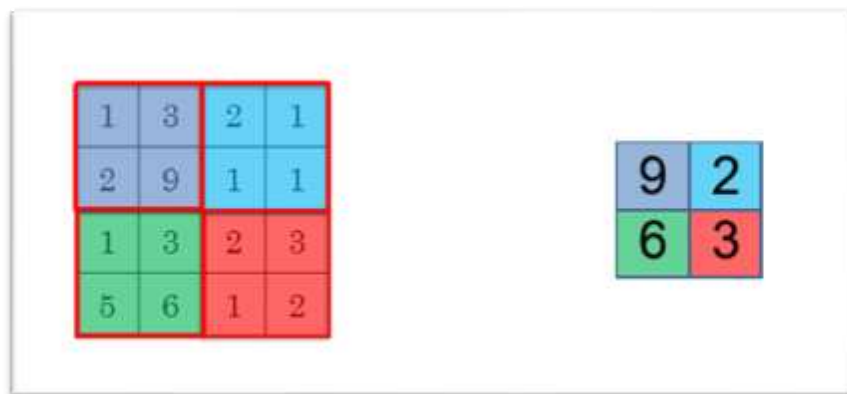
$n \times n$  image       $f \times f$  filter

padding  $p$       stride  $s$

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

### ۴-۲-۲- لایه ی pooling در کانولوشن

با افزایش لایه های یک شبکه ی کانولوشنال، ابعاد تصویر کوچک و کوچک تر شده و به عمق تصویر افزوده می شود. اطلاعات نیز در بعدهای مختلف کدگذاری می شوند. از لایه ی pooling برای کاهش سریع بعد تصویر برای تسهیل محاسبات لایه های بعد استفاده می شود. در این لایه هیچ پارامتری برای یادگیری وجود ندارد.



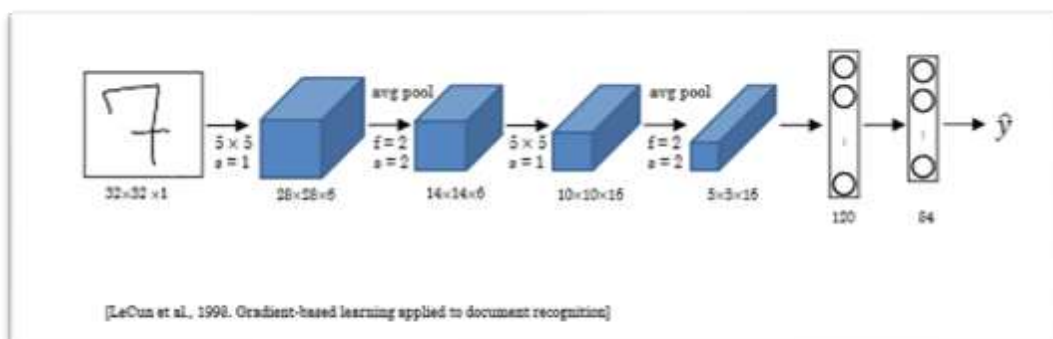
شکل ۲-۱۳ عملکرد لایه ی pooling

در شکل بالا، یک ماتریس  $4 \times 4$  وجود دارد. خانه های ماتریس به چهار ماتریس  $2 \times 2$  تقسیم می شوند و مقادیر هر ماتریس کوچک، با هم مقایسه شده و بیشترین مقدار در ماتریس  $2 \times 2$  ی خروجی قرار داده می شود که به این نوع pooling لایه ی max pooling می گویند. همچنین به جای انتخاب بیشترین مقدار می توانیم میانگین هر ماتریس کوچک را محاسبه و در ماتریس خروجی قرار دهیم که به این نوع pooling، avg pooling می گویند. در شکل ۲-۱۳ نوع پولینگ max pooling است.

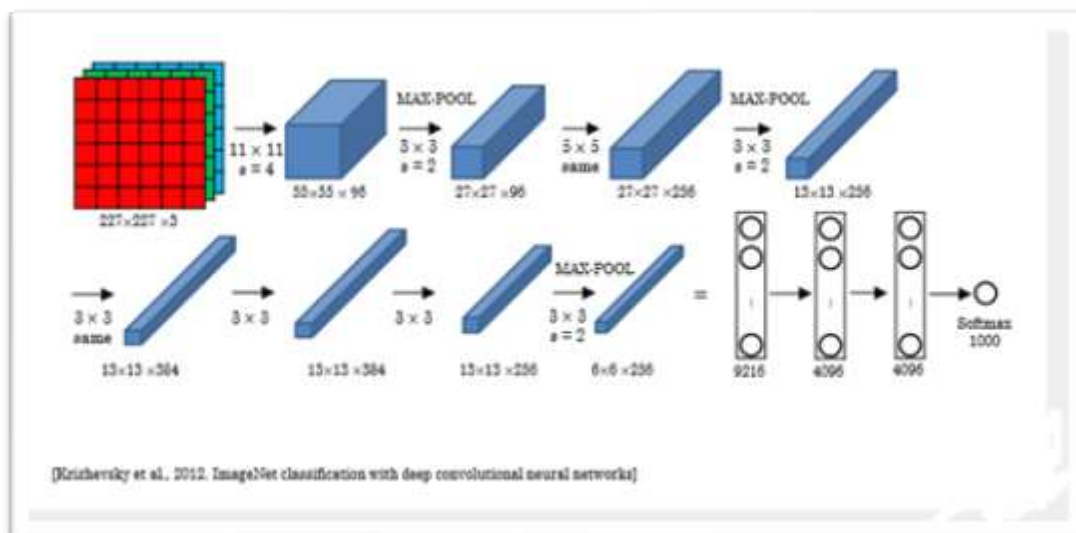
بین دو نوع پولینگ، max pooling رایج تر است و بیشتر استفاده می شود. در صورتی که activation map ورودی چند لایه باشد، pooling برای هر لایه به صورت مجزا محاسبه می شود.

قابل ذکر است از لایه های fully connected و dropout نیز می توانیم در شبکه کانولوشنال استفاده کنیم. نحوه ی ترتیب و چینش لایه ها بستگی به نظر طراح شبکه دارد و می تواند معماری های مختلفی را برای شبکه خود در نظر بگیرد.

در ادامه تصویر دو شبکه کانولوشنالی معروف به نام LeNet و AlexNet به صورت گرافیکی ذکر شده است:



شکل ۲-۱۴ شبکه ی LeNet



شکل ۲-۱۵ شبکه ی AlexNet

همانطور که در شکل های ۲-۱۴ و ۲-۱۵ دیده می شود تعداد و ترتیب لایه ها در شبکه های مختلف متفاوت است و می تواند بسته به سختی مسئله عمیق تر و پیچیده تر شود. اما پترن رایج در انتخاب لایه ها به صورت زیر است:

Conv<sup>26</sup>-pool<sup>27</sup>-conv-pool-...-FC<sup>28</sup>-FC-FC-softmax

**Softmax:** یک نوع تابع فعالیت است معمولا برای لایه ی آخر استفاده می شود تا احتمال وجود یک چیز در تصویر را به دست آوریم. به طور مثال اگر هدف شبکه ی ما شناسایی اعداد دست نویس یک تا ۹ باشد در لایه ی آخر شبکه طراحی شده از softmax استفاده می کنیم. به این ترتیب احتمال وجود هر عدد در تصویر مشخص می شود. هر عددی که احتمال بیشتری داشته باشد نهایتا انتخاب می شود.

## ۵-۲-۲- انتقال یادگیری<sup>۲۹</sup> در شبکه عصبی

مبحث مهم دیگری که در شبکه عصبی وجود دارد انتقال یادگیری می باشد

“Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.”Wikipedia

انتقال یادگیری یعنی از دانشی که در یک زمینه کسب کرده ایم در زمینه های نسبتا مشابه استفاده کنیم. به بیان دقیق تر یعنی ویژگی های استخراج شده در سایر شبکه ها را در شبکه ی خود استفاده کنیم. مثالی از این کاربرد را می توان استفاده از شبکه ی معروف vgg16 برای شناسایی اشکال و اشیای مختلف در تصویر به جای طراحی مجدد یک شبکه که عمل vgg16 را برای ما انجام دهد. Vgg16 یک شبکه عصبی معروف است که می تواند ۱۰۰۰ شی مختلف در تصویر را شناسایی کند. انتقال یادگیری در شبکه عصبی از طریق feature extraction اتفاق می افتد.

Feature extraction عمل استخراج ویژگی از برخی از لایه های شبکه های معروف و استفاده در شبکه خود است.

---

<sup>26</sup> Convolutional Layer

<sup>27</sup> Pooling Layer

<sup>28</sup> Fully Connected Layer

<sup>29</sup> Transfer learning



با استفاده از این ویژگی ما می توانیم تعداد دلخواهی از لایه های شبکه ی عصبی دیگری را در شبکه خود لود کنیم. از وزن ها و فیچرهای استخراج شده در آن شبکه ها برای تسریع عملیات آموزش مدل خود استفاده کنیم. می توانیم تمام یک شبکه را در شبکه ی خود لود کنیم یا فقط لایه های اول که برای شناسایی خط و لبه ها و اشکال ساده هندسی می باشد را لود کنیم. می توانیم لایه های لود شده را قفل کنیم تا آموزش مدل جدید سبب بهم ریختن لایه ها لود شده نشود و...

تاکنون با مفاهیم اصلی شبکه عصبی و یادگیری ماشین آشنا شده ایم. در فصل های بعدی، شبکه ی عصبی استفاده شده در برنامه ی خود را شرح داده و وارد مباحث تشخیص چهره خواهیم شد.

## فصل سوم

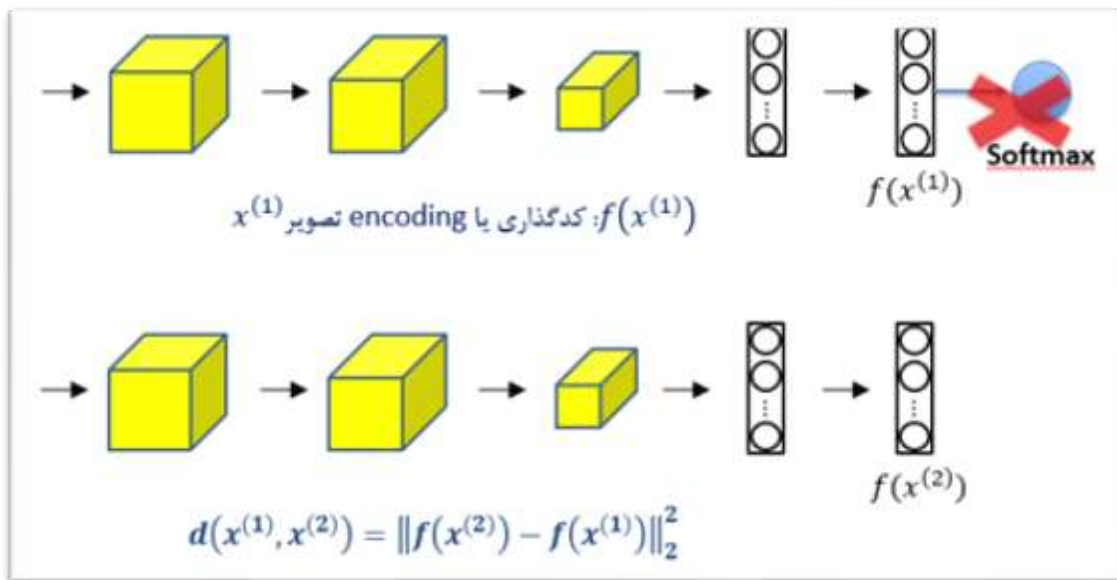
### مبانی تشخیص چهره و شبکه عصبی MTCNN

## one shot learning-۳-۱

در شبکه هایی که تاکنون بررسی کردیم، از مجموعه داده های چندهزارتایی و یک شبکه عصبی، عمل آموزش و استخراج ویژگی ها و مدل انجام می شد. اما در شبکه هایی که برای شناسایی چهره ی افراد استفاده می شود ما برای شناسایی هویت هر فرد، با مسئله ی کمبود عکس های کافی برای train مدل مواجهیم. به علاوه اگر بخواهیم از روش هایی که تاکنون ذکر شد استفاده کنیم فقط می توانیم چهره ی افرادی را شناسایی کنیم که قبل از فرآیند training مشخص شده اند و برای افزودن افراد جدید راه حل مناسبی وجود ندارد.

“ One-shot learning is an object categorization problem, found mostly in computer vision. Whereas most machine learning based object categorization algorithms require training on hundreds or thousands of samples/images and very large datasets, one-shot learning aims to learn information about object categories from one, or only a few, training samples/images.” Wikipedia

برای مدیریت این مشکل one/low shot learning مطرح می شود. یعنی عمل آموزش و شناسایی افراد تنها با وجود یک عکس از آن فرد انجام شود که به آن one shot learning گفته می شود. در صورتی که به جای یک عکس از چند عکس از آن فرد استفاده کنیم low shot learning گفته می شود.



شکل ۳-۱ شبکه ی کانولوشنال برای شناسایی چهره

مانند آنچه در شکل ۳-۱ نشان داده شده است شبکه ی عصبی با لایه ی آخر softmax که احتمال وجود یک شی در تصویر را برمی گرداند در مسئله ی تشخیص چهره نمی تواند به خوبی عمل کند چون ما در تشخیص چهره شناسایی دقیق هویت یک فرد است. پس به جای تابع فعالیت softmax باید از چه تابع فعالیت استفاده شود؟ اینجاست که تابع فعالیت شباهت<sup>۳۰</sup> مطرح می شود.

### ۱-۳-۱- تابع فعالیت شباهت

اگر دو تصویر برای مقایسه با هم داشته باشیم در صورتی که میزان تفاوت دو تصویر را با  $d$  نمایش دهیم:

$$d(img1, img2)$$

و اگر میزان حد آستانه<sup>۳۱</sup> (معیار یکسان بودن یا نبودن تصاویر) را با علامت  $T$  نمایش دهیم در این صورت:

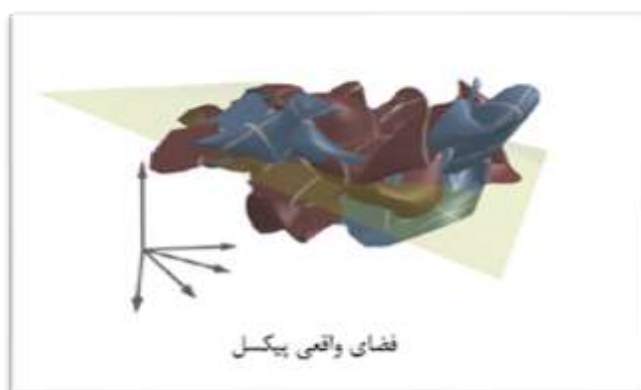
If  $d(img1, img2) \leq T$   $\longrightarrow$  دو تصویر متعلق به یک شخص است

Else if  $d(img1, img2) > T$   $\longrightarrow$  دو تصویر متعلق به دو فرد مختلف است

از این تابع فعالیت در شبکه های عصبی تشخیص چهره استفاده می شود که با نام Siamese Network شناخته می شوند.

