

به نام خدا



دانشگاه تهران



دانشکده مهندسی برق و کامپیوتر

|                                      |                    |
|--------------------------------------|--------------------|
| آناهیتا هاشم زاده - پرهام بیچرانلویی | نام و نام خانوادگی |
| ۸۱۰۱۰۰۳۰۳ - ۸۱۰۱۰۰۵۰۲                | شماره دانشجویی     |
| ۱۴۰۱،۰۹،۲۸                           | تاریخ ارسال گزارش  |

**درس شبکه‌های عصبی و یادگیری عمیق**

**تمرین اضافه**

## فهرست

|    |  |
|----|--|
| ۴  | پاسخ ۱. تشخیص تقلب یا استفاده از شبکه عمیق |
| ۴  | ۱-۱  |
| ۴  | ۱-۲  |
| ۷  | ۱-۳  |
| ۸  | ۱-۴  |
| ۹  | ۱-۵  |
| ۱۱ | ۱-۶  |
| ۲۴ | ۱-۷  |
| ۲۶ | پاسخ ۳ - تشخیص کاراکتر نوری                |
| ۲۶ | ۱-۳. تفاوت بین شبکه‌های CNN و DCNN         |
| ۲۶ | ۲-۳. سه روش بهینه سازی                     |
| ۲۸ | ۳-۳. پیاده سازی معماری DCNN                |
| ۳۱ | ۴-۳. رسم نمودارها و مقادیر معیارها         |
| ۳۷ | ۵-۳. بهترین شبکه                           |

## شکل‌ها

|  |    |
|--|----|
| شکل ۱. سه مرحله اصلی استفاده شده در مقاله.....   | ۵  |
| شکل ۲. شبکه عصبی denoising autoencoder.....  | ۶  |
| شکل ۳. ماتریس طبقه بندی داده های تست.....  | ۱۰ |
| شکل ۴. دقت و recall شبکه با thresholdهای مختلف.....  | ۱۲ |
| شکل ۵. نمودار loss و accuracy بدست آمده برای داده های آموزش و validation با thresholdهای مختلف.....                  | ۲۳ |
| شکل ۶. نمودار loss و accuracy بدست آمده برای داده های آموزش و validation با داده های متعادل و داده های نامتعادل..... | ۲۵ |
| شکل ۷. عنوان تصویر نمونه.....  | ۲  |
| شکل ۸. عنوان تصویر نمونه.....  | ۲  |
| شکل ۹. عنوان تصویر نمونه.....  | ۲  |
| شکل ۱۰. عنوان تصویر نمونه.....   | ۲  |
| شکل ۱۱. عنوان تصویر نمونه.....   | ۲  |

## جدول‌ها

- ۶ ..... denoised autoencoder  
جدول ۱. معماری مدل
- ۷ ..... classifier  
جدول ۲. معماری مدل
- ۹ ..... تقسیم بندی داده‌ها  
جدول ۳.
- ۹ ..... Net hyper parameters  
جدول ۴.
- ۱ **Error! Bookmark not defined.** ..... نتایج شبکه برای داده‌های تست
- Error! ..... threshold بر روی داده‌های تست ۱  
جدول ۶. نتایج ارزیابی شبکه برای مختلف
- Bookmark not defined.
- ۲۴ ..... unbalanced و balanced ..... نتایج شبکه برای داده‌های تست با استفاده از داده‌های
- Error! Bookmark not defined. ..... عنوان جدول نمونه
- جدول ۸. عنوان جدول نمونه ..... عنوان جدول نمونه
- Error! Bookmark not defined. ..... عنوان جدول نمونه
- Error! Bookmark not defined. ..... عنوان جدول نمونه
- Error! Bookmark not defined. ..... عنوان جدول نمونه
- Error! Bookmark not defined. ..... عنوان جدول نمونه
- جدول ۱. عنوان جدول نمونه ..... عنوان جدول نمونه
- Error! Bookmark not defined. ..... عنوان جدول نمونه
- Error! Bookmark not defined. ..... عنوان جدول نمونه
- Error! Bookmark not defined. ..... عنوان جدول نمونه
- Error! Bookmark not defined. ..... عنوان جدول نمونه

## پاسخ ۱. تشخیص تقلب یا استفاده از شبکه عمیق

.۱-۱

تکنیک های داده کاوی یک از روش های اصلی در حل مشکل تشخیص تقلب می باشد. نحوه تشخیص تقلب بسیار مهم است، امروزه به علت کلان داده استفاده از روش های سنتی غیرممکن است به همین دلیل مراکز زیادی توجه خود را به روش های محاسباتی جدید تشخیص تقلب متوجه کرده اند.

مسئله کلیدی طبقه بندی تنها زمانی میتواند نتیجه مطلوبی داشته باشد که دارای مجموعه داده مطلوب باشد، این در حالی است که در کاربردهای عملی تعداد زیادی مجموعه داده نامتعادل وجود دارد. در مسئله تقلب نیز کلاس اقلیت، که تراکنش های غیرعادی می باشد، اهمیت دارد. به طور مثال اگر کلاس اقلیت کمتر از درصد کل مجموعه باشد، دقت مدل به بیش از ۹۹ درصد می رسد حتی اگر تمام کلاس اقلیت نیز به اشتباه طبقه بندی شده باشد. پس امروز تلاش های زیادی برای حل اسن مشکل صورت گرفته است.

یک روش رایج برای رسیدگی به مشکل طبقه بندی داده های نامتعادل نمونه گیری کلاس اقلیت است. هدف اصلی oversampling افزایش تعداد نمونه های کلاس اقلیت به منظور بهتر شدن اطلاعات طبقه بندی اصلی است. بنابراین در مسائلی که تقاضا بیشتری برای دقت طبقه بندی هست عموما الگوریتم oversampling استفاده می شود. در این مقاله نیز برای حل این مشکل و دستیابی به مدل مناسب از denoising autoencoder و همچنین oversampling استفاده شده است.

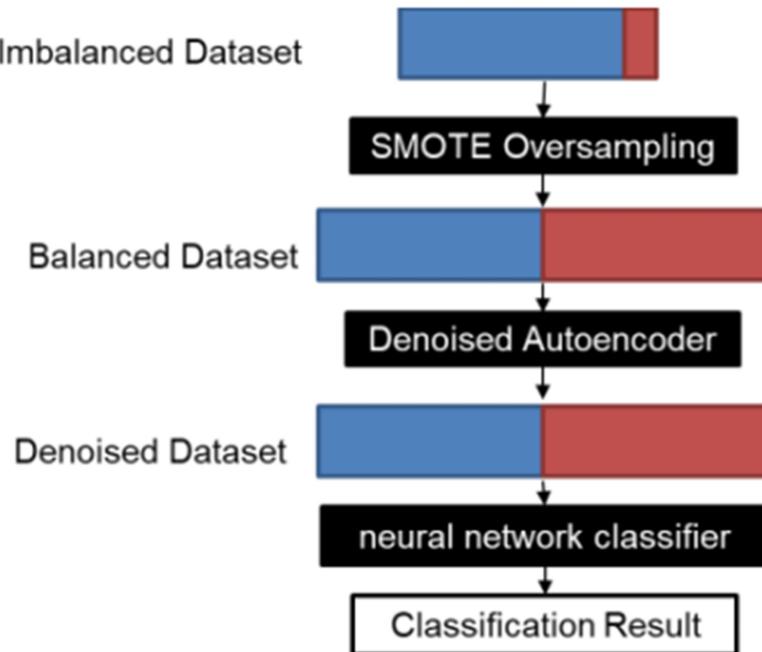
.۱-۲

معماری مدل از دو شبکه classifier و autoencoder تشکیل شده که داده ها را denoise کرده و classifier وظیفه طبقه بندی داده ها را به عهده دارد. در این مقاله پروسه انجام کار دارای سه مرحله اصلی می باشد:

- در ابتدا از oversampling برای تبدیل دیتاست نامتعادل به دیتاست متعادل استفاده می شود.

- سپس از denoised autoencoder برای از بین بردن نویز داده ها استفاده میشود.

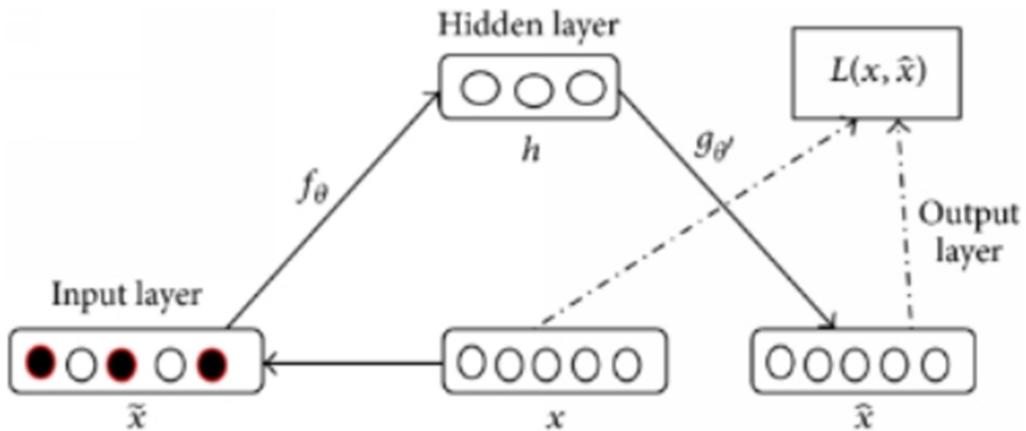
- در نهایت از مدل deep fully connected برای طبقه بندی نهایی استفاده می شود.



