به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

هوش مصنوعی قابل اعتماد

تمرین دوم

نام و نام خانوادگی	نیما مدیرکیاسرایی
شماره دانشجویی	۸۱۰۱۰۲۳۳۹
تاریخ ارسال گزارش	۱۴۰۳/۰۲/۲۹

فهرست

سؤال اول: تفسیر پذیری داده جدولی

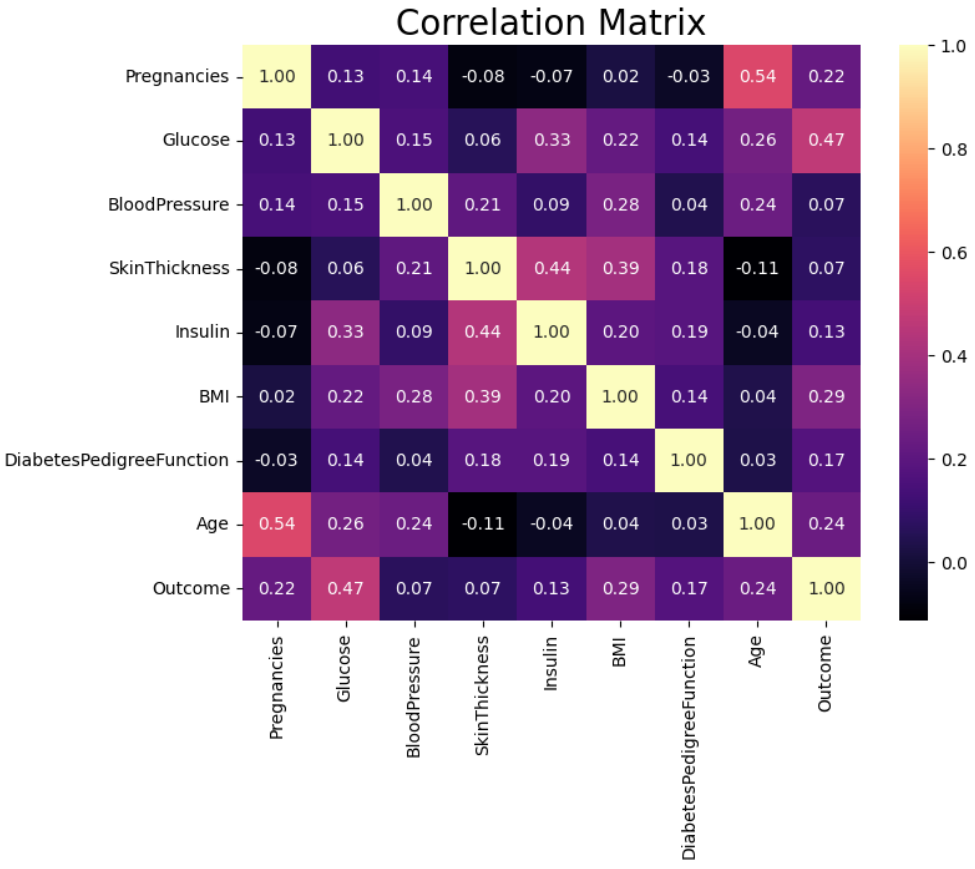
بارگذاری داده

در ابتدا دیتاست Diabetes را بارگذاری می کنیم و به تحلیل کاوشگرانه آن می پردازیم.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

شکل ۱ - توصیف آماری دیتاست

الف) در این قسمت وابستگی ویژگی ها را بررسی می کنیم.



شکل ۲- ماتریس همبستگی

شکل ۳- نمودار pairplot

طبق اشکال بالا می توانیم ببینیم که ویژگی ای که با خروجی (Outcome) بیشترین وابستگی را دارد، مقدار گلوکز (Glucose) می باشد. از طرف دیگر به جز ستون Outcome، ویژگی های سن (Age) و بارداری (Pregnancies) به یکدیگر و همچنین ویژگی های انسولین (Insulin) و ضخامت پوست (Skin Tickness) به یکدیگر بیشترین وابستگی را دارند.

در کل ویژگی های غالب تر بر خروجی برابر هستند با : Glucose – Pregnancies – Age – BMI Insulin و ویژگی های کمتر تاثیر گذار برابر هستند با : Blood Pressure – Skin Thickness

نتایج دیگری که می توان گرفت: برای جلوگیری از مقدار بالای Glucose و Insulin باید BMI را کنترل کرد و همچنین با افزایش سن باید Glucose و Insulin را کنترل کرد.

نتایجی که می توان از pairplot گرفت:

۱. مقدار گلوکز بالا در حاملگی ریسک دیابت را افزایش می دهد.

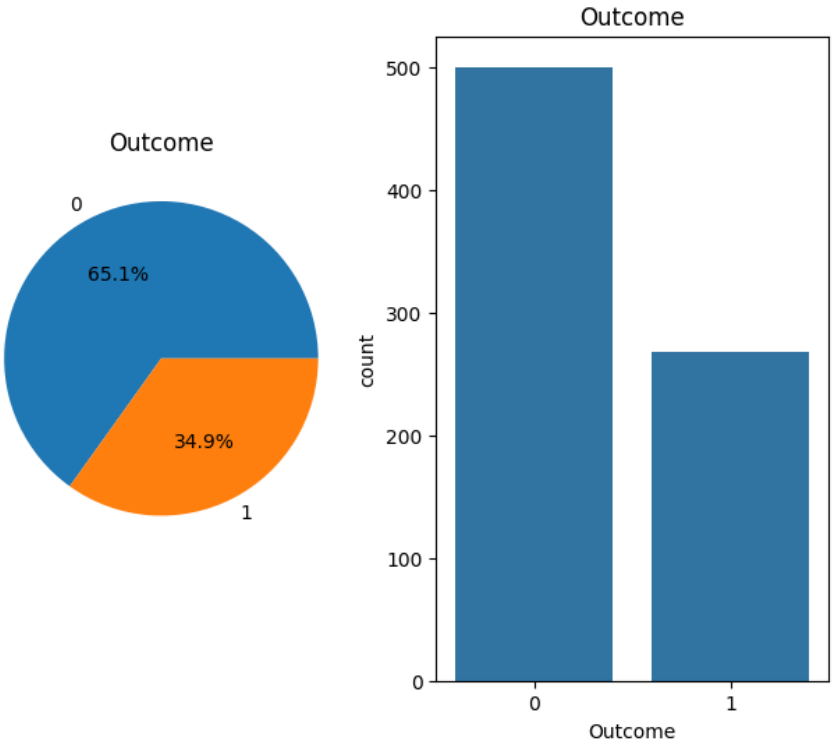
۲. BMI بالای ۳۰ و مقدار بالای Glucose با یکدیگر ریسک دیابت را افزایش می دهند.

۳. می توانیم ببینیم که افزایش سطح گلوکز خون تاثیر کلیدی ای در افزایش ریسک دیابت دارد.

ب) در این قسمت توزیع افراد سالم و مبتلا به دیابت را بررسی می کنیم.

Outcome	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	3.298000	109.980000	68.184000	19.664000	68.792000	30.304200	0.429734	31.190000
1	4.865672	141.257463	70.824627	22.164179	100.335821	35.142537	0.550500	37.067164

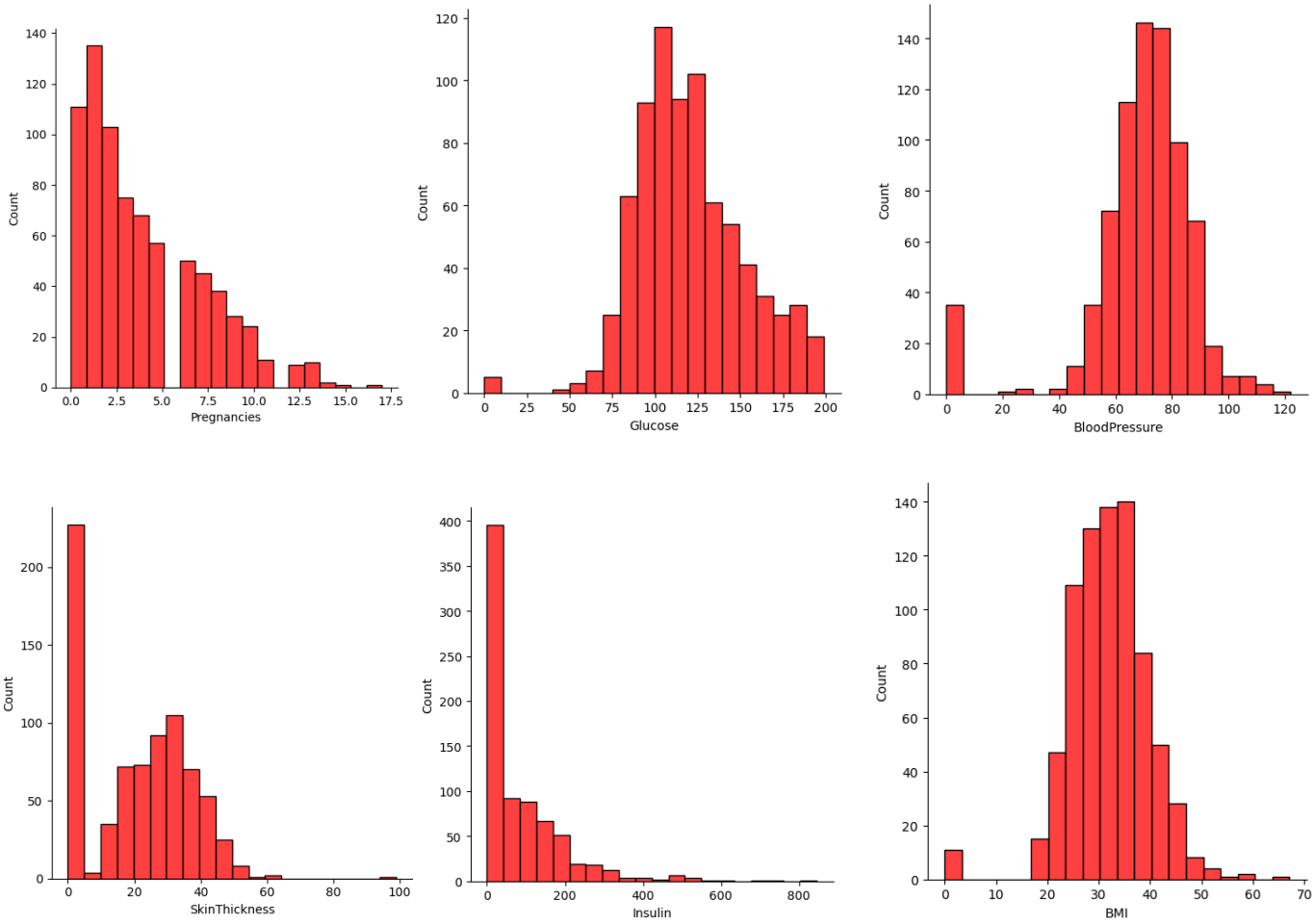
شکل ۴- میانگین ویژگی های مختلف بر اساس خروجی

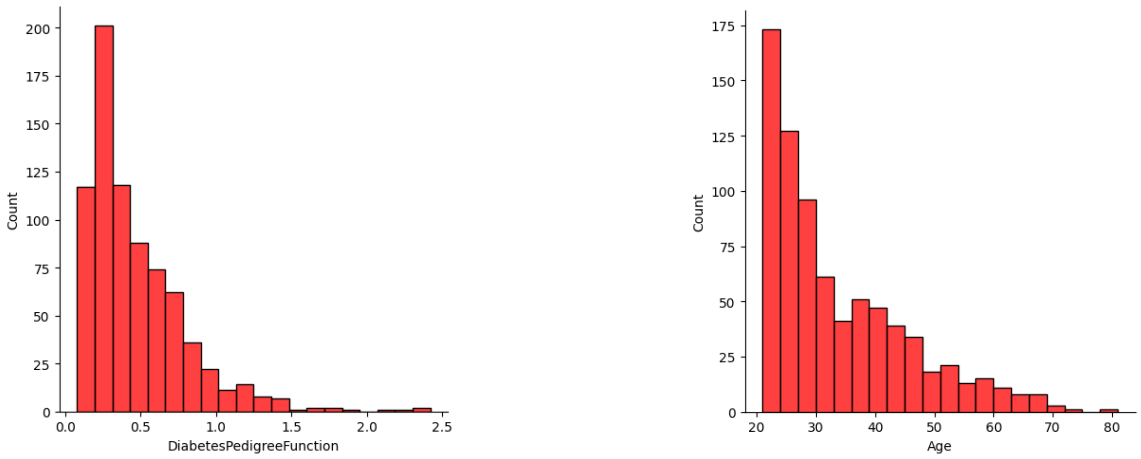


شکل ۵- توزیع افراد سالم و مبتلا به دیابت

طبق شکل بالا می توانیم ببینیم که تقریبا ۶۵ درصد داده ها دارای کلاس ۰ یا همان افراد سالم و ۳۵ درصد داده ها متعلق به افراد بیمار هستن که این یعنی توازن مناسبی در داده ها وجود ندارد و بهتر بود این درصد ها به همدیگر نزدیکتر باشند. این اختلاف ممکن است باعث ایجاد بایاس در جهت سالم پیش بینی کردن بیمار هنگام پیش بینی مدل باشد که خطای مدل را افزایش می دهد.