### ۲-۳- Siamese network

برای حل چالش تشخیص چهره مطرح شد. در این شبکه ها هدف شبکه استخراج encoding تصاویر است. بدین معنا که تصاویر را از شبکه ی CNN عبور می دهیم فیچرهای هر تصویر استخراج می شود و فضای در هم تنیده ی پیکسل ها در دنیای حقیقی برای هر فرد جدا می شود.



شکل ۳-۲ فضای واقعی پیکسل ها

<sup>30</sup> Similarity Function

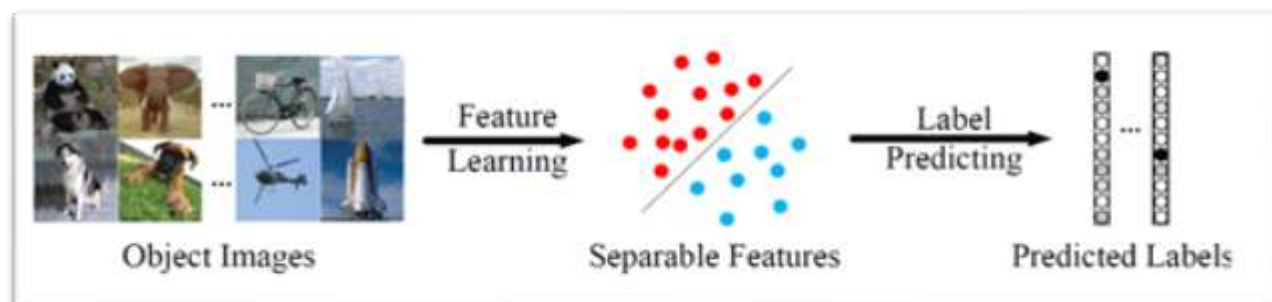
<sup>31</sup> Threshold

اطلاعات تصاویر در دنیای حقیقی پیکسل ها مانند آنچه در شکل ۳-۲ نشان داده شده است در هم تنیده می باشد. کار شبکه های عصبی تفکیک اطلاعات موجود در این فضا می باشد. فضای نرونی خوب فضای نرونی است که بتواند اطلاعات را در این فضای در هم تنیده از هم تفکیک کند.

بدین ترتیب در لایه ی آخر شبکه های سایمیس پیش بینی و برگرداندن احتمال وجود اشیا در تصویر نیست. بلکه یک encoding که یک آرایه ی ۱۲۸ تایی از تصویر می باشد با عبور تصویر از شبکه برگردانده می شود، آرایه با تصاویر جدید هر فرد که encoding آنان نیز استخراج شده است مقایسه می شود. فرمول تابع شباهت در شکل ۳-۱ با حرف d نشان داده شده است. از این تابع برای مقایسه ی دو encoding استفاده می شود و نتیجه بر اساس معیار حد آستانه مشخص می شود.

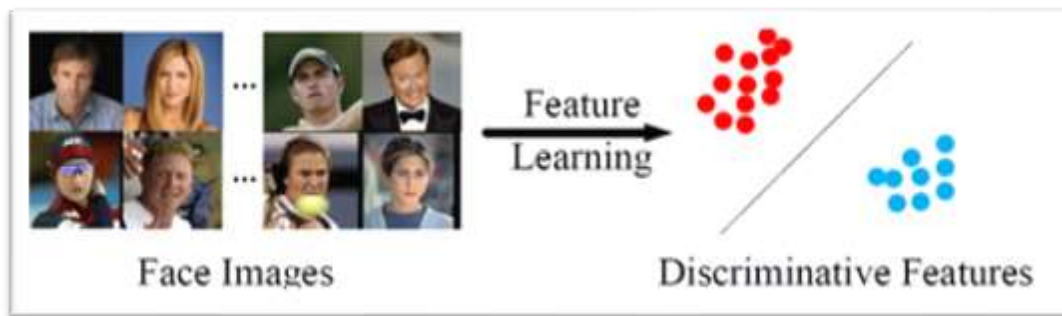
### ۳-۲-۱- یادگیری متریک<sup>۳۲</sup>

در لایه ی softmax، فیچرهایی که استخراج می شوند همانند شکل زیر قابل تفکیک برای هر تصویر هستند (seperable) اما یک ویژگی مهم را دارا نیستند و آن قابلیت خوشه بندی فیچرهاست (discriminative). برای این که بتوانیم در مدل های آموزش دیده ی تشخیص چهره عملکرد مناسبی مشاهده کنیم لازم است که دو ویژگی ذکر شده یعنی seperable و discriminative در فیچرهای استخراج شده وجود داشته باشد.



شکل ۳-۳ seperable features

<sup>32</sup> Metric Learning

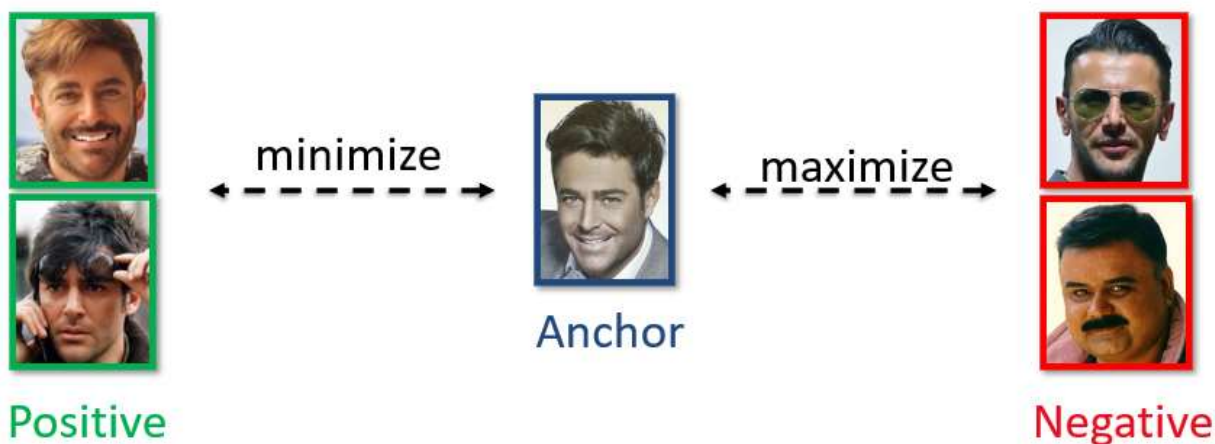


شکل ۳-۴ discriminative features

گفتیم یکی از موارد مهمی که در مورد شبکه های عصبی وجود دارد مسئله ی cost/lost function می باشد که معیار بد بودن مدل را مشخص می کند و باید به حداقل برسد. برای این که بتوانیم ویژگی discriminative بودن فیچرها را پیاده سازی کنیم، نیاز به استفاده از نوع جدیدی از تابع هزینه داریم. مثل Triplet loss و center loss

## ۲-۲-۳- تابع هزینه Triplet loss

در سال ۲۰۱۵ توسط شرکت گوگل مطرح شد. در این تابع هزینه به جای محاسبه ی خطا برای یک عکس، خطا را روی دسته های سه تایی از تصاویر یک شخص حساب می کنیم. دیتاست ما برای این loss به صورت بسته های سه تایی A، P و N است.



$$d(\text{Anchor}, \text{Positive}) + \alpha < d(\text{Anchor}, \text{Negative})$$

هدف ما در این تابع هزینه:

- حداقل کردن فاصله ی Anchor با Positive ها
- حداکثر کردن فاصله Anchor با Negative ها

$$\mathcal{L}(A, P, N) = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0)$$

فرمول بالا برای محاسبه ی loss به کار می رود. حال اگر ما دیتاستی شامل ۱۰۰۰۰ تصویر از ۱۰۰۰ نفر داشته باشیم نحوه ی انتخاب بسته های سه تایی مهم است. بسته های سه تایی نباید به صورت رندم انتخاب شوند اما باید از انتخاب بسته های خیلی سخت هم پرهیز کنیم تا به جواب صحیح برسیم.

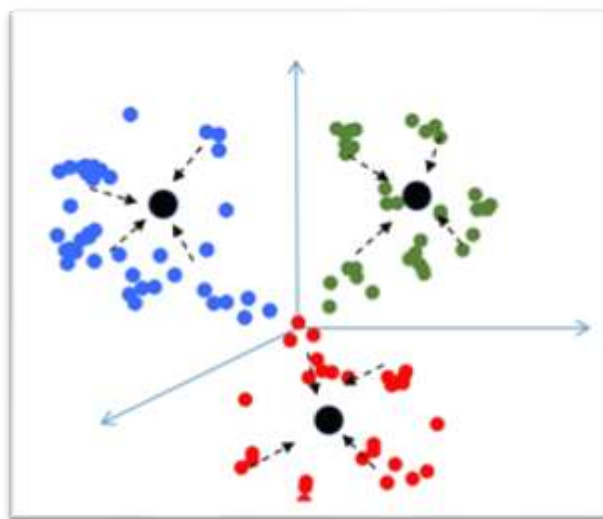
شرکت گوگل با استفاده از این تابع هزینه در سال ۲۰۱۵ توانست در چالش LFW<sup>33</sup> با ۹۹.۶۳ درصد برنده این چالش شود. همچنین در چالش MegaFace<sup>34</sup> نیز به درصد ۷۰.۵ برسد.

مشکلات triplet loss:

- نیاز به سایز mini-batch بزرگ (برای پردازش به سخت افزار قوی نیاز دارد) و پیدا کردن  $\alpha$  مناسب دارد.
- همگرایی کند و آهسته ای دارد.

### ۳-۲-۳- تابع هزینه Center loss

در سال ۲۰۱۶ مطرح شد. در این تابع هزینه با تشکیل خوشه<sup>35</sup> هایی تلاش می شود هر فیچر از هر کلاس به مرکز خوشه نزدیک شود.



شکل ۳-۵ خوشه بندی فیچرها

<sup>33</sup> Labeled Faces in the Wild-یکی از چالش های معروف تشخیص چهره

<sup>34</sup> یکی دیگر از چالش های تشخیص چهره که هر ساله برگزار می شود

<sup>35</sup> cluster

در شکل ۳-۵ به صورت شماتیک آنچه برای تحقق آن در center loss انجام می شود نشان داده شده است. فرمول center loss به صورت زیر است:

$$\mathcal{L}_C = \frac{1}{2} \sum_{i=1}^m \|x_i - c_{y_i}\|_2^2$$

$m$ : mini-batch size

$c_{y_i}$ :  $y_{th}$  class center in  $d$  dimension

$x_i$ : feature vector in  $R^d$  ( $d$  is the feature dimension)

تابع نهایی:

● **SOFTMAX LOSS:** encouraging the separability of features.  
 ● **CENTER LOSS:** simultaneously learning a center for deep features of each class and penalizing the distances between the deep features and their corresponding class centers.  
 ● **JOINT SUPERVISION:** minimizing the intra-class variations while keeping the features of different classes separable.

$$\mathcal{L} = \mathcal{L}_S + \lambda \mathcal{L}_C$$

$$= - \underbrace{\sum_{i=1}^m \log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^n e^{W_j^T x_i + b_j}}}_{\text{Inter-class Separability}} + \underbrace{\frac{\lambda}{2} \sum_{i=1}^m \|x_i - c_{y_i}\|_2^2}_{\text{Intra-class Compactness}}$$

شکل ۳-۶ فرمول تابع هزینه center loss

خلاصه ی center loss:

- ایجاد توده ی درون کلاسی و جدایی بین کلاس ها
- کارایی خوب، همگرایی ساده

این تابع هزینه توانست در چالش LFW به درصد ۹۹.۲۸ و در چالش MegaFace به ۶۵.۲۳۴ درصد برسد.



### ۴-۲-۳- مقایسه ی Center loss و Triplet loss

حال با معرفی این دو تابع هزینه به بررسی تفاوت های آنها پرداخته و در نهایت تابع هزینه ی مناسب را معرفی می کنیم.

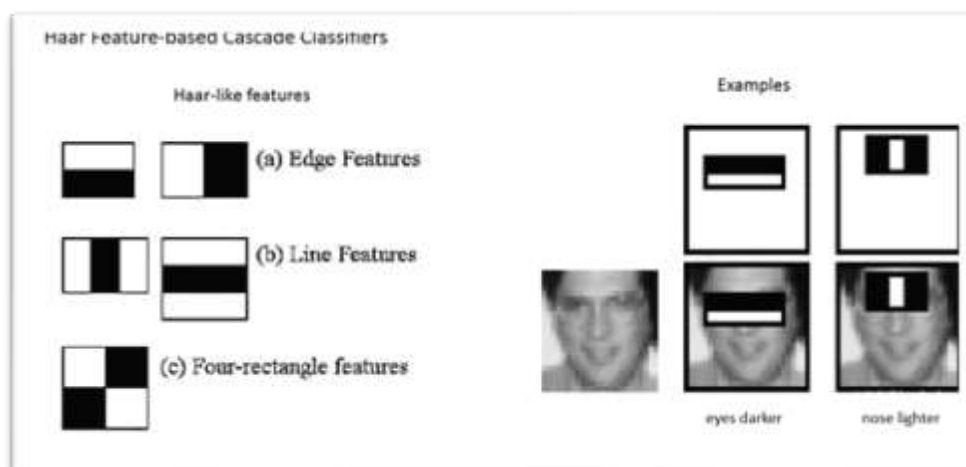
دیتاستی که شرکت گوگل برای آموزش مدل خود استفاده کرده به صورت عمومی منتشر نشده و یکی از بزرگترین دیتاست های موجود در زمینه ی چهره می باشد. این دیتاست شامل ۱۰۰ تا ۲۰۰ میلیون عکس از ۸ میلیون شخص متفاوت می باشد. در حالی که برای آموزش مدلی که از center loss استفاده شده در چالش LFW تنها از دیتاستی شامل ۷۰۰ هزار عکس از ۱۷ هزار و ۱۸۹ شخص متفاوت می باشد. همچنین در center loss و نیز در چالش MegaFace تنها از ۴۹۰ هزار عکس استفاده شده است.

با توجه به کوچک تر بودن دیتاست تابع center loss رسیدن به درصد دقتی مثل triplet loss نشان از عملکرد خوب این تابع هزینه است.