شکل ۱. سه مرحله اصلی استفاده شده در مقاله

در ادامه به اختصار به جزئیات این مراحل می پردازیم:

در واقع autoencoder یک شبکه عصبی برای یادگیری unsupervised است که دارای دو بخش encoder و decoder می باشد. در autoencoder لایه خروجی تعداد نورون برابر با لایه ورودی دارد و هدف آن بازسازی ورودی خود به جای پیشیبینی مقدار هدف از روی ورودی است. در اینجا از یک مدل autoencoder با ۷ لایه به منظور رفع نویز دادگان استفاده شده است. در لایه اول داده نویزدار را دریافت می کند و سپس ورودی را به لایه های fully connected داده و در نهایت square loss function را اعمال و خروجی بدون نویز را نیز ایجاد میکند.



شکل ۲. شبکه عصبی denoising autoencoder

جدول ۱. معماری مدل denoised autoencoder

|                           |
|---------------------------|
| Dataset with noise(29)    |
| Fully-Connected-Layer(22) |
| Fully-Connected-Layer(15) |
| Fully-Connected-Layer(10) |
| Fully-Connected-Layer(15) |
| Fully-Connected-Layer(22) |
| Fully-Connected-Layer(29) |
| Square Loss Function      |

بعد از گرفتن خروجی بدون نویز، در ادامه داده ها را به یک مدل classifier با ۵ لایه fully connected برای طبقه بندی وارد کرده است. در آخر نیز از softmax و همچنین cross-entropy function برای طبقه بندی نهایی استفاده شده است.

جدول ۲. معماری مدل classifier

|                                     |
|-------------------------------------|
| Denoised Dataset(29)                |
| Fully-Connected-Layer(22)           |
| Fully-Connected-Layer(15)           |
| Fully-Connected-Layer(10)           |
| Fully-Connected-Layer(5)            |
| Fully-Connected-Layer(2)            |
| SoftMax Cross Entropy Loss Function |

### .۱-۳

هدف اصلی از متعادل کردن کلاس ها افزایش فراوانی طبقه اقلیت یا کاهش فراوانی طبقه اکثریت است. این کار برای به دست آوردن تقریباً همان تعداد نمونه برای هر دو کلاس انجام می شود. تکنیک های resampling به شرح زیر است:

#### :Random under-sampling

در این روش با کم کردن داده های کلاس majority تلاش می شود دیتاست را متعادل کند. این روش زمانی استفاده میشود که تعداد داده کافی باشد. به این صورت انجام می شود که با حفظ تمام داده های اقلیت، بطور تصادفی تعداد مساوی داده از کلاس اکثریت را حذف میکنیم و این کار را تا جایی ادامه می دهیم که نمونه ها متعادل شوند.

**مزایا:** این روش زمانی که مجموعه داده آموزش بسیار زیاد باشد می تواند به بهبود زمان اجرا و همچنین مسائل storage کمک کند.

**معایب:** در این روش ممکن است اطلاعات مفید دور ریخته شود و همچین ممکن است داده های انتخاب شده در این روش biased باشد و نماینده خوبی برای جامعه نباشد، در نتیجه منجر به نتایج نادرست با داده ها تست شود.

## **:Random Over-Sampling**

این روش تعداد نمونه های کلاس اقلیت را با تکرار رندوم افزایش می دهد تا تعداد بیشتری داده از کلاس اقلیت را در سمبول ارائه دهد و دیتاست به تعادل برسد.

**مزایا:** بر خلاف روش پیشین در این روش منجر به از دست دادن اطلاعات نخواهد شد و ممکن است عملکرد بهتری داشته باشد.

**معایب:** از آنجایی که این روش رویداد کلاس اقلیت را تکرار میکند، احتمال overfitting را بالا میبرد.

## **:Synthetic Minority Over-sampling**

این روش برای جلوگیری از overfitting زمانی که نمونه های دقیقی از داده اقلیت به دیتاست اضافه می شود، استفاده می شود. به این صورت که یک زیرمجموعه از داده کلاس اقلیت به عنوان نمونه گرفته می شود و سپس نمونه های مشابه مصنوعی ایجاد می شوند و به دیتاست اصلی اضافه می شوند. در نهایت دیتاست جدید به عنوان نمونه ای برای آموزش مدل طبقه بندی استفاده می شود.

**مزایا:** در این روش اطلاعات مفید از دست نمی رود و همچنین مشکل overfitting موجود در روش پیشین را با استفاده از ایجاد داده های مصنوعی به جای تکرار نمونه کاهش می یابد.

**معایب:** حین تولید داده های مصنوعی SMOTE داده های نزدیک از کلاس های دیگر را در نظر نمیگیرد، و این مسئله می تواند منجر به افزایش همپوشانی کلاس ها و ایجاد نویز اضافی شود.

به طور کلی بین هیچ یک از روش های resampling نسبت به روش دیگر هیچ مزیت مطلقی وجود ندارد و استفاده از این روش ها بستگی به کاربرد مورد استفاده از دیتاست دارد. ترکیب این روش ها با هم معمولاً موفق آمیز بوده است.

---

.۱-۴

ابتدا دیتاست موردنظر را لود کرده و داده های مربوط به TIME را حذف و همچنین داده های AMOUNT را با استفاده از دستور StandardScaler() نرمال میکنیم. در ادامه ۲۰ درصد دادگان را به تست و ۸۰ درصد دادگان را نیز به آموزش اختصاص می دهیم و از میان داده های آموزش ۲۰ درصد برای دادگان validation انتخاب شده است.

جدول ۳. تقسیم بندی داده ها

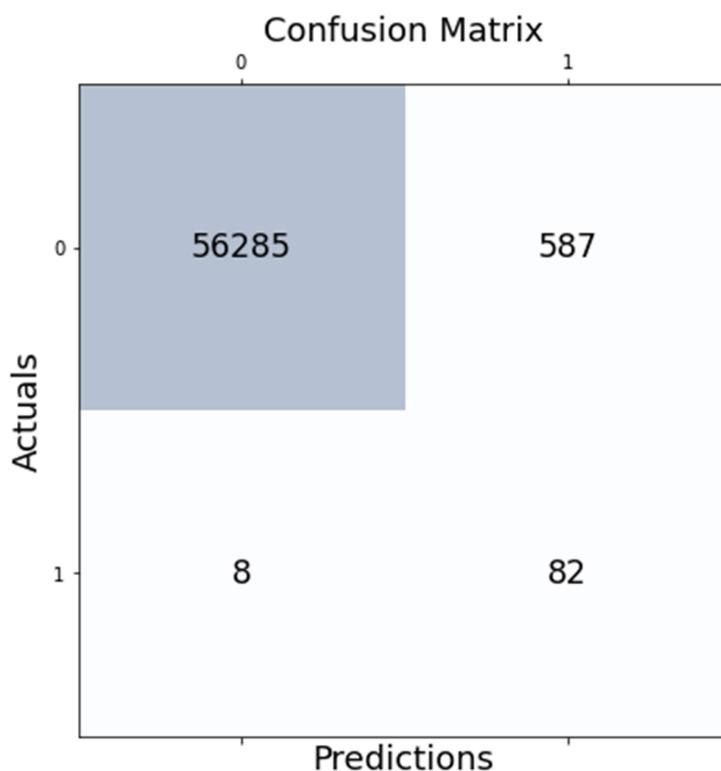
|                       |              |
|-----------------------|--------------|
| Shape of x_train      | (182276, 29) |
| Shape of y_train      | (182276, 1)  |
| Shape of x_validation | (45569, 29)  |
| Shape of y_validation | (45569, 1)   |
| Shape of x_test       | (56962, 29)  |
| Shape of y_test       | (56962, 1)   |

همانطور که پیشتر گفته شد، از آنجایی که داده ها نامتعادل است می بایست دیتاست را پیش از آموزش متعادل کنیم، پس با استفاده از دستور SMOT() دادگان آموزش را (با threshold های مختلف مطابق با بخش ۶ سوال) oversample میکنیم. بعد از ساخت دیتاست متعادل با استفاده از Gaussian noise به دادگان آموزش نویز اضافه کرده، سپس آن را به denoised autoencoder وارد میکنیم. بعد از بدست آوردن آموزش denoised dataset، خروجی autoencoder را به classifier وارد کرده و طبقه بندی می کنیم.

