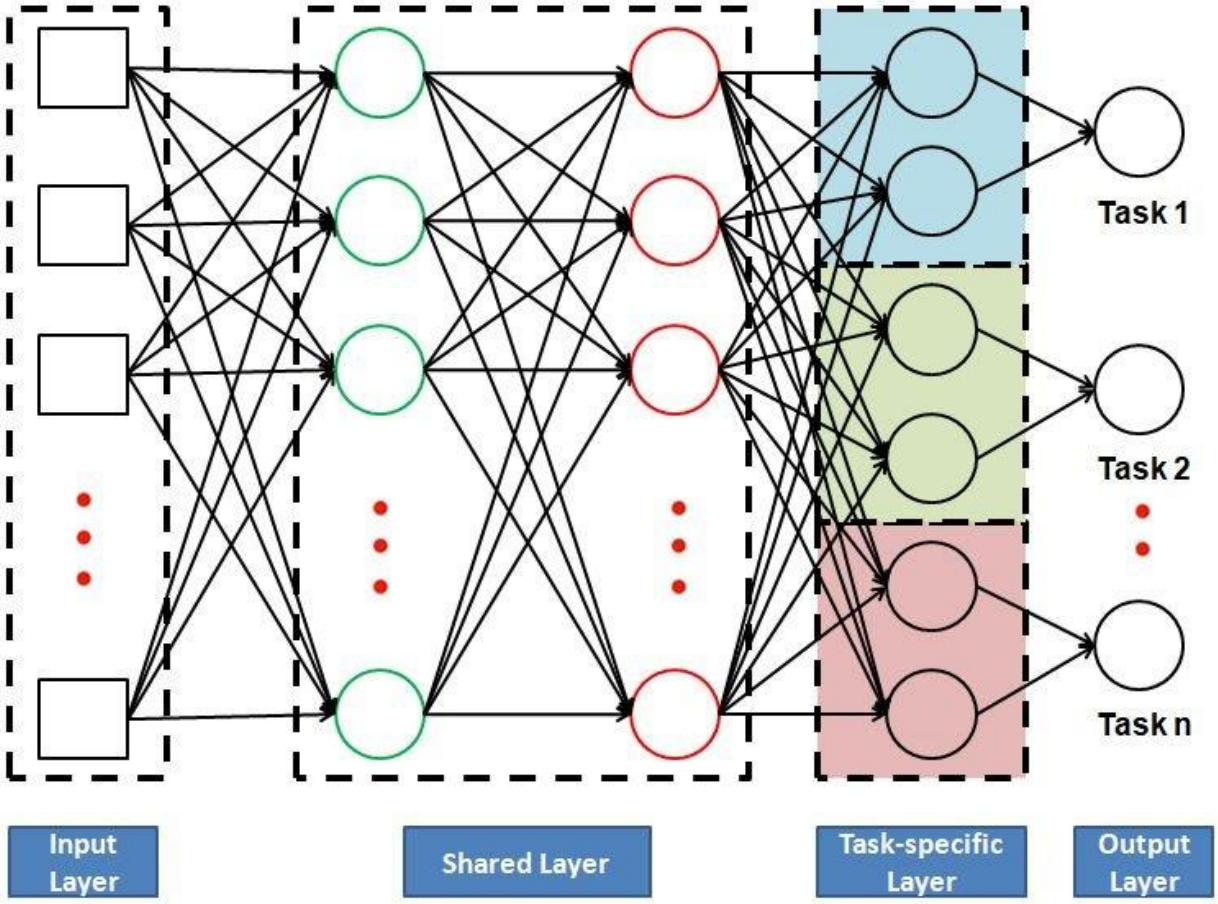
گزارش چالش ها – پروژه شبکه عصبی – هوش مصنوعی – یزدان ماستری فراهانی

مقدمه:

ابتدا به یوتیوب رفتم و تعدادی ویدیو مربوط به شبکه های عصبی رو مشاهده کردم تا کلا بفهمم داستان از چه قراره. داخل ویدیو از یک شبکه عصبی استفاده شده بود تا با توجه به عکس تقریبا ناواضح از یک عدد، شبکه عصبی رقم آن را تشخیص دهد. کارکرد آن به این صورت بود که در چهار لایه که لایه اول شامل تمامی پیکسل های عکس که یک عدد بین 0 و 1 بود را شامل میشد و در لایه اخر خروجی که یکی از ده رقم بود را نشان میداد. در دو لایه پنهانی میانی هم متوجه تیکه تیکه ساختن حروف از پیکسل های عکس و مفاهیم bias و backpropagation و نورون ها و وزن هر متغیر و در آخر استفاده از یک تابع برای محدود کردن آن میان بازه ای که مد نظر داریم و نحوه نمایش این ها آشنا شدم. که بعدا متوجه شدم سوال دو و سه مشابه همین مسئله هستند.