### ۳-۳- الگوریتم Viola-Jones

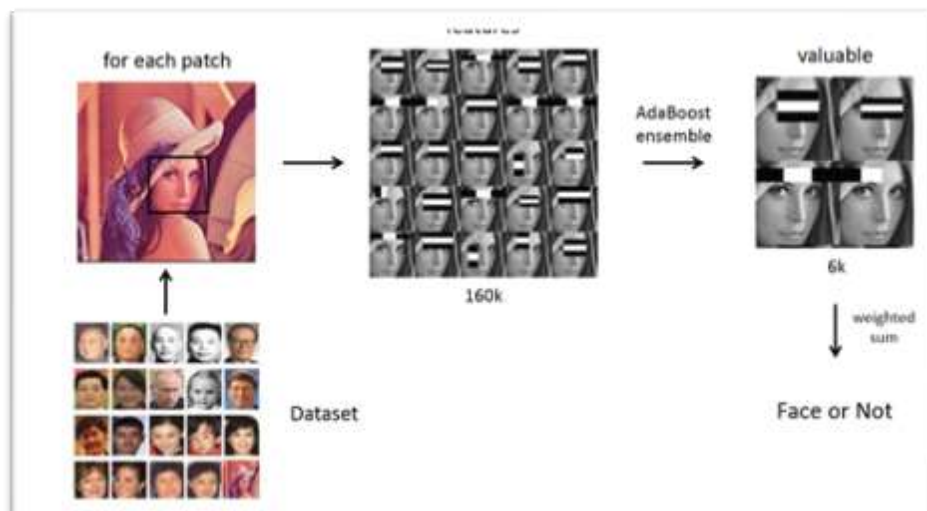
“ The Viola-Jones object detection framework is the first object detection framework to provide competitive object detection rates in real-time proposed in 2001 by Paul Viola and Michael Jones. “ Wikipedia

در این الگوریتم کلاسیک یک سری فیچر مهندسی شده با نام Haar Feature-based Cascade Classifiers وجود دارد. از این مجموعه فیچرها برای شناسایی نواحی مختلف چهره مانند چشم ها، پیشانی، لب ها و... استفاده می شود.



شکل ۳-۷ عملکرد الگوریتم کلاسیک تشخیص چهره

در فرآیند Training، این فیلترها روی قسمت های مختلف تصویر قرار می گرفت و با درصدهایی مشخص می شد بهتر است کدام فیلتر برای کدام بخش صورت استفاده شود.



شکل ۸-۳ الگوریتم Viola Jones

برای بهینه سازی عملکرد این الگوریتم پیشنهاد شد فیچرها با هم گروه هایی به نام stage تشکیل دهند. اگر patch انتخاب شده در هر stage رد شد الگوریتم غیر چهره برگردانده و خارج شود. این الگوریتم سرعت بالایی دارد و در سال های گذشته برای تشخیص چهره ی گوشی های هوشمند از این روش استفاده شده است.

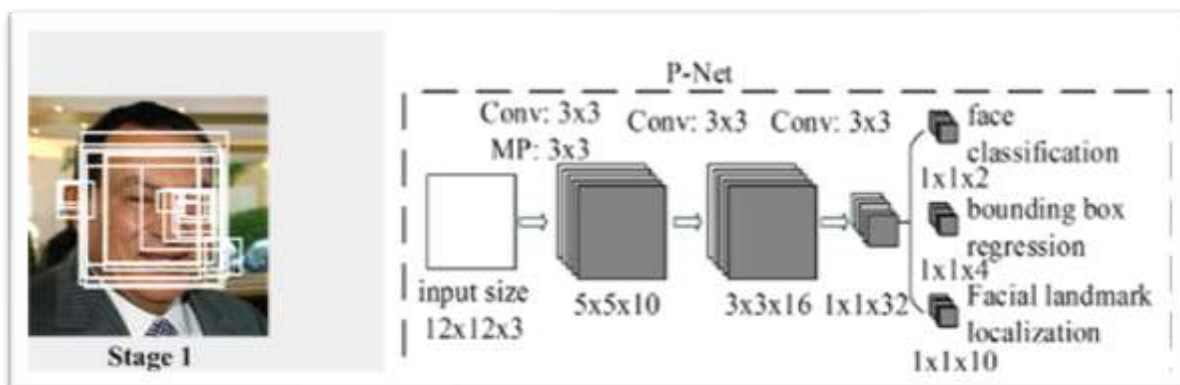
## ۴-۳- شبکه عصبی<sup>36</sup> MTCNN

MTCNN یک روش تشخیص چهره (Face Detection) مبتنی بر یادگیری عمیق است. از ایده ی الگوریتم Voila-Jones استفاده می کند یعنی فیچرها را روی قسمت های مختلف تصویر حرکت می دهد تا چهره های موجود در تصویر را شناسایی کرده و landmarkهای مربوط به ۵ ناحیه از صورت شامل چشم ها، نوک بینی، گوشه های دهان را بر می گرداند. اما تفاوت این روش با Voila-Jones در این است که فیچرها به صورت دستی استخراج نمی شوند بلکه در فرآیند آموزش و با شبکه cnn استخراج خواهند شد. از سه شبکه cnn به صورت آبشاری با نام های P-net، R-net و O-net تشکیل شده است.

<sup>36</sup> Multi-task Cascaded Convolutional Neural Networks

### ۱-۴-۳- شبکه ی پیشنهاد دهنده یا P-net<sup>37</sup>

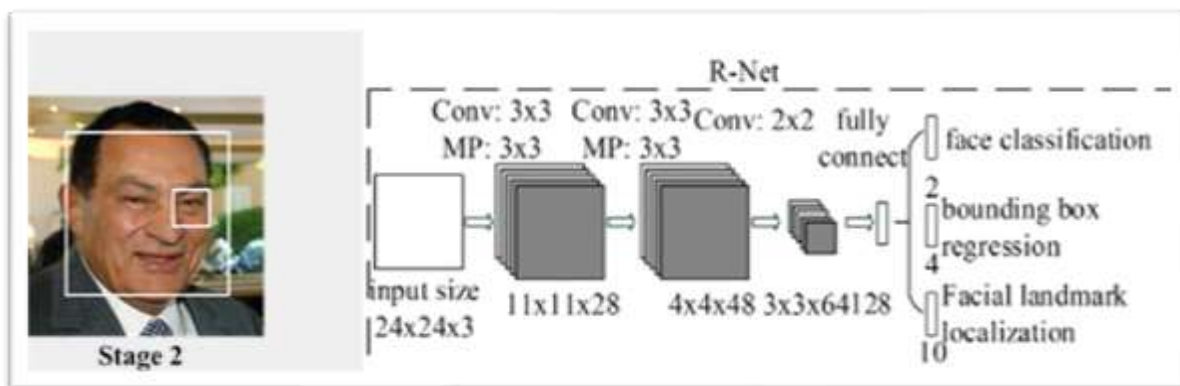
بسیار سبک می باشد. از این شبکه برای پیدا کردن تمامی نواحی مشکوک به وجود چهره استفاده می شود. این شبکه با سرعت خیلی بالا و دقت خیلی پایین عمل می کند و تمامی نواحی مربوط به چهره را برای ما پیدا می کند. این شبکه نرخ مثبت کاذب خیلی بالایی دارد.



شکل ۹-۳ شبکه پیشنهاد دهنده ی mtcnn

### ۲-۴-۳- شبکه دوم، شبکه ی محدود کننده یا R-net<sup>38</sup>

ورودی این شبکه تصویر خروجی از شبکه ی اول می باشد. با دقت بیشتری نسبت به شبکه قبلی، نواحی اشتباه غیر چهره را حذف می کند و نواحی اصلی چهره را با یک کادر مربعی مشخص می کند. این شبکه تعداد پارامترهای بیشتری دارد. در طول آموزش یک مدل، تعدادی پارامتر قابل یادگیری و در مواردی غیرقابل یادگیری از هر لایه استخراج می شود که نشان دهنده ی حجم و قدرت شبکه است.



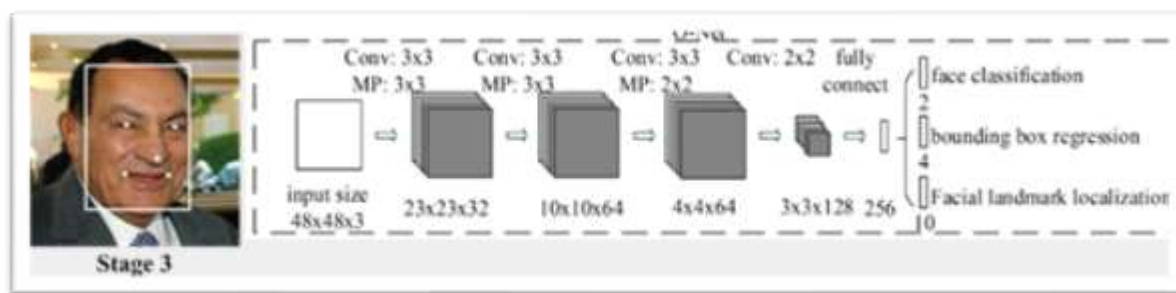
شکل ۱۰-۳ شبکه ی محدود کننده ی mtcnn

<sup>37</sup> Proposal Network

<sup>38</sup> Refine Network

### ۳-۴-۳- شبکه ی سوم، شبکه خروجی یا O-net<sup>39</sup>

کار این شبکه، شبیه به عملکرد شبکه دوم برای نواحی باقی مانده است. از لحاظ تعداد پارامتر و تعداد لایه این شبکه از دو شبکه قبلی پیچیده تر و کندتر است. دقت شبکه را بالا می برد و با قطعیت وجود یا عدم وجود چهره در نواحی باقی مانده که توسط دو شبکه قبل تشخیص داده شده است را مشخص می کند.



شکل ۱۱-۳ شبکه خروجی mtcnn

هر سه شبکه ی mtcnn، همزمان به پیدا کردن landmarkهای چهره نظیر چشم ها و بینی و گوشه های دهان می پردازند. شبکه ی آخر در نهایت، دقت رگرسیون بالایی برای این نقاط دارد. روش mtcnn از دقت بالا و سرعت پایین برای تشخیص چهره برخوردار است. معمولاً در پروژه ها و مقالات مربوط به Face Recognition از وزن های شبکه های pretrain مربوط به face detection مثل MTCNN استفاده می کنند. زیرا که زمان آموزش این مدل ها زیاد است و نیاز به سخت افزار نسبتاً قوی دارد.

در این پروژه نیز از شبکه ی MTCNN برای انجام مبحث face detection استفاده شده است. سپس با عبور دادن تصاویر از شبکه، embedding تصاویر استخراج شده و با حد آستانه مشخص شده در مقاله ی mtcnn به بررسی و مقایسه ی تصاویر می پردازیم تا خروجی مورد نظر تولید شود.

در فصل بعدی نحوه ی پیاده سازی این پروژه بررسی خواهد شد.

<sup>39</sup> Output Network

## **فصل چهارم**

### **پیاده سازی**

## ۱-۴-پیش نیاز ها

برای شروع کار نیاز است برخی برنامه ها در سیستم نصب باشد. شرح نرم افزارهای مورد نیاز و کاربردها در روند پروژه به شرح زیر است:

**Python:** یک زبان شی گراست<sup>۴۰</sup> که امروزه در کارهای هوش مصنوعی و علم داده<sup>۴۱</sup> بسیار مورد استفاده قرار می گیرد. برای این پروژه بهتر است نسخه ی ۳.۷ نصب شده باشد.

**IDE<sup>۴۲</sup> یا Text Editor** برای زبان پایتون: وجود یکی از این دو برنامه برای امکان درج و ویرایش کدهای Python لازم و ضروری است. تفاوت IDE و Text Editor در امکاناتی است که در اختیار برنامه نویس قرار می دهد. در IDE برنامه نویس با قابلیت های بیشتری برای عیب یابی<sup>۴۳</sup> و رفع خطاها و برخی امکانات جانبی دیگر رو به روست و معمولا هر IDE توانایی پشتیبانی از تعداد محدودی از زبان های برنامه نویسی دارد در حالی که با Text Editor برنامه های زبان های مختلف را می توان مشاهده و ویرایش کرد. برای زبان پایتون Pycharm و Spyder دو IDE معروف می باشد که استفاده از آنها رایج است. ما در این پروژه از Spyder که در برنامه ی Anaconda قرار دارد استفاده کرده ایم.

**Pip یا Anaconda:** Pip برنامه ی مدیریت کتابخانه های پایتونی می باشد که برای نصب و حذف کتابخانه ها مورد استفاده قرار می گیرد. Anaconda نیز مجموعه ای از ابزارهایی است که برای کار روی علم داده مورد نیاز است. با استفاده از آناکوندا می توان IDE هایی نظیر Spyder یا Pycharm را نصب کرد. همچنین ابزارهای دیگری مثل Jupyter Notebook که برای تست و آموزش کدهای پایتونی قابل استفاده است و... موجود می باشد. ما در این پروژه از آناکوندا نسخه ی ۳ استفاده می کنیم.

---

<sup>۴۰</sup> OOP-Object Oriented Programming language

<sup>۴۱</sup> Data Science

<sup>۴۲</sup> محیط توسعه ی مجتمع - Integrated Development Environment

<sup>۴۳</sup> Debug

## ۲-۴- کتابخانه <sup>۴۴</sup>های مورد نیاز

کتابخانه ها مجموعه ای از توابع از پیش تعریف شده و آماده هستند که با فراخوانی <sup>۴۵</sup> آنها به پروژه می توانیم از تمامی توابع و امکانات آن ها استفاده کنیم. کتابخانه ها گاه از سوی توسعه دهندگان <sup>۴۶</sup> زبان های برنامه نویسی و گاه به صورت متن باز <sup>۴۷</sup> از سوی جمعی از برنامه نویسان عرضه می شوند و در اختیار سایرین قرار می گیرند. ما نیز برای این پروژه از برخی کتابخانه های مطرح در این زمینه استفاده خواهیم کرد.

ذکر این نکته قابل اهمیت است که آیا می توان بدون بهره گیری از برخی کتابخانه ها به توسعه ی برنامه ها از جمله این پروژه پرداخت؟ جواب بله است اما با توجه به پیشرفت های موجود در این علوم، لازم است از بهترین و کم خطاترین روش ها برای رسیدن به جواب درست استفاده کرد و برای این کار لازم است از ابزارهای لازم استفاده کنیم.

### ۱-۲-۴- NumPy

NumPy یک کتابخانه برای زبان برنامه نویسی پایتون است. با استفاده از این کتابخانه امکان استفاده از آرایه ها و ماتریس های بزرگ چند بعدی فراهم می شود. همچنین می توان از تابع های ریاضیاتی سطح بالا بر روی این آرایه ها استفاده کرد. هدف اصلی NumPy فراهم ساختن امکان کار با آرایه های چندبعدی همگن است. این آرایه ها جدولی از عناصر (معمولاً اعداد) هستند که همگی از یک نوع می باشند و با یک چندتایی، از اعداد صحیح مثبت اندیس گذاری می شوند. در NumPy ابعاد به نام محور (axe) شناخته می شوند. تعداد محورها رتبه (rank) نامیده می شود. برای نصب NumPy از یکی از دستورات زیر استفاده می شود (با توجه به محیط توسعه)

برای pip

Pip install numpy

برای anaconda

Conda install numpy

---

<sup>44</sup> Library

<sup>45</sup> Import

<sup>46</sup> Developers

<sup>47</sup> Open-Source

با درج یکی از دستورات بالا در محیط cmd یا prompt برنامه ی آنکوندا فایل های مورد نیاز دانلود شده و کتابخانه نصب می شود. قبل از نصب باید ارتباط سیستم با اینترنت برقرار باشد.