**جدول ۴. Net hyper parameters**

| Network Used Hyper Parameters |             |               |
|-------------------------------|-------------|---------------|
| • Batch Size                  |             | 300           |
| • Epochs                      |             | 100           |
| • Input                       |             | 29            |
| • Output                      | Autoencoder | 29            |
|                               | Classifier  | 2             |
| • Loss Function               | Autoencoder | MSE           |
|                               | Classifier  | Cross Entropy |
| • Optimizer                   |             | SGD           |
| • Learning rate               |             | 0.001         |
| • momentum                    |             | 0.9           |

استفاده از accuracy برای ارزیابی مدل طبقه بندی، خصوصاً برای مدل هایی که از دیتاست نامتعادل استفاده میکنند، معیار مناسبی نیست. چون تعداد داده های majority زیاد بوده و مثلاً اگر ۹۹ درصد از داده ها نرمال باشند و فقط ۱ درصد از داده ها غیرعادی باشند همچنان مدل می تواند دقت ۹۹ درصد بدست آورد، در صورتی که تشخیص میزان کلاس غیرعادی اهمیت دارد. در همچین مواردی از confusion matrix استفاده می شود. recall نسبت بین تعداد ناهنجاری های درست تشخیص داده شده و تعداد کل ناهنجاری ها را نشان می دهد، در واقع ارزیابی میکند که چه میزان میتوان ناهنجاری ها را در مدل طبقه بندی شناسایی کرد. با توجه به اینکه recall بدست آمده برای شبکه برابر  $0.95$  است پس یعنی این مدل تا ۹۵ درصد میتواند ناهنجاری ها را تشخیص دهد.



شکل ۳. ماتریس طبقه بندی داده های تست

جدول ۵. نتایج شبکه برای داده های تست

|               |       |
|---------------|-------|
| Test Accuracy | 0.989 |
|---------------|-------|

|              |       |
|--------------|-------|
| Recall score | 0.950 |
| F1-score     | 0.605 |
| Precision    | 0.561 |

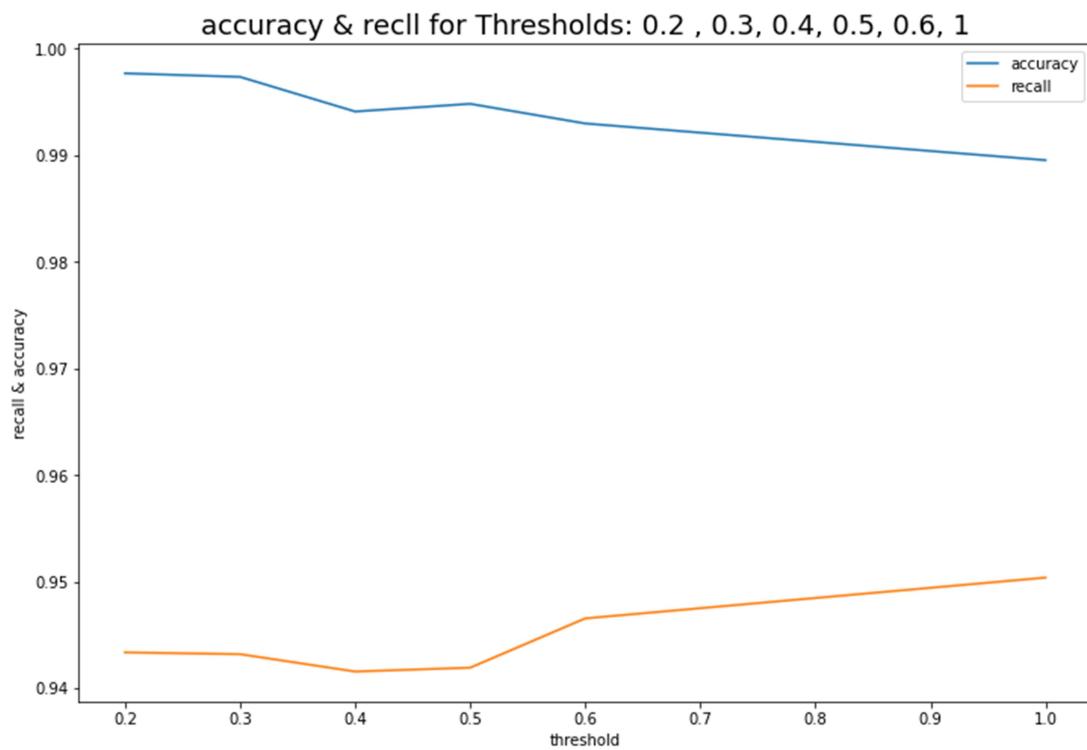
## ۱-۶

در این بخش از سوال همانطور که خواسته شده oversampling داده ها را با threshold های مختلف انجام داده و سپس accuracy و recall به ازای دادگان تست مربوط به هر حالت گزارش شده است.

جدول ۶ نتایج ارزیابی شبکه برای threshold های مختلف بر روی داده های تست

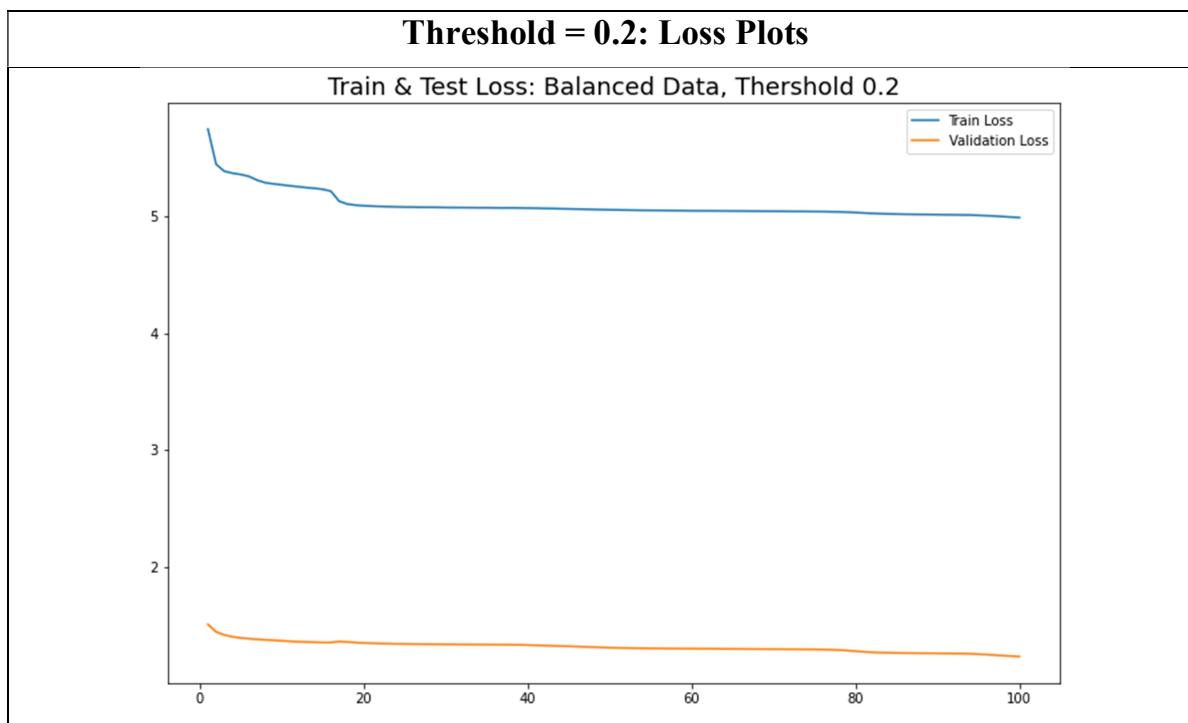
| Threshold | Accuracy | Recall |
|-----------|----------|--------|
| 0.2       | 0.9977   | 0.9433 |
| 0.3       | 0.9973   | 0.9432 |
| 0.4       | 0.9941   | 0.9415 |
| 0.5       | 0.9948   | 0.9419 |
| 0.6       | 0.9929   | 0.9465 |
| 1         | 0.9895   | 0.9503 |