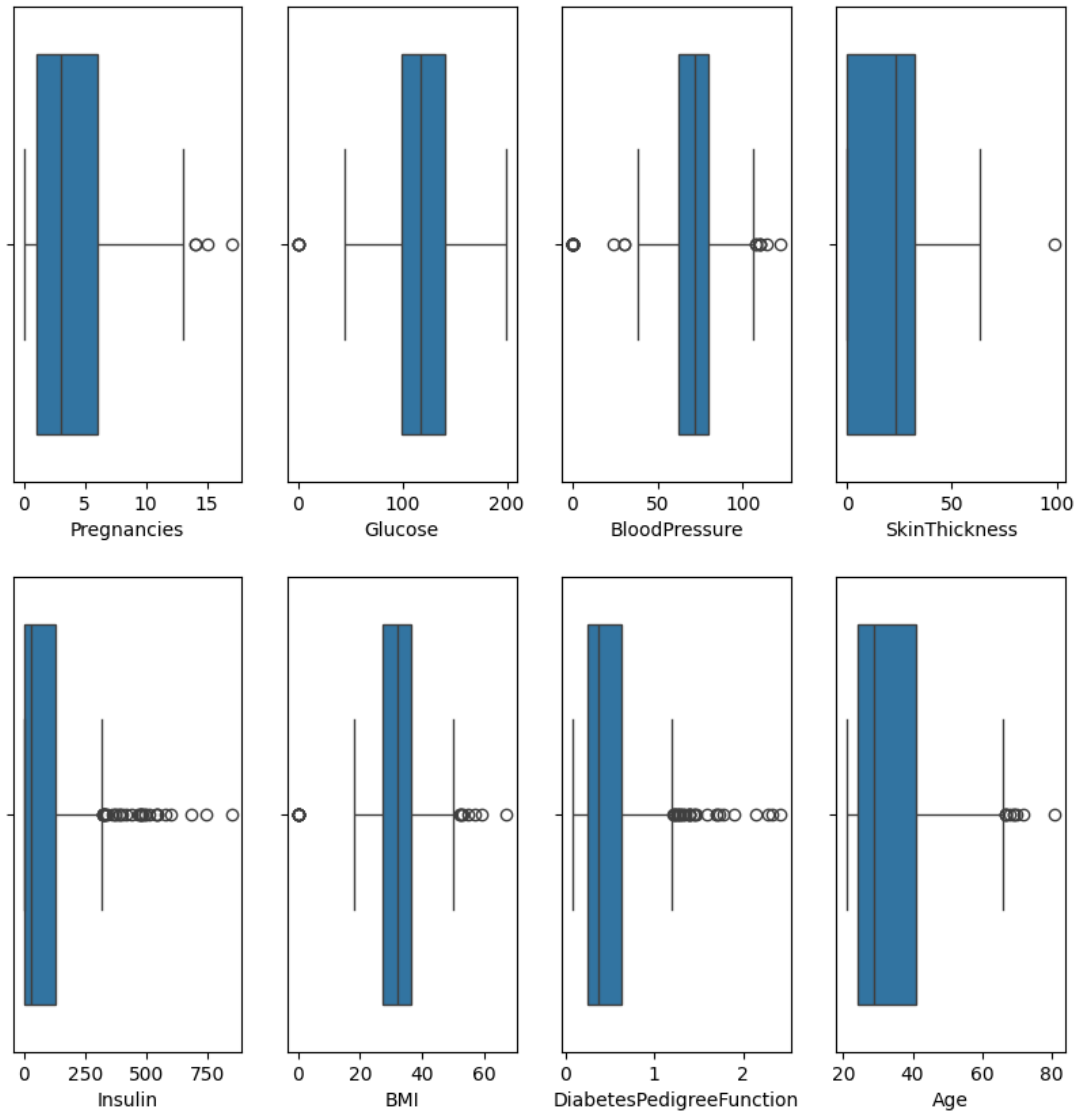
ج) در ابتدا توزیع دیتا برای ویژگی های مختلف را رسم می کنیم.





شکل ۶- توزیع دیتا برای ویژگی های مختلف

در ادامه پراکندگی داده ها را با استفاده از Boxplot نمایش می دهیم.



شکل ۷- نمودار جعبه ای

همانطور که می بینیم در اکثر ستون ها به جز Age، Glocose و Pregnancies داده های پرت وجود دارد. نکته دیگری که می توان به آن توجه کرد این است که ستون Insulin انحراف معیار تقریبا ۱۱۵ دارد که بین بقیه ویژگی ها انحراف معیار خیلی بالایی به حساب می آید که در کل مشکل زا می باشد.

بله، داده‌های پرت (outliers) می‌توانند تاثیرات قابل توجهی روی دقت و تحلیل مدل ها داشته باشند. داده‌های پرت می‌توانند باعث شوند که مدل به طور نامناسبی به این داده‌ها واکنش نشان دهد و به جای تمرکز بر الگوهای اصلی داده‌ها، سعی کند این نمونه‌های نادر را نیز به درستی پیش‌بینی کند. این موضوع می‌تواند دقت کلی مدل را کاهش دهد. معیارهای ارزیابی مانند میانگین مطلق خطا (MAE) و میانگین مربعات خطا (MSE) می‌توانند به شدت تحت تاثیر داده‌های پرت قرار بگیرند. این معیارها در حضور داده‌های پرت افزایش می‌یابند و ممکن است نمایانگر خطای واقعی مدل نباشند. الگوریتم‌های مختلف یادگیری ماشین ممکن است به داده‌های پرت حساس باشند و در نتیجه مدل به خوبی آموزش نبیند.

در ادامه می خواهیم به بررسی داده های نامعلوم (Nan) بپردازیم. در نگاه اول اگر به دیتاست نگاه کنیم می توانیم ببینیم که هیچ داده ی Nan وجود ندارد. اما در واقع در جاهایی که داده نامعلوم است، صفر قرار داده شده است (به جز ستون Pregnancies که مقدار صفر، یک مقدار طبیعی برای آن می باشد.). در ادامه تعداد داده های نامعلوم در هر ستون را مشاهده می کنیم.

Pregnancies	0
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0
Outcome	0

شکل ۸- تعداد داده های نامعلوم در هر ستون

	n_miss	ratio
Insulin	374	48.70
SkinThickness	227	29.56
BloodPressure	35	4.56
BMI	11	1.43
Glucose	5	0.65

شکل ۹- درصد داده های نامعلوم در هر ستون

در بخش پیش پردازش به رفع مشکلات داده های نامعلوم و داده های پرت می پردازیم.

در ادامه به پیش پردازش داده ها می پردازیم. در ابتدا قصد داریم تا به نحوی داده های نامعلوم را پر کنیم. روش های مختلفی برای انجام این کار داریم اما در اینجا برای ستون هایی که صرفا درصد کمی از داده های آن ها نامعلوم هستند(Blood Pressure – BMI – Glucose)، از روش KNN computer استفاده می کنیم که یک نوع روش پیش بینی کننده است و برای ستون هایی که درصد بالایی از آنها نامعلوم هستند (Insulin – Skin Tickness) با استفاده از داده آماری مد این کار را انجام می دهیم.

در ادامه به سراغ حل کردن مشکل داده های پرت می رویم. روشی که ما در این قسمت استفاده می کنیم روش IQR می باشد که در ابتدا برای هر متغیر یا هر ستون threshold های بالا و پایین آن را محاسبه می کنیم که از شکل زیر بدست می آید.

```
low_limit = quartile1 - 1.5 * interquartile_range
up_limit = quartile3 + 1.5 * interquartile_range
```

شکل ۹- روش IQR

در نهایت داده هایی که در هر ستون خارج از این بازه هستند را پیدا می کنیم، اگر پایین تر از این بازه بودند مقدار آنها را به low_limit و اگر بالاتر از این بازه بودند مقدار آنها را به up_limit تغییر می دهیم و نتیجتا توانسته ایم مشکل داده های پرت را حل کنیم.

در مرحله آخر Preprocessing برای تمام ستون ها به جز Outcome، از اسکیلر StandardScaler استفاده می کنیم که میانگین داده های هر ستون را به صفر و واریانس آنها را به ۱ تبدیل می کند و در نهایت مدل را به سه بخش آموزش، ارزیابی و تست تقسیم می کنیم.

آموزش و ارزیابی مدل

در این بخش یک شبکه عصبی مطابق با معماری موجود در جدول ۱ طراحی میکنیم و این شبکه را با دیتاست مورد نظر آموزش می دهیم.

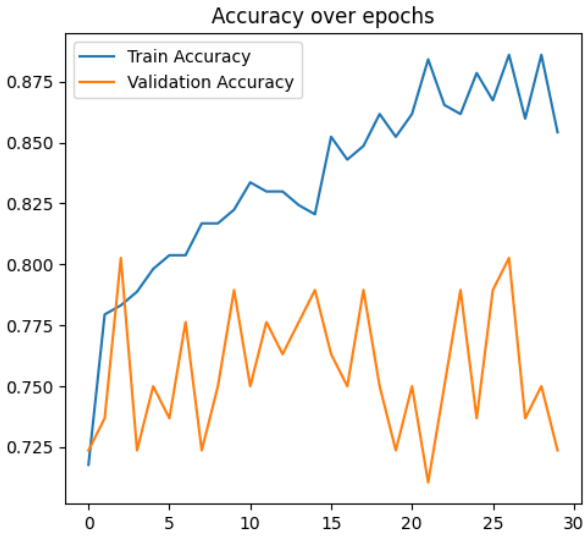
Layer	Config
Linear	input_dim=8, output_dim=100, activation: Relu
Batch Norm	size=100
Linear	input_dim=50, output_dim=50, activation: Relu
Dropout	p=0.2
Linear	input_dim=50, output_dim=50, activation: Relu
Linear	input_dim=50, output_dim=20, activation: Relu
Linear	input_dim=10, output_dim=1

```
class DiabetesNN(nn.Module):
    def __init__(self):
        super(DiabetesNN, self).__init__()
        self.layer1 = nn.Linear(8, 100)
        self.bn1 = nn.BatchNorm1d(100)
        self.layer2 = nn.Linear(100, 50)
        self.dropout = nn.Dropout(0.2)
        self.layer3 = nn.Linear(50, 50)
        self.layer4 = nn.Linear(50, 20)
        self.layer5 = nn.Linear(20, 1)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = self.bn1(x)
        x = F.relu(self.layer2(x))
        x = self.dropout(x)
        x = F.relu(self.layer3(x))
        x = F.relu(self.layer4(x))
        x = torch.sigmoid(self.layer5(x))
        return x
```

شکل ۱۰ – معماری شبکه عصبی مورد نظر

قابل ذکر است که آموزش مدل با پارامترهای زیر انجام می شود:

تابع خطای Binary Cross Entropy Loss – بهینه ساز Adam با نرخ یادگیری ۰.۰۱ – تعداد Epoch ۳۰



شکل ۱۱ – نمودار دقت

شکل ۱۲ – نمودار خطا

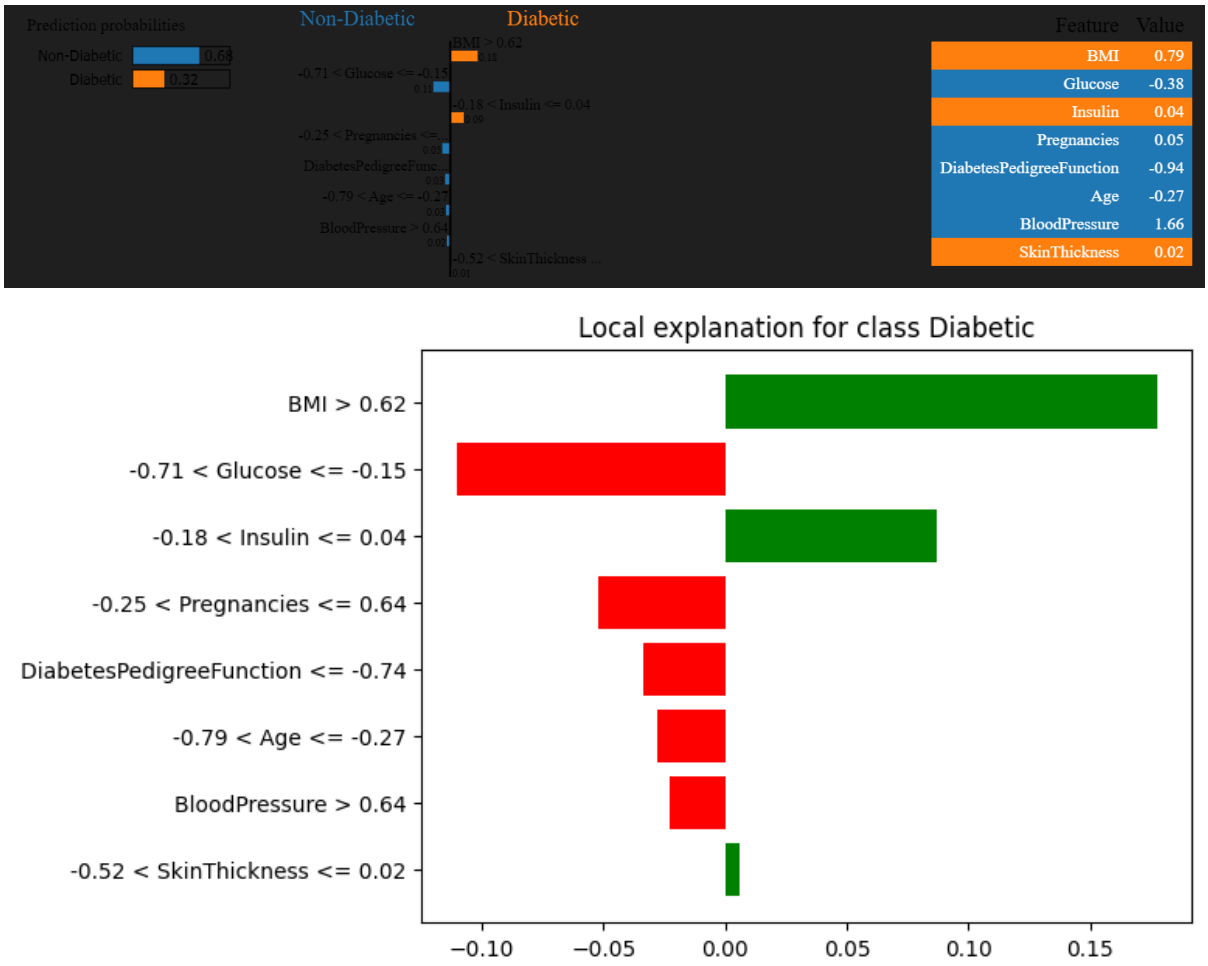
```
Accuracy: 0.7727, Recall: 0.6852, F1 Score: 0.6789
Confusion Matrix:
[[82 18]
 [17 37]]
              precision    recall  f1-score   support

         0.0       0.83      0.82      0.82       100
         1.0       0.67      0.69      0.68        54

    accuracy                           0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154
```

شکل ۱۲ – معیارهای ارزیابی و ماتریس آشفتگی

تفسیر مدل – روش Lime

در این قسمت به صورت تصادفی سه نمونه از داده های داخل جدول را انتخاب می کنیم و با استفاده از تابع explain_instance داخل کتابخانه Lime به تفسیر شبکه آموزش دیده شده می پردازیم.