## Matplotlib-۴-۲-۲

مصورسازی داده یا تجسم سازی داده<sup>۴۸</sup> ارائه ی گرافیکی داده است که هدف اصلی آن انتقال بهینه ی اطلاعات به کاربران از طریق نمایش روابط میان داده ها به کمک نمودارها است همچنین فرآیند تحلیل و تصمیم گیری مبتنی بر اطلاعات را راحت تر میکند. Matplotlib از کتابخانه های معروف پایتون برای ترسیم نمودارهاست. برای نصب این کتابخانه از یکی از دستورات زیر استفاده می شود:

برای Pip

Pip install matplotlib

و برای آنکوندا

Conda install matplotlib

## SciPy-۴-۲-۳

کتابخانه ی SciPy بر مبنای کتابخانه ی NumPy است و امکان کار با آرایه های n بعدی را فراهم می کند. این کتابخانه برای کار با آرایه های Numpy ایجاد شده است و بسیاری از عملیات محاسباتی و بهینه سازی را به طور کارا ممکن می کند. هر دو پکیج Numpy و Scipy روی سیستم عامل های موجود کار می کنند و به راحتی نصب می شوند. برای نصب آن:

در pip

Pip install scipy

در anaconda

Conda install scipy

---

<sup>48</sup> Data Visualization



## OpenCV<sup>۴۹</sup>-۴-۲-۴

کتابخانه OpenCV بدون شک قدرت مند ترین کتابخانه کار با تصاویر است. OpenCV برای فراهم آوردن یک زیرساخت مشترک برای برنامه های کاربردی بینایی کامپیوتری ( computer vision ) و سرعت بخشیدن به استفاده از ادراک ماشین در محصولات تجاری ساخته شده است. این کتابخانه دارای بیش از ۲۵۰۰ الگوریتم بهینه سازی شده است که شامل مجموعه ای جامع از کلاسیک و پیشرفته ترین بینایی کامپیوتر و الگوریتم های یادگیری ماشین است. برای نصب:

در pip

Pip install opencv

و در آنکوندا

Conda install opencv

## Keras-۴-۲-۵

کتابخانه Keras یک کتابخانه متن باز شبکه عصبی است که در پایتون نوشته شده است. این کتابخانه روی کتابخانه های TensorFlow, Microsoft Cognitive Toolkit, R Theano, یا PlaidML قابل اجرا است. Keras در شبکه تصاویر کاربرد دارد و می تواند برای پیش بینی طبقه بندی، استخراج ویژگی ها و تنظیمات مربوط به مدل استفاده شود. در حقیقت کراس یک فرانت اند (front-end) برای چارچوب<sup>۵۰</sup> های یادگیری عمیق تنسرفلو، CNTK و تیانو است و آن ها پشت شبکه های عصبی را می سازند و آموزش می دهند و برای همین به آن یک چهارچوب سطح بالا می گوییم چون کراس پیچیدگی استفاده از این کتابخانه ها را تا حد خوبی حذف می کند. یک ویژگی خاص دیگر کراس این است که محدود به یک کتابخانه یادگیری عمیق نیست و همانطور که گفتیم می توانیم از تنسرفلو، CNTK و یا تیانو برای محاسبات پشت پرده آن استفاده کنیم.

برای نصب در pip:

Pip install keras

و برای نصب در آنکوندا

Conda install keras

---

<sup>49</sup> Open Source Computer Vision

<sup>50</sup> Framework

## ۶-۲-۴- تفاوت چارچوب با کتابخانه

کتابخانه، مجموعه‌ای از کدهای مختلف است که توابع پر استفاده را در اختیار کاربر قرار می‌دهد تا در وقت او صرفه‌جویی شود. برای مثال، یک کتابخانه ریاضی، توابع مهم ریاضی مانند تابع نمایی، لگاریتمی، مثلثاتی و ... را در برمی‌گیرد. زبان‌های برنامه‌نویسی، معمولاً کتابخانه‌های متعددی برای انجام امور مختلف دارند. کتابخانه‌هایی با موضوعات پردازش داده، رسم نمودار، آنالیز متن و ... نوشتن این توابع از ابتدا، معمولاً انرژی و زمان زیادی را از برنامه‌نویس می‌گیرد.

چارچوب، یک چارچوب یا اسکلت استاندارد برای ساخت و توسعه نرم‌افزارهاست. فریم ورک، اسکلت کلی یک نرم‌افزار را برای شما فراهم می‌کند و شما می‌توانید جاهای خالی آن را با کدهای مدنظر خود پر کنید. با استفاده از یک فریم ورک، برای نوشتن برنامه‌های مشابه، صرفه‌جویی زیادی در زمان می‌شود و کدها منظم‌تر نوشته می‌شوند.

به طور کلی، تفاوت چارچوب و کتابخانه، نحوه‌ی کنترل کدهاست. در کتابخانه، شما کد را فراخوانی می‌کنید و از آن استفاده می‌کنید؛ اما، در مورد فریم ورک، داستان کاملاً متفاوت است. فریم ورک، به اصطلاح Inversion of Control (کنترل‌پذیر معکوس) است. به این معنی که بر خلاف کتابخانه، کاربر، فریم ورک را فراخوانی نمی‌کند، بلکه فریم ورک، کدهای اضافه شده توسط کاربر را فراخوانی می‌کند.

## ۷-۲-۴- Tensorflow

محبوب‌ترین کتابخانه یادگیری ماشینی دنیا، در حال حاضر، کتابخانه‌ی TensorFlow گوگل است. تنسورفلو یک پلتفرم متن‌باز و پایانه به پایانه (end-to-end) است. یکی از ویژگی‌های متمایزکننده‌ی TensorFlow از سایر پلتفرم‌ها و کتابخانه‌ها، آن است که این کتابخانه می‌تواند بر روی GPU یا CPU اجرا شود. بنابراین، برای اجرای برنامه در سطح GPU نیازی به استفاده از زبان‌های سطح پایین‌تر مانند C++ نیست. تنسورفلو از یک سیستم چند لایه استفاده می‌کند. این سیستم شما را قادر می‌سازد، به سرعت، شبکه‌های عصبی را با حجم زیادی از داده، نصب کنید و آموزش دهید.

TensorFlow فریم ورکی است که داده‌های آن به‌صورت تنسور تعریف و اجرا می‌شوند. همین دلیل سبب می‌شود تا TensorFlow به یک فریم ورک محبوب برای آموزش و اجرای شبکه‌های عصبی عمیق در برنامه‌های کاربردی مبتنی بر هوش مصنوعی تبدیل شود و به‌صورت گسترده در تحقیقات مرتبط با یادگیری عمیق به کار گرفته شود.

تنسورفلو در اصل یک کتابخانه سطح پایین است که برای کتابخانه ی keras، به عنوان back-end عمل می کند. البته از کتابخانه ی keras به صورت مستقل نیز می توان استفاده نمود. کتابخانه کراس استفاده از کتابخانه تنسورفلو را راحت تر می کند. به دلیل نقش back-end داشتیم تنسورفلو برای کراس به چارچوب نیز معروف است. اما نحوه ی استفاده از آن در پروژه ها مانند سایر کتابخانه های پایتون با استفاده از کلمه ی کلیدی import می باشد.

همچنین دلیل دیگر فریمورک بودن تنسورفلو این است که در صورت اضافه نمودن آن به پروژه لازم است برای تعریف شبکه عصبی و آموزش و تست گرفتن برخی نکات و چارچوب های تعریف شده را رعایت کنیم.

گفتیم مزیت اصلی تنسورفلو قابلیت اجرای آن بر روی پردازنده ی اصلی یعنی CPU و نیز پردازنده کارت گرافیکی یا GPU می باشد. دلیل استفاده از GPU به جای پردازنده اصلی سیستم این است که CPU یک منبع پردازشی گران قیمت بوده و محاسباتی که در شبکه های یادگیری عمیق انجام می شوند مجموعه ای از توابع ریاضی با تعداد تکرار زیاد می باشند به همین دلیل برای جلوگیری از هدررفت منابع سیستم توصیه می شود از GPU برای لود مدل و آموزش آن استفاده نمود.

البته می توان از CPU نیز برای اجرای تنسورفلو استفاده نمود. به صورت پیش فرض با وارد کردن دستور زیر نسخه ی قابل اجرا روی CPUی تنسورفلو بر روی سیستم نصب خواهد شد:

برای pip

Pip install tensorflow

و برای آنکوندا

Conda install tensorflow

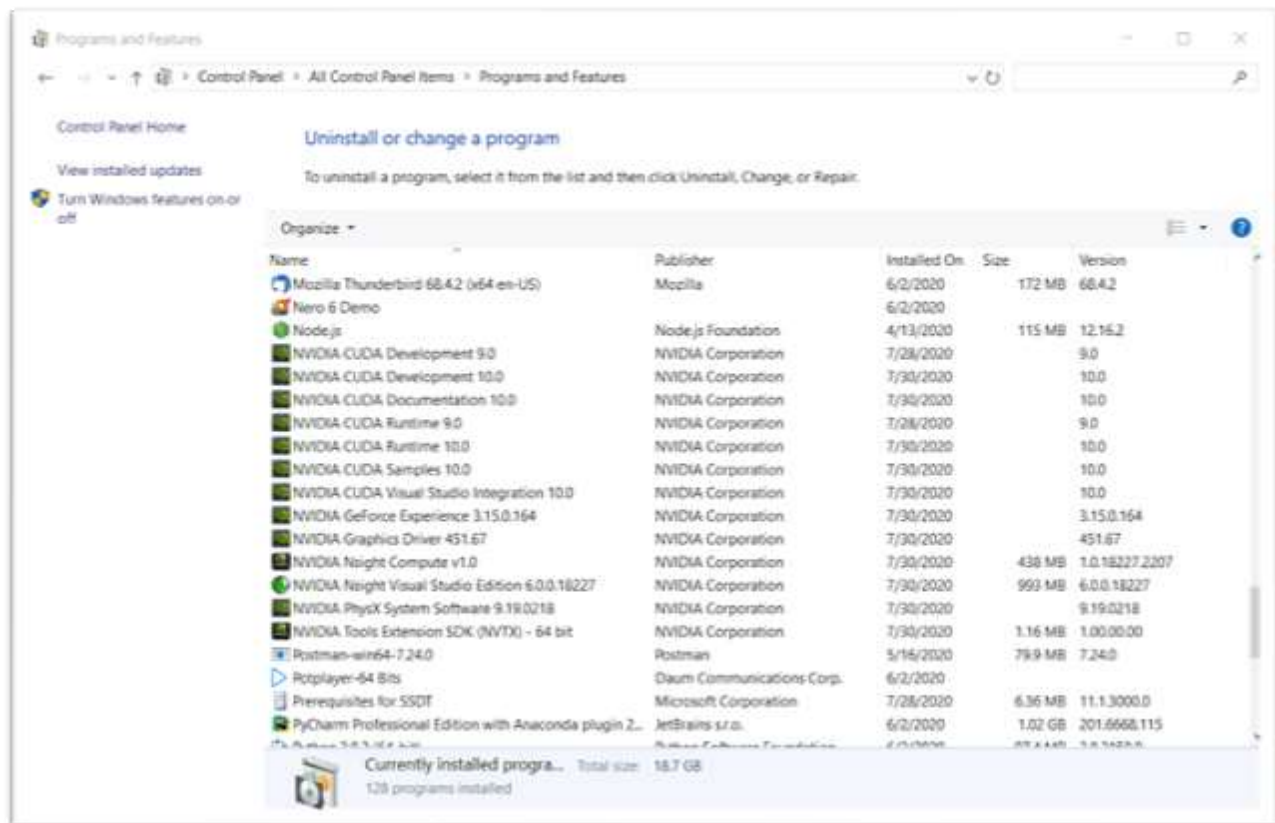
اما نصب تنسورفلو بر روی GPU مراحل بیشتری دارد که در ادامه شرح داده خواهد شد.

## ۸-۲-۴- نصب تنسورفلو روی GPU

در ابتدا باید این نکته را ذکر کنیم که اگر کارت گرافیک سیستم مورد نظر، از کارت گرافیک های nvidia باشد نسخه ی تنسورفلوی GPU قابل نصب در آن سیستم است و در غیر این صورت باید از تنسورفلوی CPU استفاده کرد. با توجه به وجود کارت گرافیک nvidia مراحل نصب این نسخه در ادامه بیان می شود.

برای نصب تنسورفلو باید ابتدا Cuda رو سیستم مورد نظر نصب باشد. برای این کار در ابتدا لازم است برنامه ی Nvidia (که در هنگام نصب ویندوز و معرفی کارت گرافیک به سیستم نصب می شود) حذف شده باشد. حذف این برنامه آسیبی به درایورهای کارت گرافیک نمی زند. برای حذف می توان در سیستم عامل ویندوز از مسیر زیر اقدام به حذف برنامه nvidia کرد:

### Control Panel/ Programs and Features



شکل ۴-۱ program and features

در مسیر یاد شده برنامه هر برنامه ای با نام nvidia موجود باشد باید حذف شود. این کار برای جلوگیری از تداخل نسخه های Cuda با یکدیگر می باشد. در صورتی که بخواهیم از مشکلات بعدی در نصب های بعدی جلوگیری

کنیم می توانیم از برنامه هایی مثل Your Uninstaller که مخصوص حذف برنامه ها می باشد. Your Uninstaller با حذف اطلاعات برنامه های حذفی از بخش رجیستری ویندوز باعث می شود از حذف و نصب ناقص برنامه ها جلوگیری کنیم.

بعد از حذف برنامه های nvidia نوبت نصب Cuda می باشد. در ابتدا از طریق سایت Tensorflow.org نسخه سازگار Cuda با نسخه ی تنسورفلوی نصبی را چک می کنیم. برای این پروژه ما از تنسورفلوی نسخه ی ۲ و برای کودا از نسخه ی ۱۰ این نرم افزار استفاده کرده ایم.

اما Cuda چیست و چرا باید نصب شود؟

"CUDA مخفف عبارت انگلیسی Compute Unified Device Architecture است یک سکوی پردازش موازی و مدل برنامه نویسی است که توسط شرکت انویدیا به وجود آمده است و در واحدهای پردازش گرافیکی این شرکت پشتیبانی می شود. کودا به توسعه دهنده گان نرم افزار اجازه می دهد تا از یک GPU که ویژگی -CUDA-enabled دارد برای هدف پردازش استفاده کنند، رویکردی که GPGPU شناخته می شود. کودا به توسعه دهنده گان امکان دسترسی مستقیم به حافظه و مجموعه دستورالعمل در واحد پردازش گرافیکی را می دهد." ویکی پدیا با استفاده از کودا می توانیم تنسورفلو را روی کارت گرافیک اجرا کنیم و نخ<sup>۵۱</sup> های پردازش موازی در GPU ایجاد کنیم که کاربرد زیادی دارد. به طور مثال به صورت همزمان با یک نخ تصاویری را از هارد دیسک به حافظه ی اصلی بارگزاری کنیم و با نخ دیگری عمل آموزش مدل پیگیری شود.