سوال یک – بخش اول:

ابتدا از یک تابع ساده برای امتحان کد سوال استفاده شد. سپس از بازه -50 تا 50 با 10000 نقطه به تمرین مدل پرداخته شد. اما از آنجایی که تعداد نقاط تاثیر بسیاری در نتیجه سوال داشت و به همین میزان بالا بودن بازه نیاز به نقاط بیشتری برای آموزش مدل داشت و این امر باعث هزینه زمانی زیادی میشد، در انتها از عدد 15 به جای 50 برای دامنه هر تابع در نظر گرفته شد.

سپس داده ها 80 به 20 میان داده ترین و تست تقسیم شدند.

```
X_train, X_test, y_train1, y_test1 = train_test_split(X, y1, test_size=0.2, random_state=42)
```

پس از آن از MLP Regressor برای آموزش مدل استفاده شده است که شامل پارامتر های متفاوتی ست که هر کدام بر روی نتیجه کار تاثیر بسیاری دارند.

پارامتر های MLP Regressor:

تعداد لایه ها و تعداد نورون های هر لایه: این پارامتر در بخش اول شامل 4 لایه که در دو لایه اول از 16 نورون و در دو لایه دوم از 8 نورون استفاده شده است که کمترین مقدار ممکن برای گرفتن بیشتر دقت لازم بودند. نکته قابل توجه در این مرحله این بود که افزایش لایه ها و یا افزایش نورون ها الزاما باعث افزایش دقت مدل نمیشد.

Activation: در این مرحله با بررسی تمامی حالت ها، بهترین حالت 'relu' بود که معمولا انتخاب میشود چرا که به خوبی به اصلاح رابطه غیر خطی میان ورودی خروجی در شبکه عصبی در خروجی هر نورون میپردازد.

Random_state: این پارامتر به کنترل میان داده های اتفاقی درون مدل میپردازد که البته با توجه به تاثیری که بر weight , bias ها دارد، سعی بر آن بود که بهترین آن انتخاب شود.

Iteration: این پارامتر تاثیر بسیاری در زمان آموزش مدل داشت و در نتیجه انتخاب عدد درست برای آن بسیار اهمیت داشت. از یک حدی به بعد افزایش iteration تاثیری در افزایش دقت مدل نداشته و صرفا باعث هزینه زمانی بیشتر میشود.

Solver: این تابع برای optimization درون MLP است که سه ورودی 'adam', 'sgd', 'lbfgs' را می پذیرد. انتخاب بهترین گزینه بستگی به اندازه و پیچیدگی دیتاست دارد که در اینجا adam خیلی بهتر از بقیه گزینه ها بود.

پس ازین موارد دیتا را درون MLP، fit کردیم و با نمایش score، درصد دقت تابع ها سنجیده شده است.

```
mlp1 = MLPRegressor(hidden_layer_sizes=(16,16,8,8), activation='relu', solver='adam', random_state=2, max_iter=504)
mlp1.fit(X_train, y_train1)
```

انتخاب درست پارامتر ها باعث شد که در کمترین زمان به بهترین دقت رسیده شود که در همگی موارد بالای 99% برای تابع هایی با پیچیدگی بسیار بودند. در صورت استفاده از تابع هایی با پیچیدگی کمتر تقریبا خطوط مربوط به تابع اصلی در پلات آن غیر قابل مشاهده بود و دقت به 100% میل میکرد.

پس از اطمینان حاصل نمودن از صحت کد 4 تابع بسیار پیچیده که شامل نقاط بحرانی بسیاری نیز بودند برای بررسی مدل استفاده شده است که شامل:

$Function1: |Sin(x) + x/2|$

$Function2: Exp(Sin(x)/2 + Cos(\sqrt{x+20})/2)$

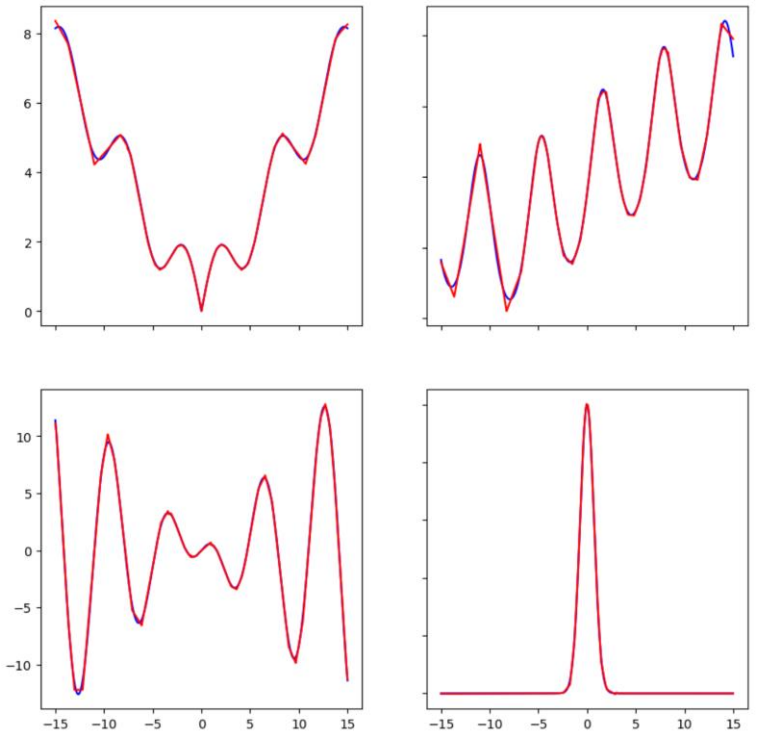
$Function3: x * Cos(x)$

$Function4: Exp(-x^2)$

```
def true_function1(x):
    return (abs(np.sin(x) + x/2))

def true_function2(x):
    return np.exp(0.5* np.sin(x)) + 0.5* np.cos(np.sqrt(x+20))

def true_function3(x):
    return (np.cos(x) * x)

def true_function4(x):
    return (np.exp(-1 * x**2))
```

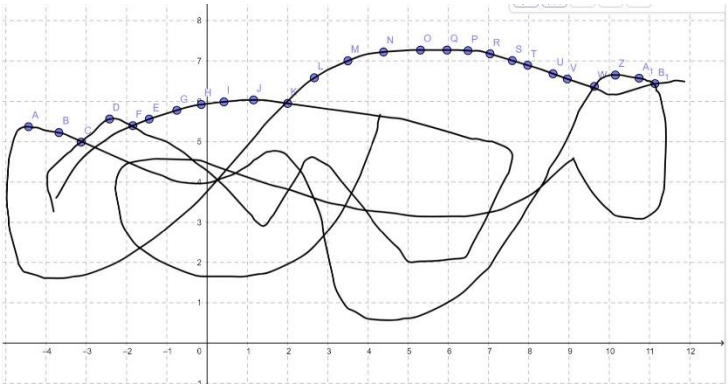
در آخر نیز با تعیین یک بازه برای دامنه و همچنین گرفتن predict از MLP، از این دو مورد برای نمایش تابع ها استفاده شده است.

خط آبی شامل خروجی اصلی تابع و خط قرمز نمایش حدس مدل است.



سوال یک – بخش دوم:

در این بخش از یک سایت آنلاین برای کشیدن یک تابع رندم استفاده شده است و سپس 26 نقطه از قسمت بالایی آن را انتخاب کرده و مختصات آنها را درون دیتاست در نظر گرفتم.

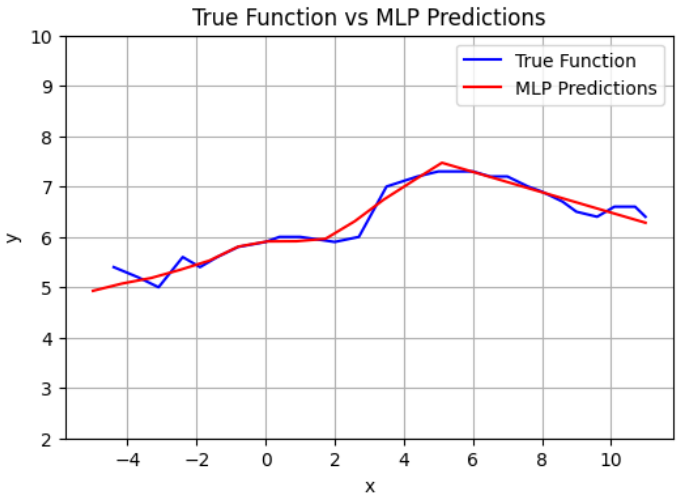


```
x = np.array([-4.4, -3.7, -3.1, -2.4, -1.9, -1.4, -0.8, 0, 0.4, 1, 2, 2.7, 3.5, 4.4, 5, 6, 6.5, 7, 7.6, 8, 8.6, 9, 9.6, 10.1, 10.7, 11 ])
y = np.array([5.4, 5.2, 5, 5.6, 5.4, 5.6, 5.8, 5.9, 6, 6, 5.9, 6, 7, 7.2, 7.3, 7.3, 7.2, 7.2, 7, 6.9, 6.7, 6.5, 6.4, 6.6, 6.6, 6.4])
```

سپس مشابه بخش اول عمل کرده با این تفاوت که از آنجایی که دیتاست در اینجا بسیار کوچکتر است نیاز به تغییر برخی از پارامتر های MLP Regressor وجود دارد. همانطور که بالاتر گفته شد، adam برای optimize کردن تابع های بزرگ و پیچیده است و در اینجا به خوبی lbfgs عمل نخواهد کرد. همچنین به تعداد کمتری از iteration نیاز خواهیم داشت و در نهایت random_state نیز در حالت 52 بهترین عملکرد را خواهد داشت.

```
mlp = MLPRegressor(hidden_layer_sizes=(16,16,8,8), activation='relu', solver='lbfgs', random_state=52, max_iter=254)
mlp.fit(X_train, y_train)
```