داده شماره ۱۰

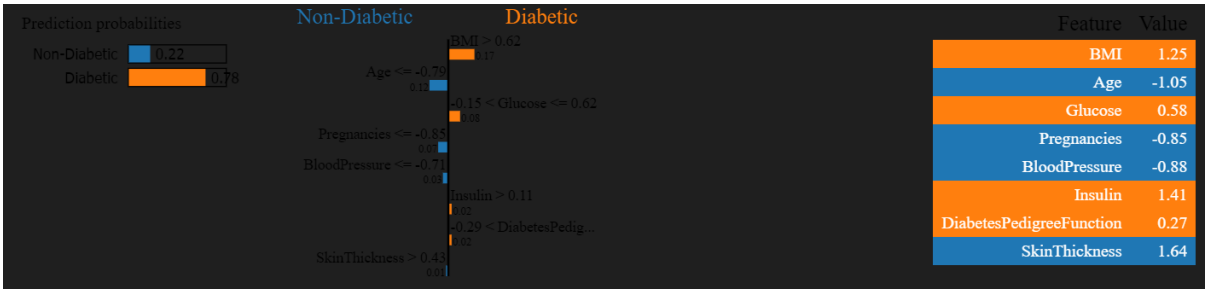


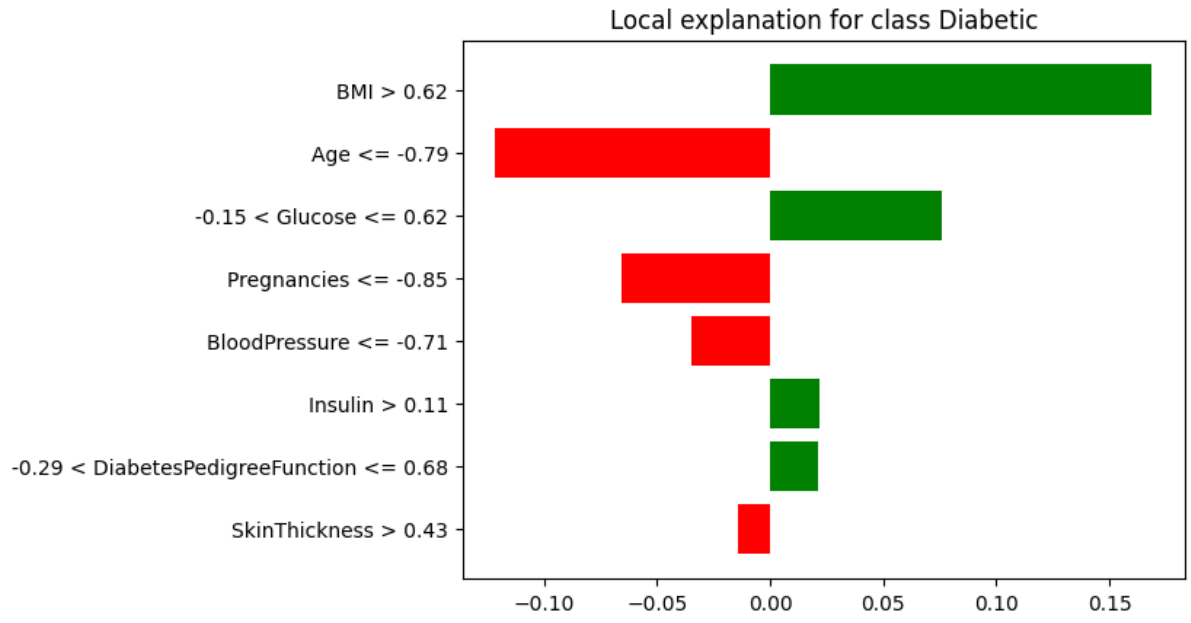
شکل ۱۳ – تفسیر LIME برای نمونه ۱۰

تصاویر بالا نشان می دهد که چگونه هر ویژگی بر پیش‌بینی تأثیر گذاشته است. احتمال پیش‌بینی شده برای دیابت ۰.۳۲ و برای غیر دیابت ۰.۶۸ است. ویژگی هایی که با رنگ نارنجی نشان داده شده اند تاثیر مثبت در پیش بینی دیابت و ویژگی هایی که با رنگ آبی نشان داده شده اند تاثیر منفی در پیش بینی شدن دیابت داشته اند. به عنوان مثال در این مثال مقدار BMI تقریبا ۰.۷۹ بوده است که مقدار بالای آن باعث افزایش احتمال دیابتی شدن فرد می شود اما مقدار گلوکز این فرد ۰.۳۸- بوده است که پایین بودن آن باعث کاهش احتمال دیابتی بودن می شود. در مجموع می توانیم ببینیم که زور ویژگی های آبی به

نارنجی چربیده و احتمال بیشتری به غیر دیابتی بودن تخصیص داده شده است. همچنین طبق دومین تصویر به ترتیب از بالا به پایین می توانیم موثر ترین ویژگی ها را ببینیم که موثرترین آنها BMI و کم اثر ترین آنها Skin Thickness بوده است.

داده شماره ۴۸۴

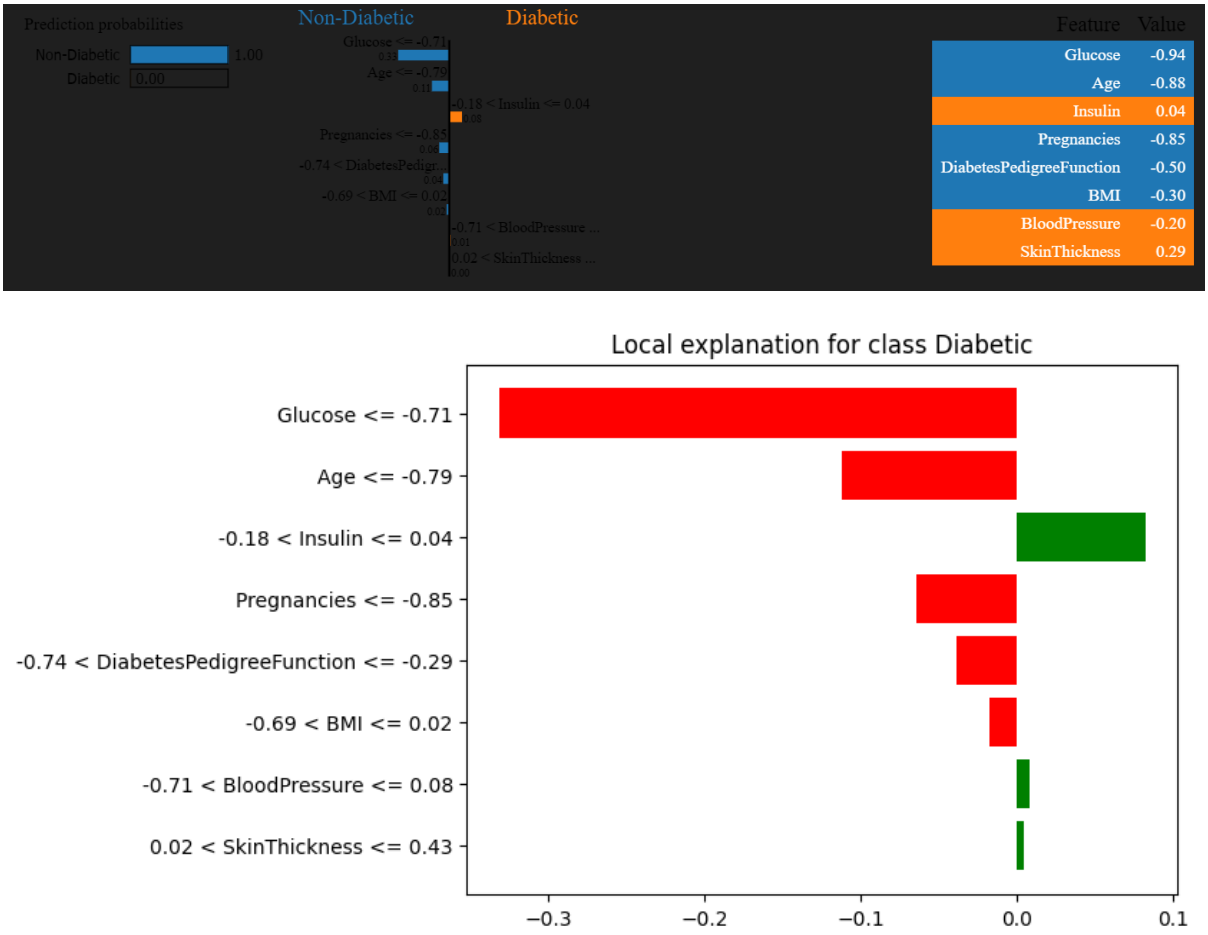




شکل ۱۴ – تفسیر LIME برای نمونه ۴۸۴

طبق تصاویر بالا میتوانیم ببینیم که ویژگی های BMI، Glucose، Insulin و DiabetesPedigreeFunction تاثیر مثبت در تشخیص دیابت دارند اما بقیه ویژگی ها تاثیر منفی. همچنین پر اثر ترین ویژگی ها BMI و سن بوده اند و کم اثر ترین Skin Thickness. در آخر نیز زور ویژگی های با زور مثبت به بقیه چربیده و مدل فرد را مبتلا به دیابت تشخیص داده است.

داده شماره ۷۶۴



شکل ۱۵– تفسیر LIME برای نمونه ۷۶۴

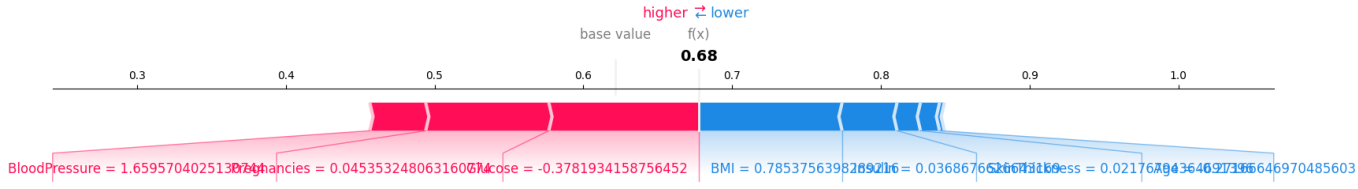
طبق تصاویر بالا میتوانیم ببینیم که ویژگی های Insulin، BloodPressure و SkinTickness تاثیر مثبت در تشخیص دیابت دارند اما بقیه ویژگی ها تاثیر منفی. همچنین پر اثر ترین ویژگی ها Glucose و Age بوده اند و کم اثر ترین Skin Thickness. در نهایت نیز زور ویژگی های با اثر منفی به شدت چربیده و مدل با قطعیت فرد را سالم تشخیص داده است.

تفسیر مدل – روش SHAP

۵) در این قسمت برای ۳ نمونه انتخاب شده در قسمت قبل، با استفاده از متد KernelExplainer داخل کتابخانه SHAP، نمودار force_plot رسم می کنیم تا بتوانیم مدل آموزش دیده شده را تا حدی تفسیر کنیم و تاثیر ویژگی های مختلف در این سه نمونه را مشاهده کنیم.

در تفسیر خروجی های force_plot برای نمودارهای SHAP باید به این نکته توجه کنیم که ویژگی هایی که با رنگ آبی مشخص شده اند سعی در افزایش ریسک ابتلا به دیابت را دارند و ویژگی های با رنگ قرمز ریسک ابتلا به دیابت را کاهش می دهند. در واقع عددی که در بالای هر نمودار این بخش دیده می شود، احتمال سالم بودن فرد از دید مدل آموزش دیده شده می باشد.

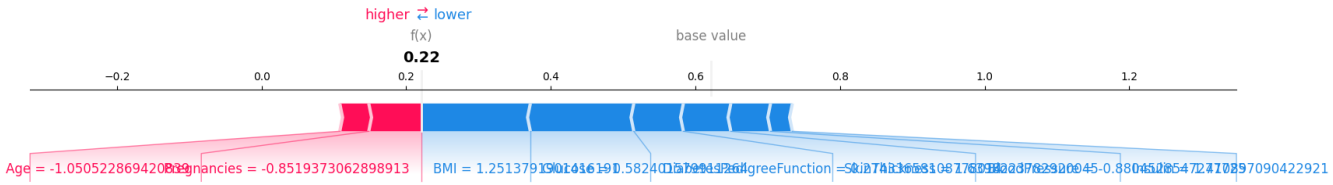
داده شماره ۱۰



شکل ۱۶– تفسیر SHAP برای نمونه ۱۰

در تصویر بالا می توانیم ببینیم که ویژگی های BMI – Insulin – Skin Thickness – Age تاثیر مثبت در دیابتی بودن فرد دارند و باقی ویژگی ها تاثیر منفی دارند. همچنین پر اثر ترین ویژگی ها از سمت مثبت BMI و از سمت منفی Glucose می باشد.

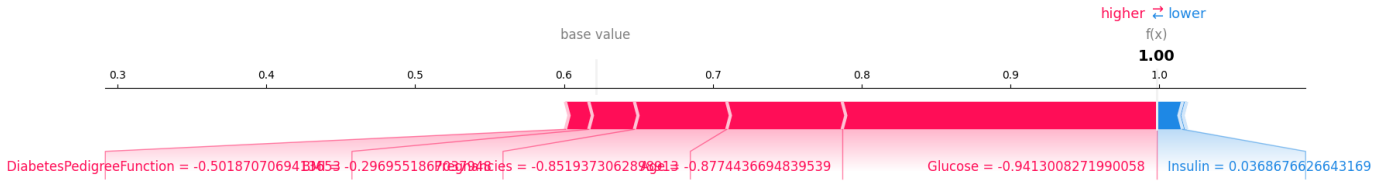
داده شماره ۴۸۴



شکل ۱۷– تفسیر SHAP برای نمونه ۴۸۴

طبق تصویر بالا می توانیم ببینیم که فقط ویژگی های Age – Pregnancies اثر کاهشی در دیابتی تشخیص دادن فرد دارند اما بقیه اثر مثبت در دیابتی بودن فرد دارند. همچنین مهم ترین ویژگی برای این نمونه نیز BMI می باشد.

