به نام خدا

1928
K. N. Toosi University of Technology

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

مبانی سیستم های هوشمند

[شهاب مقدادی نیشابوری]

[۴۰۰۰۹۴۴۳]

استاد : آقای دکتر مهدی علیاری

دی ۱۴۰۳

فهرست مطالب

عنوان شماره صفحه

سوال اول

https://drive.google.com/file/d/1WzZx-UP_kLm_GvJ7PRR38O9iOtsw79nh/view?usp=sharing

بخش ۱

اگر در یک مساله طبقه بندی دو کلاسه، دو لایه اخر شبکه به ترتیب ReLU و sigmoid باشند، میدانیم که خروجی ReLU عددی صفر یا مثبت است، بنابراین ورودی لایه آخر عددی صفر یا مثبت خواهد بود. پس خروجی لایه اخر که activation function آن sigmoid است، حتما بزرگتر از ۰.۵ شده و و بین ۰.۵ تا ۱ قرار خواهند گرفت. بنابراین اگر فرض کنیم که کلاس های مد نظر ما دو کلاس ۰ و ۱ باشند، تمام خروجی ها همواره در کلاس ۱ طبقه بندی خواهند شد.

بخش ۲

مزیت استفاده از تابع ELU به جای ReLU، آن است که این تابع در نقاط منفی مشتق ۰ ندارد، بنابراین در هنگام Back Propagation، برخلاف تابع ReLU، تابع ELU امکان گیر افتادن در یک نقطه را(به دلیل ۰ شدن مشتق) ندارد.

$$\frac{d}{dx}\mathrm{ELU}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ \alpha e^x & \text{if } x < 0 \end{cases}$$

بخش ۳

در این بخش از ما خواسته شده مقادیر داخل یک مثلث را از بقیه صفحه جدا کنیم، برای این کار از سه نورون McCulloch-Pitts استفاده میکنیم.

هر یک از این نورون ها، معادله خط یکی از اضلاع مثلث را شامل میشوند که با قرار دادن نقطه مورد نظر در معادله خط و با مشاهده مثبت یا منفی شدن نتیجه، میتوان فهمید که نقطه در کدام سمت خط قرار دارد. ۳ نورون شامل معادله خطوط زیر میباشد:

$y = 0$

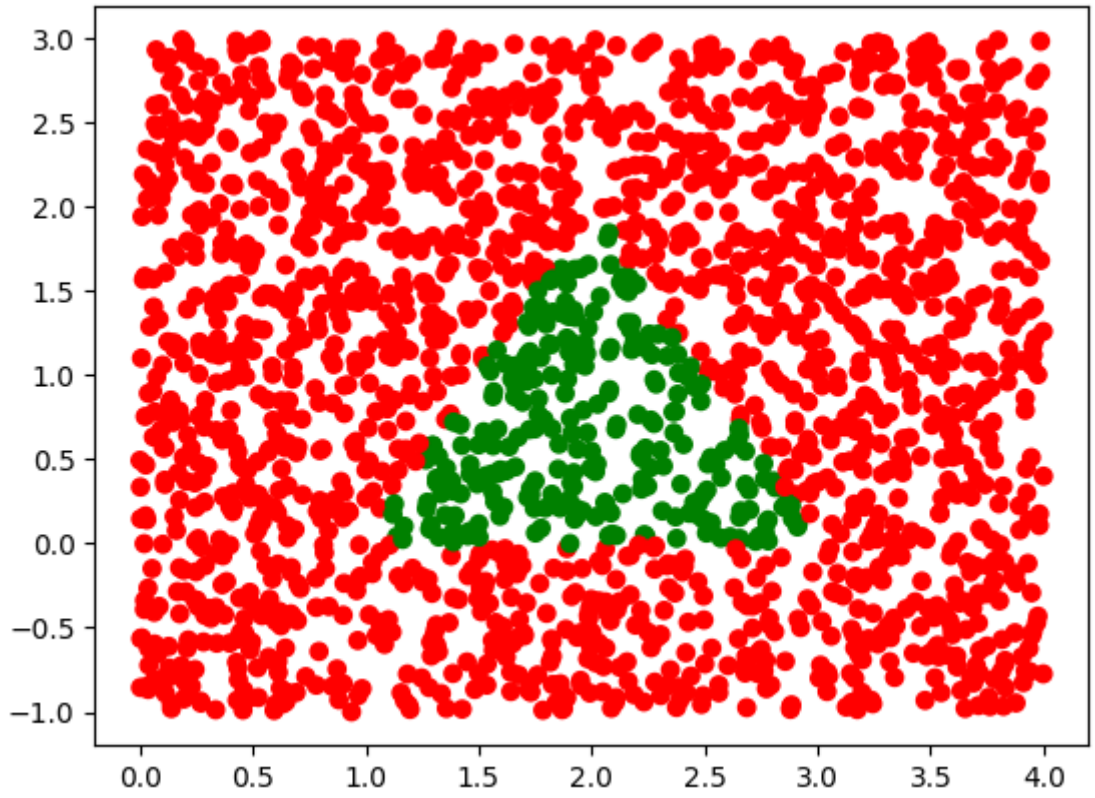
$2x - y - 2 = 0$

$-2x - y + 6 = 0$

که تنها در حالتی نقطه داخل مثلث است، که در صورتی که نقطه را در تمامی معادله های بالا جایگزاری کنیم، حاصل تمامی آنها عددی مثبت (یا ۰) شود.(که در این حالت خروجی تمام نورون های ما برابر با ۱ خواهد شد)

در نهایت نیز، یک نورون برای لایه خروجی اضافه میکنیم، این نرون دارای ۳ ورودی که از خروجی نرون های قبلی با وزن های ۱، و بایاس منفی سه میباشد، اگر خروجی این نورون ۱ باشد، به آن معناست که نقطه داخل مثلث قرار دارد و در غیر این صورت، نقطه در خارج مثلث میباشد.

تابع فعال ساز دیگری نیز نمیتوان به این مدل نورون اضافه کرد چرا که اساس کار نورون McCulloch-Pitts آن است که بردار ورودی را در بردار وزن ها ضرب کرده و حاصل را با bias جمع کند، اگر این عدد کوچک تر از حدی مشخص بود ۰ و در غیر این صورت ۱ خروجی دهد. بنابراین از تابع فعال ساز دیگری نمیتوان استفاده کرد.

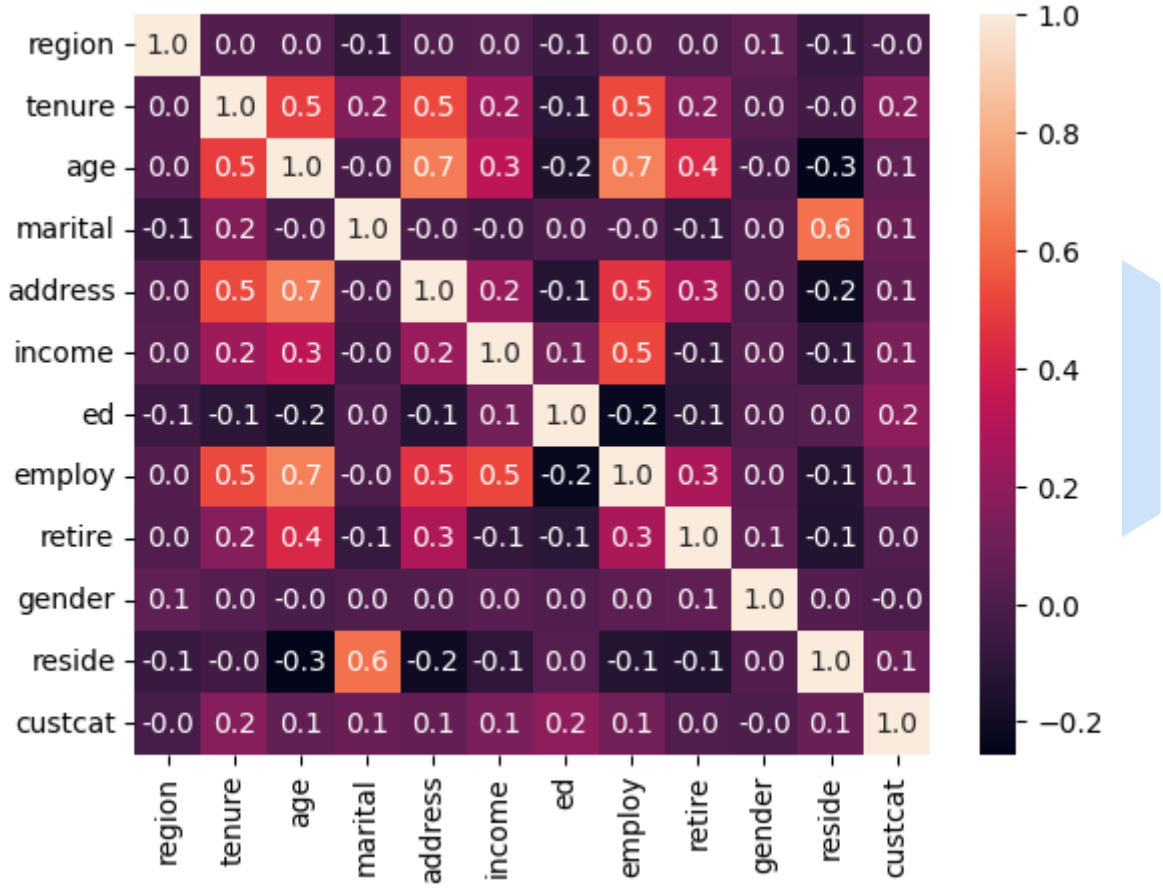


سوال دوم

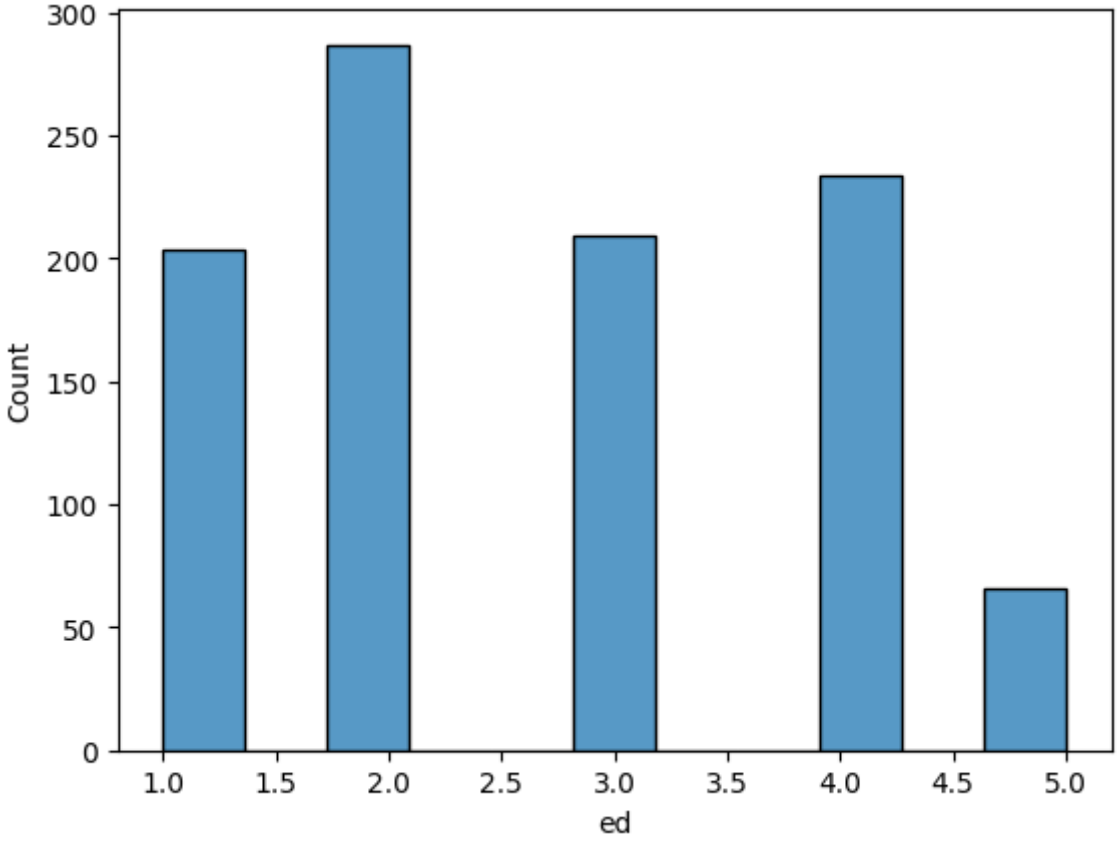
https://drive.google.com/file/d/14Z6P2zVTTFq07uYuUWJS9Ph30BXUAAMs/view?usp=sharing