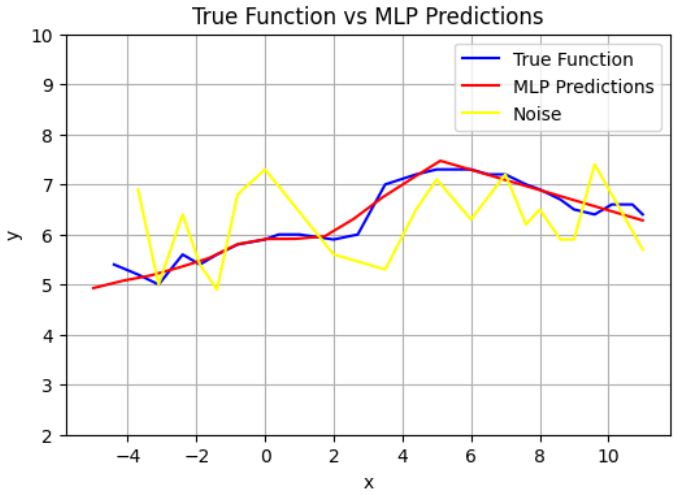
از آنجایی که حجم داده ورودی بسیار کم است و 26 نقطه بیشتر نمیباشد دقت مدل 87 درصد بوده و تنها راه افزایش آن، افزایش تعداد نقاط است که دستی وارد کردن تعداد زیادی از نقاط کار بسیار زمانبری میشود.



در مرحله بعد از ما خواسته شده تا نویزی را به تابع اضافه کنیم و سپس میزان تاثیر پذیری شبکه عصبی را نسبت به نویز مقایسه کنیم. برای این کار اعداد رندمی را به داده آموزشی اضافه کردیم. از آنجایی که اکنون میدانیم برخی از داده ها پرت حساب خواهند شد از یک مکانیزم پیش پردازش بر روی دیتاست استفاده میکنیم و یک داده تمیز از دل داده آمیخته با نویز خارج میکنیم تا دقت بهتری داشته باشیم.

میزان دقت هر دو حالت داده آمیخته به نویز و داده عادی نمایش داده شده اند. در حالت عادی در صورت نویز داشتن داده، دقت حدودا 20 درصد کاهش پیدا میکند که در حالتی که پیش پردازش انجام شود حدودا 15 درصد آن جبران می‌شود. (دلیل استفاده از قید حدودا به خاطر تقریبی بودن میزان نویز است)

بعد از نمایش تابع های پیشبینی شده و تابع اصلی و نویز مشاهده میکنیم که به نسبت نویزی که اضافه شده، شبکه عصبی تغییر چندانی نداشته و همچنان دقت بالایی را دارا میباشد.



پاسخ سوال تاثیر پرش های ناگهانی: پرش‌های ناگهانی در دیتاست تأثیرات منفی بر نتیجه خروجی شبکه عصبی دارند. این پرش‌ها باعث ایجاد اختلالات در فرآیند آموزش شبکه شده و باعث افزایش خطاها و عدم پایداری در پیش‌بینی‌های مدل شوند. از آنجا که شبکه‌های عصبی به داده‌های پیوسته و پیوسته وار وابسته هستند، وجود پرش‌های ناگهانی می‌تواند توانایی مدل در یادگیری الگوهای صحیح را کاهش دهد. بنابراین، بهتر است، دیتاست‌ها از پرش‌های ناگهانی پاکسازی شده و داده‌های پیوسته‌ای برای آموزش شبکه‌های عصبی استفاده شود تا دقت و پایداری مدل بهبود یابد. در بخش اول نیز استفاده از تابع هایی با مجانب عمودی باعث به هم ریختن کامل سیستم ارزیابی می‌شود.