داده شماره ۷۶۴



شکل ۱۸– تفسیر SHAP برای نمونه ۷۶۴

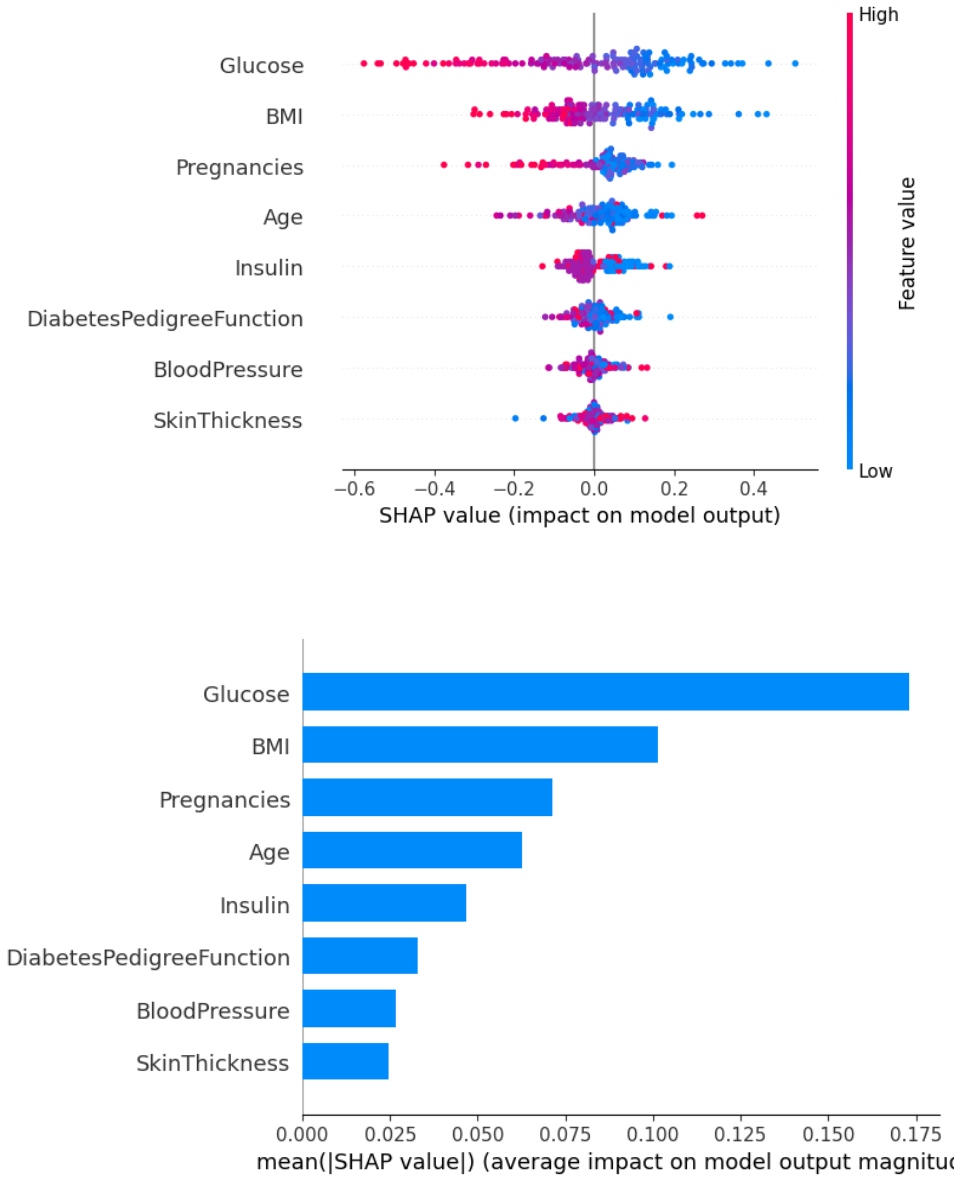
طبق تصویر بالا که مدل با قاطعیت فرد را سالم تشخیص داده است، تنها ویژگی ای که اثر افزایشی در ریسک فرد به دیابت دارد Insulin است و بقیه به تشخیص سالم بودن فرد کمک می کنند و منطقی است که مدل با قاطعیت فرد را سالم تشخیص داده است. همچنین مهم ترین ویژگی در این بخش Glucose می باشد.

۶) هر دو روش SHAP و LIME از روش های محلی تفسیر مدل ها هستند. SHAP بر اساس تئوری بازی‌ها است و اهمیت ویژگی‌ها را به عنوان مقدار Shapley از تئوری بازی‌ها محاسبه می‌کند. این رویکرد تضمین می‌کند که توزیع منصفانه‌ای از اهمیت ویژگی‌ها بین تمام ویژگی‌ها وجود دارد. اما LIME بر اساس مدل‌های محلی خطی است. این روش یک مدل خطی محلی را در نزدیکی نمونه مورد نظر برازش می‌کند تا رفتار مدل پیچیده را در آن ناحیه توضیح دهد. همانطور که طبق نتایج مشخص است روش SHAP اهمیت ویژگی ها را با دقت بیشتری نسبت به روش LIME بیان می کند و چون بر اساس تئوری بازی ها است توزیع منصفانه تری بین ویژگی ها تقسیم می کند اما چون روش LIME با استفاده از یک مدل خطی تخمین می زند ممکن است خطای بیشتری داشته باشد. بر اساس تصاویر بالا اکثر مشاهدات از دو روش LIME و SHAP یکسان بودند اما این نکته قابل توجه است که روش LIME برای ویژگی ها بازه نیز تعریف می کند اما با استفاده از force_plot در SHAP این بازه ها را نمی توانیم ببینیم و صرفا یک عدد به هر ویژگی نسبت می دهد.

طبق ماتریس همبستگی ای که در تصویر شماره ۲ دیدیم، پر اثر ترین ویژگی ها بر روی خروجی ویژگی های BMI و Glucose و کم اثر ترین آنها Skin Thickness بودند. همچنین اگر به خروجی های LIME و

SHAP برای هر ۳ نمونه تصادفی نگاه کنیم، تقریبا در تمامی آنها یا ویژگی BMI و یا ویژگی Glucose تاثیر گذارترین ویژگی در جهت مثبت یا منفی بوده است و کم اثر ترین نیز Skin Thickness بوده است که نشان می دهد این ماتریس همبستگی با تفسیر مدل با استفاده از این دو روش همخوانی دارد.

۷) در ابتدا با استفاده از روش SHAP اهمیت ویژگی ها را برای تمامی داده های تست بدست می آوریم.



شکل ۱۹ – تفسیر SHAP برای کل دیتاست تست

در ادامه برای روش LIME این کار را انجام می دهیم. برای این روش چون تابع آماده ای وجود ندارید تا بتوانیم کل دیتاست تست را به آن بدهیم، دیتاها را جدا جدا به آن می دهیم و در نهایت برای هر ویژگی میانگین گیری انجام می دهیم.

```
Normalized Feature Importances:
Pregnancies: 0.059566234400517 46
Glucose: 0.1865245943908868
BloodPressure: 0.034588642619307 36
SkinThickness: 0.028662653265622916
Insulin: 0.078440476289954 35
BMI: 0.108262575570725 54
DiabetesPedigreeFunction: 0.014881110283374943
Age: 0.046323658089336 62
```

شکل ۲۰ – تفسیر LIME برای کل دیتاست تست

به طور کلی می توانیم ببینیم در مجموع هم ویژگی های BMI و Glucose در صدر لیست پر اهمیت ترین ویژگی ها و BloodPressure و Skin Thickness در انتهای لیست هستند که مطابقت کامل با ۳ نمونه دیده شده در قسمت قبل دارد. اما بین نتایج کلی بین روش های SHAP و LIME تفاوت هایی نیز دیده می شود. مثلا رتبه سوم در روش SHAP متعلق به Pregnancies است در صورتی که در روش LIME برای Insulin می باشد. همچنین کم اهمیت ترین ویژگی در روش LIME برابر با DiabetesPedigreeFunction است در صورتی که در روش SHAP ضخامت پوست می باشد.

تفسیر مدل – روش NAM

الف)

تفاوت‌های مدل NAM با مدل های Black Box هوش مصنوعی:

شفافیت و تفسیرپذیری:

NAMs: به عنوان مدل‌های قابل تفسیر (glass-box) طراحی شده‌اند که هر ویژگی به طور جداگانه و مستقل از دیگر ویژگی‌ها در نظر گرفته می‌شود. تفسیر NAMها با ترسیم توابع شکل‌گیری هر ویژگی امکان‌پذیر است.

مدل‌های Black-Box: به دلیل پیچیدگی و تعاملات بالا بین ویژگی‌ها، تفسیرپذیری کمتری دارند و درک فرآیند تصمیم‌گیری آنها دشوارتر است.

ساختار مدل:

NAMs: ساختارشان به گونه‌ای است که هر ویژگی به وسیله یک شبکه عصبی مستقل مدل می‌شود و نتیجه نهایی از ترکیب این شبکه‌ها بدست می‌آید.

مدل‌های Black-Box: همه ویژگی‌ها به صورت یکجا و به صورت پیچیده مدل می‌شوند.

قابلیت استفاده در مسائل چندوظیفه‌ای (Multitask Learning):

NAMs: به راحتی قابل تنظیم برای مسائل چندوظیفه‌ای هستند، زیرا هر ویژگی می‌تواند توسط چندین شبکه عصبی مدل شود.

مدل‌های Black-Box: پیاده‌سازی مسائل چندوظیفه‌ای پیچیده‌تر است و نیاز به تغییرات اساسی در ساختار مدل دارند.

مزایای استفاده از NAM:

تفسیرپذیری بالا: به دلیل ساختار خاصشان، توضیح نتایج و نحوه تصمیم‌گیری مدل آسان‌تر است.

انعطاف‌پذیری: قابلیت تنظیم برای مسائل چندوظیفه‌ای و چند کلاسه بدون تغییرات اساسی در ساختار مدل.

قابلیت ترکیب با دیگر روش‌های یادگیری عمیق: می‌توانند با سایر روش‌های یادگیری عمیق ترکیب شوند تا مدل‌های پیچیده‌تر و دقیق‌تری ایجاد کنند.

معایب استفاده از NAM:

پیچیدگی محاسباتی: به دلیل نیاز به آموزش چندین شبکه عصبی برای هر ویژگی، ممکن است محاسبات بیشتری نیاز داشته باشند.

پایداری کمتر در برخی موارد: ممکن است در مواردی که نیاز به تعاملات پیچیده بین ویژگی‌ها است، کارایی کمتری نسبت به مدل‌های Black-Box داشته باشند.

ب)

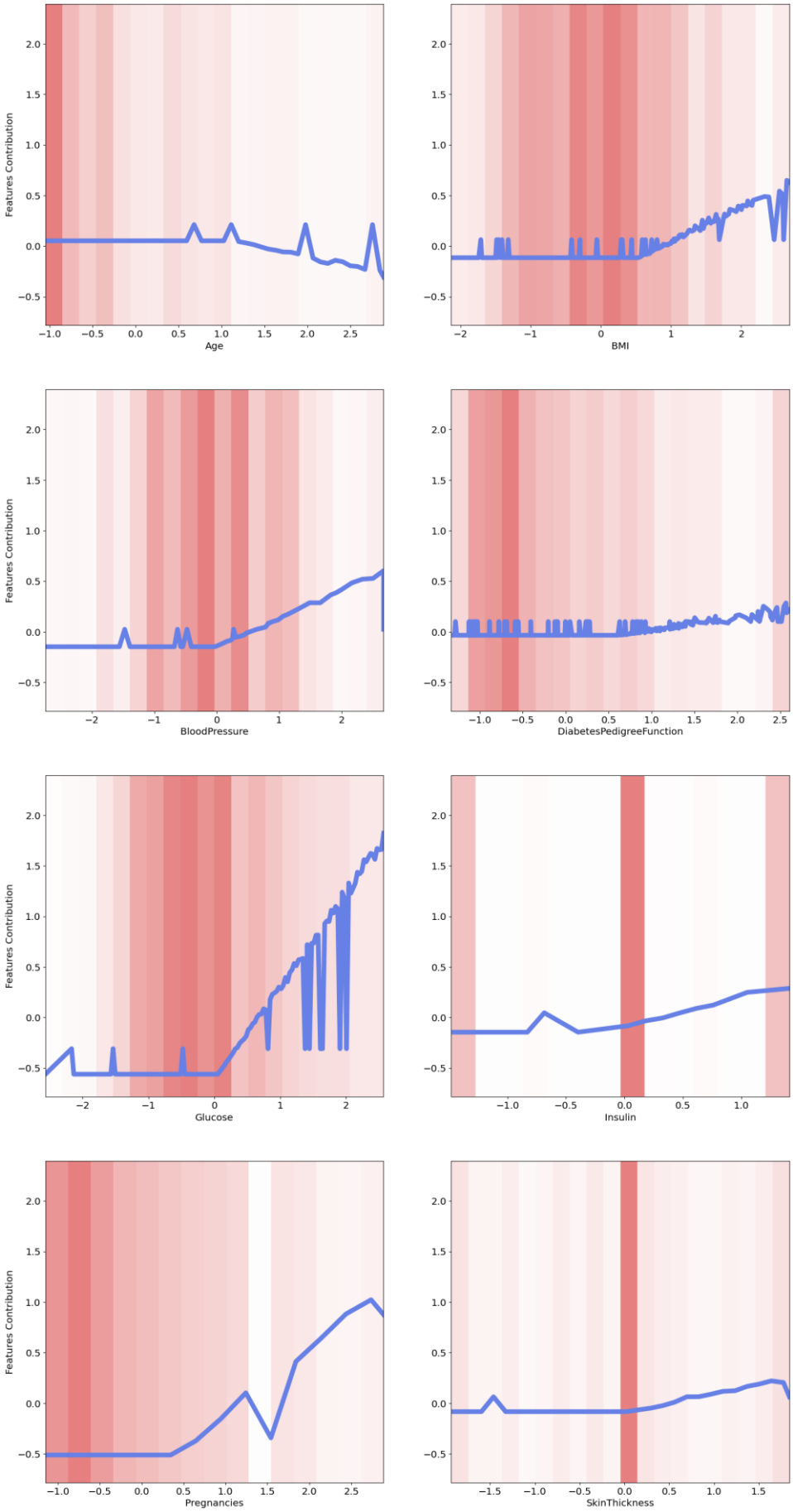
برای پیاده سازی این بخش یک فولدر nam در کنار کدها قرار داده شده است که از یک لینک گیتهاب[۱] آورده شده است و این فولدر برای اجرای کدهای این بخش نیاز است که در Directory شما حضور داشته باشد.

```
model = NAM(
            config=config,
            name=config.experiment_name,
            num_inputs=len(dataset[0][0]),
            num_units=get_num_units(config, dataset.features),
        )
model
```

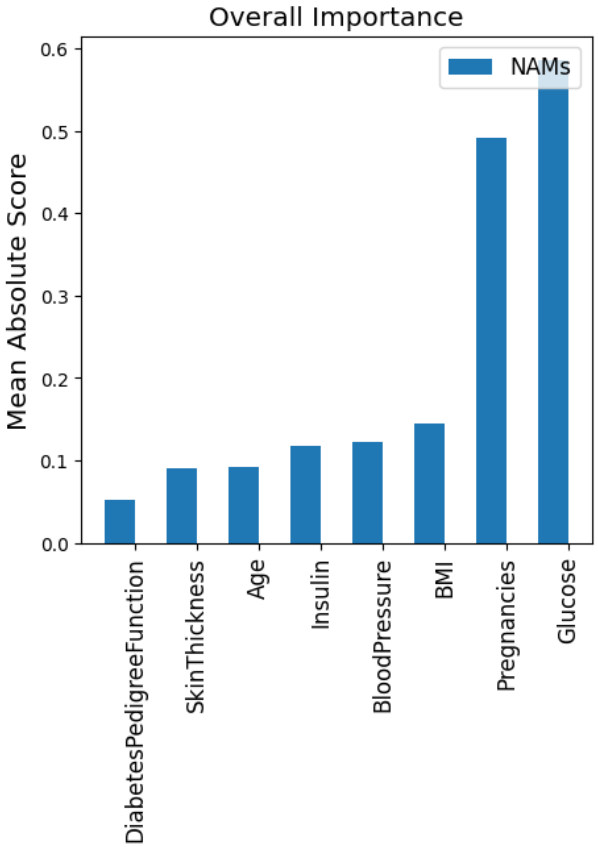
```
NAM(
  (dropout): Dropout(p=0.1, inplace=False)
  (feature_nns): ModuleList(
    (0): FeatureNN(
      (dropout): Dropout(p=0.1, inplace=False)
      (model): ModuleList(
        (0): ExU(in_features=1, out_features=30)
        (1): Linear(in_features=30, out_features=1, bias=True)
      )
    )
    (1): FeatureNN(
      (dropout): Dropout(p=0.1, inplace=False)
      (model): ModuleList(
        (0): ExU(in_features=1, out_features=278)
        (1): Linear(in_features=278, out_features=1, bias=True)
      )
    )
    (2): FeatureNN(
      (dropout): Dropout(p=0.1, inplace=False)
      (model): ModuleList(
        (0): ExU(in_features=1, out_features=124)
        (1): Linear(in_features=124, out_features=1, bias=True)
      )
    )
    (3): FeatureNN(
...
        (1): Linear(in_features=94, out_features=1, bias=True)
      )
    )
  )
)
```

شکل ۲۱ – پیاده سازی و معماری مدل NAM

ج)



شکل ۲۲ – تابع شکل گیری و اهمیت ویژگی ها بر اساس مدل NAM



شکل ۲۳ – اهمیت ویژگی های دیتاست بر اساس مدل NAM

نمودارهای ارائه شده طبق مدل NAM نشان می‌دهند که چگونه هر ویژگی به صورت مستقل بر پیش‌بینی مدل تأثیر می‌گذارد. خطوط آبی نشان‌دهنده تابع شکل‌گیری هر ویژگی و نواحی رنگی نشان‌دهنده میزان تأثیر هر ویژگی است.