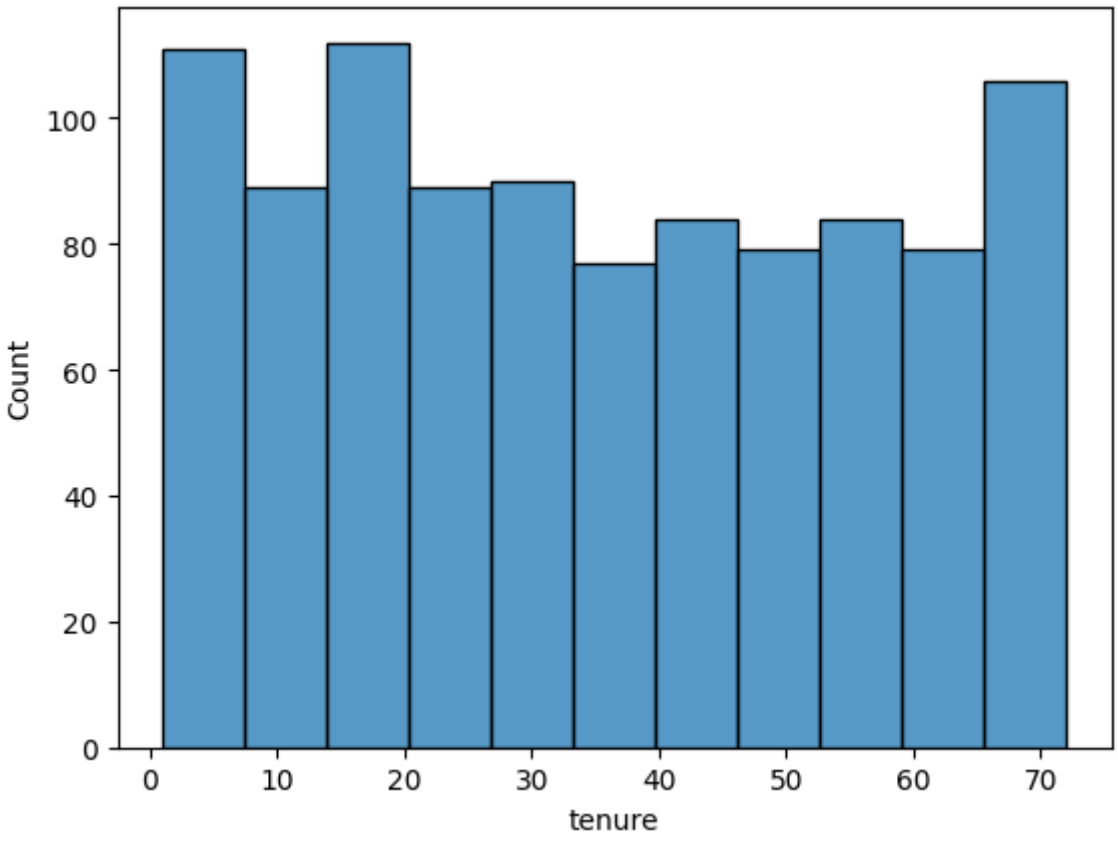
بخش ۱ و ۲

ابتدا با استفاده از pandas، فایل csv اطلاعات را به محیط پایتون وارد میکنیم(مشاهده میکنیم که اطلاعات حاوی ۱۲ ستون بوده که یکی از این ۱۲ ستون خروجی مد نظر برای classification است) و سپس طبق خواسته سوال و با استفاده از کتابخانه seaborm، heatmap داده ها را رسم میکنیم:



مشاهده میشود که دو ستون ed و tenure بیشترین همبستگی را با فیلد هدف دارند، بنابراین هیستوگرام این دو فیچر را رسم میکنیم:


300
250
200
150
100
50
0
Count
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
ed


100
80
60
40
20
0
Count
0
10
20
30
40
50
60
70
tenure

بخش ۳

سپس داده ها را به نسبت ۰.۸ برای آموزش مدل، ۰.۱۵ برای تست مدل و ۰.۰۵ برای راستی آزمایی مدل تقسیم بندی میکنیم و با کمک کتابخانه sklearn، داده ها را با روش MinMaxScaler نرمالایز میکنیم.

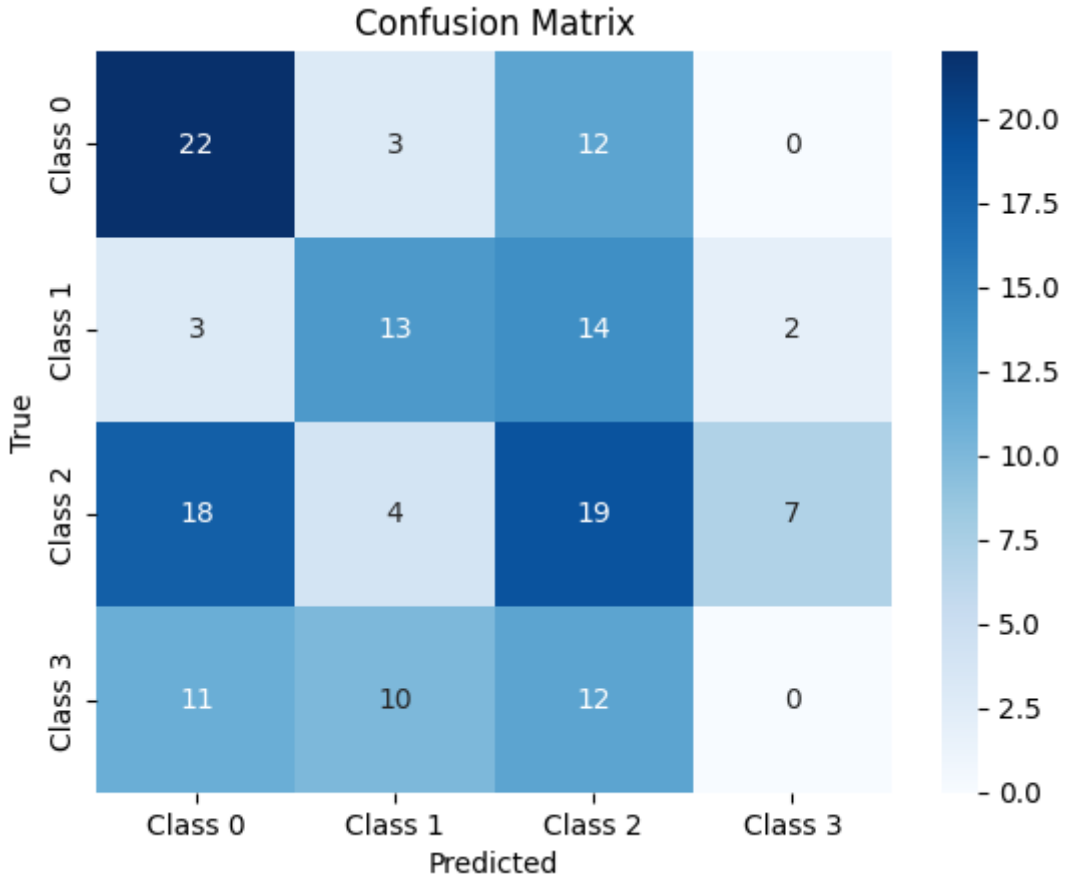
بخش ۴

در مرحله بعدی دو مدل برای classification طراحی میکنیم، که یکی از آنها دارای ۲ لایه پنهان و دیگری دارای ۱ لایه پنهان میباشد(در همه حالات، اندازه batch را برابر با ۴۰۰ و تعداد epoch را برابر با ۲۰۰ عدد درنظر میگیریم). از آنجا که تعداد خروجی های ممکن عدد ۴ است، بنابرین در لایه خروجی ۴ نورون softmax قرار میدهیم و از تابع هزینه CategoricalCrossentropy استفاده میکنیم و بقیه لایه ها را به صورت زیر قرار میدهیم:

برای شبکه با ۲ لایه پنهان:

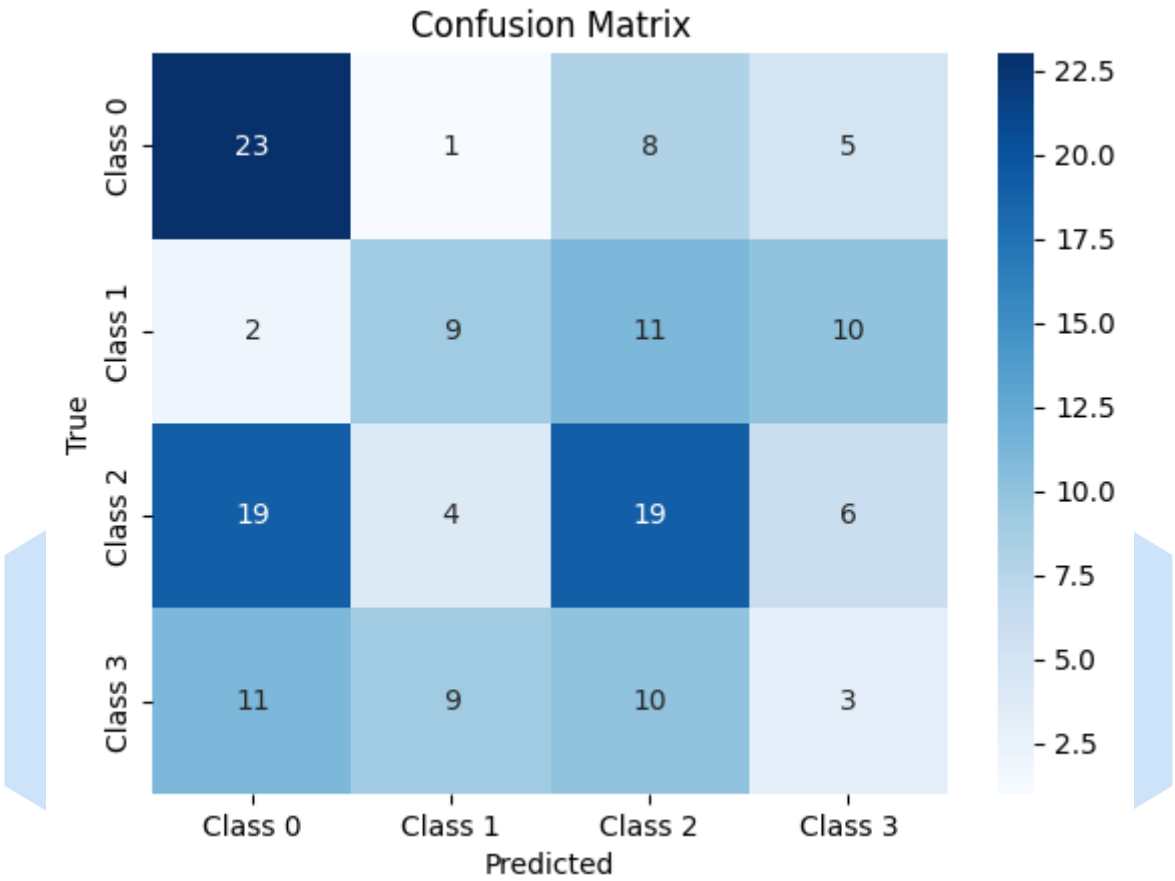
- ۱۰ نورون با تابع فعال ساز linear برای لایه ورودی
- ۸ نورون با تابع فعال ساز linear برای لایه پنهان اول
- ۶ نورون با تابع فعال ساز linear برای لایه پنهان دوم