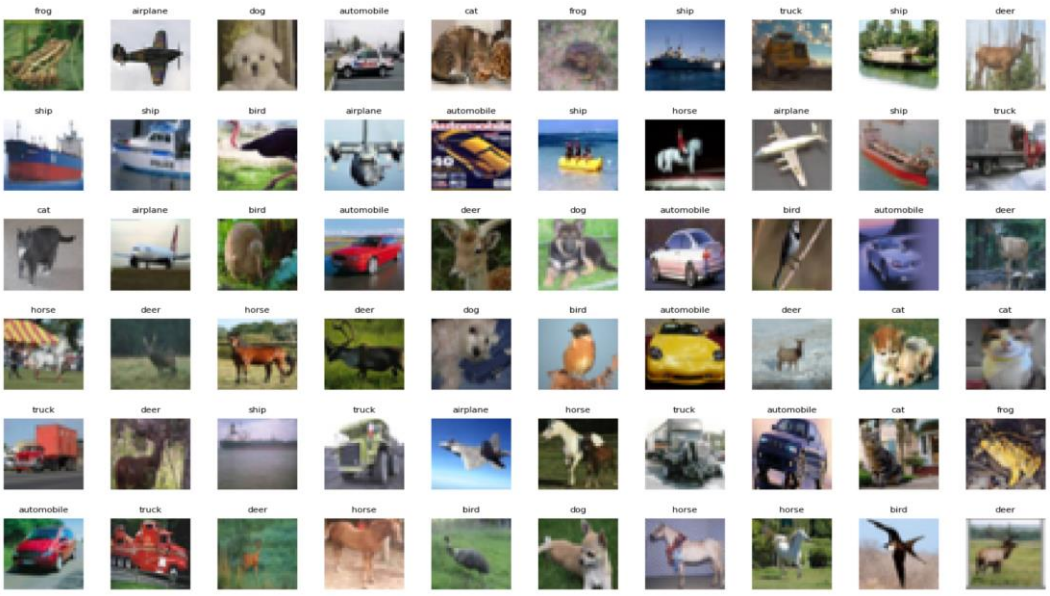
مقایسه نسبت به بخش اول: از آنجایی که تابع استفاده شده در این بخش کاملا رندم است حدود 12 درصد کاهش دقت داشتیم که منطقیست چرا که همانطور که در بالا توضیح داده شد، پرش های ناگهانی در این تابع بیشتر بوده و همچنین پیشبینی آن به خاطر رندم بودن سخت تر است.

سوال دو:

ابتدا به بررسی دیتاست CIFAR میپردازیم. از قرار معلوم دو دیتاست CIFAR-10 و CIFAR-100 وجود داشته که به ترتیب شامل ده و صد دسته بندی هستند. در اینجا ما از دیتاست CIFAR-10 استفاده کردیم که شامل 60000 عکس 32*32 بوده که یعنی هر کلاس شامل 6000 عکس هستند.

این ده کلاس به ترتیب شامل:

1. **هواپیما**
2. **ماشین**
3. **پرنده**
4. **گربه**
5. **آهو**
6. **سگ**
7. **قورباغه**
8. **اسب**
9. **کشتی**
10. **کامیون**



همانطور که داخل داک پروژه اشاره شد در این مرحله به سرچ درباره Image classification پرداخته شد و از کد های به انتشار گذاشته درون سایت Kaggle بسیار کمک گرفته شده است.

تقسیم بندی

در ابتدا کتابخانه های مورد نظر را اضافه شده و سپس دیتاست را به نسبت 50000 داده آموزشی و 10000 تست تقسیم شده است.

سپس یک بخشی از داده را به عنوان نمونه به نمایش گذاشته و میزان تراکم عکس ها را در میان دسته بندی های مختلف بررسی کرده که از قرار معلوم تمامی آنها به صورت یکسان میان هر 6 کلاس تقسیم شده بودند.

پیش‌پردازش

در این مرحله ابتدا پیکسل ها را که داده آنها عددی میان 0 تا 255 هست را به بین 0 و 1 تقسیم بندی میکنیم و داده ها را نیز میان ده کلاس دسته بندی میکنیم.

ساخت مدل

در این بخش که پیچیده ترین بخش کار است، لایه های **convolutional, Pooling, Dropout** را اضافه میکنیم. در هر لایه با تعداد فیلتر های متفاوت و padding یکسان به محاسبه کانولوشن میپردازیم تا تمامی بخش های مختلف تصویر پوشش داده شود.

لایه **dropout** نیز برای جلوگیری از overfit شدن می‌باشد و همچنین به مدل کمک می‌کند تا مستقل باشد و نسبت به داده هایی که با آنها مواجه نشده بهتر عمل کند. **MaxPooling2D** نیز برای تقویت مدل در مقایسه و تشخیص در اندازه های مختلف است.

در آخر برای شبکه عصبی خود از دو لایه که لایه اول 128 نورون و لایه دوم 10 نورون داشته استفاده شده است. activation = softmax در لایه خروجی شبکه های عصبی استفاده میشود که خروجی را به احتمال توضیع به کلاس های ممکن تبدیل می‌کند.

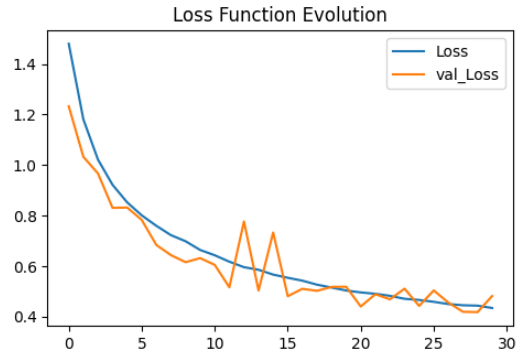
Early Stopping

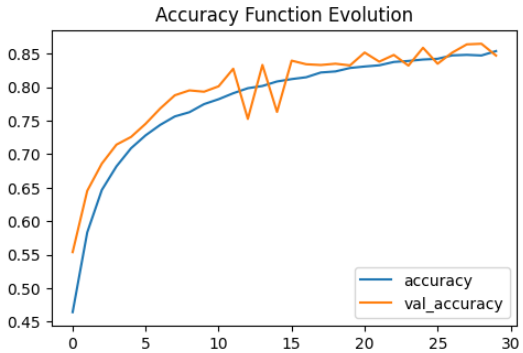
در صورتی که لاس به مقدار لازم در ایپاک مورد نظر نرسیده باشد از توقف آن جلوگیری می‌کند. دو پارامتر دارد که اولین آن یعنی monitor بر روی val_loss ست می‌شود که در راستای همین تعریفی که اکنون گفته شد است، اما patience مربوط به تعداد ایپاک های لازم برای ایجاد callback است.