:Age

تأثیر کم و نسبتاً ثابت بر پیش‌بینی دارد.

:BMI

با افزایش BMI، تأثیر مثبت آن بر پیش‌بینی افزایش می‌یابد.

:BloodPressure

تأثیر نسبتاً خطی و مثبت دارد.

:DiabetesPedigreeFunction

تأثیر کم و نسبتاً ثابت بر پیش‌بینی دارد.

Glucose:

تأثیر قوی و مثبت دارد، به ویژه در مقادیر بالاتر.

Insulin:

تأثیر کم و نسبتاً ثابت بر پیش‌بینی دارد.

Pregnancies:

تأثیر مثبت با افزایش تعداد بارداری‌ها.

SkinThickness:

تأثیر کم و نسبتاً ثابت بر پیش‌بینی دارد.

نتیجه‌گیری:

Glucose و BMI بیشترین تأثیر مثبت را بر پیش‌بینی دیابت دارند.

ویژگی‌های Age، DiabetesPedigreeFunction، Insulin، و SkinThickness تأثیر کمتری بر پیش‌بینی دارند.

BloodPressure و Pregnancies نیز تأثیر مثبتی دارند، اما به اندازه Glucose و BMI نیستند.

چون این روش به طور مستقل به ویژگی ها نگاه می کند بنابراین تفسیر بهتری می تواند نسبت به دو روش قبلی داشته باشد. در اکثر موارد با روش های قبلی یکسان است اما تفاوت هایی نیز دارد به عنوان مثال در اینجا زایمان از BMI مهم تر است اما در روش های قبلی برعکس بود.

تفسیر مدل – روش GRACE

برای اجرای کدهای این بخش یک فولدر به اسم src_grace در کنار کدها قرار گرفته است [۲] که نیاز است در Directory شما حضور داشته باشد.

در ابتدا مدل را با استفاده از Trainer کتابخانه GRACE آموزش می دهیم.

```
Training...
Val loss: 0.6788 Val acc: 0.5714
```

شکل ۲۴ – آموزش مدل GRACE

در ادامه می توانیم مدل آموزش دیده شده را روی دیتاست تست کنیم.

```
Val loss: 0.6157 Val acc: 0.6286

  Dataset     Accuracy    F1
============================
Validation    0.629     0.613
Test          0.688     0.688
```

شکل ۲۵ – تست مدل GRACE

در نهایت با استفاده از این روش می توانیم نمونه جدید را طوری تولید کنیم که کلاس آن تغییر کند.

```
 Dataset   #avgFeatChanged   Fidelity
==================================
Test           2.948         0.740

         sample  prediction  Glucose BloodPressure SkinThickness Pregnancies
0      Original           1  117.000       86.000       30.000       5.000
1  Contrastive            0  117.000       86.000       30.000       0.000
```

شکل ۲۶ – تولید نمونه جدید

همانطور که در تصویر بالا می بینیم تمامی متغیر ها به جز Pregnancies ثابت باقی مانده و صرفا تعداد زایمان از ۵ اگر به صفر تغییر کند می توان گفت که کلاس پیش بینی شده از ۱ به ۰ تغییر می کند یعنی مدل پیش بینی می کند که فرد سالم است. همانطور که در روش های قبل نیز دیدیم ویژگی Pregnancies جز ویژگی های پر تاثیر در کاهش ریسک به دیابت بود که در اینجا نیز این حرف صادق است و برای سالم نشان دادن بیمار مقدار آن را به صفر کاهش داد.

سؤال دوم: تفسیر پذیری در حوزه تصویر

روش GRAD-CAM

در این روش، از طریق بررسی گرادیان‌های پس انتشار شده (انتشار داده شده در جهت عکس جریان اطلاعاتی شبکه عصبی) به سمت «نقشه‌های فعال‌سازی کلاسی» (Class Activation Maps)، میزان تأثیرگذاری بخش‌های مختلف یک تصویر در خروجی (پیش‌بینی) تولید شده توسط شبکه عصبی محاسبه می‌گردد.

گام‌های استفاده از این روش به‌صورت زیر می‌باشد

۱- ورودی: با یک تصویر شروع کنید و پیش‌بینی خاصی را انتخاب کنید که می‌خواهید بفهمید.

۲- اصلاح مدل: یک مدل شبکه عصبی پیچشی از پیش آموزش‌دیده را انتخاب کنید (در اینجا VGG16) و آن را در آخرین لایه پیچشی حذف کنید. لایه‌های کاملاً متصل را در بالای مدل اضافه کنید تا یک پیش‌بینی بر اساس ویژگی‌های استخراج شده توسط آخرین لایه ایجاد کنید.

۳- مرحله انتشار رو به جلو: تصویر انتخاب شده را از مدل اصلاح شده عبور دهید تا خروجی‌های زیر را به‌دست آورید:

پیش‌بینی: خروجی مدل، که احتمال پیش‌بینی انتخابی را نشان می‌دهد (به عنوان مثال، احتمال تصویر حاوی یک گربه).

هزینه: تفاوت بین مقدار تخمین‌زده‌شده توسط مدل و مقدار واقعی را محاسبه کنید.

خروجی آخرین لایه: خروجی آخرین لایه کانولوشن را که نمایان‌گر ویژگی‌های آموخته‌شده تصویر است را بازیابی کنید.

پس‌انتشار: با استفاده از الگوریتم پس‌انتشار، گرادیان خروجی آخرین لایه را نسبت به هزینه محاسبه کنید. این گرادیان نشان می‌دهد که چگونه هر عنصر از خروجی لایه آخر به پیش‌بینی کلی کمک می‌کند.

بخش‌های گرادیان مرتبط: بخش‌هایی از گرادیان را که تأثیر مستقیمی بر پیش‌بینی انتخابی دارند، شناسایی کنید. این بخش‌ها مهم‌ترین ویژگی‌های تصویر را که بر تصمیم مدل تأثیر گذاشته‌است برجسته می‌کنند. (به عنوان مثال، مناطقی که نمایان‌گر حضور یک گربه هستند).

تولید نقشه‌حرارتی: بخش‌های شناسایی‌شده از گرادیان را برای مطابقت با ابعاد تصویر ورودی، کاهش، اندازه و تغییر مقیاس دهید. این فرایند تضمین می‌کند که نقشه‌حرارتی همان شکل تصویر اصلی را خواهد داشت. نقشه‌حرارتی به‌دست آمده، مهم‌ترین بخش‌های تصویر را که به پیش‌بینی انتخاب‌شده کمک کرده‌اند. (به عنوان مثال، مناطقی که گربه در آن قرار دارد یا قسمت‌هایی از بدن گربه) را به تصویر می‌کشد.

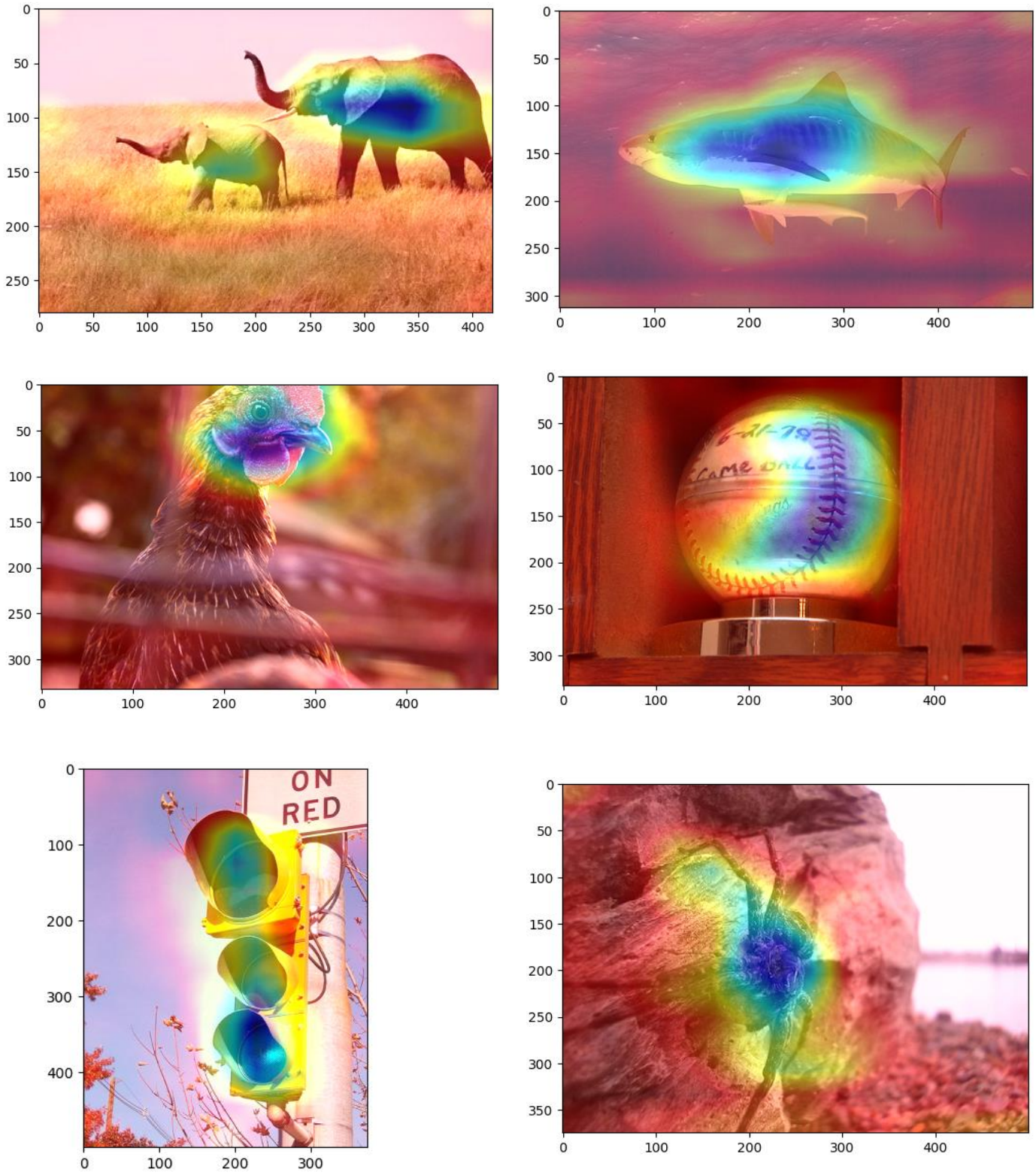
$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{gradients via backprop}} \qquad (۱)$$

$$L_{\text{Grad-CAM}}^c = ReLU\underbrace{\left(\sum_k \alpha_k^c A^k\right)}_{\text{linear combination}} \qquad (۲)$$

طبق رابطه اول وزن های α^k برای هر نقشه ویژگی A^k از طریق میانگین گیری گرادیان ها در تمامی موقعیت های مکانی (i, j) محاسبه می شوند. همچنین Z تعداد پیکسل ها در نقشه ویژگی A^k می باشد.

طبق رابطه دوم نقشه حرارتی Grad-CAM با ترکیب خطی نقشه های ویژگی A^k و وزن های α^k به دست می آید. برای اطمینان از اینکه فقط ویژگی های با تاثیر مثبت در نظر گرفته می شوند از تابع Relu نیز استفاده می کنیم. این تابع تضمین می کند که تنها مقادیر مثبت در نظر گرفته شوند و مقادیر منفی صفر شوند.

در هنگام پیاده سازی نیز دقیقا همین الگوریتم را به صورت کد می نویسیم و کلاس GradCAM را تشکیل می دهیم. در ادامه با استفاده از این کلاس، saliency map مربوط به ۶ عکس مختلف را تولید می کنیم.