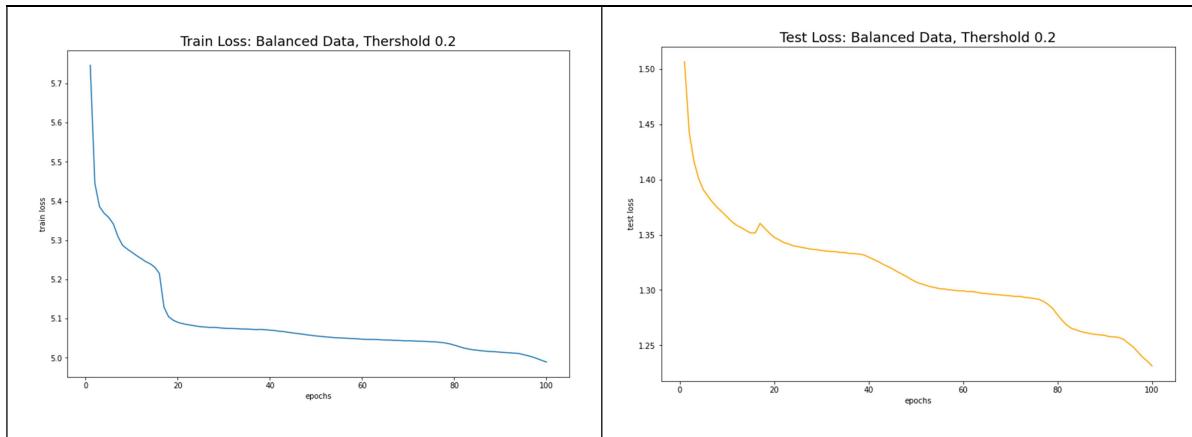
همانطور که در جدول ۶ دیده می شود حدودا در تمام حالت ها دقت تقریبا برابر ۹۹ درصد می باشد، هرچند با افزایش threshold یعنی افزایش میزان oversampling داده های اقلیت و بالا بردن تعادل دیتاست مقدار recall را به افزایش است هر چند این اختلاف بسیار ناچیز است اما این بدين معنی است که مدل قادر است میزان بیشتری ناهنجاری موجود در دیتاست را تشخیص دهد.



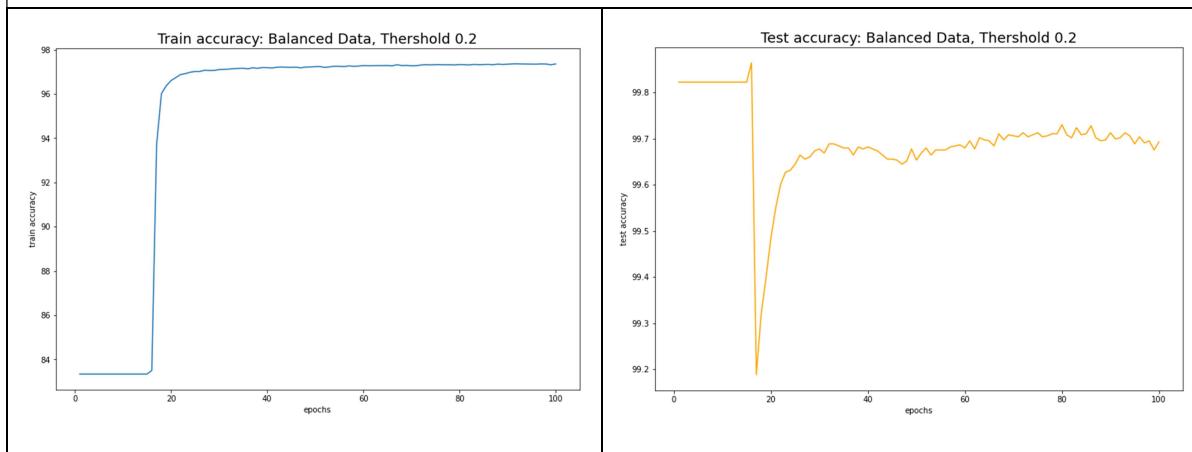
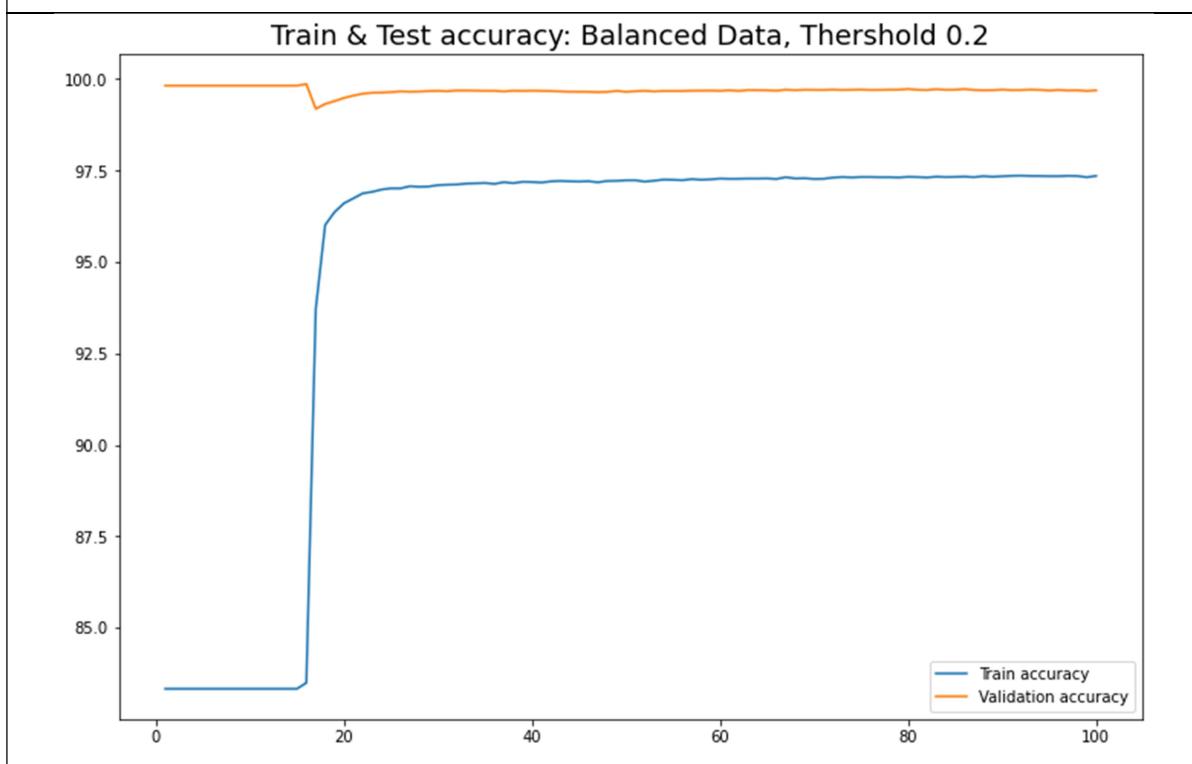
شکل ۴. دقت و recall شبکه با threshold های مختلف

- نتایج بدست آمده برای آموزش شبکه با threshold های مختلف در ادامه آورده شده است:



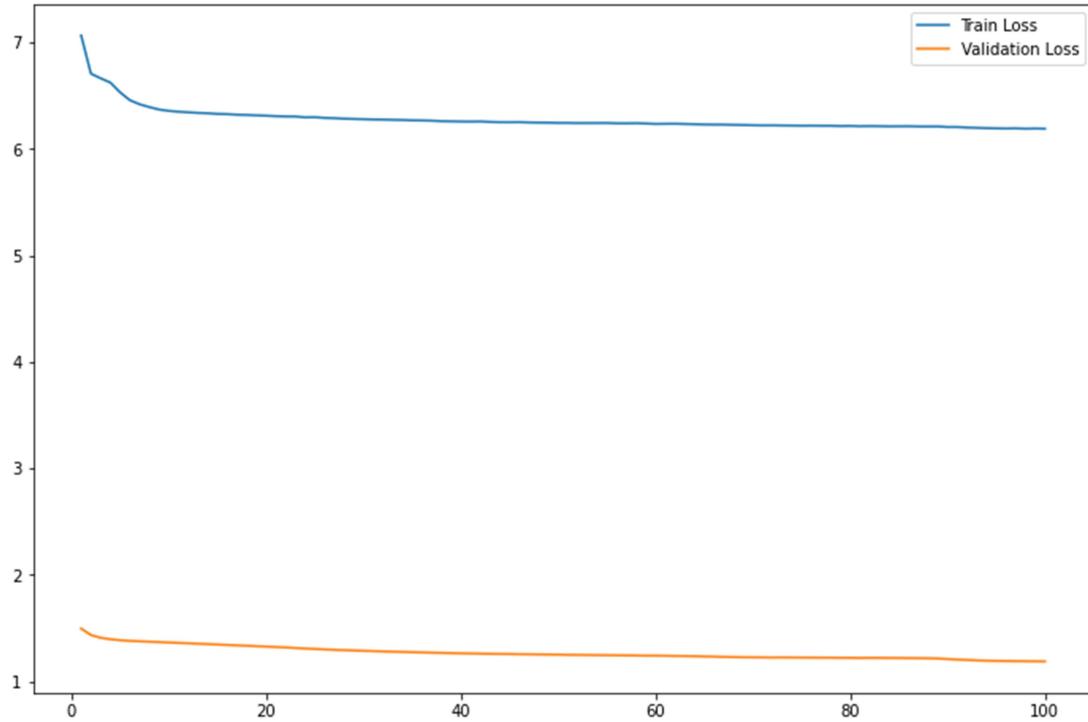


**Threshold = 0.2: Accuracy Plots**

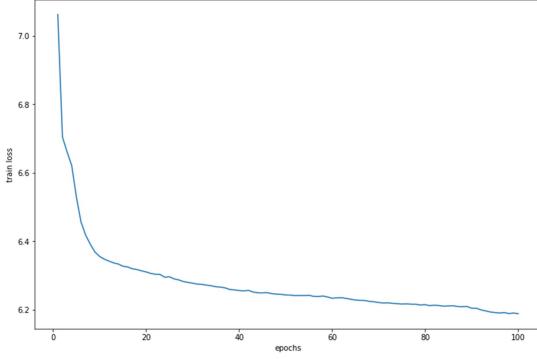


## Threshold = 0.3: Loss Plots

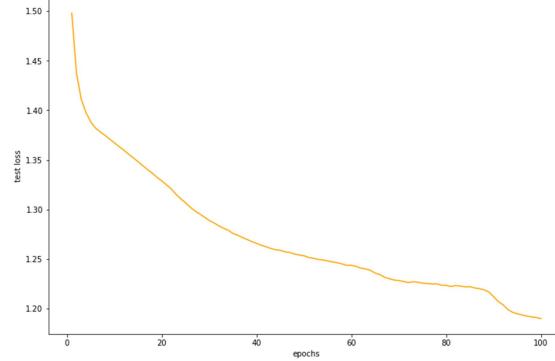
Train & Test Loss: Balanced Data, Thersholt 0.3



Train Loss: Balanced Data, Thersholt 0.3

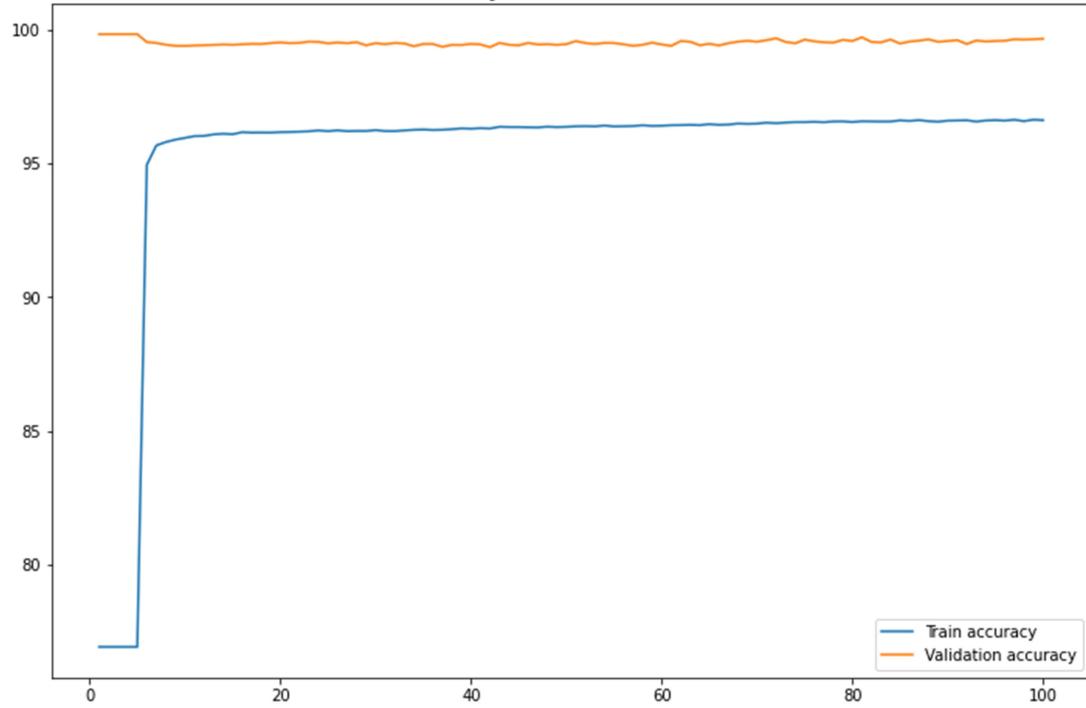


Test Loss: Balanced Data, Thersholt 0.3

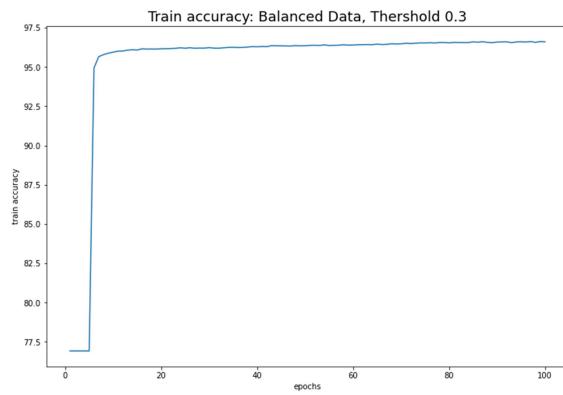


## Threshold = 0.3: Loss Plots

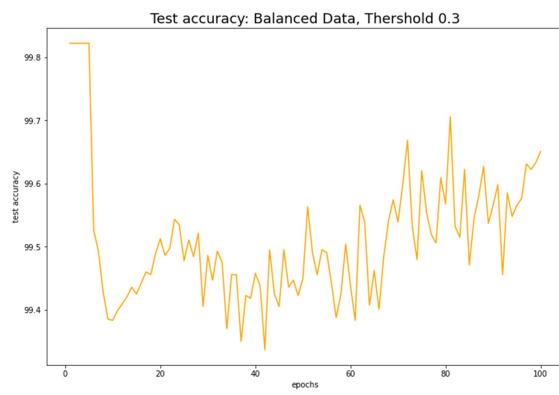
Train & Test accuracy: Balanced Data, Thersholt 0.3



Train accuracy: Balanced Data, Thersholt 0.3

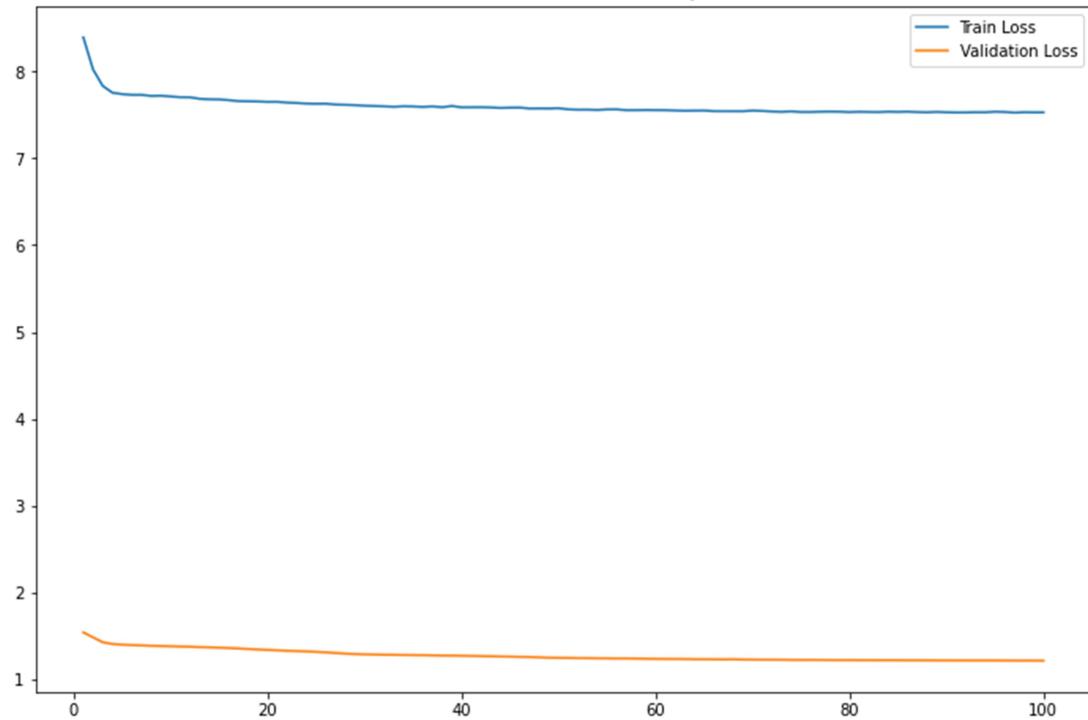


Test accuracy: Balanced Data, Thersholt 0.3

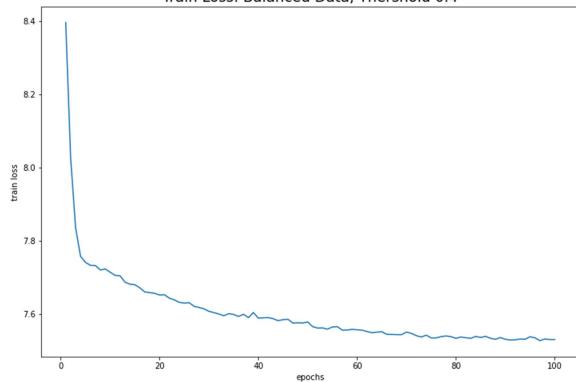


## Threshold = 0.4: Loss Plots

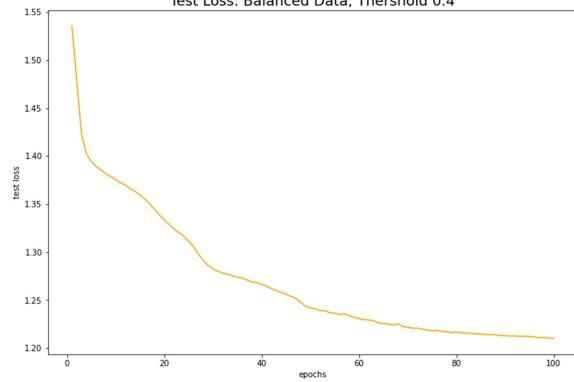
Train & Test Loss: Balanced Data, Thersholt 0.4