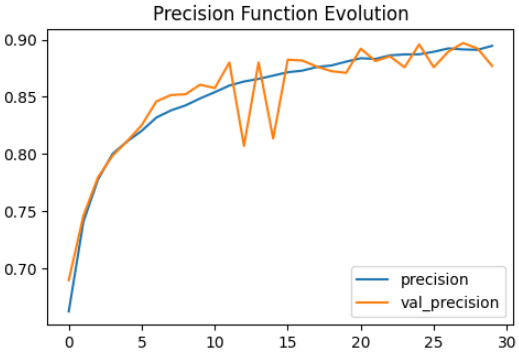
Model Evaluation

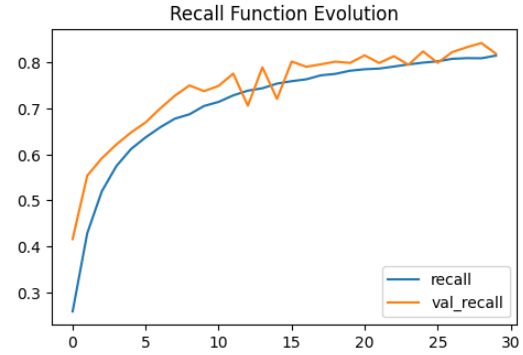
با توجه به نمودار هایی که در ادامه مشاهده خواهید کرد،هر چه تعداد ایپاک ها بیشتر باشد به طبع میزان دقت مدل نیز بالاتر خواهد رفت برای مثال ابتدا با batch_size =64 و تعداد 10 ایپاک به ساخت مدل پرداخته شد که به Accuracy = 79% رسیدیم. اما در انتها با batch_size = 32 و 30 ایپاک به Accuracy = 85% که دقت خوبی برای این مدل است. البته با 20 ایپاک بیشتر یعنی 50، حتی میتوانستیم تا 88% دقت برسیم که به همان میزان زمان بیشتر از ما برای ترین مدل میگرفت و با توجه به کم شدن شیب نمودار Accuracy، ارزش آن را ندارد.

خروجی آن را با استفاده از نمودار های زیر میتوانید مشاهده کنید. سپس میزان دقت مدل را چاپ میکنیم که 84.71% است.