شکل ۲۷ – خروجی های Grad Cam

روش Guided GRAD-CAM

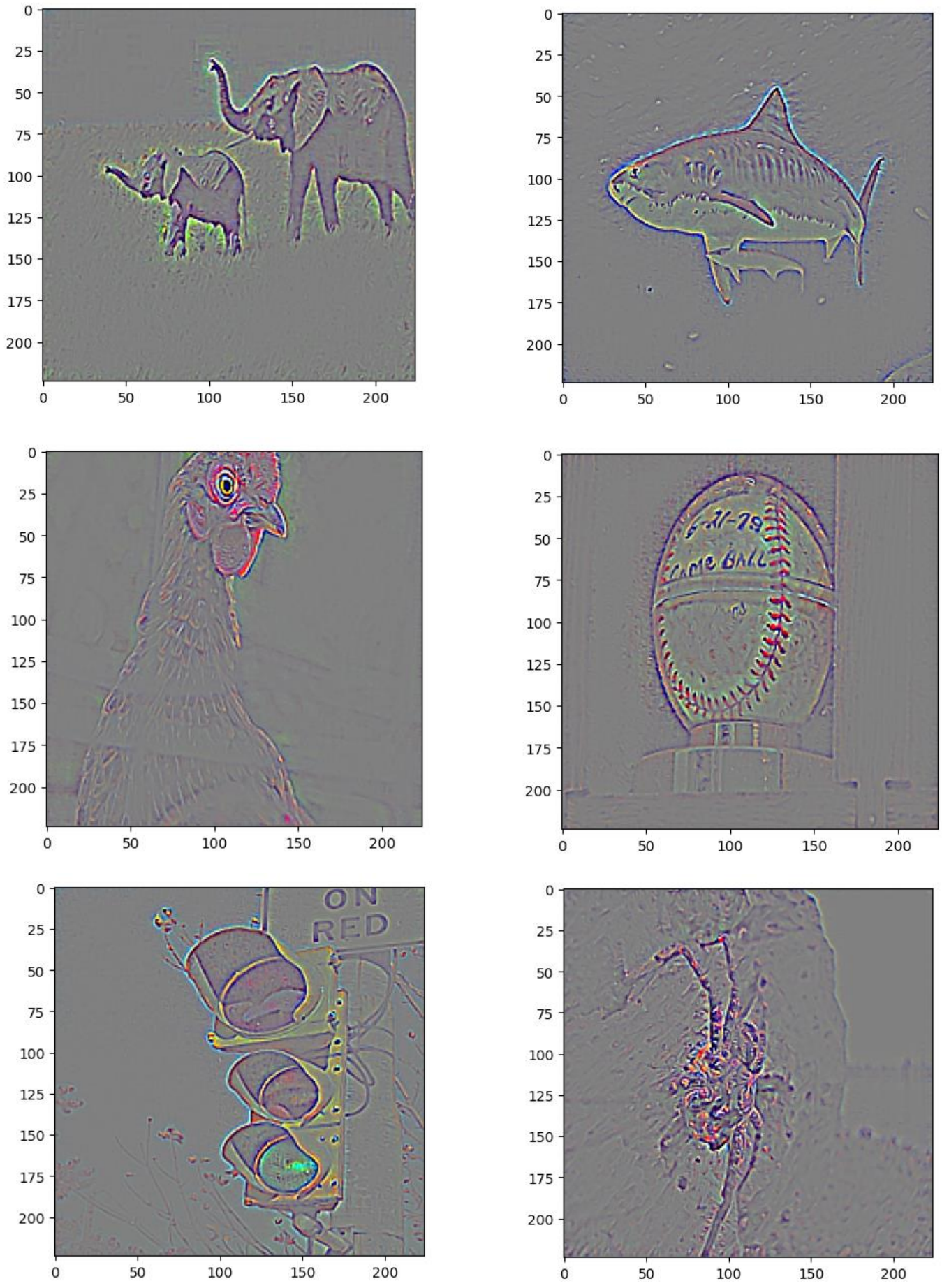
در روش Backpropagation معمولی، برای ایجاد saliency maps، گرادیان نمره خروجی نسبت به ورودی محاسبه می‌شود. این گرادیان‌ها نشان می‌دهند که چگونه تغییرات جزئی در پیکسل‌های ورودی روی نمره خروجی تأثیر می‌گذارند. saliency maps حاصل از این روش، نواحی مهم تصویر را برجسته می‌کنند، اما ممکن است شامل نویز زیادی باشند و جزئیات دقیقی را نشان ندهند.

روش Guided Backpropagation، یک تکنیک بهبود یافته از Backpropagation معمولی است که نویز را کاهش داده و وضوح نقشه‌های توجه را افزایش می‌دهد. در این روش، هنگام محاسبه گرادیان‌ها، تنها گرادیان‌های مثبت به لایه‌های پایین‌تر بازپراکندی می‌شوند و گرادیان‌های منفی فیلتر می‌شوند. به عبارت دیگر، فقط تغییراتی که تأثیر مثبت روی نمره خروجی دارند، در نظر گرفته می‌شوند.

در ادامه کلاس Guided_Backprop را پیاده سازی می کنیم و با استفاده از آن برای ۶ عکس مختلف saliency map ها را تولید می کنیم.

```
def backward_hook_fn(module, grad_in, grad_out):
    grad = self.activation_maps.pop()
    grad[grad > 0] = 1
    positive_grad_out = torch.clamp(grad_out[0], min=0.0)
    new_grad_in = positive_grad_out * grad

    return (new_grad_in,)
```

شکل ۲۸ – عبور دادن گرادیان های مثبت در روش Guided

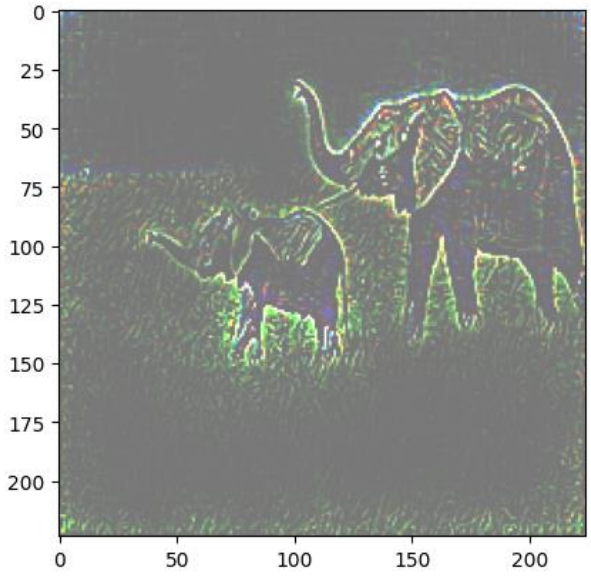


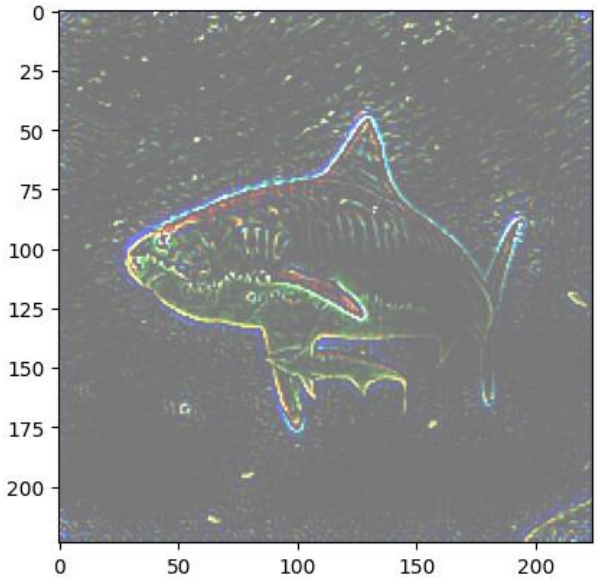
شکل ۲۹ – خروجی های روش Guided Backpropagation

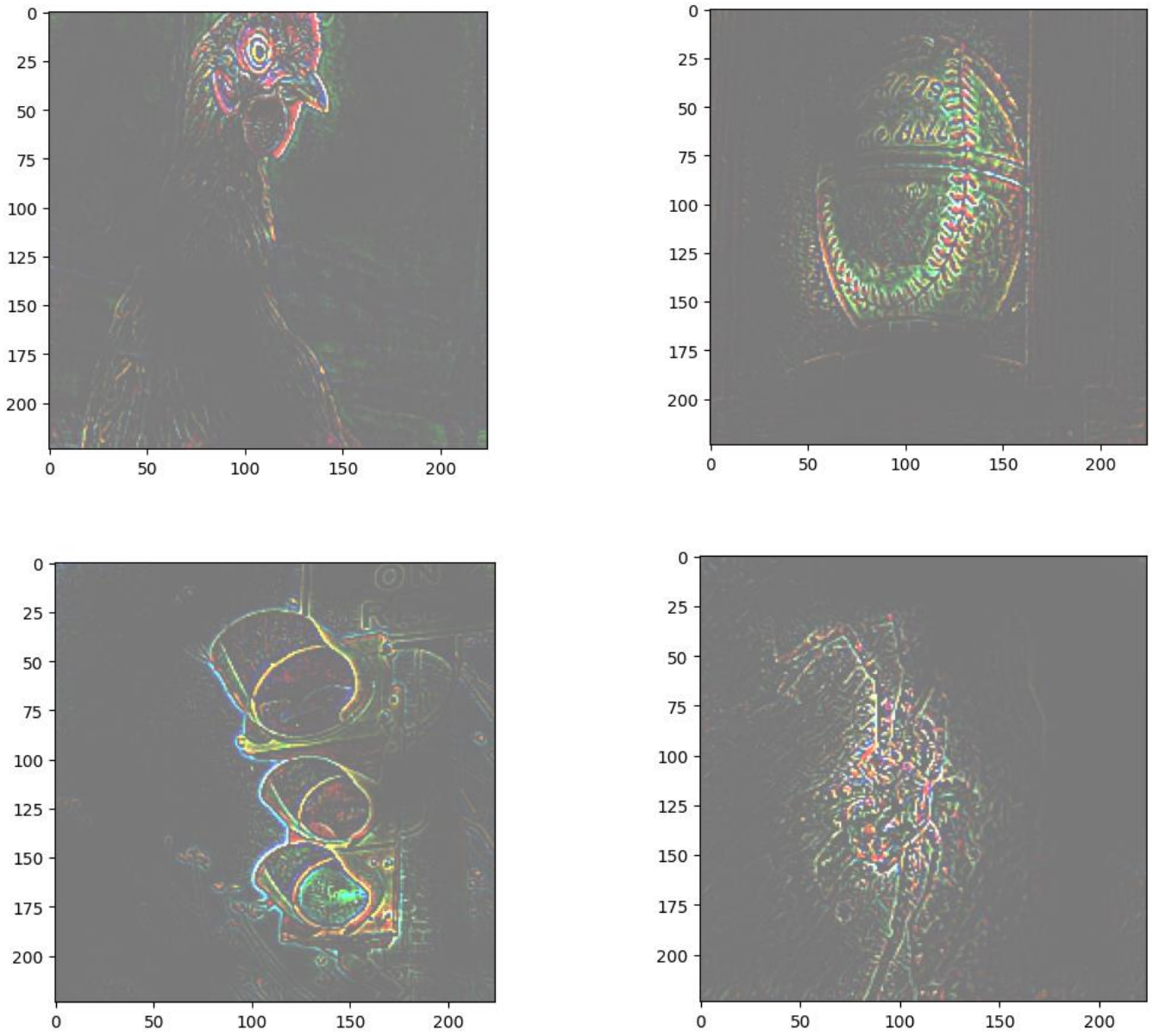
در ادامه به بررسی ترکیب این دو روش می پردازیم. هدف از این ترکیب، بهره‌برداری از مزایای هر دو روش برای ایجاد نقشه‌های توجه (saliency maps) با وضوح بالا (بدست آمده از روش Guided) و خاصیت کلاسی بهتر (بدست آمده از روش GRAD-CAM) است. بعد از اینکه هر دو روش را به طور جداگانه روی تصویر مورد نظر اعمال کردیم و برای هر روش یک ماتریس جداگانه بدست آمد حالا برای ترکیب این دو روش از ضرب درایه به درایه این دو ماتریس استفاده می کنیم تا ماتریس Guided-GRAD-CAM بدست آید.

```python
def guided_grad_cam(gradcam_heatmap, guided_backprop_gradients):
    # Resize Grad-CAM heatmap to match the size of the gradients
    gradcam_heatmap_resized = cv2.resize(gradcam_heatmap, guided_backprop_gradients.shape[:2])
    # Convert the 2D heatmap to 3D by repeating it across the color channels
    gradcam_heatmap_resized = np.repeat(gradcam_heatmap_resized[:, :, np.newaxis], 3, axis=2)
    # Element-wise multiplication
    guided_grad_cam_output = np.multiply(gradcam_heatmap_resized, guided_backprop_gradients)
    # Normalize the result to make it suitable for visualization
    guided_grad_cam_output = np.maximum(guided_grad_cam_output, 0)  # ReLU to ensure non-negative
    guided_grad_cam_output /= np.max(guided_grad_cam_output) if np.max(guided_grad_cam_output) > 0 else 1  # Avoid division by zero
    return guided_grad_cam_output
```

شکل ۳۱ – کد ترکیب Grad Cam و Guided Backpropagation







شکل ۳۱ – خروجی های روش Guided Backpropagation Grad Cam

روش SMOOTHGRAD

این روش با اضافه کردن نویز به داده‌های ورودی، نقشه‌های حساسیت (sensitivity maps) تولید شده توسط مدل‌های شبکه عصبی عمیق را بهبود می‌بخشد. در ابتدا به تصویر ورودی چندین بار نویز گاوسی اضافه می‌شود. هر تصویر تغییریافته یک نسخه نویزی از تصویر اصلی است. برای هر تصویر نویزی، نقشه حساسیت (گرادیان نمره کلاس نسبت به ورودی) محاسبه می‌شود. در نهایت نقشه‌های حساسیت محاسبه‌شده برای هر نسخه نویزی تصویر با هم میانگین‌گیری می‌شوند تا یک نقشه حساسیت نهایی به دست آید. این میانگین‌گیری باعث کاهش نویز و افزایش وضوح نقشه‌های حساسیت می‌شود.

روش SmoothGrad تلاش دارد تا چندین مشکل موجود در نقشه‌های حساسیت تولید شده توسط روش‌های پایه‌ای مانند گرادیان‌های خام (vanilla gradients) را بهبود بخشد:

کاهش نویز:

نقشه‌های حساسیت اولیه معمولاً نویز زیادی دارند که می‌تواند باعث تفسیر غلط و نادرست از نواحی مهم تصویر شود. با اضافه کردن نویز و سپس میانگین‌گیری، نویز کاهش یافته و نقشه‌ها صاف‌تر و دقیق‌تر می‌شوند.

وضوح بهتر نقشه‌ها:

نقشه‌های حساسیت تولید شده توسط SmoothGrad به صورت بصری واضح‌تر هستند و نواحی مهم تصویر را به شکلی که برای انسان‌ها قابل درک‌تر است، نشان می‌دهند.