در سایت developer.nvidia.com می توانیم نسخه ی کودای سازگار با نوع سیستم خود را پیدا کرده و دانلود کنیم. بعد از اتمام دانلود Cuda Toolkit 10.0 به سراغ نصب آن می رویم. فرآیند نصب ساده است و به راحتی نصب می شود.

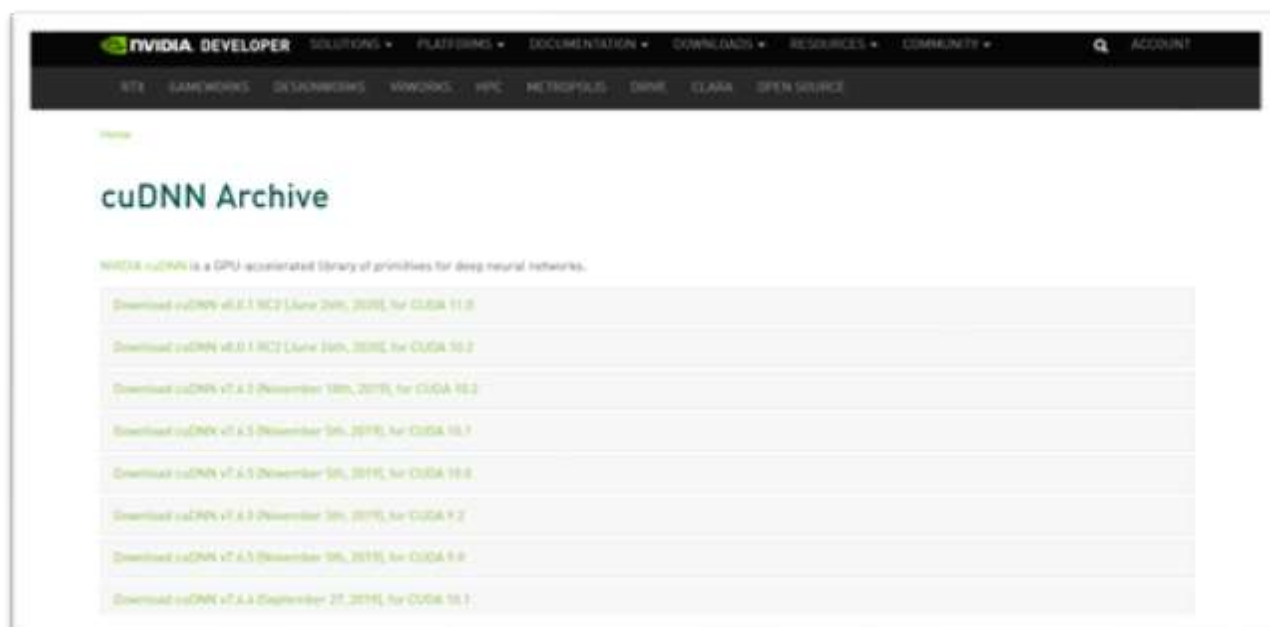
---

<sup>51</sup> Thread



شکل ۴-۲ cuda toolkit v.10.0

بعد از نصب Cuda Toolkit نوبت به cuDNN می باشد. از سایت nvidia پس از ساخت یک حساب کاربری و ورود می توان نسخه ی سازگار cuDNN با کودای نصب شده را دانلود کنیم.



شکل ۴-۳ cuDNN

پس از دانلود cuDNN مشاهده می شود فایل فشرده شامل سه پوشه به نام های bin, include و lib می باشد. لازم است محتویات پوشه include و lib در مسیر زیر کپی شود:

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0

در مسیر ذکر شده سه پوشه ی lib, include و bin موجود است. محتویات فایل های دانلود شده را درون پوشه های نامبرده کپی می کنیم.

پس از کپی لازم است تغییراتی را در سیستم عامل ویندوز ایجاد کنیم. برای این کار به قسمت control panel ویندوز وارد شده و در قسمت System روی گزینه Advanced system setting کلیک می کنیم. بعد از کلیک در کادر باز شده روی گزینه Environment Variables کلیک می کنیم.

در کادر باز شده از بخش system variables کلمه ی PATH را پیدا کرده و روی دکمه ی Edit کلیک می کنیم. در بخش جدید روی New کلیک کرده و دو عبارت زیر را وارد می کنیم:

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\libnvvp

تغییرات را ذخیره کرده و خارج می شویم. حال در محیط Anaconda یا Pip و در قسمت prompt دستور زیر را وارد می کنیم:

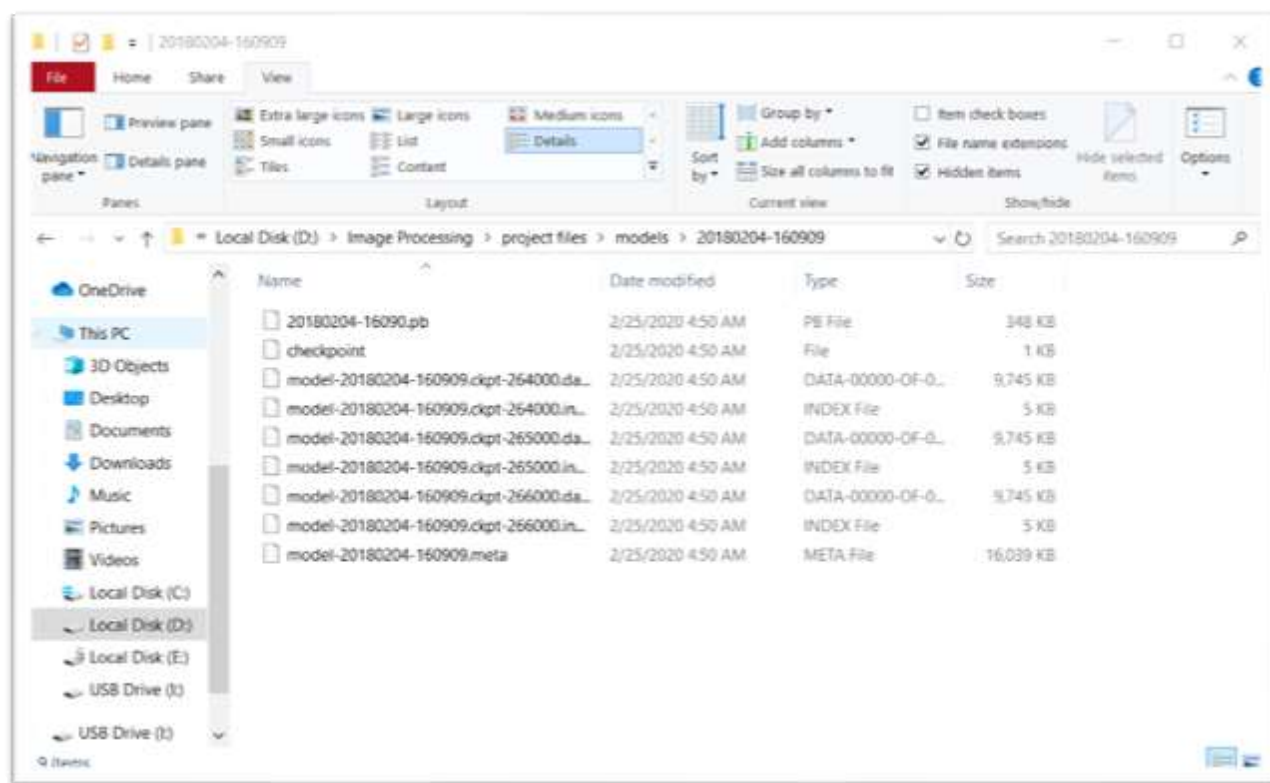
Pip/Conda install --ignore-installed --upgrade tensorflow-gpu

بدین ترتیب توانستیم نسخه ی قابل اجرای تنسورفلو روی کارت گرافیک را نصب و راه اندازی کنیم. سپس می توانیم درایور جدید Nvidia را دانلود و نصب کنیم.

تا این مرحله تمام پیش نیازها برای اجرای کدها و برنامه نصب و راه اندازی شده است. حال به سراغ بررسی کدها و خرجی برنامه خواهیم رفت.

### ۳-۴- پیاده سازی

هر مدل شبکه عصبی بعد از اتمام فرآیند آموزش، وزن های خود را در فایل هایی ذخیره می کند تا در موارد بعدی به جای آموزش مجدد از وزن های آماده استفاده شود. بنابراین وزن های شبکه ی MTCNN در کنار فایل های پروژه و در پوشه ی models قرار دارد:



شکل ۴-۴: محتویات پوشه model

محتویات پوشه ی model در هنگام اجرای برنامه فراخوانی خواهند شد و به ما در فرآیند تشخیص چهره کمک خواهند نمود.



**نکته مهم:** به دلیل وجود دو نسخه متفاوت از تنسورفلو<sup>۵۲</sup>، کدهای نوشته شده در برنامه ی ما با تنسورفلوی نسخه ی ۱ کار می کند به همین دلیل در هنگام فراخوانی تنسورفلو از قطعه کد زیر استفاده می کنیم تا تفاوت های موجود در نسخه های مختلف منجر به بروز مشکل در برنامه ما نشود:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

### ۱-۳-۴-FaceRecognition.py

اجرای برنامه از این فایل شروع می شود. در این فایل کدهایی برای فراخوانی تصاویر چهره های افراد از پوشه ی images و استخراج embedding آنان و نیز عمل جستجوی چهره و رسم کادر مستطیل سبزرنگ اطراف هر چهره ی شناسایی شده و ثبت log تردها در فایل log.txt با زمان عبور و مرور انجام میشود. در ادامه به بررسی همه ی کدها می پردازیم:

#### گام اول: فراخوانی کتابخانه های مورد نیاز

برای فراخوانی کتابخانه های نصب شده در پایتون از ساختار دستور زیر استفاده می شود:

نام مستعار as نام کتابخانه Import

نام کتابخانه، نام یکی از کتابخانه های از قبل نصب شده روی سیستم ما می باشد که بنا به نیاز قبل از استفاده در برنامه باید آن را فراخوانی کنیم. نام مستعار نیز، نامی دلخواه است که به جای نام اصلی کتابخانه در طول برنامه از آن استفاده می شود. مقداردهی نام مستعار اختیاری است ولی همه ی کتابخانه های مورد نیاز باید قبل از استفاده در طول برنامه فراخوانی شوند.



```
1  #-*- coding: utf-8 -*-
2  """
3  @author: ZahraAkbari
4  Final Project of Bachelor Degree
5  Technical and Vocational University
6  Dr.Shariati Technical College
7  """
8  import tensorflow.compat.v1 as tf
9  tf.disable_v2_behavior()#تنسورفلو 1 نسخه ی 2 به نسخه ی 1
10 import numpy as np
11 import cv2
12 from detection.mtcnn import detect_face#کدهای ریچون دیوید مندرکز
13 from keras.preprocessing import image
14 from scipy import misc
15 import FaceToolKit as ftk
16 import os
17 import datetime
```

نسخه ی بتای تنسورفلوی ۲ در سال ۲۰۱۹ معرفی و در سال ۲۰۲۰ نسخه ی اصلی معرفی شد. <sup>52</sup>

از کتابخانه `datetime` OS برای مسيردهی فایل های موردنیاز برنامه در حین اجرا استفاده می شود. کتابخانه `datetime` نیز برای استخراج زمان کنونی سیستم به کار می رود.

```
from detection.mtcnn import detect_face
```

فایل `detect_face` یک فایل پایتونی است که ساختار شبکه ی `mtcnn` را پیاده سازی کرده و عمل جستجو و تشخیص چهره را با عبور تصاویر از شبکه ی `mtcnn` با وزن های آماده و استخراج `bounding_box` های مورد نیاز انجام میدهد. این فایل کتابخانه نیست بلکه کدهای ریپازیتوری<sup>۵۳</sup> گیت هاب<sup>۵۴</sup> دیوید سندبرگ<sup>۵۵</sup> است که ما برای تعریف `mtcnn` و لود وزن ها و عمل تشخیص چهره از آن استفاده می کنیم.

منظور از کد بالا فایل `detect_face` موجود در پوشه ی `mtcnn` که در پوشه ی `detection` (هم مسیر با فایل اصلی پروژه) قرار دارد می باشد.

فایل `FaceToolkit.py` نیز کدهای نوشته شده توسط ما برای استخراج `embedding` تصاویر می باشد. به صورت زیر فراخوانی می شود:

```
import FaceToolkit as ftk
```

گام دوم: تعریف متغیرها، ساخت اشیاء و سایر پیش نیازها

```
tf.reset_default_graph()
```

این دستور برای جلوگیری از بروز خطا در هنگام اجرای دوباره فایل `FaceRecognition` می باشد. بدون استفاده از این قطعه کد، با خطای عدم بارگزاری مجدد ماژول های مورد نیاز می شویم.

```
v=ftk.Verification();
```

از کلاس `Verification` موجود در فایل `FaceToolkit` یک شی جدید ساخته می شود.

نگاهی به تابع `__init__` این کلاس که در هنگام ساخت شی فراخوانی می شود می اندازیم:

---

<sup>53</sup> هر پروژه در گیت هاب ریپازیتوری نامیده می شود

<sup>54</sup> [www.github.com](http://www.github.com)

<sup>55</sup> David Sandberg

```

7
8 import tensorflow.compat.v1 as tf
9 tf.disable_v2_behavior()
10 import numpy as np
11 from facenet import face
12 class Verification:
13     def __init__(self):
14         gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.333)
15         self.session = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
16         self.images_placeholder = ""
17         self.embeddings = ""
18         self.phase_train_placeholder = ""
19         self.embedding_size = ""
20         self.session_closed = False

```

در هنگام ساخت شی از این کلاس، با استفاده از دستور GPUOptions فضای حافظه<sup>۵۶</sup> ی کارت گرافیک را به بخش‌هایی تقسیم می‌کند تا تمام قدرت یک GPU صرف یک عملیات نشود و بتواند کارهای مختلفی را در یک زمان واحد مدیریت کند.

هر گراف در تنسورفلو برای انجام محاسبات لازم استفاده می‌شود. مفهوم session نیز برای اجرای گراف‌ها یا بخشی از یک گراف استفاده می‌شود. بدین ترتیب با استفاده از session منابع لازم به گراف‌ها اختصاص پیدا می‌کند و نتایج گراف‌ها نیز در session‌ها ذخیره می‌شود.

پس از تنظیمات مربوط به حافظه ی کارت گرافیک و session چند متغیر تعریف شده است که در طول اجرای برنامه مورد استفاده قرار می‌گیرد. ادامه ی کدهای فایل FaceRecognition را بررسی می‌کنیم:

```
v.load_model("./models/20180204-160909/")
```

پس از تعریف شی و اجرای تابع \_\_init\_\_ به فراخوانی مدل از پیش آموزش دیده بر روی شبکه ی تعریف شده برای استخراج ویژگی‌هاست. نگاهی به تابع load\_model فایل FaceToolkit می‌اندازیم:

```
def load_model(self, model):
```

```
    face.load_model(model, self.session)
```

در این تابع ورودی به تابع load\_model فایل face انتقال داده شده است. از فایل face.py برای لود وزن‌های مدل و ساخت تنسورهای تصاویر استفاده می‌شود.