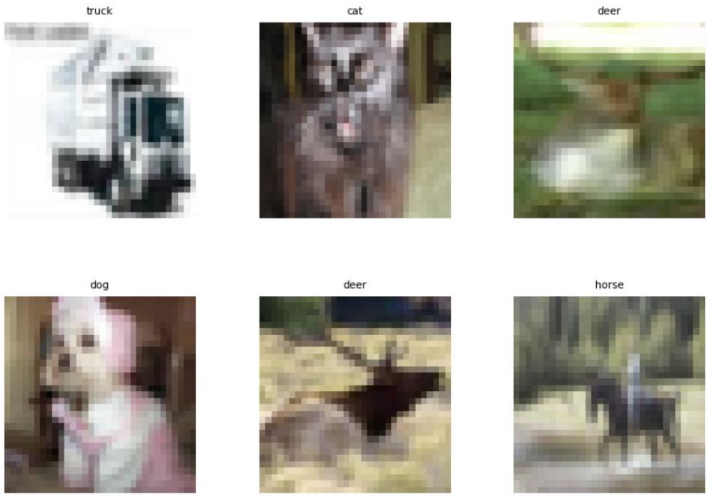


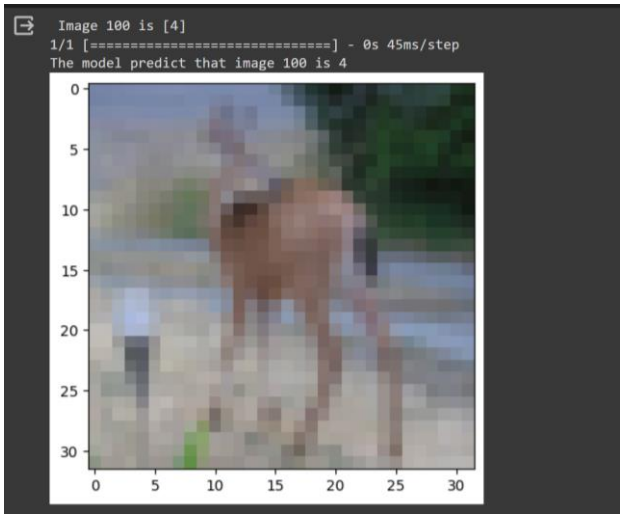


	precision	recall	f1-score	support
0	0.77	0.85	0.81	1000
1	0.90	0.90	0.90	1000
2	0.91	0.53	0.67	1000
3	0.66	0.61	0.63	1000
4	0.82	0.73	0.77	1000
5	0.73	0.71	0.72	1000
6	0.75	0.90	0.82	1000
7	0.80	0.88	0.84	1000
8	0.87	0.87	0.87	1000
9	0.77	0.94	0.85	1000
accuracy			0.79	10000
macro avg	0.80	0.79	0.79	10000
weighted avg	0.80	0.79	0.79	10000

Test on some Images

با توجه به مطالب خواسته شده در داک پروژه در انتها دو تابع رندم برای تست دستی کد در نظر گرفته شده که نمونه ای از آن را مشاهده میکنید: (class4 = “deer”)





سوال سه

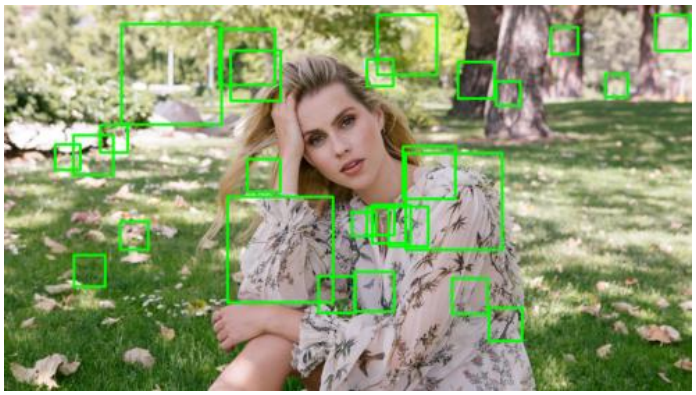
اولین چالش در حل این سوال تفاوت فاحش میان دو دیتاست بود، چرا که دیتاست اول از حجم دیتای بسیار کمتر و همچنین صورت های بزرگتر و تصویر هایی با سایر های بسیار کوچکتر برخوردار است، در صورتی که در دیتاست دوم کاملا برعکس این سه مورد روی می‌دهد.

انتخاب Face Detector

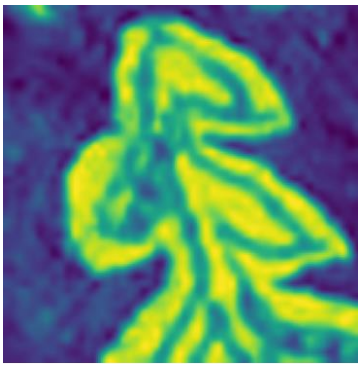
در ابتدا که هدف بررسی صورت ها بود قطعه کدی که از آن کمک گرفته شده بود از MTCNN برای تشخیص چهره استفاده میکرد که همین دلیل بر تغییر هدف و استفاده از دیتاست اصلی شد. اما اشکال اصلی MTCNN تشخیص ندادن چهره در تصویر های خیلی بزرگ بود. در نتیجه فانکشن مورد نیاز برای تشخیص چهره را درون کد عوض کرده و از cascade classifier درون cv2 برای پیدا کردن چهره درون عکس استفاده شد.

دو چالش اصلی استفاده از cascade classifier

1. عدم تشخیص چهره های کوچک در تصویر (لانگ شات ها)



2. تشخیص اشتباه چهره در برخی تصاویر



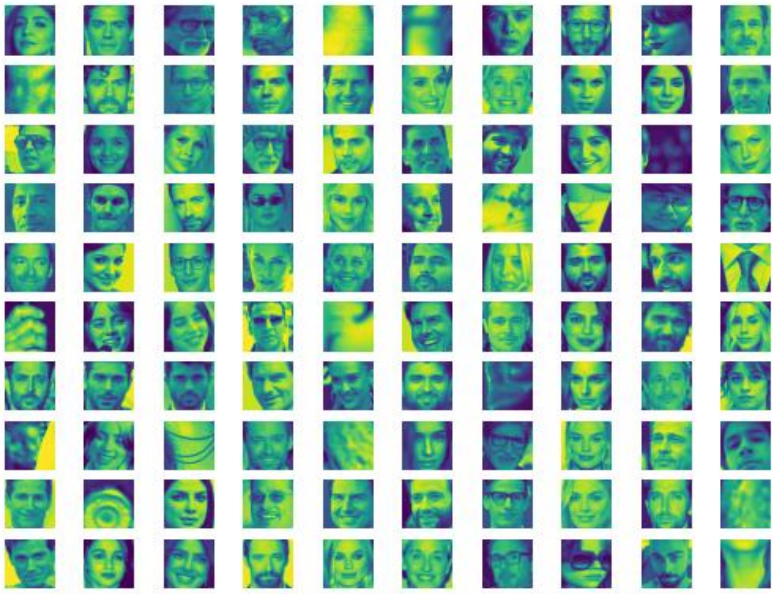
برای حل چالش اول تابع تشخیص چهره را به دو بخش تقسیم شده است تا هر زمان که خروجی از تابع cv2 خالی بود از MTCNN برای تشخیص چهره استفاده کند. این روش به طور کامل باعث حل این مشکل شد.