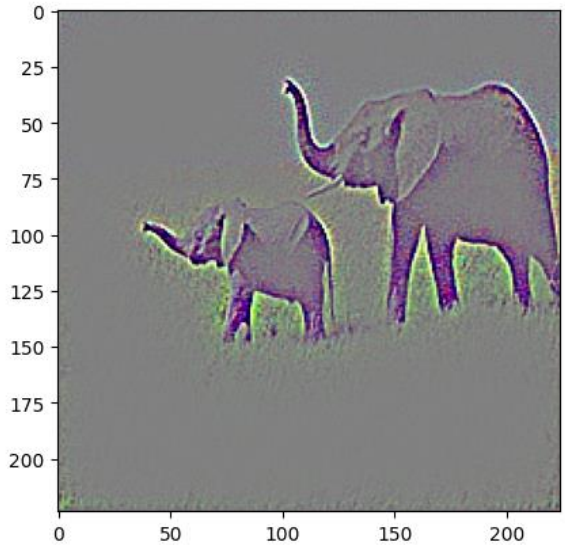
پایدارسازی نقشه‌های حساسیت:

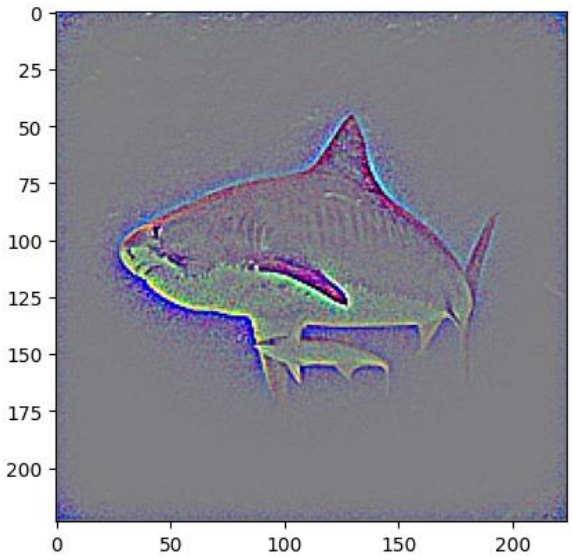
تغییرات کوچک در ورودی‌ها می‌تواند به تغییرات بزرگ در نقشه‌های حساسیت منجر شود. با استفاده از روش SmoothGrad، نقشه‌ها پایدارتر و به تغییرات کوچک حساسیت کمتری دارند.

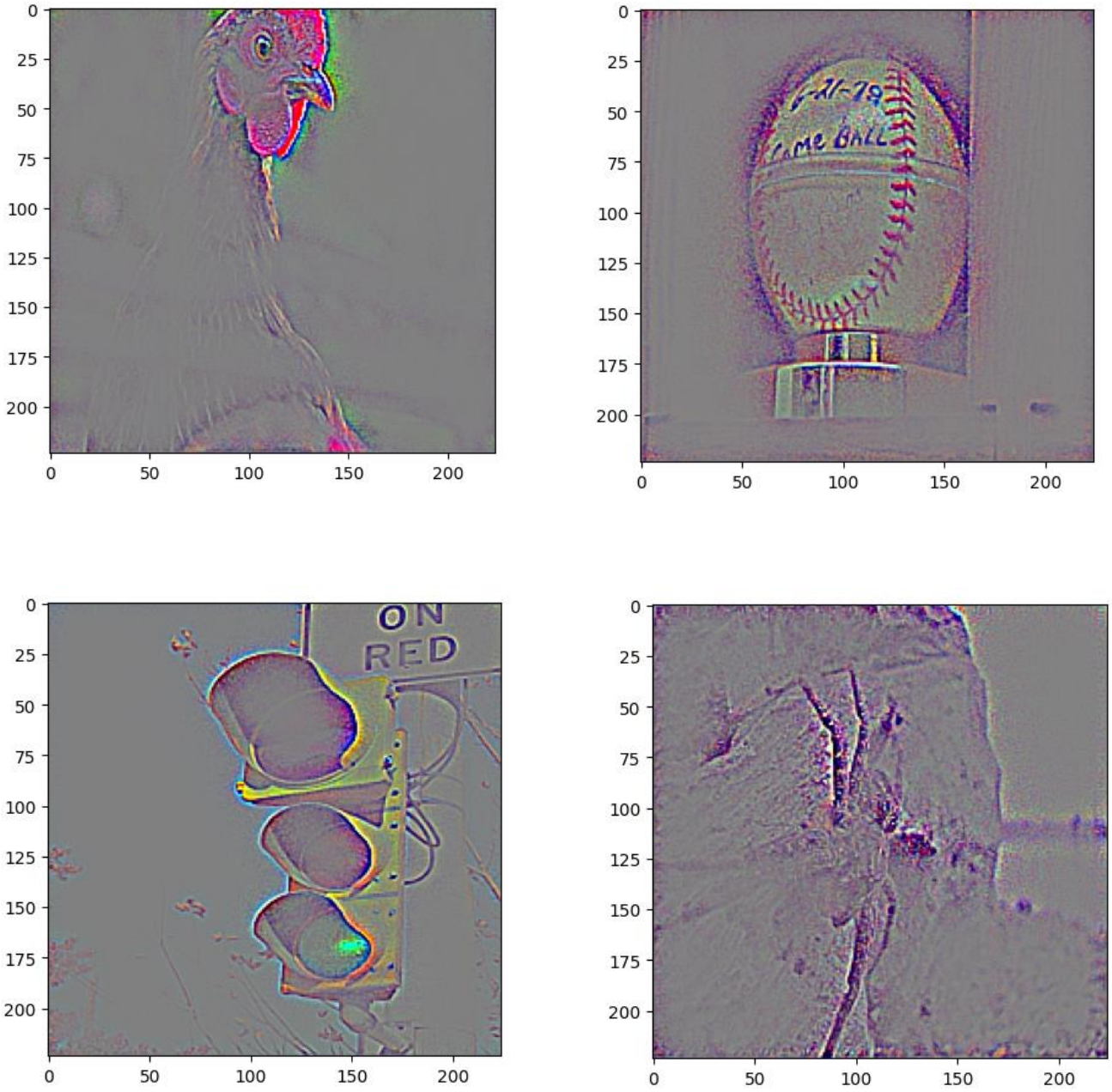
```python
def generate_smooth_grad(guided_bp, tensor, target_class, param_n, param_sigma_multiplier):
    # Generate an empty image/matrix
    smooth_grad = np.zeros(tensor.size()[1:]).transpose(1, 2, 0)
    mean = 0
    sigma = param_sigma_multiplier / (torch.max(tensor) - torch.min(tensor)).item()
    for x in range(param_n):
        # Generate noise
        noise = Variable(tensor.data.new(tensor.size()).normal_(mean, sigma**2))
        # Add noise to the image
        noisy_img = tensor + noise
        # Calculate gradients
        vanilla_grads = guided_bp.visualize(noisy_img, target_class)
        # Add gradients to smooth_grad
        smooth_grad = smooth_grad + vanilla_grads
    # Average it out
    smooth_grad = smooth_grad / param_n
    return smooth_grad
```

شکل ۳۲ – کد پیاده سازی SmoothGrad

طبق کد بالا در هر تکرار نویز گاوسی با میانگین صفر و واریانس سیگما به تصویر ورودی اضافه می شود. تصویر نویزی به مدل داده می شود تا گرادیان های معمولی (vanilla gradients) محاسبه شوند. گرادیان های محاسبه شده به مجموع گرادیان ها اضافه میشود. در نهایت مجموع گرادیان ها بر تعداد تکرار ها تقسیم می شود تا نقشه حساسیت نهایی بدست آید. همچنین در این بخش برای هر ۶ تصویر تعداد تکرار را برابر با ۵۰ در نظر می گیریم و مقدار سیگما را برای تصویر کوسه برابر با ۲ و بقیه تصاویر برابر با ۴ قرار می دهیم که این مقادیر با آزمون و خطا بدست آمده است.

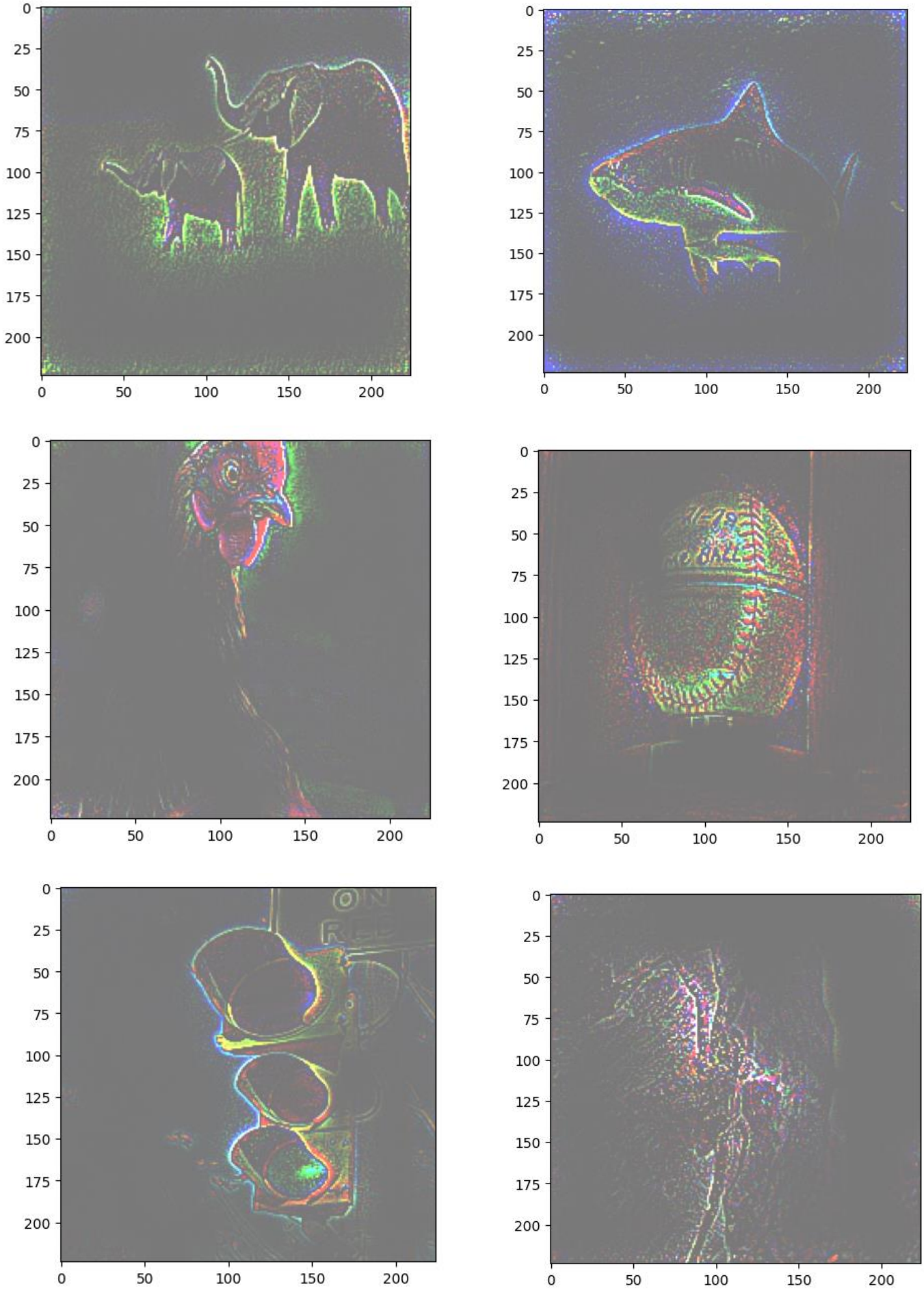






شکل ۳۳ – خروجی روش Smoothed Guided Backpropagation

در ادامه با ضرب درایه به درایه روش GRAD-CAM را نیز با این دو روش ترکیب می کنیم تا بتوانیم تقریبا واضح ترین و بهترین تفسیر را مشاهده کنیم.

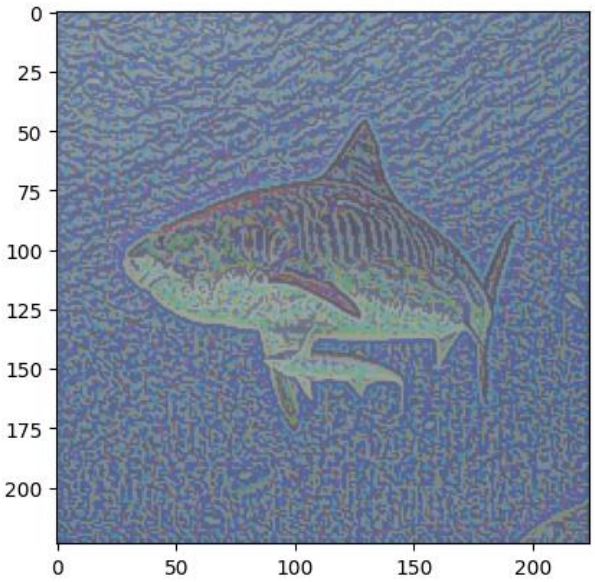


شکل ۳۴ – خروجی روش Smoothed Guided Backpropagation + Grad Cam

Adversarial Perturbation and Pixel Attribution

در این قسمت از عکس کلاس کوسه که در پایین آورده شده است استفاده می کنیم. همچنین قابل ذکر است که از حمله FGSM از کتابخانه Cleverhans با مقدار اپسیلون ۰.۸ استفاده می کنیم.

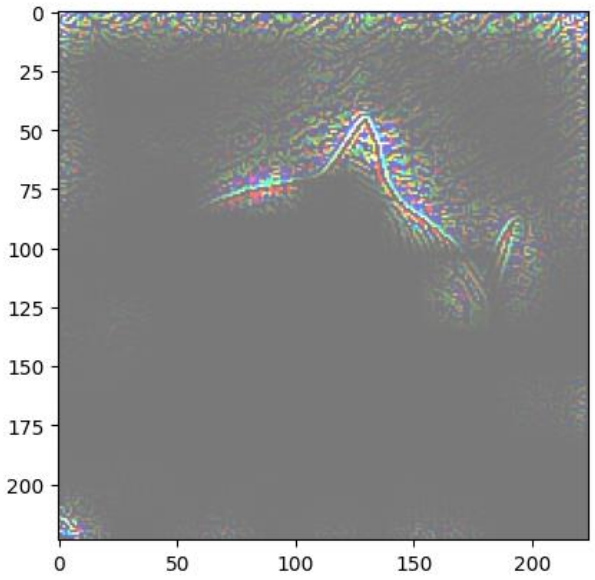
شکل ۳۵ – تصویر نمونه کوسه



شکل ۳۶ – تصویر نمونه کوسه تحت حمله FGSM با مقدار اپسیلون ۰.۸

تصویر بالا را به مدل می دهیم و کلاس خروجی آن برابر با ۶۵ می شود که مطابق با sea snake می باشد.

تصویر حاصل را طبق ترکیب روش Smoothed guided backpropagation و grad cam تفسیر می کنیم و در نهایت تصویر زیر بدست می آید.