---

<sup>56</sup> memory

```

56 def load_model(model, session):
57     model_exp = os.path.expanduser(model)
58     if os.path.isfile(model_exp):
59         with gfile.FastGFile(model_exp, 'rb') as f:
60             graph_def = tf.GraphDef()
61             graph_def.ParseFromString(f.read())
62             tf.import_graph_def(graph_def, name='')
63     else:
64         meta_file, ckpt_file = get_model_filenames(model_exp)
65         saver = tf.train.import_meta_graph(os.path.join(model_exp, meta_file))
66         saver.restore(session, os.path.join(model_exp, ckpt_file))

```

در تابع `load_model` فایل `face`، گراف های محاسباتی از مدل آموزش دیده بارگزاری شده و گراف ها برای محاسبات بعدی نگهداری می شوند.

`v.initial_input_output_tensors()`

در `Tensorflow`، تمام محاسبات شامل تنسور است. تنسور یک بردار یا ماتریس با ابعاد `n` است که انواع داده ها را نشان می دهد. همه مقادیر در یک تنسور دارای یک نوع داده یکسان با یک شکل شناخته شده (یا جزئی شناخته شده) هستند. در تابع فوق نیز، تنسورهای لازم برای محاسبات بارگزاری خواهند شد.

در ادامه ی کدهای `FaceRecognition`، برخی متغیرها و مسیرها به برنامه معرفی می شوند.

`image_size = 160`

`verification_threshold=1.188`

مقادیر متغیرهای `image_size` و `verification_threshold` نباید به صورت تصادفی یا دلخواه مقداردهی شوند، برای تنظیم این دو متغیر به مقدار ذکر شده در مقاله ی `mtcnn` رجوع می شود.

`default_color = (0, 255, 0) #BGR`

`default_thickness = 2`

کدهای بالا نیز برای رسم کادر مستطیل دور هر چهره به کار می رود که مقادیر دلخواهی دارد. در این برنامه ما از کادر سبز رنگ استفاده کرده ایم.

`file_object = open('log.txt', 'a')`

برای ثبت تردد رفت و آمدها از یک فایل متنی استفاده می کنیم. کد فوق یک فایل با نام `log` ایجاد کرده و برای درج اطلاعات باز می کند.

```
base_dir=os.path.expanduser("./Cross_logs")
```

```
os.makedirs(base_dir,exist_ok=True)
```

```
now = datetime.datetime.now()
```

```
path=os.path.join(base_dir, str(now.year)+str(now.month)+str(now.day))
```

```
os.makedirs(path,exist_ok=True)
```

از هر چهره ی شناسایی شده که کادر سبز رنگی دور آن کشیده می شود یک نمونه در پوشه ی Cross\_logs قرار داده می شود. در داخل پوشه ی ذکر شده یک پوشه ی دیگر با تاریخ جاری سیستم ساخته می شود تا مشخص شود ترددها مربوط به چه زمانی است. کدهای بالا ساخت پوشه و نامگذاری پوشه ها را انجام می دهند.

```
c=0
```

```
counter=0
```

```
flag=False
```

متغیر c یک شمارنده برای تعداد عکس های ذخیره شده است که در ادامه برنامه مورد استفاده قرار می گیرد. متغیرهای falg و counter هم در هنگام فیلمبرداری از دوربین مورد استفاده قرار می گیرند که در ادامه کاربردشان را شرح می دهیم.

### گام سوم: تعریف توابع مورد نیاز

```
with tf.Graph().as_default():
```

```
sess = tf.Session()
```

```
pnet, rnet, onet = detect_face.create_mtcnn(sess, None)
```

```
minsize = 20
```

```
threshold = [ 0.6, 0.7, 0.7 ]
```

```
factor = 0.709
```

کدهای بالا برای ایجاد گراف ها و session برای عمل جستجوی چهره در تصویر و بارگذاری سه شبکه ی mtcnn است. متغیر minsize کوچک ترین ابعاد چهره ای است که در برنامه برای جستجوی چهره استفاده می شود. مقادیر حد آستانه نیز به تفکیک هر شبکه ی mtcnn، در مقاله ذکر شده است که از همان مقادیر استفاده می کنیم. متغیر factor نیز برای فاکتور pyramid استفاده می شود.

**Image Pyramid**: در فرآیند **mtcnn** شبکه با یک فاکتور ابعادی را برای تصویر چهره در نظر می گیرد سپس طول و عرض تصویر را یک و نیم برابر افزایش می دهد تا اندازه های بزرگ و کوچک تصویر را پوشش دهد. استفاده از این ویژگی سبب می شود شبکه در برابر تغییرات منعطف تر عمل کند.

```
def image_to_encoding(img):
```

```
    img = image.load_img(img, target_size=(160, 160))
```

```
    y = image.img_to_array(img)
```

```
    return v.img_to_encoding(y, image_size)
```

تابع فوق در طول اجرای برنامه در دو جا فراخوانی می شود. یکبار زمانی که بخواهیم embedding تصاویر موجود در پوشه ی **images** که افراد مجاز می باشند را استخراج کنیم و یکبار برای شناسایی چهره ی جدیدی که شناسایی شده است. در این تابع با استفاده از توابع موجود در کراس، پیش پردازش هایی را روی تصاویر ورودی انجام می دهد. این پیش پردازش ها شامل کارهایی مثل تغییر اندازه تصویر به ابعاد ذکر شده در مقاله، تغییر کانال های رنگی به RGB و تبدیل تصویر به آرایه برای استخراج embedding ها. همه ی این توابع در مسیر **keras.preprocessing** و در بخش **image** موجود است.

تابع **img\_to\_encoding** فایل **FaceToolkit** به شرح زیر است:

```
38
39 def img_to_encoding(self, img, image_size):
40     image = face.make_image_tensor(img, image_size)
41     feed_dict = {self.images_placeholder: image, self.phase_train_placeholder: False }
42     emb_array = np.zeros((1, self.embedding_size))
43     emb_array[0, :] = self.session.run(self.embeddings, feed_dict=feed_dict)
44     return np.squeeze(emb_array)
45
46
```

در این تابع تنسورهای موردنیاز برای محاسبات ساخته شده ات. برای ساخت نیز از تابع **make\_image\_tensor** قابل **Face** استفاده شده است. سپس کارهای لازم برای استخراج **encoding** تصویر که یک آرایه چند بعدی می باشد انجام شده است.

گفتیم که برای شناسایی و تشخیص هویت چهره ی افراد لازم است ابتدا یک تصویر از چهره ی هر شخص را به برنامه بدهیم تا برنامه بتواند عمل مقایسه را انجام دهد. کدهای زیر برای خواندن تصاویر موجود در پوشه ی images و ساخت دیتابیس از آن ها نوشته شده است:

```
database={ }
```

```
valid_images = [".jpg",".png"]
```

فرمت های استاندارد خواندن تصویر را مشخص می کند که دو نوع png و jpg را شامل می شود. با وجود توابع پیش پردازش کراس می توانیم تصاویر را در فرمت های مختلف خوانده و عملیات لازم را روی آن ها انجام دهیم.

```
valid_path="./images"
```

```
for f in os.listdir(valid_path):
```

```
    ext = os.path.splitext(f)[1]
```

```
    name = os.path.splitext(f)[0]
```

```
    if ext.lower() not in valid_images:
```

```
        continue
```

کد بالا هر فایل موجود در پوشه ی تصاویر را انتخاب کرده، پسوند را از نام انتسابی جدا می کند، اگر فرمت و پسوند فایل جزو پسوندهای مجاز نباشد از آن صرف نظر می کند در غیر این صورت با کد زیر encoding تصویر را در آرایه ی دو بعدی دیتابیس و با نام مشخصی ذخیره می کند.

```
    database[name]=image_to_encoding(os.path.join(valid_path,f))
```

پس از ساخت encoding ها نوبت به مقایسه ی آنان می رسد در ادامه توابع نوشته شده بدین منظور را شرح می دهیم:

```
def distance(emb1,emb2):
```

```
    diff=np.subtract(emb1,emb2)
```

```
    return np.sum(np.square(diff))
```

تابع فوق معیار شباهت دو چهره را مشخص می کند. برای تنظیم این معیار نیز از مقالات استفاده شده است. فرمول ریاضی تابع فوق به صورت زیر است:

$$(emb1-emb2)^2 + (emb1-emb2)^2$$

محاسبه ی معیار شباهت بسیار مهم است و نشان دهنده ی دقت عملکرد برنامه ی ما در شناسایی و تمیز چهره های مختلف از یکدیگر است.

در تابع `who_is_it` تصاویر موجود در دیتابیس با تصویر ورودی از دوربین مقایسه می شود. اگر فرد از قبل به برنامه معرفی شده باشد و در دیتابیس موجود باشد پیغام مناسبی روی ترمینال چاپ شده سپس یک `log` از شناسایی فرد در ساعت و تاریخ جاری در فایل `log.txt` نوشته خواهد شد. در صورتی که چهره ی یک فرد ناشناس باشد پیغامی مبنی بر موجود نبودن چهره ی فرد در دیتابیس روی ترمینال چاپ می شود.

```
def who_is_it(image_path, database):  
    encoding = image_to_encoding(image_path)  
    min_dist = 2000  
    for (name, db_enc) in database.items():  
        dist = distance(encoding, db_enc)  
        if min_dist > dist:  
            min_dist = dist  
            identity = name  
    if min_dist > verification_threshold:  
        print("Not in the database.")  
    else:  
        print ("it's " + str(identity) + ", the distance is " + str(min_dist))  
        now = datetime.datetime.now()  
        file_object.write("[log] "+str(identity) + " " + str(now) + "\r\n")  
    return min_dist, identity
```

توابع تعریف شده در فایل `FaceRecognition` تا زمانی که در روند اصلی برنامه فراخوانی نشوند استفاده نخواهند شد. در ادامه تصویر را از دوربین وب کم گرفته و کادر مستطیل شکل را رسم نموده و هر ۵۰ فریم یکبار عمل تشخیص هویت چهره های موجود در تصویر را انجام می دهیم.



## گام آخر: گرفتن تصویر از وب کم و بازشناسی چهره

متغیر `cap` با استفاده از تابع `videocapture` کتابخانه `opencv` تصویر دوربین را به صورت لحظه به لحظه می خواند. با استفاده از تابع `videocapture` علاوه بر تصویر دوربین می توانیم یک فیلم را هم بخوانیم. برای این کار کافی است آدرس فیلم را در داخل پرانتز قرار دهیم.

```
cap=cv2.VideoCapture(0)
```

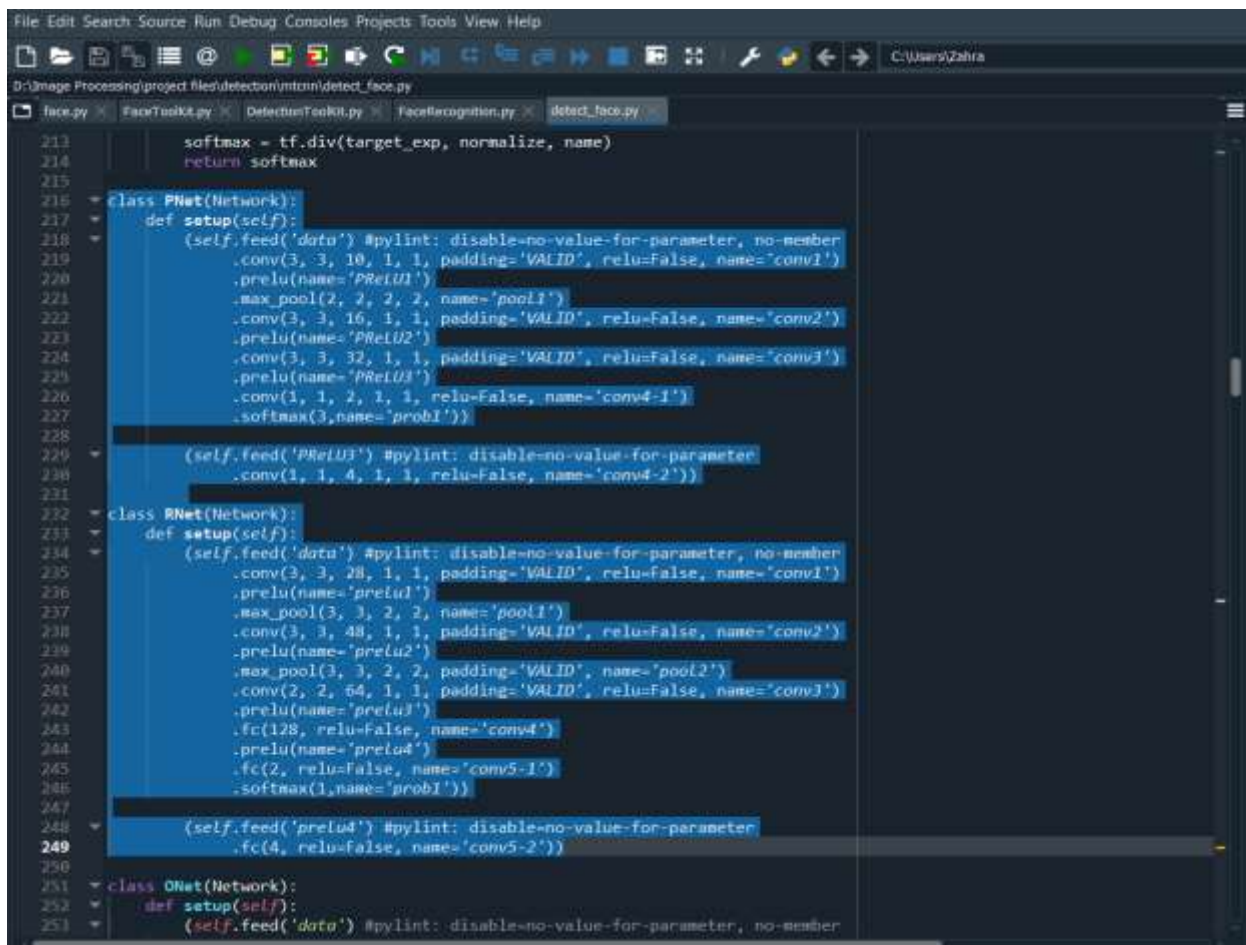
```
while True:
```

```
    __, frame = cap.read()
```

```
    bounding_boxes, points = detect_face.detect_face(frame, minsize, pnet, rnet, onet, threshold, factor)
```

`bounding_boxes` مختصات نقاط چهره های شناسایی شده توسط شبکه ی `mtcnn` می باشد. برای استخراج این مختصات تمامی فاکتورهای تنظیم شده و مورد نیاز را به تابع `detect_face` دیوید سندبرگ میفرستیم تا جستجوی تصویر شروع شود.