Train Loss: Balanced Data, Thersholt 0.4

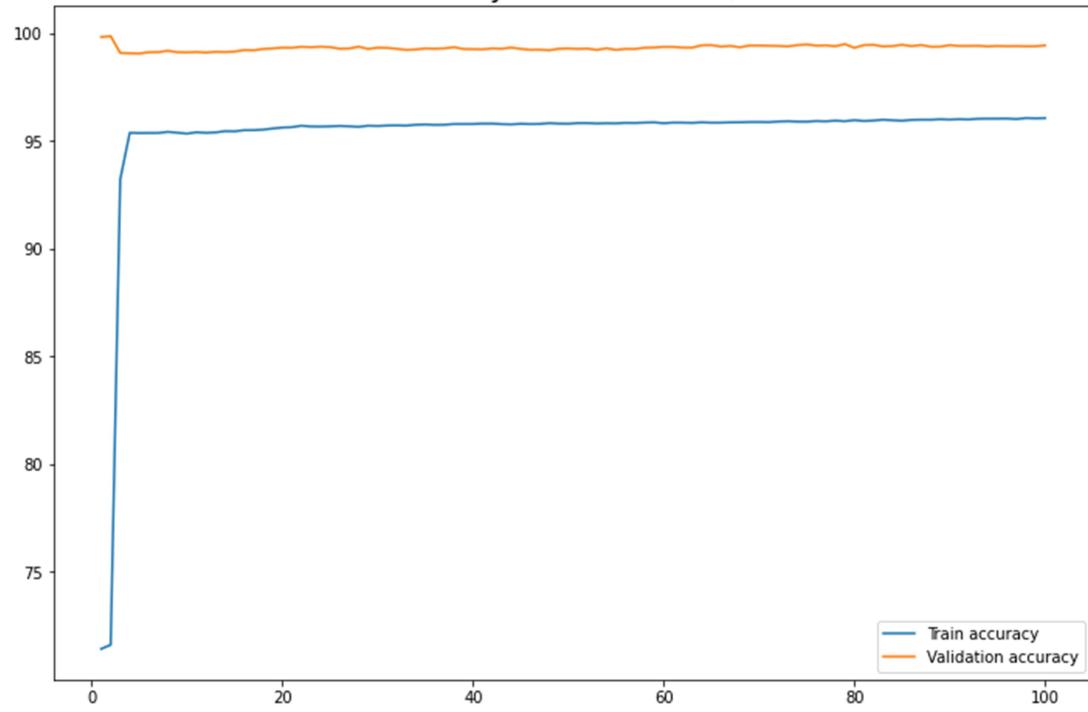


Test Loss: Balanced Data, Thersholt 0.4

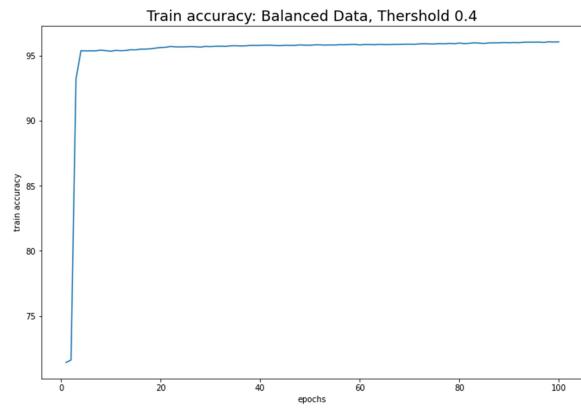


## Threshold = 0.4: Accuracy Plots

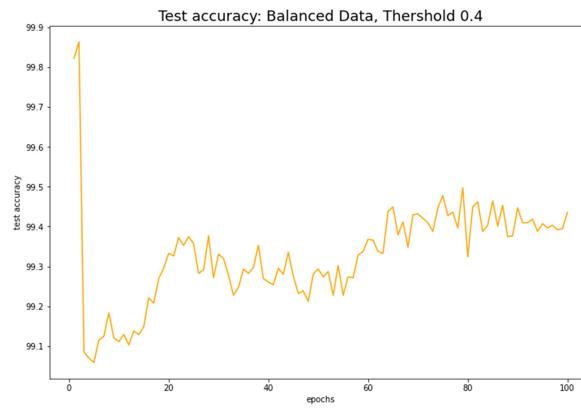
Train & Test accuracy: Balanced Data, Thersholt 0.4



Train accuracy: Balanced Data, Thersholt 0.4

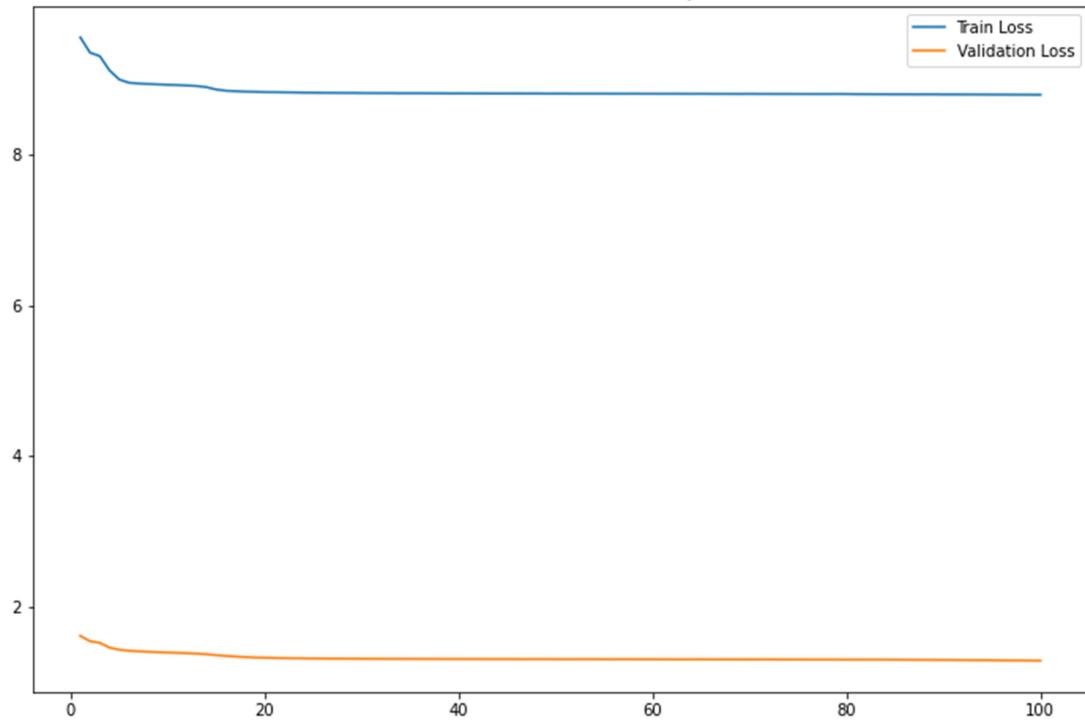


Test accuracy: Balanced Data, Thersholt 0.4

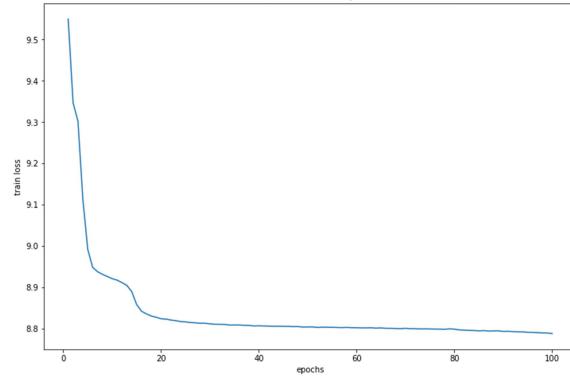


## Threshold = 0.5: Loss Plots

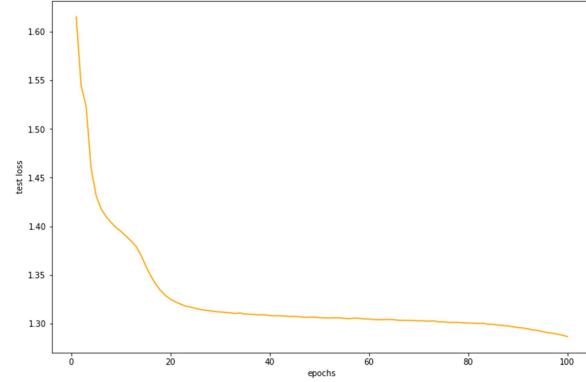
Train & Test Loss: Balanced Data, Thersholt 0.5