شکل ۳۷ – خروجی روش Smoothed Guided Backpropagation + Grad Cam برای نمونه کوسه تحت حمله

طبق تصویر بالا می توانیم ببینیم که بخش اصلی تصویر که مشخص کننده کلاس کوسه است سیاه شده است و مدل برای طبقه بندی این تصویر به آن بخش دقت نمی کند. به همین دلیل کلاس تصویر را اشتباه پیش بینی می کند و در واقع حمله FGSM کارش را درست انجام داده است. اما اگر به تصویر خروجی کوسه در قسمت های قبل دقت کنیم می بینیم که به طور کامل آن بخش از تصویر که کوسه در آن است مشخص است و مدل به آن دقت می کند بنابراین به درستی طبقه بندی را انجام می داد.

Feature Visualization

در این قسمت قصد داریم برای کلاس مرغ در مدل VGG16 از پیش آموزش دیده شده تصویری را پیدا کنیم به صورتی که کلاس پیش بینی شده توسط مدل برای کلاس مرغ بیشینه شود.

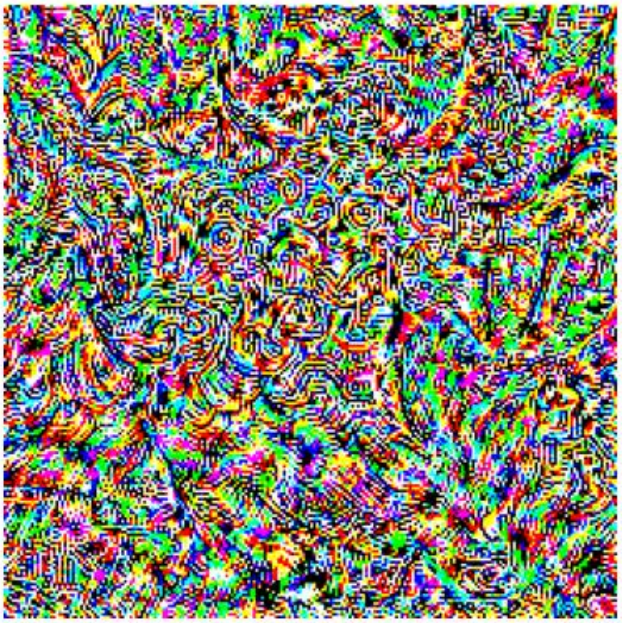
(۱)

```python
model = vgg16(pretrained = True)
# model.load_state_dict(torch.load('vgg16-397923af.pth'))
model.eval()  # Set the model to evaluation mode

# Class index for 'hen' in ImageNet
hen_idx = 8

# Create an initial random image
image = torch.rand(1, 3, 224, 224, requires_grad=True)

# Optimizer
optimizer = torch.optim.Adam([image], lr=0.1)

# Optimization loop
for _ in range(200):  # Number of iterations
    optimizer.zero_grad()
    output = model(image)
    loss = -output[0, hen_idx]  # Negative logit for 'hen'
    loss.backward()
    optimizer.step()

image = image.detach().to('cpu').squeeze().permute(1, 2, 0)
image = torch.clamp(image, 0, 1)
plt.imshow(image)
plt.axis('off')
plt.show()
```

شکل ۳۸ – پیاده سازی روش Activation Maximization برای کلاس مرغ (Hen)

در ابتدا یک تصویر تصادفی با ابعاد ۲۲۴*۲۲۴ ایجاد می‌شود که قابلیت گرادیان‌گیری دارد. سپس حلقه بهینه‌سازی برای ۲۰۰ تکرار انجام می‌شود که در هر تکرار:

۱. گرادیان‌های قبلی صفر می‌شوند.
۲. تصویر ورودی به مدل داده می‌شود و خروجی مدل محاسبه می‌شود.
۳. تابع هزینه (loss) به صورت منفی لگاریتم مربوط به کلاس مرغ تعریف می‌شود تا با به‌روزرسانی تصویر، لگاریتم این کلاس بیشینه شود.
۴. گرادیان‌های تابع هزینه محاسبه شده و تصویر به‌روزرسانی می‌شود.



شکل ۳۹ – خروجی روش Activation Maximization برای کلاس مرغ

۲) همانطور که مشخص است تصویر خروجی، تصویر قابل معنایی برای ما نیست که در ادامه به برخی از دلایل آن اشاره می کنیم:

الگوهای فرکانس بالا: همانطور که میبینیم الگوهای با فرکانس بالا در تصویر تولیدی دیده می شود که بالا بودن فرکانس آنها مانع دیده شدن اطلاعات بامعنا برای ما می شود.

عدم وجود محدودیت ها در بهینه سازی: الگوریتم بهینه‌سازی به دنبال تغییر پیکسل‌ها به گونه‌ای است که کلاس هدف (در اینجا کلاس مرغ) بیشینه شود، اما بدون محدودیت‌های خاصی که تصویر را در یک فضای طبیعی و قابل درک نگه دارد. این باعث می‌شود که الگوریتم به پیکسل‌هایی که بیشترین تأثیر را بر خروجی دارند توجه کند، حتی اگر این پیکسل‌ها در ترکیب با هم یک تصویر طبیعی نسازند.

عدم وجود اطلاعات ساختاری در تصویر اولیه: با شروع از یک تصویر کاملاً تصادفی، الگوریتم بهینه‌سازی ممکن است در مسیرهایی حرکت کند که به تصاویر نویزی و غیر طبیعی منجر شود، زیرا هیچ اطلاعات ساختاری یا محتوایی در تصویر اولیه وجود ندارد.

۳) Total Variation (TV) Regularization یکی از تکنیک‌های محبوب در پردازش تصویر است که برای کاهش نویز در تصاویر استفاده می‌شود و در عین حال جزئیات و لبه‌های مهم تصویر را حفظ می‌کند. این روش به خصوص برای تولید تصاویر با وضوح و معنای بهتر بسیار مفید است.

Total variance regularization به صورت زیر تعریف می شود:

$$TV(y) = \sum_{i,j} (y_{i+1,j} - y_{i,j})^2 + (y_{i,j+1} - y_{i,j})^2$$

دلیل کمک‌کننده بودن TV Regularization و Random Shift

TV Regularization:

TV Regularization با کاهش تغییرات ناگهانی در پیکسل‌های همجوار، نویز را کاهش می‌دهد. این باعث می‌شود که تصویر نهایی صاف‌تر و طبیعی‌تر به نظر برسد. برخلاف فیلترهای ساده میانگین‌گیری که ممکن است لبه‌ها و جزئیات تصویر را هموار کنند، TV Regularization به طور خاص طراحی شده است که لبه‌ها را حفظ کند. این ویژگی به تولید تصاویری با جزئیات بهتر و معنای بیشتر کمک می‌کند.

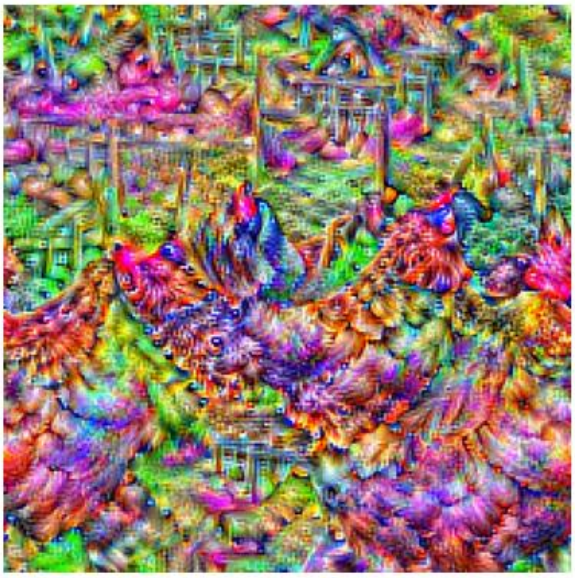
Random Shift:

با جابجا کردن تصادفی پیکسل‌ها در هر forward pass، مدل از گیر افتادن در مینیمم‌های محلی که ممکن است تصاویر نویزی و غیرطبیعی ایجاد کنند، جلوگیری می‌کند. این کمک می‌کند تا تصویر بهینه‌سازی شده به یک فضای جستجوی گسترده‌تری دست یابد. جابجا کردن تصادفی پیکسل‌ها باعث می‌شود که هر بار تصویر متفاوتی به مدل داده شود. این تنوع باعث می‌شود که بهینه‌سازی مدل به یک تصویر معنای بیشتری دست یابد که نه تنها برای کلاس هدف بهینه شده بلکه ویژگی‌های طبیعی‌تری دارد.

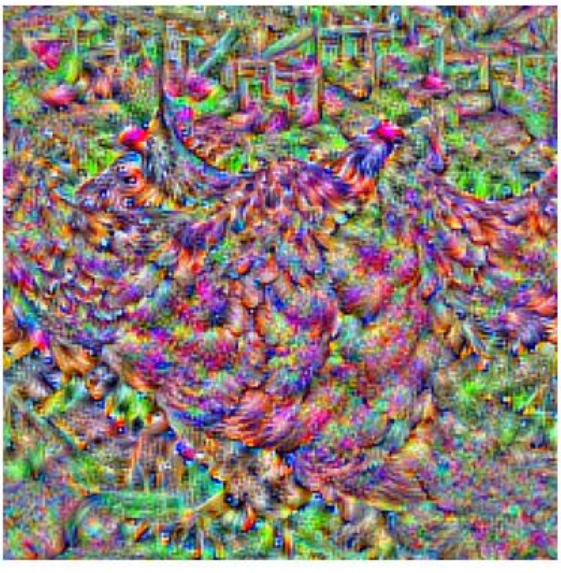
۴)

برای تولید تصویر خروجی ۳ پارامتر داریم که می توانیم آنها را تغییر دهیم. تعداد تکرار مرحله بهینه سازی - ضریب TV Regularization – ابتدا و انتهای بازه عدد رندوم برای شیفت

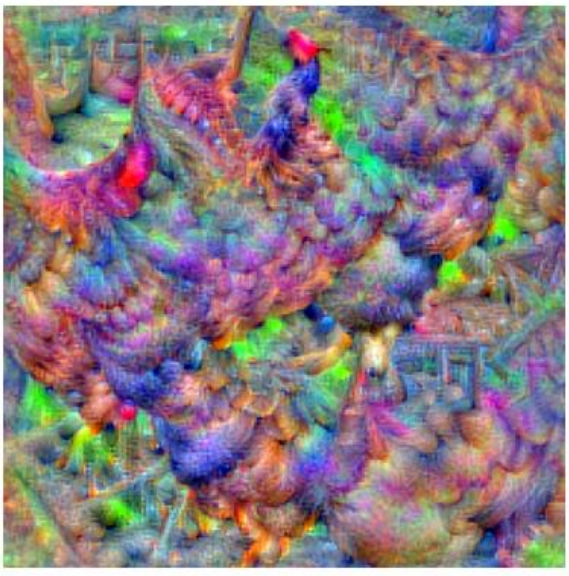
شکل ۴۰ – خروجی با ضریب ۰.۰۱، شیفت بین ۱۰ و ۱۰-، تعداد تکرار ۲۰۰



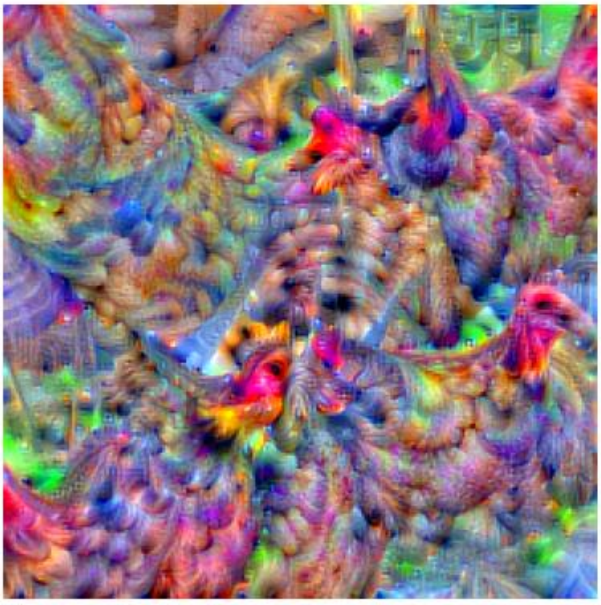
شکل ۴۱ – خروجی با ضریب ۰.۰۱، شیفت بین ۱۰ و ۱۰-، تعداد تکرار ۳۰۰



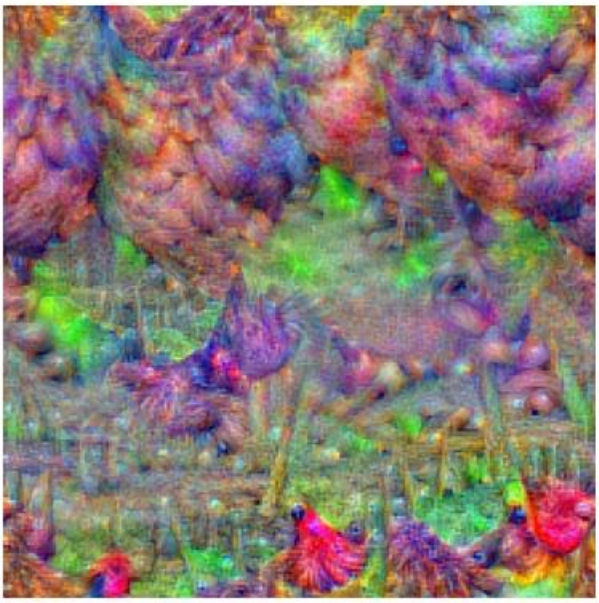
شکل ۴۲ – خروجی با ضریب ۰.۰۱، شیفت بین ۳۰ و ۳۰-، تعداد تکرار ۲۰۰



شکل ۴۳ – خروجی با ضریب ۰.۱، شیفت بین ۱۰ و ۱۰-، تعداد تکرار ۲۰۰



شکل ۴۴ – خروجی با ضریب ۰.۱، بدون شیفت، تعداد تکرار ۲۰۰



شکل ۴۵ – خروجی با ضریب ۰.۱، شیفت بین ۱۰ و ۱۰-، تعداد تکرار ۵۰۰

همانطور که در تصاویر بالا می بینیم، می توان در هر تصویر تعدادی مرغ به طور واضح پیدا کرد.

مراجع

1. https://github.com/AmrMKayid/nam

2. https://github.com/lethaiq/GRACE_KDD20

3.https://fa.wikipedia.org/wiki/%D9%87%D9%88%D8%B4_%D9%85%D8%B5%D9%86%D9%88%D8%B9%DB%8C_%D9%82%D8%A7%D8%A8%D9%84_%D8%AA%D9%88%D8%B6%DB%8C%D8%AD

4. https://en.wikipedia.org/wiki/Total_variation_denoising