از جمله کارهایی که در فایل `detect_face.py` انجام می شود (به جز تشخیص و جستجوی چهره) تعریف معماری شبکه عصبی می باشد که کدهای آن به شرح زیر است:



```
213         softmax = tf.div(target_exp, normalize, name)
214         return softmax
215
216     class PNet(Network):
217     def setup(self):
218         (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
219          .conv(3, 3, 10, 1, 1, padding='VALID', relu=False, name='conv1')
220          .prelu(name='prelu1')
221          .max_pool(2, 2, 2, 2, name='pool1')
222          .conv(3, 3, 16, 1, 1, padding='VALID', relu=False, name='conv2')
223          .prelu(name='prelu2')
224          .conv(3, 3, 32, 1, 1, padding='VALID', relu=False, name='conv3')
225          .prelu(name='prelu3')
226          .conv(1, 1, 2, 1, 1, relu=False, name='conv4-1')
227          .softmax(3, name='prob1'))
228
229         (self.feed('prelu3') #pylint: disable=no-value-for-parameter
230          .conv(1, 1, 4, 1, 1, relu=False, name='conv4-2'))
231
232     class RNet(Network):
233     def setup(self):
234         (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
235          .conv(3, 3, 28, 1, 1, padding='VALID', relu=False, name='conv1')
236          .prelu(name='prelu1')
237          .max_pool(3, 3, 2, 2, name='pool1')
238          .conv(3, 3, 48, 1, 1, padding='VALID', relu=False, name='conv2')
239          .prelu(name='prelu2')
240          .max_pool(3, 3, 2, 2, padding='VALID', name='pool2')
241          .conv(2, 2, 64, 1, 1, padding='VALID', relu=False, name='conv3')
242          .prelu(name='prelu3')
243          .fc(128, relu=False, name='conv4')
244          .prelu(name='prelu4')
245          .fc(2, relu=False, name='conv5-1')
246          .softmax(1, name='prob1'))
247
248         (self.feed('prelu4') #pylint: disable=no-value-for-parameter
249          .fc(4, relu=False, name='conv5-2'))
250
251     class ONet(Network):
252     def setup(self):
253         (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
```

```

245         .fc(2, relu=False, name='conv5-1')
246         .softmax(1, name='prob1'))
247
248     (self.feed('prelu4') #pylint: disable=no-value-for-parameter
249     .fc(4, relu=False, name='conv5-2'))
250
251 class ONet(Network):
252     def setup(self):
253         (self.feed('data') #pylint: disable=no-value-for-parameter, no-member
254         .conv(3, 3, 32, 1, 1, padding='VALID', relu=False, name='conv1')
255         .prelu(name='prelu1')
256         .max_pool(3, 3, 2, 2, name='pool1')
257         .conv(3, 3, 64, 1, 1, padding='VALID', relu=False, name='conv2')
258         .prelu(name='prelu2')
259         .max_pool(3, 3, 2, 2, padding='VALID', name='pool2')
260         .conv(3, 3, 64, 1, 1, padding='VALID', relu=False, name='conv3')
261         .prelu(name='prelu3')
262         .max_pool(2, 2, 2, 2, name='pool3')
263         .conv(2, 2, 128, 1, 1, padding='VALID', relu=False, name='conv4')
264         .prelu(name='prelu4')
265         .fc(256, relu=False, name='conv5')
266         .prelu(name='prelu5')
267         .fc(2, relu=False, name='conv6-1')
268         .softmax(1, name='prob1'))
269
270     (self.feed('prelu5') #pylint: disable=no-value-for-parameter
271     .fc(4, relu=False, name='conv6-2'))
272
273     (self.feed('prelu5') #pylint: disable=no-value-for-parameter
274     .fc(10, relu=False, name='conv6-3'))
275
276 def create_mtcnn(sess, model_path):
277     if not model_path:
278         model_path, _ = os.path.split(os.path.realpath(__file__))
279
280     with tf.variable_scope('pnet'):
281         data = tf.placeholder(tf.float32, (None, None, None, 3), 'input')
282         pnet = ONet({'data': data})
283         pnet.load(os.path.join(model_path, 'det1.npy'), sess)
284     with tf.variable_scope('rnet'):
285         data = tf.placeholder(tf.float32, (None, 24, 24, 3), 'input')

```

counter+=1

بعد از مشخص شدن مختصات نوبت به رسم نقاط روی فریم ورودی از دوربین است. حلقه ی زیر این عمل را انجام می دهد.

for bounding\_box in bounding\_boxes:

pts = bounding\_box[:4].astype(np.int32)

pt1 = (pts[0], pts[1])

pt2 = (pts[2], pts[3])

```
cv2.rectangle(frame,pt1,pt2,color=default_color, thickness=default_thickness)
```

مدل شبکه ی mtcnn به صورت real-time عمل جستجو و رسم مختصات را انجام می دهد اما به دلیل فرآیند زمانبر encoding و مقایسه ی encoding عمل بازشناسی چهره را محدود به هر ۵۰ فریم یکبار کرده ایم تا برنامه فرصت مناسب را داشته باشد. البته این بدین معناست که عمل بازشناسی چهره حدودا هرثانیه یکبار انجام میشود. در ادامه ی برنامه در شمارش فریم ها، فریم ۵۰ ام انتخاب شده و در صورت وجود چهره در فریم تصویر چهره از فریم برش داده می شود سپس به تاریخ و ساعت سیستم و شماره تصویر یعنی متغیر C در مسیر پوشه ی Cross\_Log و در پوشه ای با نام تاریخ سیستم ذخیره می شود.

سپس تابع who\_is\_it فراخوانی شده تا encoding چهره ی برش داده شده استخراج شده و با دیتابیس مقایسه شود. در صورت شناسایی و مطابقت چهره با دیتابیس نام فرد با رنگ قرمز برای چاپ روی تصویر وب کم فرستاده می شود. در ادامه تغییرات انجام شده که شامل رسم مستطیل سبز رنگ و هویت فرد است برای چاپ روی تصویر نمایش داده شده از وب کم ارسال می شود.

```
if counter>=50:
```

```
    cropped = frame[pts[1]:pts[3], pts[0]:pts[2], :]
```

```
    scaled = misc.imresize(cropped, (image_size, image_size), interp='bilinear')
```

```
    now = datetime.datetime.now()
```

```
    image_path=os.path.join(path,str(c) + "-" + str( now.strftime("%Y-%m-%d-%H-%M"))+".png")
```

```
    cv2.imwrite(image_path,scaled)
```

```
    r=who_is_it(image_path,database)
```

```
    flag=True
```

```
    c+=1
```

```
    cv2.putText(frame,str(r[1]), (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, cv2.LINE_4)
```

```
    if flag==True:
```

```
        counter=0
```

```
        flag=False
```

```
    cv2.imshow('FaceRecognition',frame)
```

```
key=cv2.waitKey(1)
```

این عملیات تا زمانی که کلید enter از صفحه کلید وارد شود ادامه پیدا می کند و پس از آن برنامه متوقف شده و منبع گرفته شده از دوربین رها می شود. فایل log.txt نیز ذخیره و بسته می شود.

```
if key==13 :
```

```
    break
```

```
file_object.close()
```

```
cap.release()
```

```
cv2.destroyAllWindows()
```

## **فصل پنجم**

### **نتیجه گیری**

## ۱-۵- نتیجه گیری

با پیاده سازی این برنامه توانستیم یک برنامه تشخیص هویت بر اساس چهره ی افراد را انجام دهیم. این برنامه می تواند با دقت و سرعت خوبی عمل کند گرچه شبکه ای سنگین بوده و سرعت بارگزاری اولیه ی شبکه به سیستم کند است اما دقت خوب این شبکه باعث می شود استفاده از آن در خیلی از پروژه ها مرسوم باشد. در گام های بعدی می توان با بهبود عملکرد آن بر اساس پیشنهادات مقاله های جدید، مشکل سنگین بودن آن را نیز حل کرده و با نوشتن یک رابط کاربری گرافیکی ساده، به توسعه و ایجاد یک محصول نرم افزاری رسید.

## ۲-۵- پیشنهادات برای کارهای آتی

یکی از چالش های این برنامه این است که در صورت استفاده به عنوان گیت ورودی یک مکان، هر شخصی می تواند با داشتن تصویر افراد مجاز، و نشان دادن آن به دوربین از مقابل دوربین عبور کرده و وارد محیط شود و تدابیر امنیتی را نقض کند. در سال های اخیر شبکه هایی برای حل این شکاف امنیتی به وجود آمده اند که هدف آنان تمیز فرد واقعی از تصاویر چهره های جعلی می باشد. به این شبکه ها Liveness Detection می گویند. ادغام و استفاده از این شبکه های عصبی در کنار mtcnn می تواند یک برنامه قوی شناسایی چهره را ایجاد کند. به صورت خلاصه پیشنهادات اینجانب برای کارهای آتی به شرح زیر است:

- افزودن قابلیت Liveness Detection به مدل mtcnn
- یافتن راه حلی برای کاهش وزن های شبکه و مشکل کندی سرعت بارگزاری شبکه به حافظه- مانند برخی شبکه ها که نسخه ی سبک خود را با نام lite یا mobile-net ارائه داده اند
- نوشتن و توسعه ی یک رابط کاربری گرافیکی به منظور پنل مدیریت نرم افزار برای تولید یک محصول نرم افزاری
- تلاش برای جمع آوری مجموعه داده چهره ی ایرانیان و آموزش مدل جدید با استناد به آخرین مقالات موجود

## مراجع

- [1] [Zhang et al.,2016, Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks]
- [2] <https://github.com/davidsandberg/facenet/tree/master/src/align>
- [3] <http://blog.class.vision/winter-96-97-syllabus/>
- [4] <http://wikipedia.org>

## پیوست ها



پیوست الف

مقاله ی MTCNN

# Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks

Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, *Senior Member, IEEE*, and Yu Qiao, *Senior Member, IEEE*

**Abstract**—Face detection and alignment in unconstrained environment are challenging due to various poses, illuminations and occlusions. Recent studies show that deep learning approaches can achieve impressive performance on these two tasks. In this paper, we propose a deep cascaded multi-task framework which exploits the inherent correlation between them to boost up their performance. In particular, our framework adopts a cascaded structure with three stages of carefully designed deep convolutional networks that predict face and landmark location in a coarse-to-fine manner. In addition, in the learning process, we propose a new online hard sample mining strategy that can improve the performance automatically without manual sample selection. Our method achieves superior accuracy over the state-of-the-art techniques on the challenging FDDB and WIDER FACE benchmark for face detection, and AFLW benchmark for face alignment, while keeps real time performance.

**Index Terms**—Face detection, face alignment, cascaded convolutional neural network

## I. INTRODUCTION

FACE detection and alignment are essential to many face applications, such as face recognition and facial expression analysis. However, the large visual variations of faces, such as occlusions, large pose variations and extreme lightings, impose great challenges for these tasks in real world applications.

The cascade face detector proposed by Viola and Jones [2] utilizes Haar-Like features and AdaBoost to train cascaded classifiers, which achieve good performance with real-time efficiency. However, quite a few works [1, 3, 4] indicate that this detector may degrade significantly in real-world applications with larger visual variations of human faces even with more advanced features and classifiers. Besides the cascade structure, [5, 6, 7] introduce deformable part models (DPM) for face detection and achieve remarkable performance. However, they need high computational expense and may usually require expensive annotation in the training stage. Recently, convolutional neural networks (CNNs) achieve remarkable progresses in a variety of computer vision tasks, such as image classification [9] and face recognition [10]. Inspired by the good per-

formance of CNNs in computer vision tasks, some of the CNNs based face detection approaches have been proposed in recent years. Yang *et al.* [11] train deep convolution neural networks for facial attribute recognition to obtain high response in face regions which further yield candidate windows of faces. However, due to its complex CNN structure, this approach is time costly in practice. Li *et al.* [19] use cascaded CNNs for face detection, but it requires bounding box calibration from face detection with extra computational expense and ignores the inherent correlation between facial landmarks localization and bounding box regression.

Face alignment also attracts extensive interests. Regression-based methods [12, 13, 16] and template fitting approaches [14, 15, 7] are two popular categories. Recently, Zhang *et al.* [22] proposed to use facial attribute recognition as an auxiliary task to enhance face alignment performance using deep convolutional neural network.

However, most of the available face detection and face alignment methods ignore the inherent correlation between these two tasks. Though there exist several works attempt to jointly solve them, there are still limitations in these works. For example, Chen *et al.* [18] jointly conduct alignment and detection with random forest using features of pixel value difference. But, the handcraft features used limits its performance. Zhang *et al.* [20] use multi-task CNN to improve the accuracy of multi-view face detection, but the detection accuracy is limited by the initial detection windows produced by a weak face detector.

On the other hand, in the training process, mining hard samples in training is critical to strengthen the power of detector. However, traditional hard sample mining usually performs an offline manner, which significantly increases the manual operations. It is desirable to design an online hard sample mining method for face detection and alignment, which is adaptive to the current training process automatically.

In this paper, we propose a new framework to integrate these two tasks using unified cascaded CNNs by multi-task learning. The proposed CNNs consist of three stages. In the first stage, it produces candidate windows quickly through a shallow CNN. Then, it refines the windows to reject a large number of non-faces windows through a more complex CNN. Finally, it uses a more powerful CNN to refine the result and output facial landmarks positions. Thanks to this multi-task learning framework, the performance of the algorithm can be notably improved. The major contributions of this paper are summarized as follows: (1) We propose a new cascaded CNNs based framework for joint face detection and alignment, and carefully

K.-P. Zhang, Z.-F. Li and Y. Q. are with the Multimedia Laboratory, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China. E-mail: kp.zhang@siat.ac.cn; zhifeng.li@siat.ac.cn; yu.qiao@siat.ac.cn

Z.-P. Zhang is with the Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong. E-mail: zz013@ie.cuhk.edu.hk

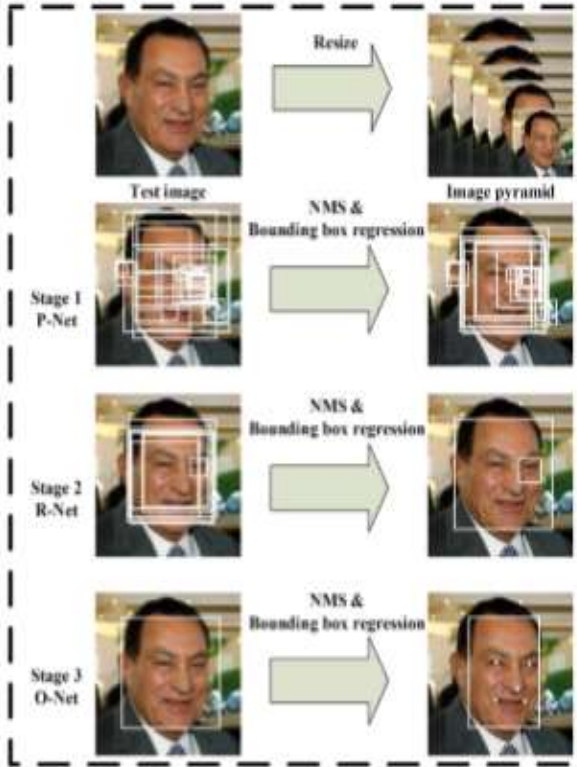


Fig. 1. Pipeline of our cascaded framework that includes three-stage multi-task deep convolutional networks. Firstly, candidate windows are produced through a fast Proposal Network (P-Net). After that, we refine these candidates in the next stage through a Refinement Network (R-Net). In the third stage, The Output Network (O-Net) produces final bounding box and facial landmarks position.

design lightweight CNN architecture for real time performance. (2) We propose an effective method to conduct online hard sample mining to improve the performance. (3) Extensive experiments are conducted on challenging benchmarks, to show the significant performance improvement of the proposed approach compared to the state-of-the-art techniques in both face detection and face alignment tasks.