برای حل چالش دوم ابتدا به بزرگ کردن حداقل مقدار ممکن برای اندازه صورت پرداخته شد. این مورد هم تا حدی باعث کاهش این مشکل شد اما در نهایت تقریبا از 16 تصویر یک الی دو تصویر تشخیص اشتباه داشتند که در این باره به بهبود بیشتری دست پیدا نکردیم.

جدای این موارد دیباگ کردن اشتباهاتی که در نام و یا ساخت فانکشن های مربوط به اینها نیز زمان بسیاری را به خود اختصاص داد چرا اکثرا خروجی های فانکشن ها مشابه یکدیگر بوده و این باعث به هم ریختن کد میشد.

ساخت و جدا کردن دیتاست (75:25)

در این مرحله ابتدا ستون کلاس را به دیتافریم اضافه کردیم که نمایانگر اسم بازیگر ها به صورت عدد بود. سپس دیتاست را به صورت 75% برای ترین و 25% برای تست جدا و شروع به خواندن فایل دیتاها و تشخیص چهره برای هر کدام شده است.



ذخیره صورت

صورت های استخراج شده را درون فایل هایی با نام آنها ذخیره کردیم تا در ادامه در مدل از آنها استفاده کنیم.

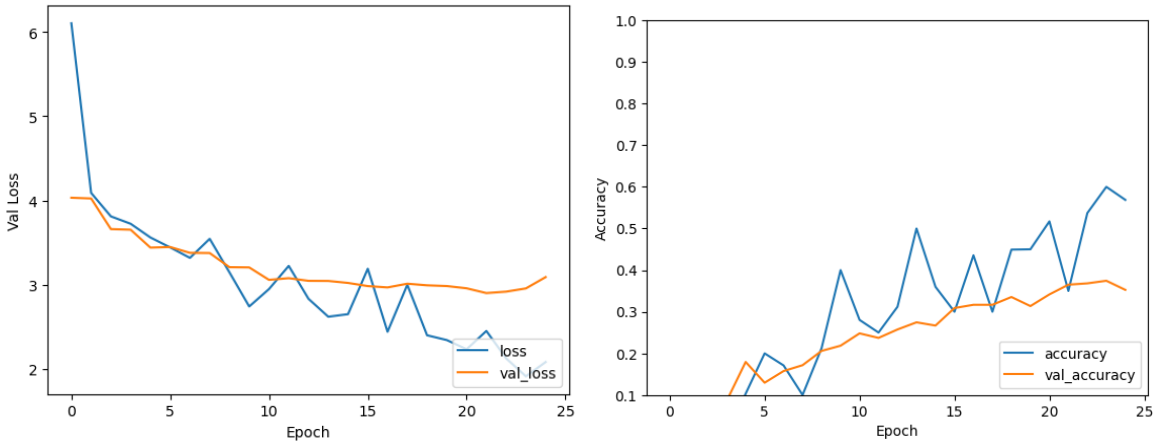
ساخت شبکه عصبی

سپس ساخت شبکه عصبی را با 25 ایپاک و batch_size = 20 ایجاد کردیم. البته به علت نبود زمان کافی، وقت برای بررسی کامل بهترین عدد وجود نداشت اما کد استفاده شده در این بخش در اصل برای دیتاست بسیار سبک تر و با کمیت کمتری بوده و قطعا در صورت استفاده از ایپاک های بیشتر میتوانستیم به دقت بالاتری دست پیدا کنیم.

مشابه بخش قبل مدل را با استفاده از لایه های مختلف از جمله کانولوشن و فیلتر های مختلف برای پوشش کامل عکس ها آموزش میدهیم.

Layer (type)	Output Shape	Param #
conv2d_1056 (Conv2D)	(None, 160, 160, 32)	320
max_pooling2d_528 (MaxPooling2D)	(None, 80, 80, 32)	0
conv2d_1057 (Conv2D)	(None, 78, 78, 64)	18,496
max_pooling2d_529 (MaxPooling2D)	(None, 39, 39, 64)	0
conv2d_1058 (Conv2D)	(None, 37, 37, 128)	73,856
max_pooling2d_530 (MaxPooling2D)	(None, 18, 18, 128)	0
flatten_176 (Flatten)	(None, 41472)	0
dense_616 (Dense)	(None, 512)	21,234,176
mc_dropout (MCDropout)	(None, 512)	0
dense_617 (Dense)	(None, 31)	15,903

CNN Evaluation



Accuracy: 60%

Test one sample