در این مدل، accuracy برابر با ۰.۳۶ شده و همانطور که مشاهده میشود نتایج این مدل، دارای صحت قابل قبولی نیستند:



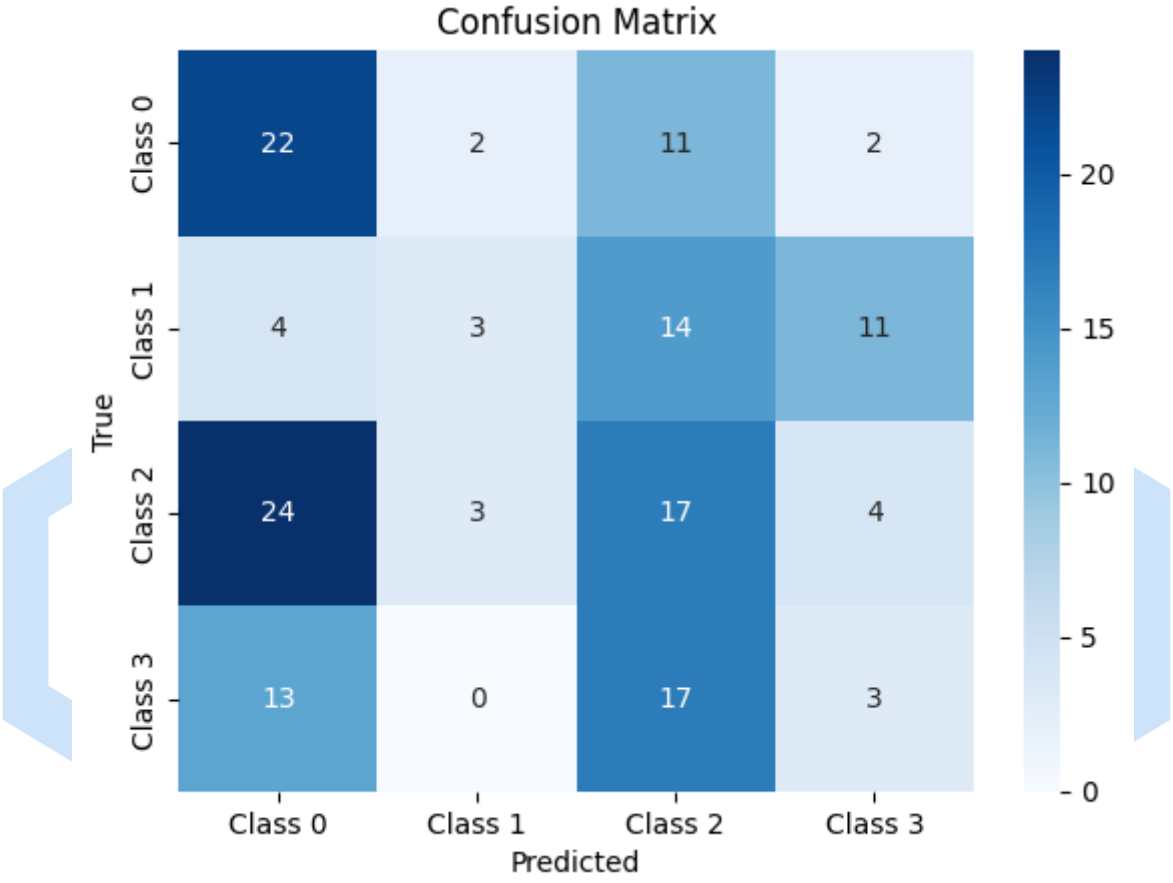
حال تعداد نرون های همه لایه ها بجز لایه خروجی را دو برابر کرده و نتایج را دوباره بررسی میکنیم.

پس از دو برابر کردن تعداد نورون ها، مقدار accuracy ثابت (۰.۳۶) باقی مانده و confusion-matrix- به شکل زیر خواهد بود:

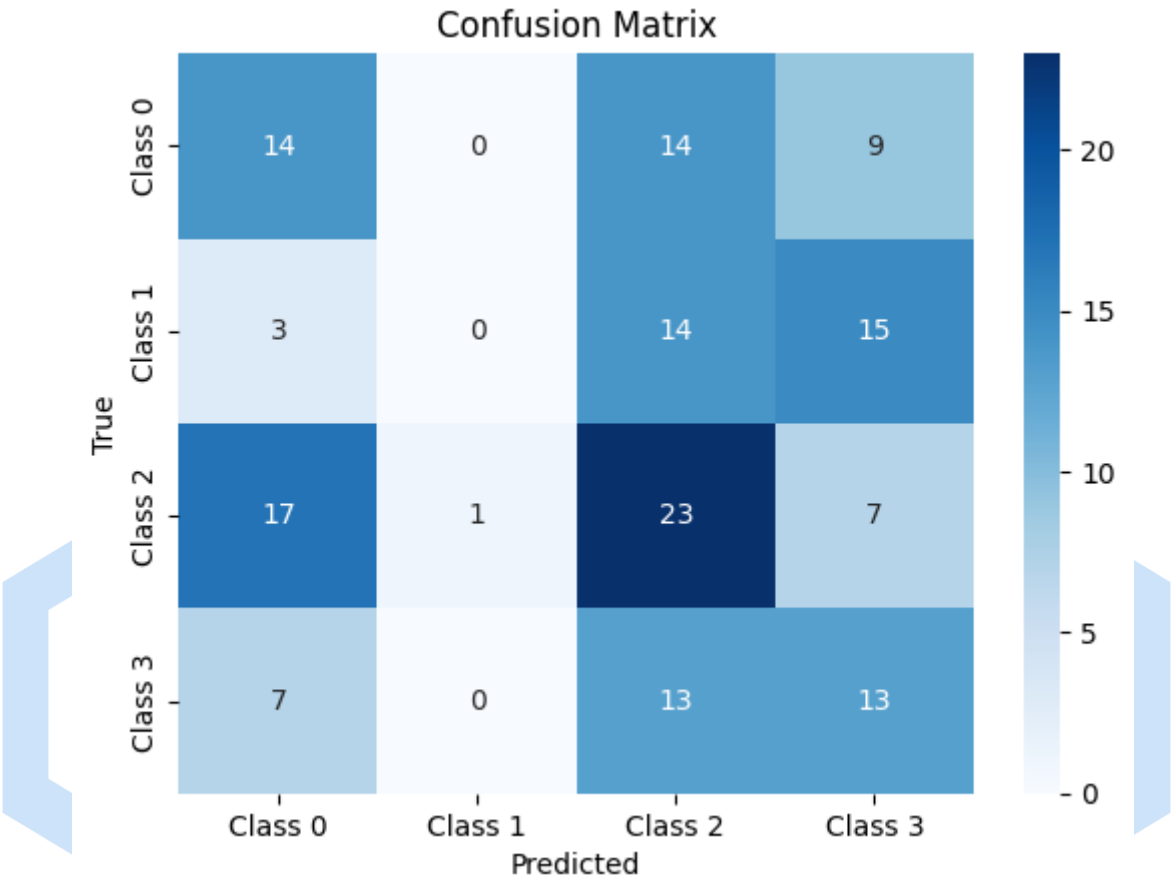


حال برای مدل دوم همین کار را تکرار میکنیم، تعداد نورون های لایه خروجی را برابر با ۱۰ نگه داشته و لایه پنهان دوم را حذف میکنیم(مدل تنها شامل یک لایه پنهان با تعداد نورون ۸ میباشد).

در این حالت مشاهده میشود که accuracy تا مقدار ۰.۳۰ کم شده و confusion matrix به شکل زیر تبدیل خواهد شد:

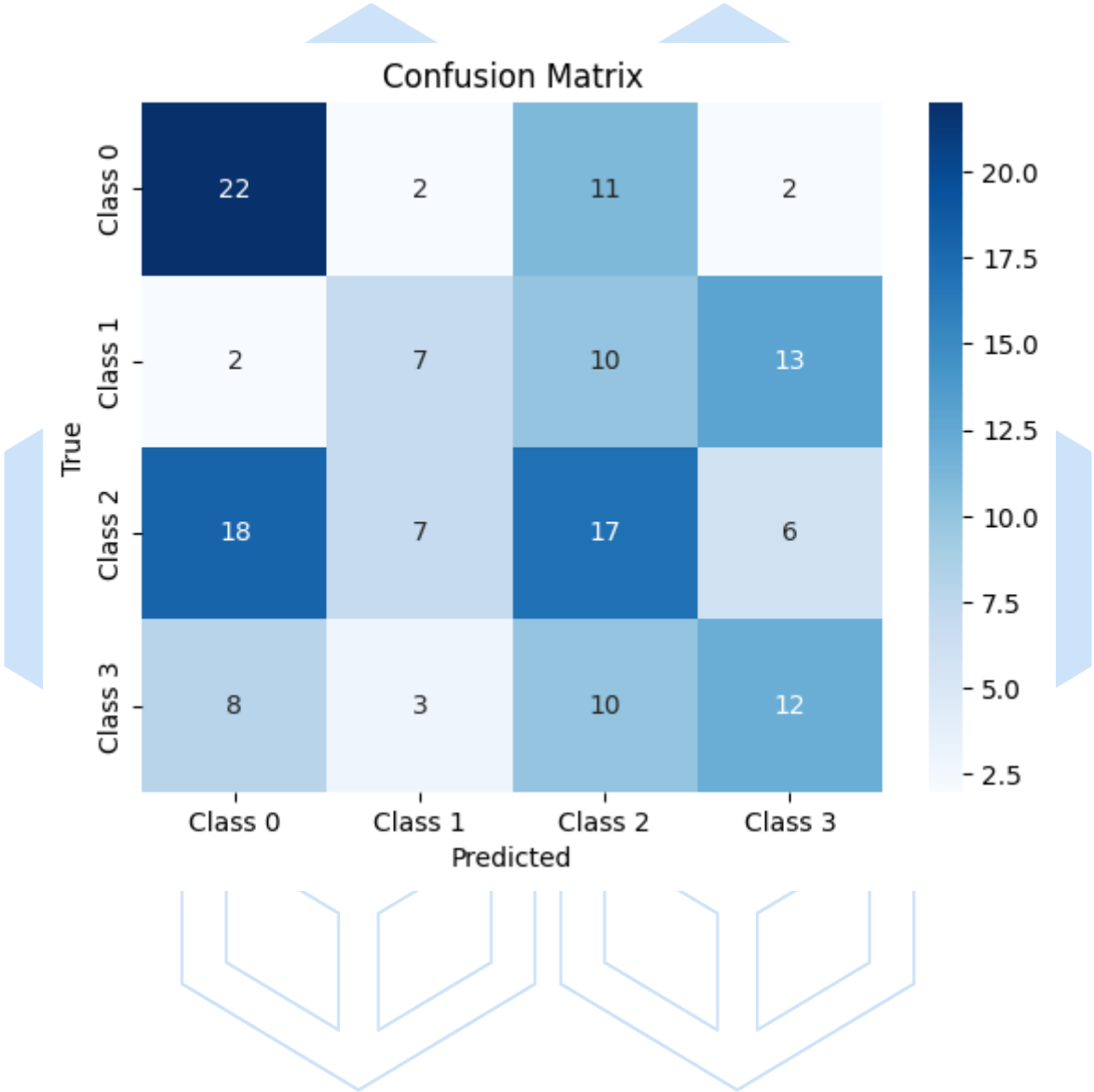


سپس تعداد نورون های لایه ورودی را به مقدار ۲۰، و تعداد نورون های لایه پنهان را به مقدار ۱۶ میرسانیم. در این حالت accuracy برابر با ۰.۳۳۳۳ شده و confusion matrix به شکل زیر تبدیل خواهد شد:

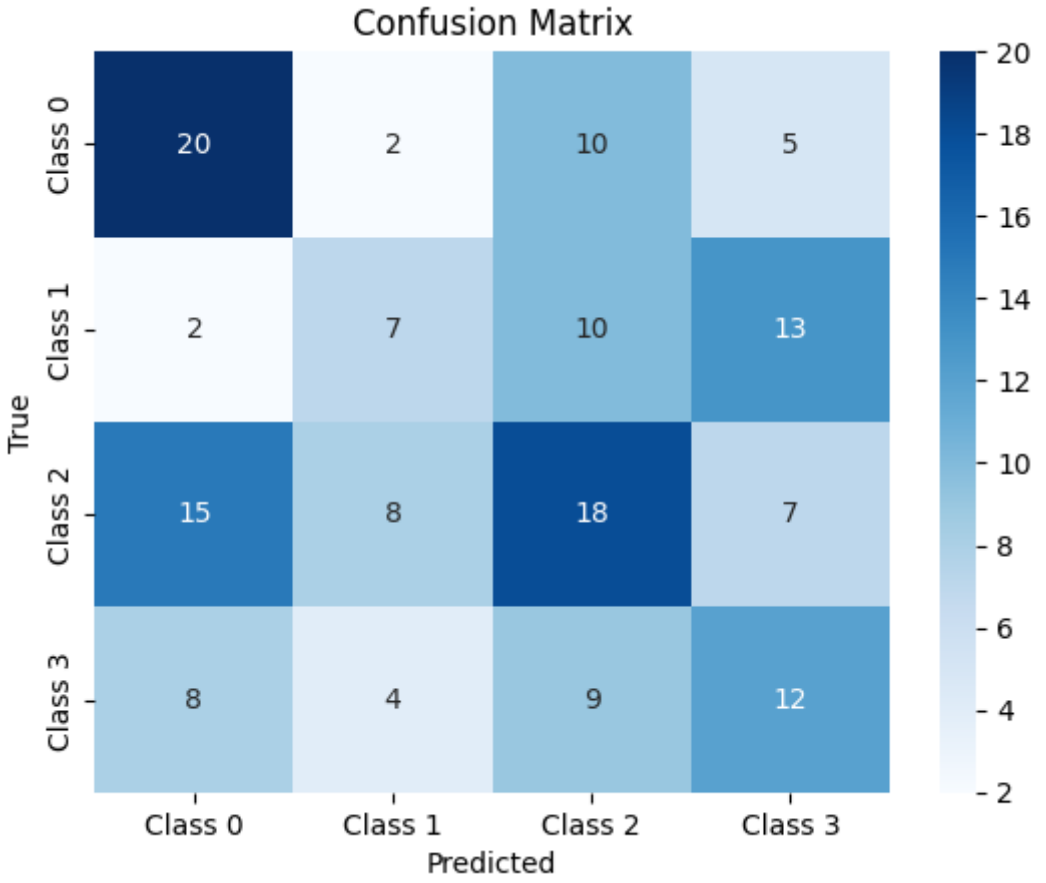


در مرحله بعد، لایه batch normalization را به مدل اضافه میکنیم(لازم به ذکر است که شبکه هایی که در قسمت قبلی بهترین خروجی ها را گرفتند در این بخش استفاده شده اند).

برای شبکه با ۲ لایه پنهان، لایه batch normalization را بعد از لایه ورودی اضافه میکنیم. مشاهده میکنیم که مقدار accuracy تا ۰.۴۰ زیاد شده و ماتریس درهم ریختگی به شکل زیر تبدیل شد:



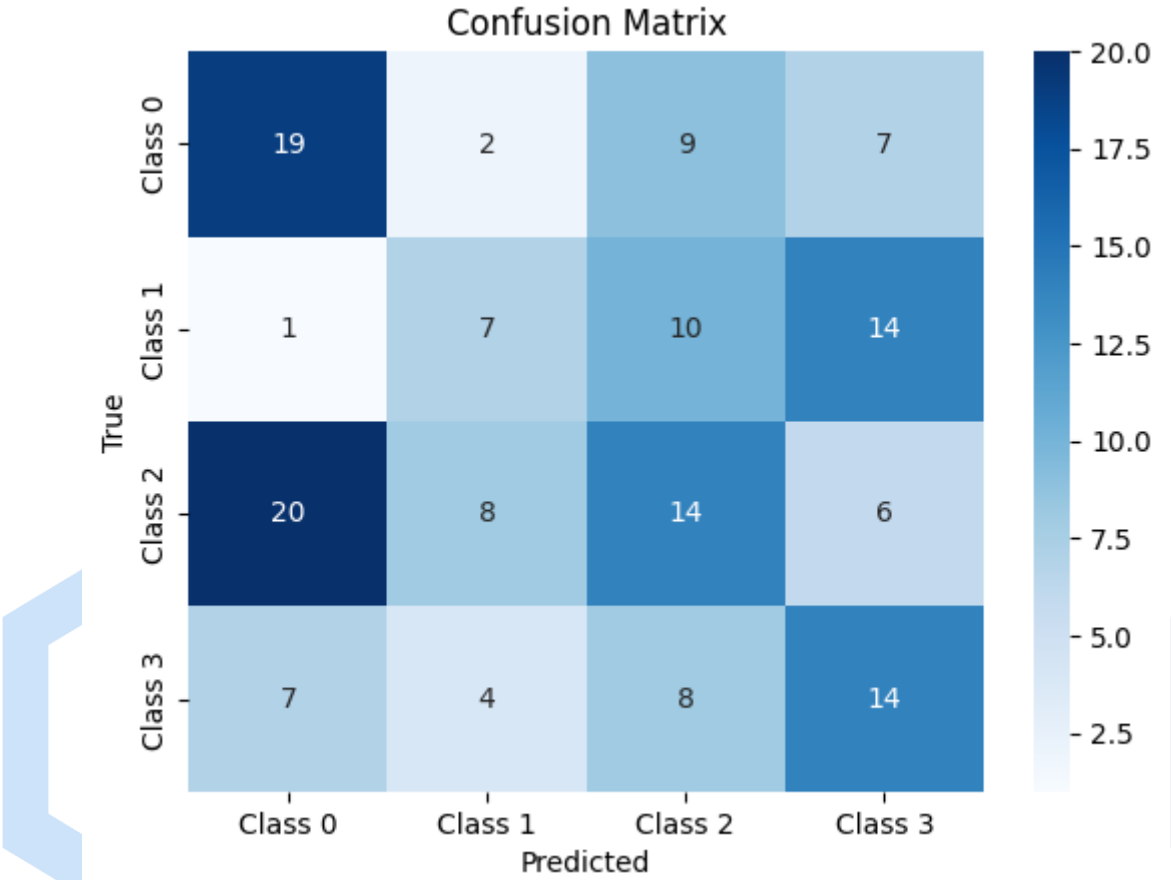
برای شبکه با ۱ لایه پنهان نیز accuracy برابر با ۰.۳۸ به دست آمد و ماتریس درهم ریختگی نیز به شکل زیر تبدیل شد:



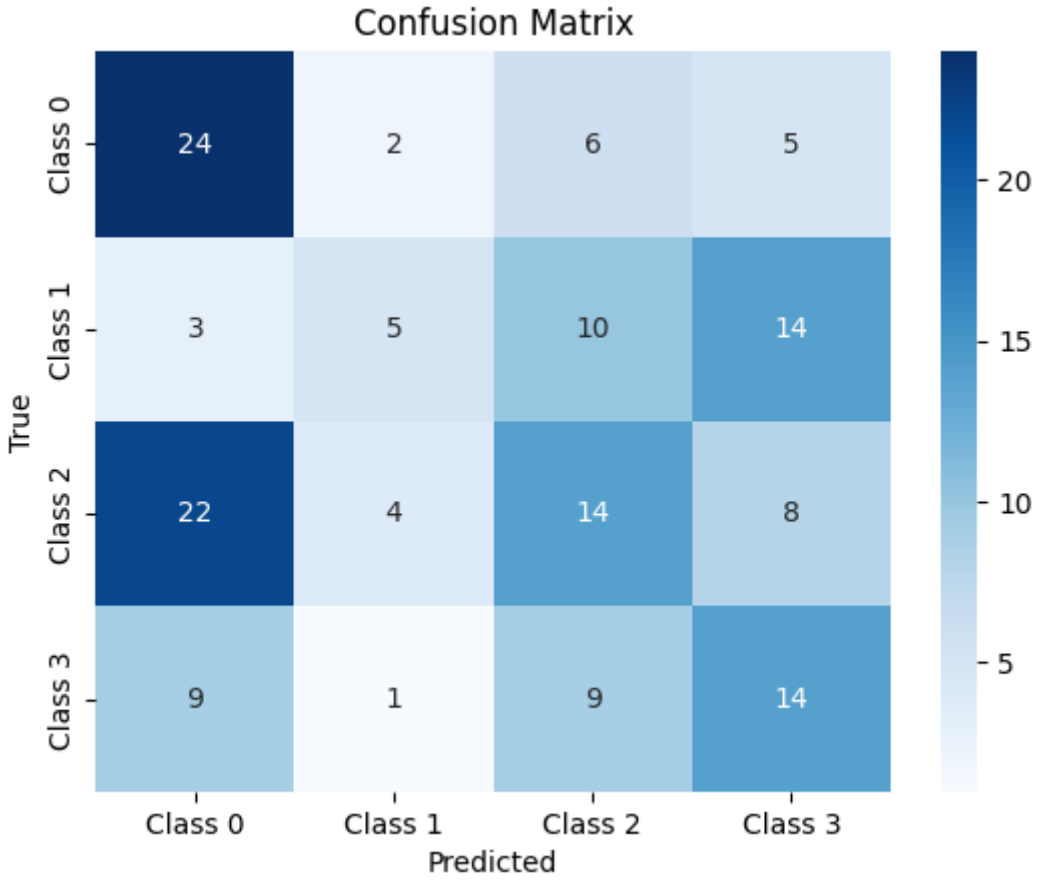
مشاهده میشود که در هر دو شبکه، اضافه کردن لایه batch normalization به بهتر شدن خروجی کمک کرد.

حال به هر دو مدل، بعد از هر لایه پنهان یک لایه DropOut با ضریب ۰.۲۵ اضافه میکنیم:

برای مدل با ۲ لایه پنهان accuracy برابر با ۰.۳۶ به دست می آید که از نتیجه مرحله قبلی بدتر است.