## II. APPROACH

In this section, we will describe our approach towards joint face detection and alignment.

### A. Overall Framework

The overall pipeline of our approach is shown in Fig. 1. Given an image, we initially resize it to different scales to build an image pyramid, which is the input of the following three-stage cascaded framework:

**Stage 1:** We exploit a fully convolutional network[?], called Proposal Network (P-Net), to obtain the candidate windows and their bounding box regression vectors in a similar manner as [29]. Then we use the estimated bounding box regression vectors to calibrate the candidates. After that, we employ non-maximum suppression (NMS) to merge highly overlapped candidates.

**Stage 2:** all candidates are fed to another CNN, called Refine Network (R-Net), which further rejects a large number of false candidates, performs calibration with bounding box regression, and NMS candidate merge.

TABLE I  
COMPARISON OF SPEED AND VALIDATION ACCURACY OF OUR CNNs AND PREVIOUS CNNs [19]

Group	CNN	300 Times Forward	Accuracy
Group1	12-Net [19]	0.038s	94.4%
Group1	P-Net	0.031s	94.6%
Group2	24-Net [19]	0.738s	95.1%
Group2	R-Net	0.458s	95.4%
Group3	48-Net [19]	3.577s	93.2%
Group3	O-Net	1.347s	95.4%

**Stage 3:** This stage is similar to the second stage, but in this stage we aim to describe the face in more details. In particular, the network will output five facial landmarks' positions.

### B. CNN Architectures

In [19], multiple CNNs have been designed for face detection. However, we noticed its performance might be limited by the following facts: (1) Some filters lack diversity of weights that may limit them to produce discriminative description. (2) Compared to other multi-class objection detection and classification tasks, face detection is a challenge binary classification task, so it may need less numbers of filters but more discrimination of them. To this end, we reduce the number of filters and change the  $5 \times 5$  filter to a  $3 \times 3$  filter to reduce the computing while increase the depth to get better performance. With these improvements, compared to the previous architecture in [19], we can get better performance with less runtime (the result is shown in Table 1. For fair comparison, we use the same data for both methods). Our CNN architectures are showed in Fig. 2.

### C. Training

We leverage three tasks to train our CNN detectors: face/non-face classification, bounding box regression, and facial landmark localization.

1) *Face classification:* The learning objective is formulated as a two-class classification problem. For each sample  $x_i$ , we use the cross-entropy loss:

$$L_i^{det} = -(y_i^{det} \log(p_i) + (1 - y_i^{det})(1 - \log(p_i))) \quad (1)$$

where  $p_i$  is the probability produced by the network that indicates a sample being a face. The notation  $y_i^{det} \in \{0,1\}$  denotes the ground-truth label.

2) *Bounding box regression:* For each candidate window, we predict the offset between it and the nearest ground truth (i.e., the bounding boxes' left top, height, and width). The learning objective is formulated as a regression problem, and we employ the Euclidean loss for each sample  $x_i$ :

$$L_i^{box} = \|\hat{y}_i^{box} - y_i^{box}\|_2^2 \quad (2)$$

where  $\hat{y}_i^{box}$  regression target obtained from the network and  $y_i^{box}$  is the ground-truth coordinate. There are four coordinates, including left top, height and width, and thus  $y_i^{box} \in \mathbb{R}^4$ .

3) *Facial landmark localization:* Similar to the bounding box regression task, facial landmark detection is formulated as a



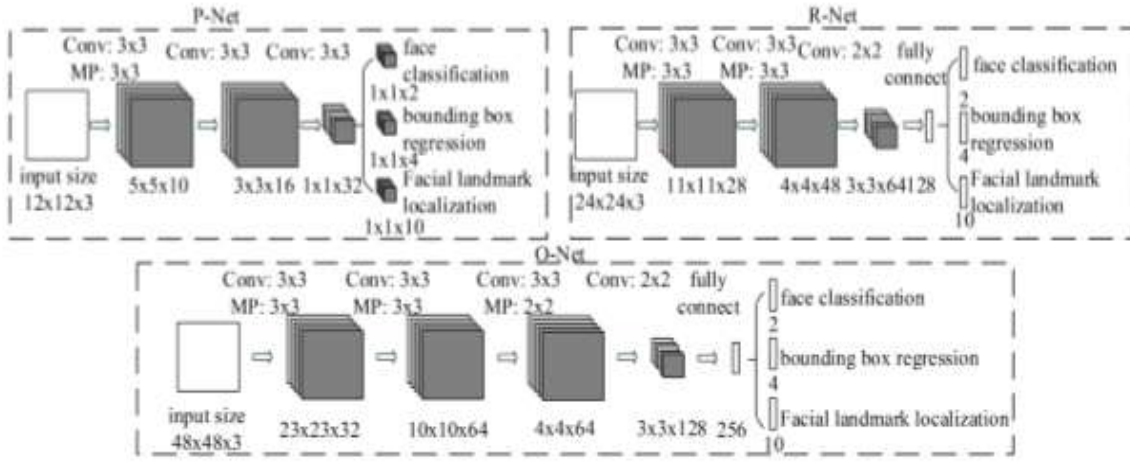


Fig. 2. The architectures of P-Net, R-Net, and O-Net, where “MP” means max pooling and “Conv” means convolution. The step size in convolution and pooling is 1 and 2, respectively.

regression problem and we minimize the Euclidean loss:

$$L_i^{\text{landmark}} = \|\hat{y}_i^{\text{landmark}} - y_i^{\text{landmark}}\|_2^2 \quad (3)$$

where  $\hat{y}_i^{\text{landmark}}$  is the facial landmark’s coordinate obtained from the network and  $y_i^{\text{landmark}}$  is the ground-truth coordinate. There are five facial landmarks, including left eye, right eye, nose, left mouth corner, and right mouth corner, and thus  $y_i^{\text{landmark}} \in \mathbb{R}^{10}$ .

4) *Multi-source training*: Since we employ different tasks in each CNNs, there are different types of training images in the learning process, such as face, non-face and partially aligned face. In this case, some of the loss functions (i.e., Eq. (1)-(3)) are not used. For example, for the sample of background region, we only compute  $L_i^{\text{det}}$ , and the other two losses are set as 0. This can be implemented directly with a sample type indicator. Then the overall learning target can be formulated as:

$$\min \sum_{i=1}^N \sum_{j \in \{\text{det}, \text{box}, \text{landmark}\}} \alpha_j \beta_i^j L_i^j \quad (4)$$

where  $N$  is the number of training samples.  $\alpha_j$  denotes on the task importance. We use  $(\alpha_{\text{det}} = 1, \alpha_{\text{box}} = 0.5, \alpha_{\text{landmark}} = 0.5)$  in P-Net and R-Net, while  $(\alpha_{\text{det}} = 1, \alpha_{\text{box}} = 0.5, \alpha_{\text{landmark}} = 1)$  in O-Net for more accurate facial landmarks localization.  $\beta_i^j \in \{0, 1\}$  is the sample type indicator. In this case, it is natural to employ stochastic gradient descent to train the CNNs.

5) *Online Hard sample mining*: Different from conducting traditional hard sample mining after original classifier had been trained, we do online hard sample mining in face classification task to be adaptive to the training process.

In particular, in each mini-batch, we sort the loss computed in the forward propagation phase from all samples and select the top 70% of them as hard samples. Then we only compute the gradient from the hard samples in the backward propagation phase. That means we ignore the easy samples that are less

helpful to strengthen the detector while training. Experiments show that this strategy yields better performance without manual sample selection. Its effectiveness is demonstrated in the Section III.

### III. EXPERIMENTS

In this section, we first evaluate the effectiveness of the proposed hard sample mining strategy. Then we compare our face detector and alignment against the state-of-the-art methods in Face Detection Data Set and Benchmark (FDDB) [25], WIDER FACE [24], and Annotated Facial Landmarks in the Wild (AFLW) benchmark [8]. FDDB dataset contains the annotations for 5,171 faces in a set of 2,845 images. WIDER FACE dataset consists of 393,703 labeled face bounding boxes in 32,203 images where 50% of them for testing into three subsets according to the difficulty of images, 40% for training and the remaining for validation. AFLW contains the facial landmarks annotations for 24,386 faces and we use the same test subset as [22]. Finally, we evaluate the computational efficiency of our face detector.

#### A. Training Data

Since we jointly perform face detection and alignment, here we use four different kinds of data annotation in our training process: (i) Negatives: Regions that the Intersection-over-Union (IoU) ratio less than 0.3 to any ground-truth faces; (ii) Positives: IoU above 0.65 to a ground truth face; (iii) Part faces: IoU between 0.4 and 0.65 to a ground truth face; and (iv) Landmark faces: faces labeled 5 landmarks’ positions. Negatives and positives are used for face classification tasks, positives and part faces are used for bounding box regression, and landmark faces are used for facial landmark localization. The training data for each network is described as follows:

- 1) *P-Net*: We randomly crop several patches from WIDER FACE [24] to collect positives, negatives and part face. Then, we crop faces from CelebA [23] as landmark faces
- 2) *R-Net*: We use first stage of our framework to detect faces



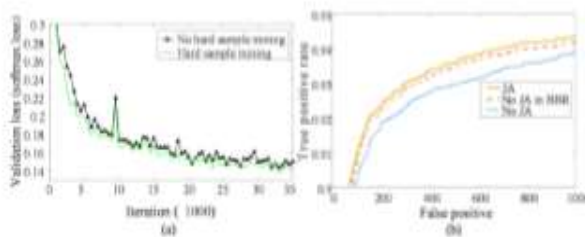


Fig. 3. (a) Validation loss of O-Net with and without hard sample mining. (b) “JA” denotes joint face alignment learning while “No JA” denotes do not joint it. “No JA in BBR” denotes do not joint it while training the CNN for bounding box regression.

from WIDER FACE [24] to collect positives, negatives and part face while landmark faces are detected from CelebA [23].

3) *O-Net*: Similar to R-Net to collect data but we use first two stages of our framework to detect faces.

#### B. The effectiveness of online hard sample mining

To evaluate the contribution of the proposed online hard sample mining strategy, we train two O-Nets (with and without online hard sample mining) and compare their loss curves. To make the comparison more directly, we only train the O-Nets for the face classification task. All training parameters including the network initialization are the same in these two O-Nets. To compare them easier, we use fix learning rate. Fig. 3 (a) shows the loss curves from two different training ways. It is very clear that the hard sample mining is beneficial to performance improvement.

#### C. The effectiveness of joint detection and alignment

To evaluate the contribution of joint detection and alignment, we evaluate the performances of two different O-Nets (joint facial landmarks regression task and do not joint it) on FDDB (with the same P-Net and R-Net for fair comparison). We also compare the performance of bounding box regression in these two O-Nets. Fig. 3 (b) suggests that joint landmarks localization task learning is beneficial for both face classification and bounding box regression tasks.

#### D. Evaluation on face detection

To evaluate the performance of our face detection method, we compare our method against the state-of-the-art methods [1, 5, 6, 11, 18, 19, 26, 27, 28, 29] in FDDB, and the state-of-the-art methods [1, 24, 11] in WIDER FACE. Fig. 4 (a)-(d) shows that our method consistently outperforms all the previous approaches by a large margin in both the benchmarks. We also evaluate our approach on some challenge photos<sup>1</sup>.

#### E. Evaluation on face alignment

In this part, we compare the face alignment performance of our method against the following methods: RCPR [12], TSPM [7], Luxand face SDK [17], ESR [13], CDM [15], SDM [21], and TCDCN [22]. In the testing phase, there are 13 images that our method fails to detect face. So we crop the central region of these 13 images and treat them as the input for O-Net. The mean error is measured by the distances between the estimated

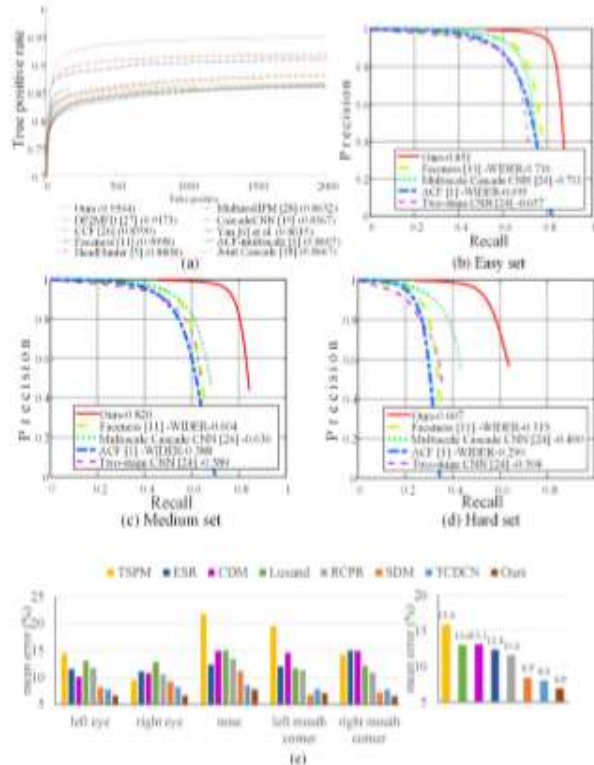


Fig. 4. (a) Evaluation on FDDB. (b-d) Evaluation on three subsets of WIDER FACE. The number following the method indicates the average accuracy. (e) Evaluation on AFLW for face alignment

landmarks and the ground truths, and normalized with respect to the inter-ocular distance. Fig. 4 (e) shows that our method outperforms all the state-of-the-art methods with a margin.

#### F. Runtime efficiency

Given the cascade structure, our method can achieve very fast speed in joint face detection and alignment. It takes 16fps on a 2.60GHz CPU and 99fps on GPU (Nvidia Titan Black). Our implementation is currently based on un-optimized MATLAB code.

### IV. CONCLUSION

In this paper, we have proposed a multi-task cascaded CNNs based framework for joint face detection and alignment. Experimental results demonstrate that our methods consistently outperform the state-of-the-art methods across several challenging benchmarks (including FDDB and WIDER FACE benchmarks for face detection, and AFLW benchmark for face alignment) while keeping real time performance. In the future, we will exploit the inherent correlation between face detection and other face analysis tasks, to further improve the performance.

### REFERENCES

- [1] B. Yang, J. Yan, Z. Lei, and S. Z. Li, “Aggregate channel features for multi-view face detection,” in *IEEE International Joint Conference on Biometrics*, 2014, pp. 1-8.

<sup>1</sup> Examples are showed in <http://kpxzhang93.github.io/SPL/index.html>