Train Loss: Balanced Data, Thersholt 0.5

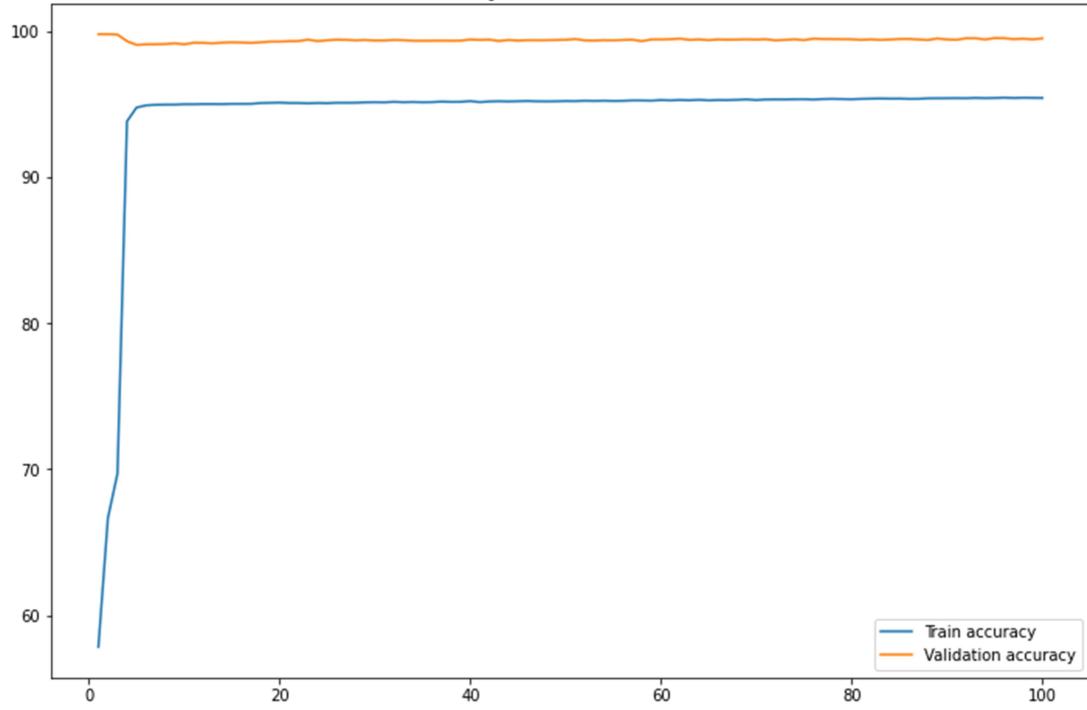


Test Loss: Balanced Data, Thersholt 0.5

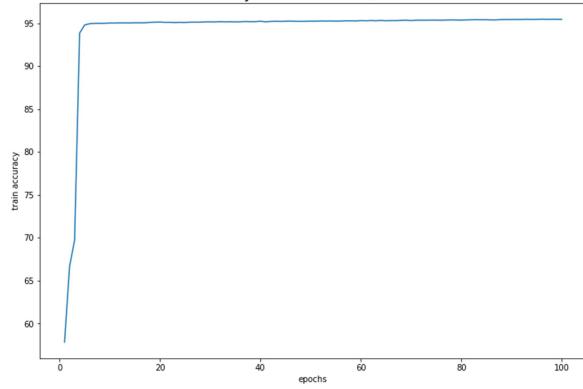


## Threshold = 0.5: Accuracy Plots

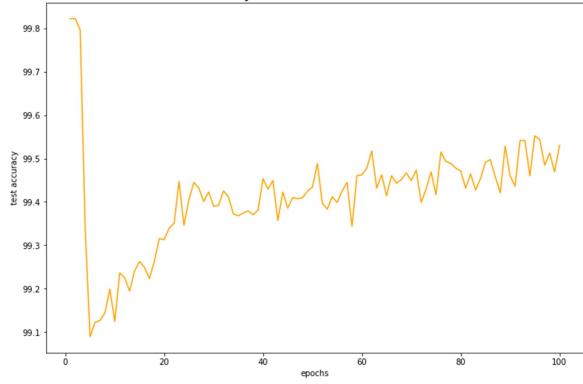
Train & Test accuracy: Balanced Data, Thersholt 0.5



Train accuracy: Balanced Data, Thersholt 0.5

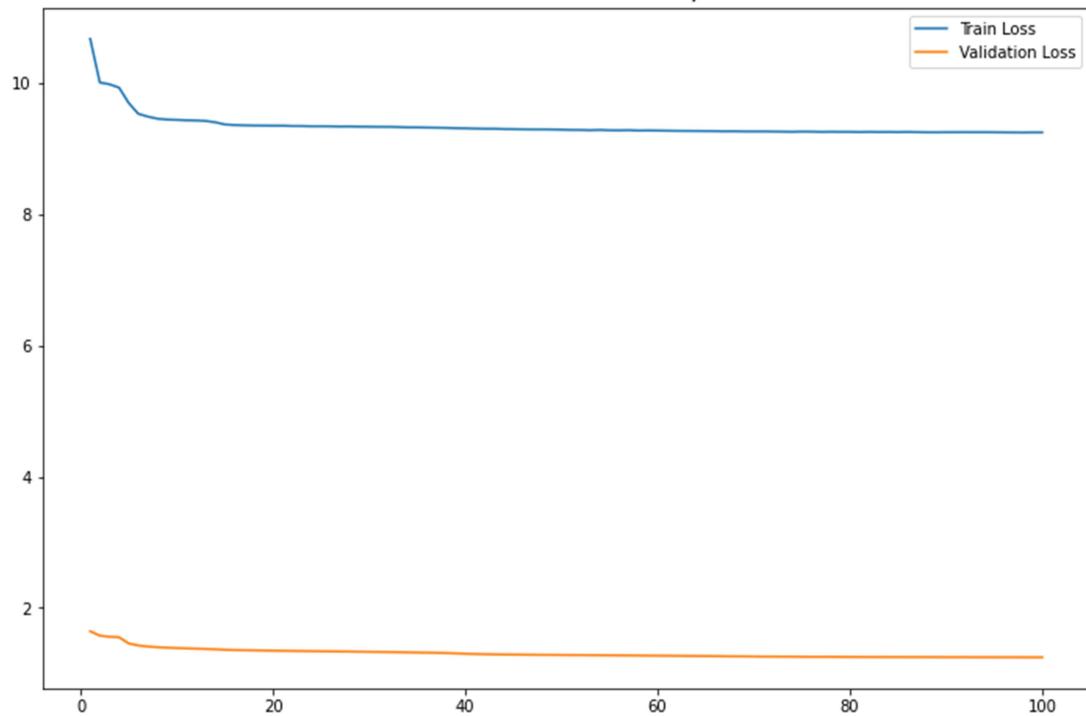


Test accuracy: Balanced Data, Thersholt 0.5

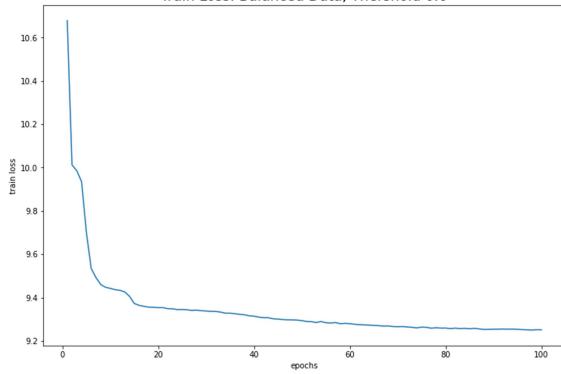


## Threshold = 0.6: Loss Plots

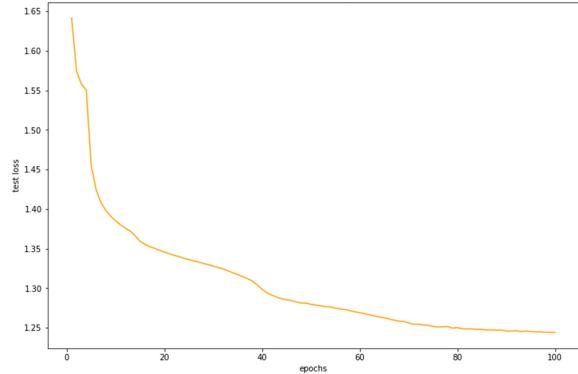
Train & Test Loss: Balanced Data, Thersholt 0.6