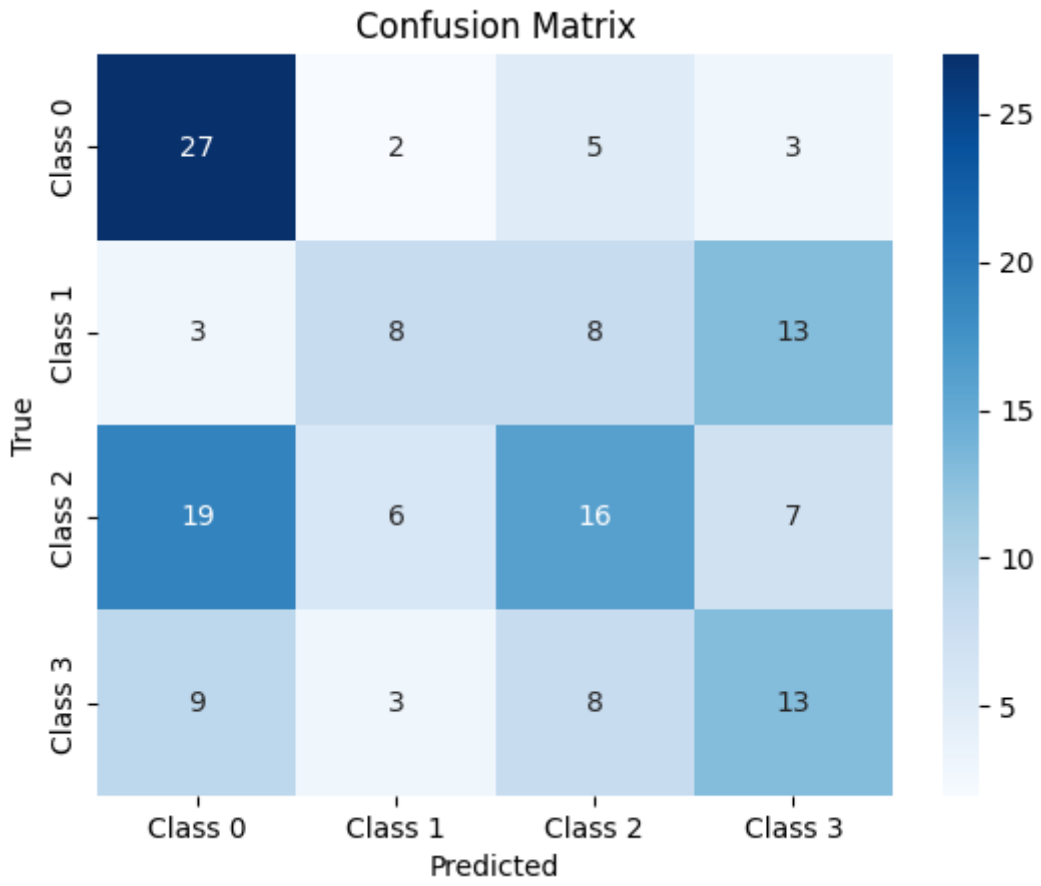


برای مدل با ۱ لایه پنهان نیز accuracy برابر با ۰.۳۸ شده که تفاوتی با حالت قبل ندارد:

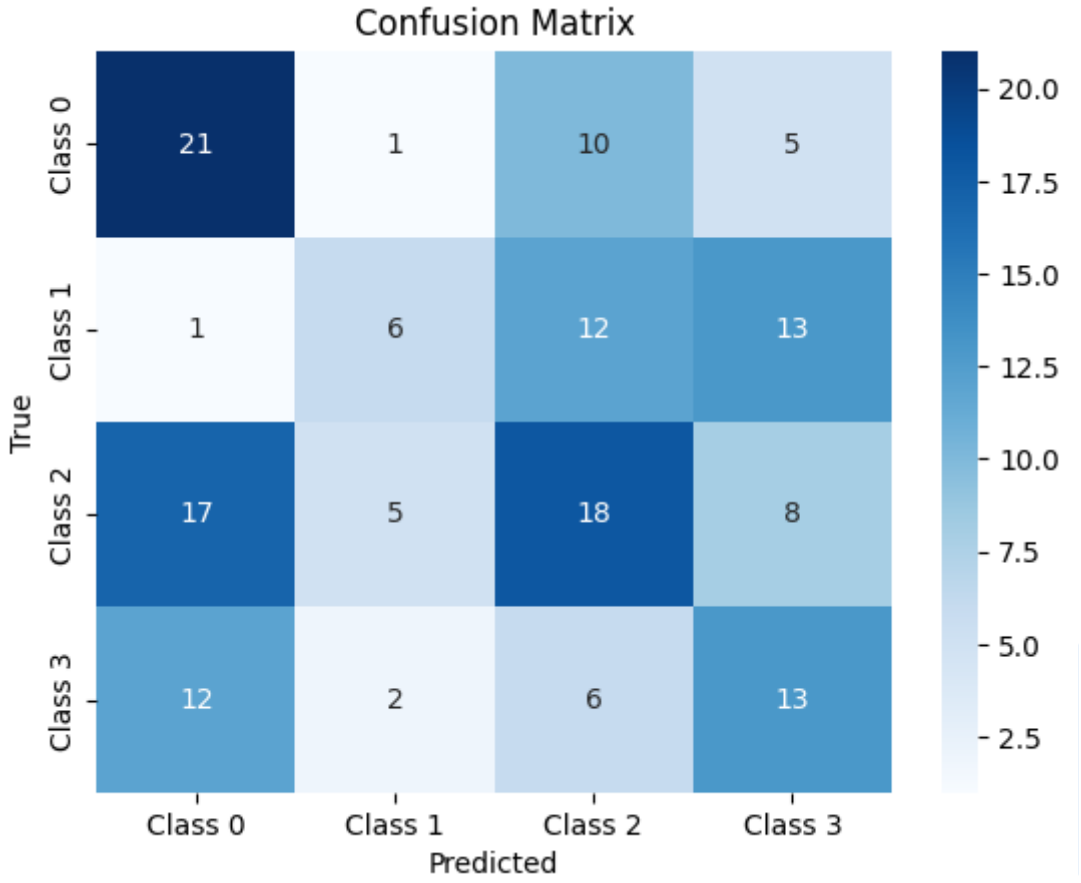


سپس در مرحله بعد، به اضافه کردن L2-regularization به لایه های پنهان میپردازیم، لازم به ذکر است که با توجه به آن که DropOut تاثیر مثبتی روی دقت نداشت، از این روش در حل این بخش استفاده نمیکنیم.

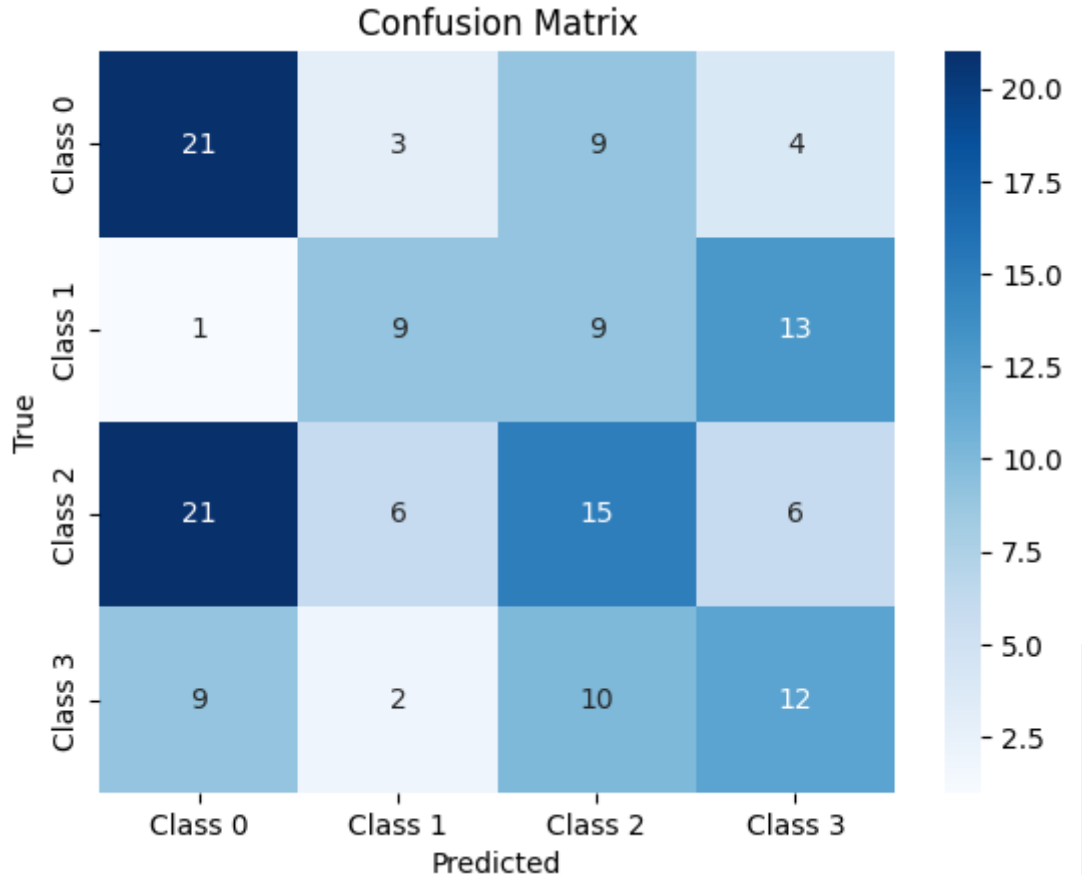
برای شبکه با ۲ لایه پنهان، accuracy به ۰.۴۳ افزایش یافت و ماتریس درهم ریختگی به شکل زیر تبدیل شد:



برای شبکه با ۱ لایه پنهان، accuracy به ۰.۳۸۶۷ افزایش یافت و ماتریس درهم ریختگی به شکل زیر تبدیل شد:



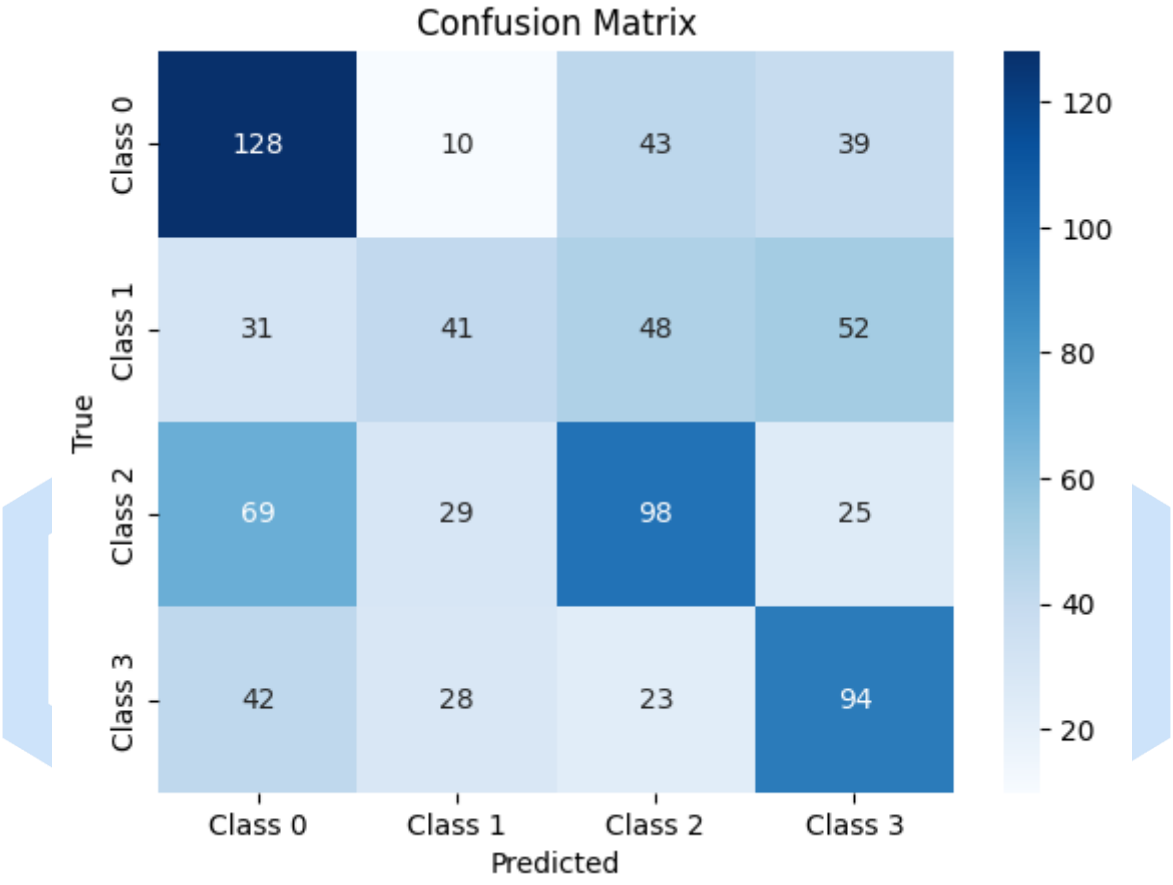
در مرحله اخر، با استفاده از optimizer RMSprop، accuracy به ۰.۳۸ کاهش یافت و ماتریس درهمریختگی به شکل زیر تبدیل شد:



بخش ۵

بهترین مدل به دست آمده، مربوط به مدل با ۲ لایه پنهان با تعداد نورون زیاد و L2-Regularization میباشد. در این مرحله دقت این مدل را بر روی داده های train گزارش میکنیم:

در این حالت دقت مدل برابر با ۰.۴۵ به دست آمده و ماتریس درهم ریختگی به شکل زیر در می آید:




```
predicted value: 0 | actuall value: 1
predicted value: 3 | actuall value: 3
predicted value: 3 | actuall value: 0
predicted value: 3 | actuall value: 3
predicted value: 3 | actuall value: 3
predicted value: 0 | actuall value: 0
predicted value: 3 | actuall value: 3
predicted value: 0 | actuall value: 2
predicted value: 2 | actuall value: 0
predicted value: 1 | actuall value: 1
```

بخش ۶

بهترین نتیجه مشاهده شده، همانطور که در قسمت قبل اشاره شد، مربوط به مدل با ۲ لایه پنهان با تعداد نورون زیاد و L2-Regularization میباشد.

اضافه کردن لایه‌ی **Batch Normalization** به مدل باعث بهبود نتایج شد، زیرا این تکنیک با نرمال‌سازی خروجی لایه‌های میانی، شبکه را نسبت به تغییرات کوچک در وزن‌ها پایدارتر کرده و روند همگرایی را سرعت بخشیده است. همچنین، این روش حساسیت مدل به مقداردهی اولیه‌ی وزن‌ها را کاهش داده و به عنوان یک مکانیزم منظم‌کننده عمل کرده است که در نهایت احتمال **overfitting** را کم می‌کند. به دلیل این ویژگی‌ها، مدل توانسته است ویژگی‌های داده‌ها را به‌صورت مؤثرتری یاد بگیرد و عملکرد بهتری روی مجموعه داده‌های ارزیابی ارائه دهد.

علاوه بر این، استفاده از **L2-Regularization** نیز نتایج را بهبود داده است، زیرا این روش با اضافه کردن جریمه به مقادیر loss ، از بزرگ شدن بیش از حد وزن‌ها جلوگیری کرده و شبکه را ساده‌تر و متعادل‌تر کرده است. این امر باعث شده تا مدل وابستگی کمتری به برخی از ویژگی‌ها داشته باشد و در نتیجه تعمیم‌پذیری آن افزایش یابد. ترکیب این دو تکنیک باعث کاهش **overfitting** ، بهبود تعمیم‌پذیری و افزایش پایداری مدل شده است. در مقابل، استفاده از **DropOut** در این مدل تأثیر مثبتی بر خروجی نداشته است و حتی در برخی موارد به دلیل حذف تصادفی نرون‌ها باعث کاهش کارایی مدل شده است.

سوال سوم

https://drive.google.com/file/d/1tpdHS0RpABZ4zBjxbv8RCX_o_szYdi0P/view?usp=sharing

بخش ۱

تابع اول، یک تصویر ورودی را به نمایش دودویی (باینری) تبدیل می‌کند که در آن هر پیکسل به‌صورت سفید (با مقدار ۱-) یا سیاه (با مقدار ۱) طبقه‌بندی می‌شود. ابتدا تصویر باز شده و ابعاد آن تعیین می‌شود، سپس با استفاده از ابزار دستکاری تصویر (ImageDraw) پیکسل‌ها پردازش می‌شوند. مجموع شدت رنگ‌های قرمز، سبز و آبی (RGB) برای هر پیکسل محاسبه می‌شود و اگر این مقدار از یک آستانه مشخص بیشتر باشد، پیکسل به سفید و در غیر این صورت به سیاه تبدیل می‌شود. نتیجه این فرایند به‌صورت یک لیست از مقادیر ۱- و ۱ بازگردانده می‌شود.

تابع دوم، برای تولید تصاویری با نویز تصادفی استفاده می‌شود. در ابتدا، لیستی از مسیرهای تصاویر ورودی تعریف شده و برای هر تصویر از این لیست، یک نسخه جدید با نویز ایجاد می‌شود و با نام متفاوت ذخیره می‌گردد. این فرایند توسط تابع اصلی generateNoisyImagesکنترل می‌شود که برای هر تصویر موجود در لیست، تابع getNoisyBinaryImageرا فراخوانی می‌کند. در نهایت، نام فایل نویزی تولید شده در خروجی چاپ می‌شود.

در تابع getNoisyBinaryImage، نویز تصادفی به هر پیکسل تصویر اضافه می‌شود. ابتدا تصویر باز شده و مقادیر رنگ قرمز، سبز و آبی (RGB) هر پیکسل خوانده می‌شود. یک مقدار تصادفی در محدوده‌ای مشخص به این مقادیر افزوده می‌شود و سپس اطمینان حاصل می‌شود که مقادیر RGB در بازه مجاز (۰ تا ۲۵۵) باقی بمانند. پیکسل‌های تغییر داده شده به تصویر اعمال می‌شوند و تصویر نهایی با نویز ایجاد شده به‌صورت فایل ذخیره می‌گردد. این کد برای شبیه‌سازی تصاویر نویزی در کاربردهایی مانند پردازش تصویر یا یادگیری ماشین مفید است.

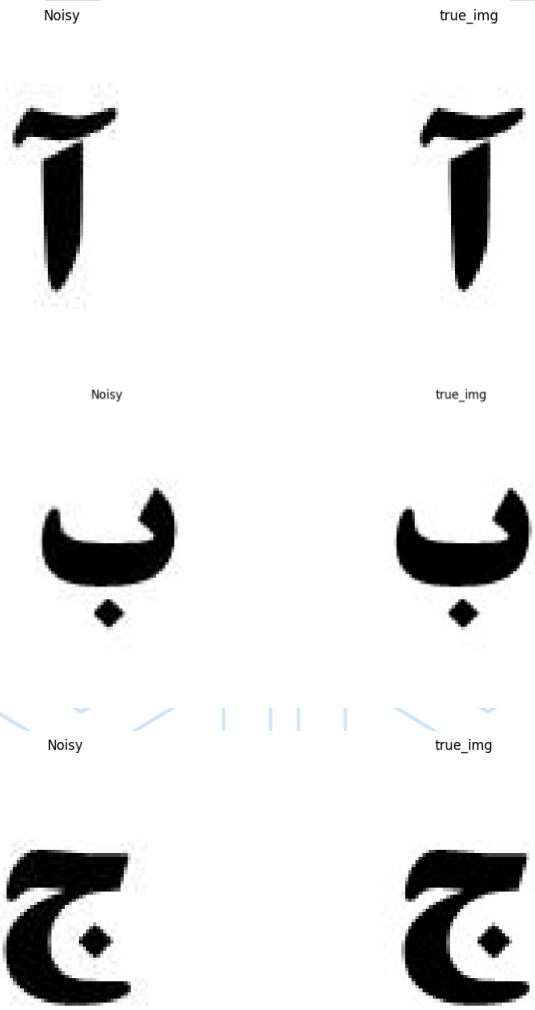
بخش ۲

شبکه همینگ برای شناسایی تعداد زیادی الگو به دلیل ساختار و ویژگی‌های خاص خود گزینه مناسبی است. این شبکه به طور ویژه برای طبقه‌بندی الگوها و محاسبه فاصله همینگ طراحی شده است و از قابلیت تطبیق با داده‌های چندکلاسه و پیچیده برخوردار است. برخلاف شبکه هاپفیلد که محدودیت‌هایی در ذخیره‌سازی تعداد الگوها دارد، شبکه همینگ می‌تواند تعداد بیشتری از الگوها را پردازش و طبقه‌بندی کند. همچنین، سرعت یادگیری و پردازش بالای این شبکه باعث می‌شود که در پروژه‌هایی با داده‌های

پیچیده‌تر، مانند تصاویر پیکسلی از حروف الفبا، عملکرد بهتری داشته باشد. از این رو، برای پروژه شناسایی حروف الفبا که نیازمند شناسایی تعداد بیشتری الگو است، شبکه همینگ انتخاب مناسب‌تری به شمار می‌رود.

برای این کار یک تابع به نام similarity تعریف میکنیم، در این تابع، عکس نویز دار به شکل باینری ورودی گرفته شده و تعداد بیت های متفاوت با هر عکس بدون نویز در آن شناسایی میشود، در نهایت خروجی تابع شماره عکسی است که در آن کمترین بیت های متفاوت با ورودی نویز دار در آن وجود دارد. حال نویز را به تدریج زیاد میکنیم تا جایی که شبکه توانایی تشخیص عکس را نداشته باشد:

با نویز فکتور ۰:



Noisy

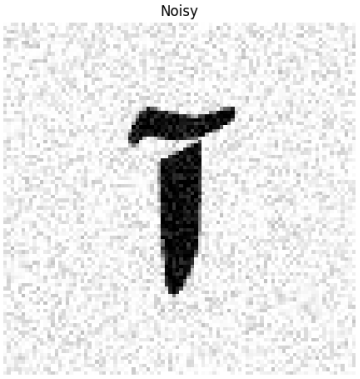
true_img

Noisy

true_img

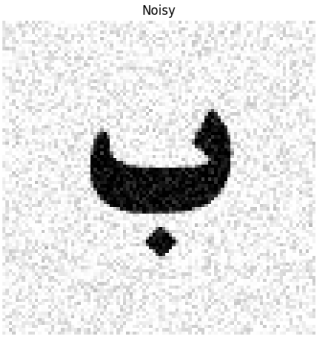
با نویز فکتور ۵۰:

Noisy

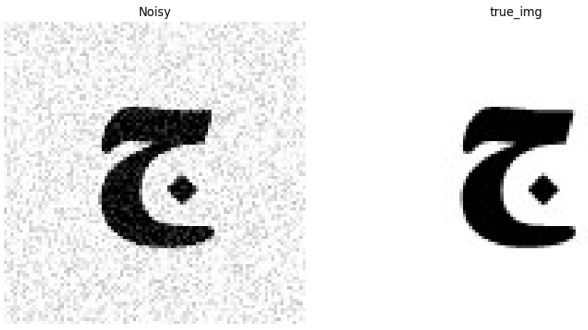


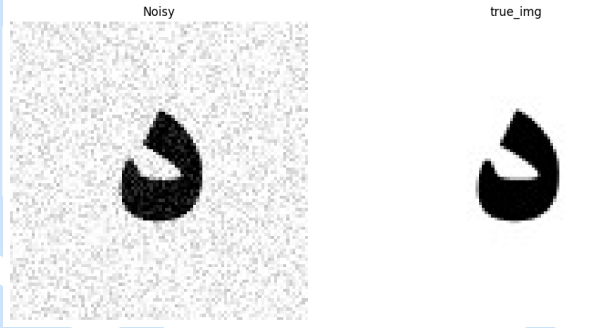
true_img

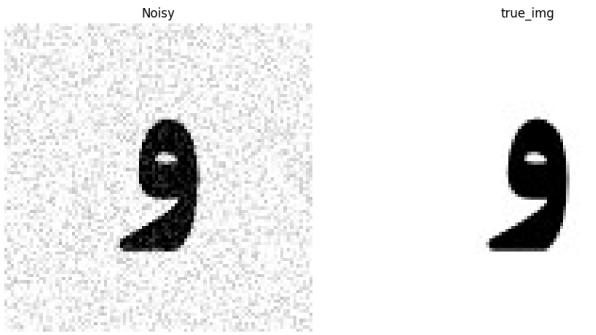
Noisy



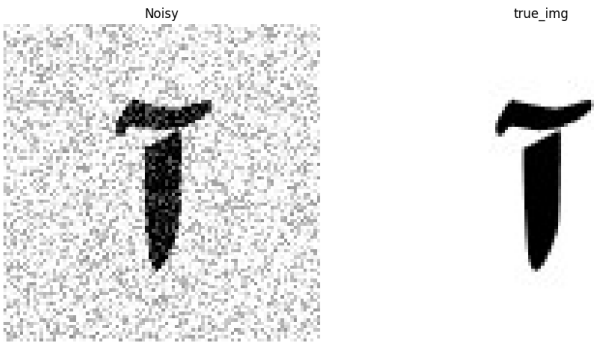
true_img



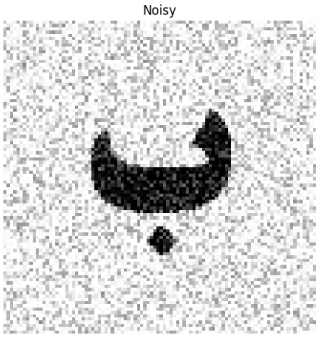




با نویز فکتور ۱۰۰:

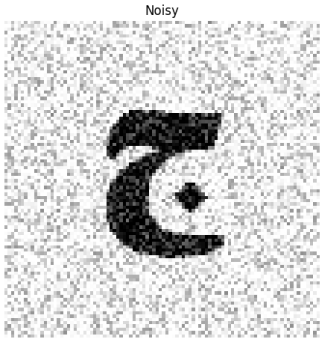


Noisy



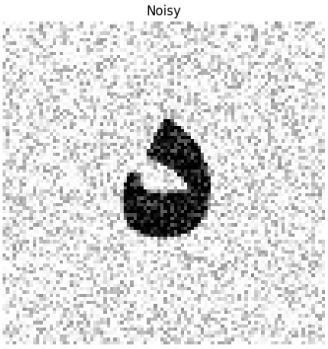
true_img

Noisy



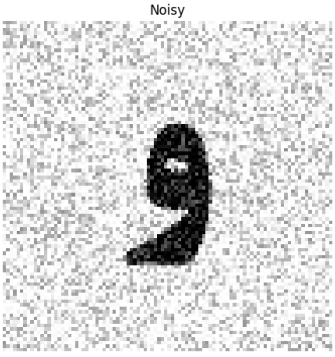
true_img

Noisy



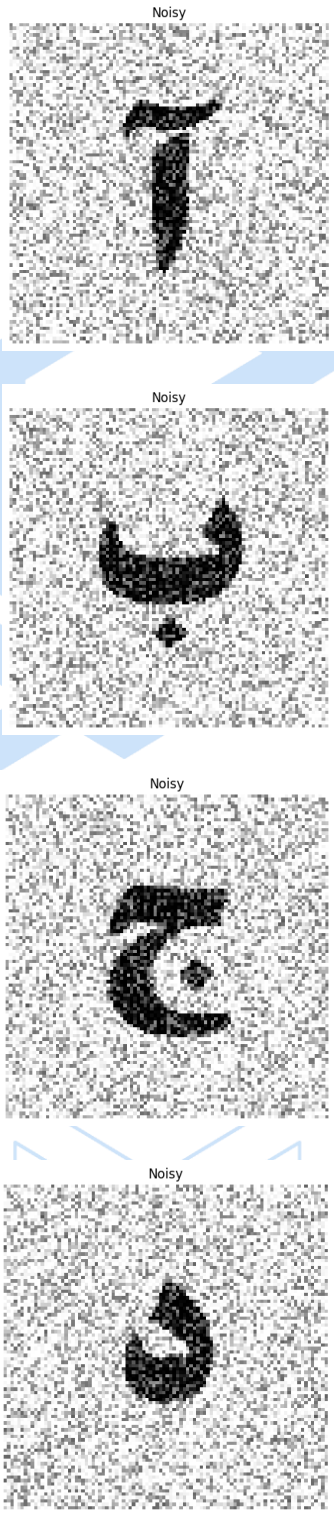
true_img

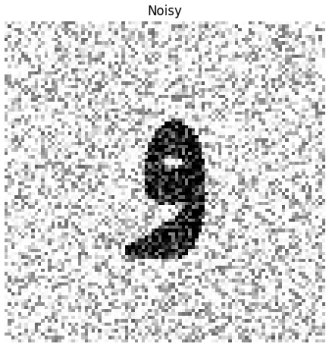
Noisy



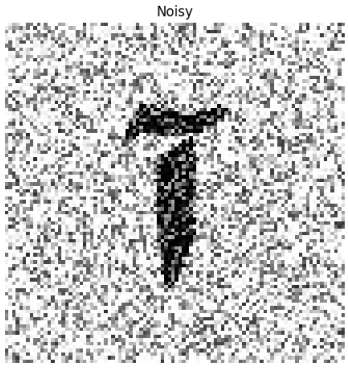
true_img

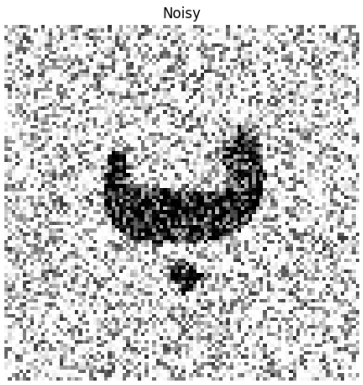
با نویز فکتور ۱۵۰:

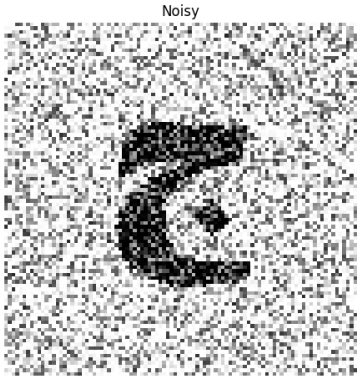


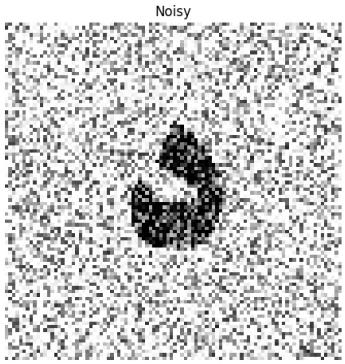


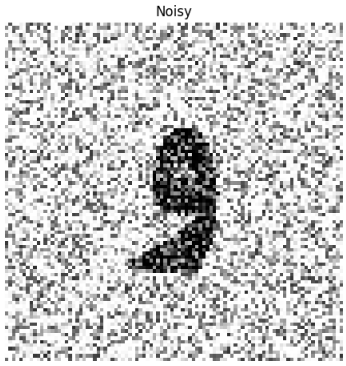
با نویز فکتور ۲۰۰:



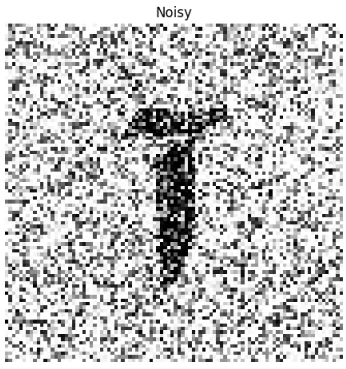


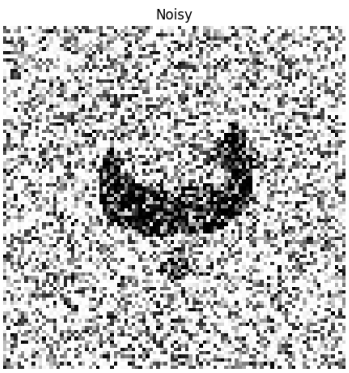


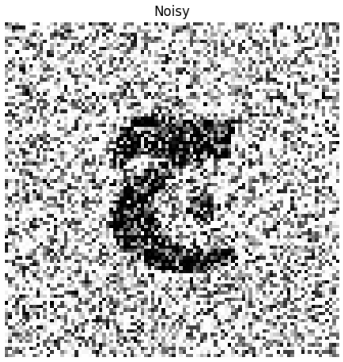


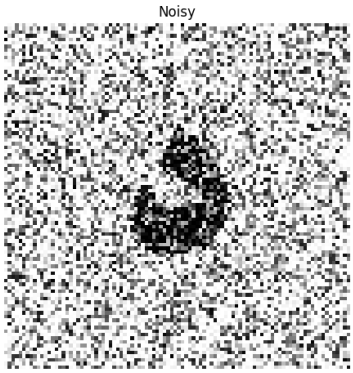


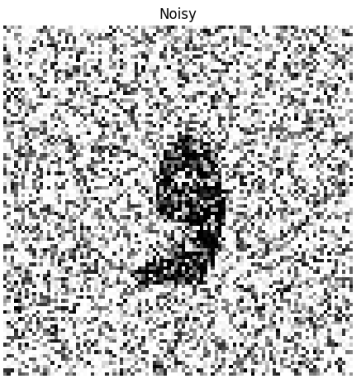
با نویز فکتور ۲۵۰:



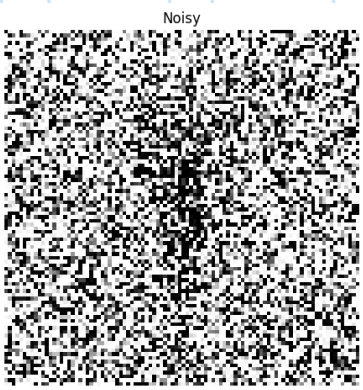




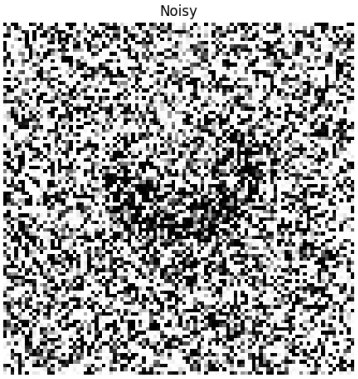




با نویز فکتور ۵۰۰:

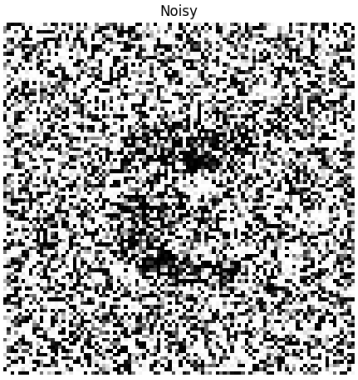


Noisy



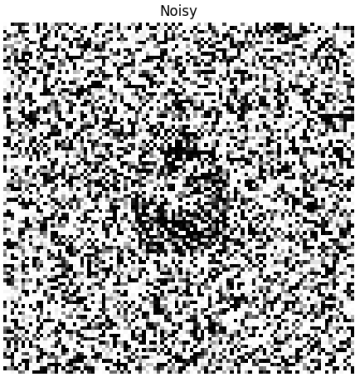
true_img

Noisy



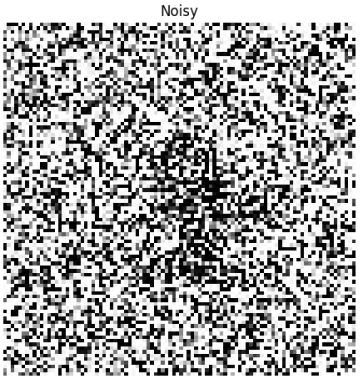
true_img

Noisy



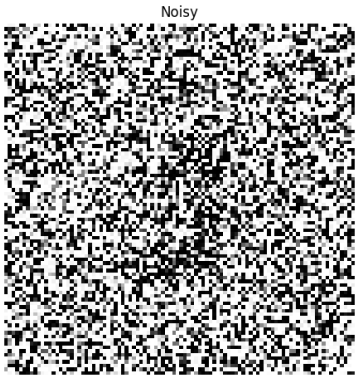
true_img

Noisy

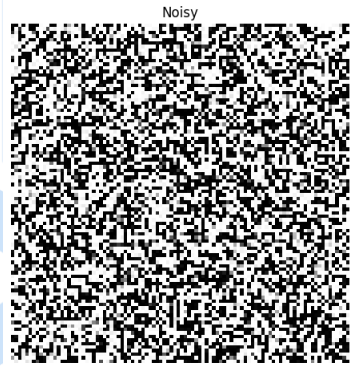


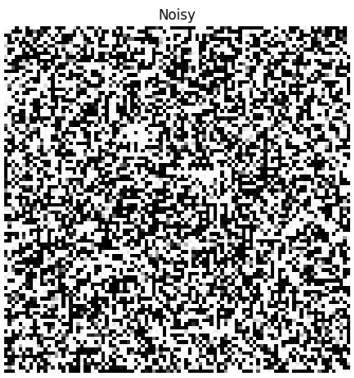
true_img

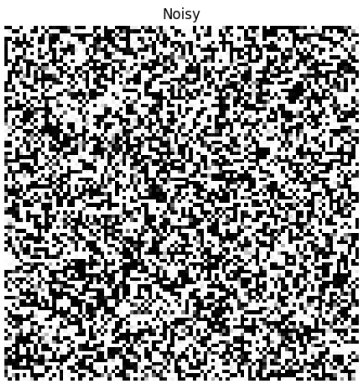
با نویز فکتور ۷۰۰:

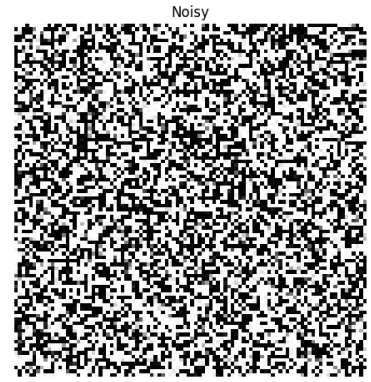


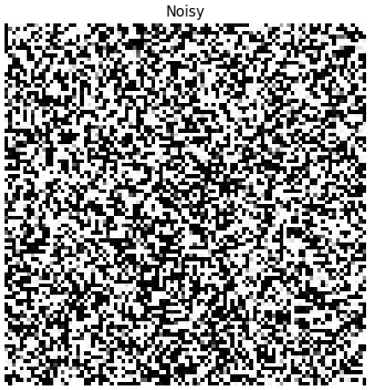
در نهایت، با افزایش نویز فکتور به ۲۰۰۰، شبکه دیگر قادر به تشخیص عکس های ورودی نمیباشد:











مشاهده میشود که شبکه، تا جایی که عکس به صورت کامل به نویز تبدیل میشود، قادر به تشخیص عکس ها میباشد چرا که شبکه بر اساس شباهت عکس ها با عکس های مرجع تصمیم گیری میکند، بنابرین برای آن که عکسی تولید کنیم که شبکه قادر به تشخیص آن نباشد، عکس باید دارای نویز خیلی زیادی باشد.

بخش ۳

ابتدا مانند توابع آماده در دفترچه کد، تابع ساخت missing point را میسازیم. در این تابع عدد رندومی در بازه ۰ تا ۱۰۰۰ ساخته میشود، اگر این عدد از miss_factor کوچک تر بود خروجی سفید شده و در غیر این صورت پیکسل به صورت دست نخورده باقی میماند.

سپس خروجی های این تابع را به شبکه خود میدهیم و miss_factor را تا جایی زیاد میکنیم که شبکه دیگر قادر به تشخیص عکس ها نباشد:

میس فکتور برابر با ۲۰۰:

Missing

true_img

Missing

true_img

Missing

true_img

Missing

true_img

Missing true_img

با میس فکتور برابر با ۵۰۰:

Missing true_img

Missing true_img

Missing true_img

Missing true_img

Missing true_img

با میس فکتور برابر با ۶۵۰:

Missing true_img

Missing true_img

Missing true_img

Missing

true_img

Missing

true_img

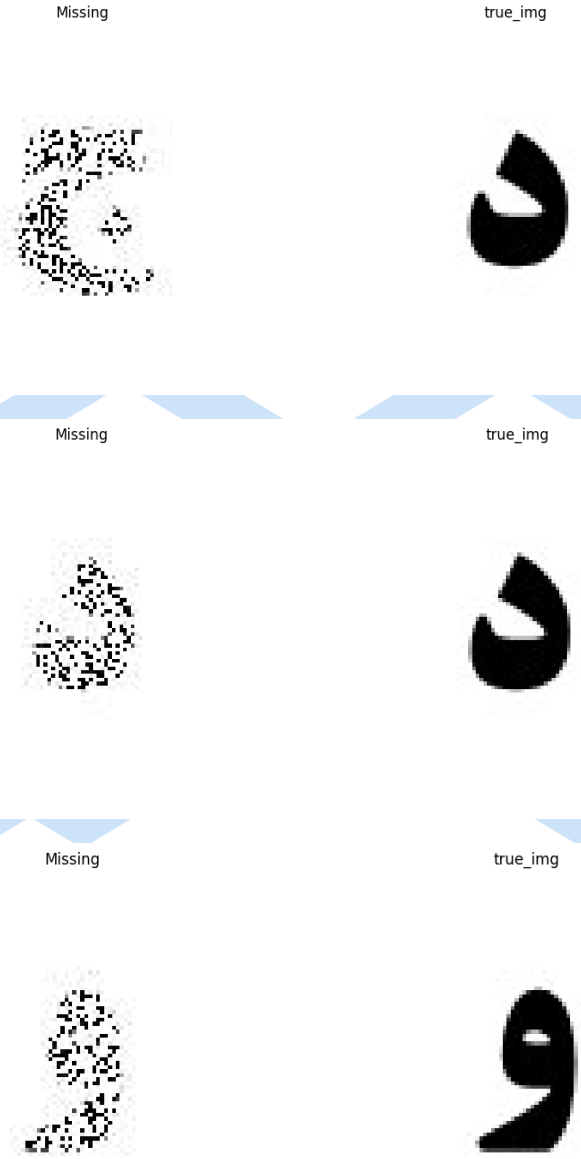
با میس فکتور برابر با ۷۰۰:

Missing

true_img

Missing

true_img



مشاهده میشود با مخدوش کردن سه جهارم پیکسل ها، شبکه دیگر قادر به تشخیص تمامی تصاویر نیست و دارای خطا میباشد. برای حل این راه حل، میتوان پیکسل های سفید را به گونه ای پر کرد که هر پیکسل سفید میانگین ۸ پیکسل اطراف خود باشد. با این راه حل بعضی از پیکسل ها بازیابی میشوند و شبکه تا اعداد بالاتری از میس فکتور میتواند تصاویر را به درستی تشخیص دهد.

سوال چهارم

https://drive.google.com/file/d/13yMeo5fSlaNEDqjTrfi5rHZG1bnA7IGn/view?usp=sharing

در این سوال، ابتدا داده های California housing را از طریق sklearn ایمپورت میکنیم. سپس داده ها را با کمک تابع standard scaler نرمالایز میکنیم و سپس آنها را با نسبت ۰.۸ به ۰.۲ به داده های آموزش و تست تقسیم بندی میکنیم.

در مرحله اول یک مدل بدون لایه RBF، از روی داده ها آموزش میدهیم، این مدل حاوی یک لایه ورودی با ۸ نورون و با اکتیویشن فانکشن linear، لایه پنهان اول دارای ۱۶ نورون و با اکتیویشن فانکشن relu، لایه پنهان دوم با ۸ نورون و با اکتیویشن فانکشن relu و در نهایت لایه خروجی با ۱ نورون و با اکتیویشن فانکشن linear میباشد. سپس مدل را با اپتیمایزر Adam و تابع هزینه MSE کامپایل میکنیم و در نهایت با کمک دستور fit، مدل را روی داده های آموزش با ۲۰۰ ایپاک، آموزش میدهیم.

سپس به کمک مدل، داده های تست را پیش بینی میکنیم و در نهایت، خطای mse برابر با ۰.۲۸ به دست می آید.

در مرحله بعد، به شبکه زیر یک لایه RBF به عنوان لایه پنهان اول اضافه میکنیم. این لایه از توابع Gaussainبرای محاسبه خروجی نودها استفاده می‌کند و به دلیل توانایی در مدل‌سازی روابط غیرخطی پیچیده بین ویژگی‌ها و خروجی‌ها، در مسائلی که داده‌ها رفتار غیرخطی دارند، بسیار مؤثر است. در لایه RBF، هر نود یک مرکز دارد و خروجی هر نود بر اساس فاصله ورودی از مرکز آن و با استفاده از تابع Gaussianمحاسبه می‌شود. این ویژگی به مدل کمک می‌کند تا الگوهای پیچیده در داده‌ها را شبیه‌سازی کند و پیش‌بینی‌های دقیقی انجام دهد. این لایه شامل ۸ نورون که در حقیقت ۸ مرکز برای دسته بندی داده ها میباشد، است. و مراکز را نیز به صورت رندوم مقدار دهی میکنیم و بقیه شبکه، همانند قبل باقی میماند.

در این حالت خطای داده های تست نسبت به مقدار پیش بینی شده مدل، مثل حالت قبل، برابر با ۰.۲۸ به دست می آید.

یکی از دلایل اصلی که باعث شد خطاها برای هر دو مدل مشابه باشد، می‌تواند این باشد که داده‌های مجموعه California Housingبه‌طور ذاتی پیچیدگی غیرخطی زیادی ندارند و ویژگی‌ها در آن به‌صورت خطی به یکدیگر وابسته هستند. در چنین شرایطی، شبکه‌های عصبی بدون لایه RBF قادرند الگوهای موجود در داده‌ها را به‌خوبی شبیه‌سازی کنند و نیاز به استفاده از لایه‌های پیچیده‌تر مانند RBF احساس نمی‌شود. بنابراین، در این شرایط، شبکه بدون RBF می‌تواند به اندازه شبکه با لایه RBF عملکرد مطلوبی داشته باشد.

دلیل دیگر می‌تواند این باشد که پارامترهای مدل، مانند تعداد نورون‌ها یا نرخ یادگیری، برای هر دو مدل مشابه تنظیم شده‌اند و این تنظیمات باعث شده که مدل‌ها به‌طور مشابهی رفتار کنند. به عبارت دیگر، اگر لایه RBF به‌درستی تنظیم نشده باشد یا پارامترهای مربوطه به طور مناسب انتخاب نشده باشند، تأثیر آن در مدل کاهش می‌یابد و در نتیجه، مدل با لایه RBF نتوانسته است بهبود قابل توجهی در پیش‌بینی ایجاد کند. در اینجا ممکن است به دلیل نداشتن علم کافی برای نحوه انتخاب مراکز به‌طور بهینه، شبکه RBFنتوانسته باشد ویژگی‌های داده‌ها را به‌طور مؤثر مدل‌سازی کند، و در نتیجه عملکرد مشابهی با مدل بدون RBF مشاهده شده است.