Train Loss: Balanced Data, Thersholt 0.6

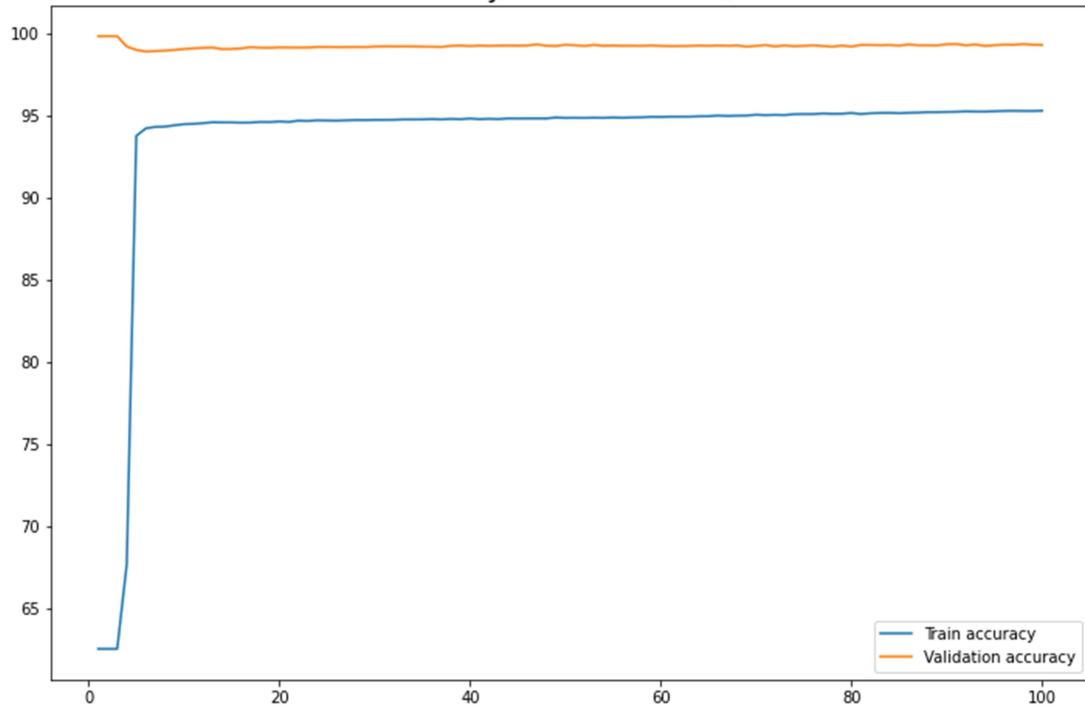


Test Loss: Balanced Data, Thersholt 0.6

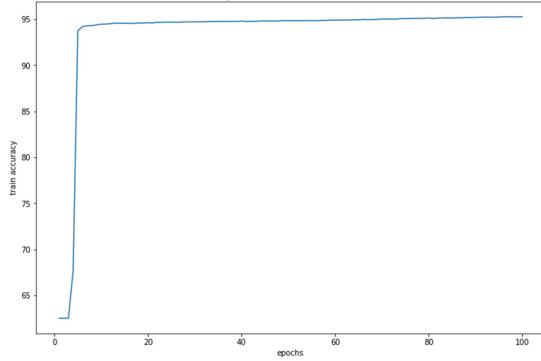


## Threshold = 0.6: Accuracy Plots

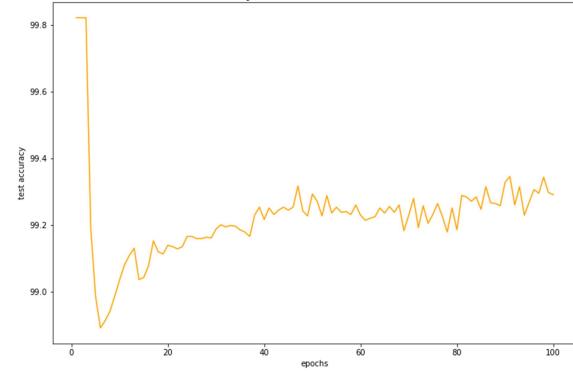
Train & Test accuracy: Balanced Data, Thersholt 0.6



Train accuracy: Balanced Data, Thersholt 0.6

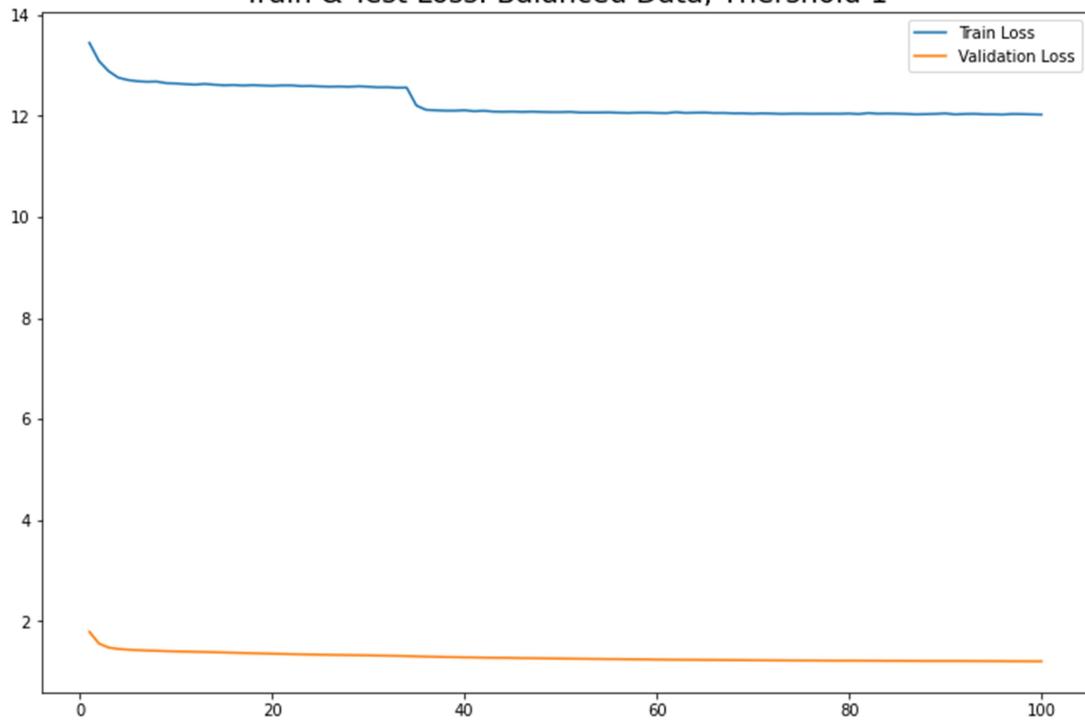


Test accuracy: Balanced Data, Thersholt 0.6

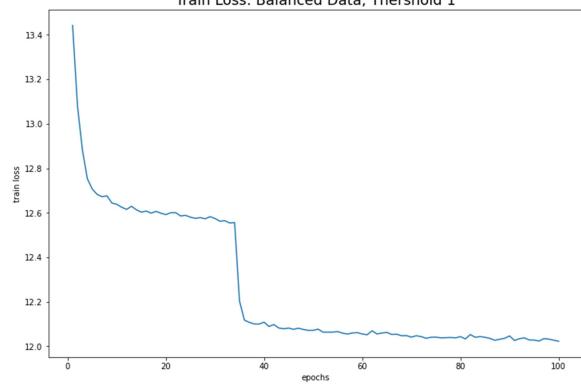


## Threshold = 1: Loss Plots

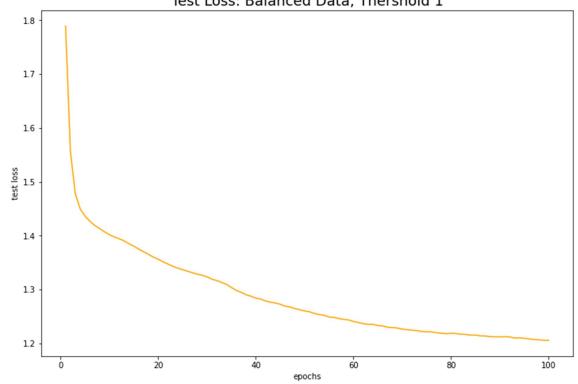
Train & Test Loss: Balanced Data, Thersholt 1

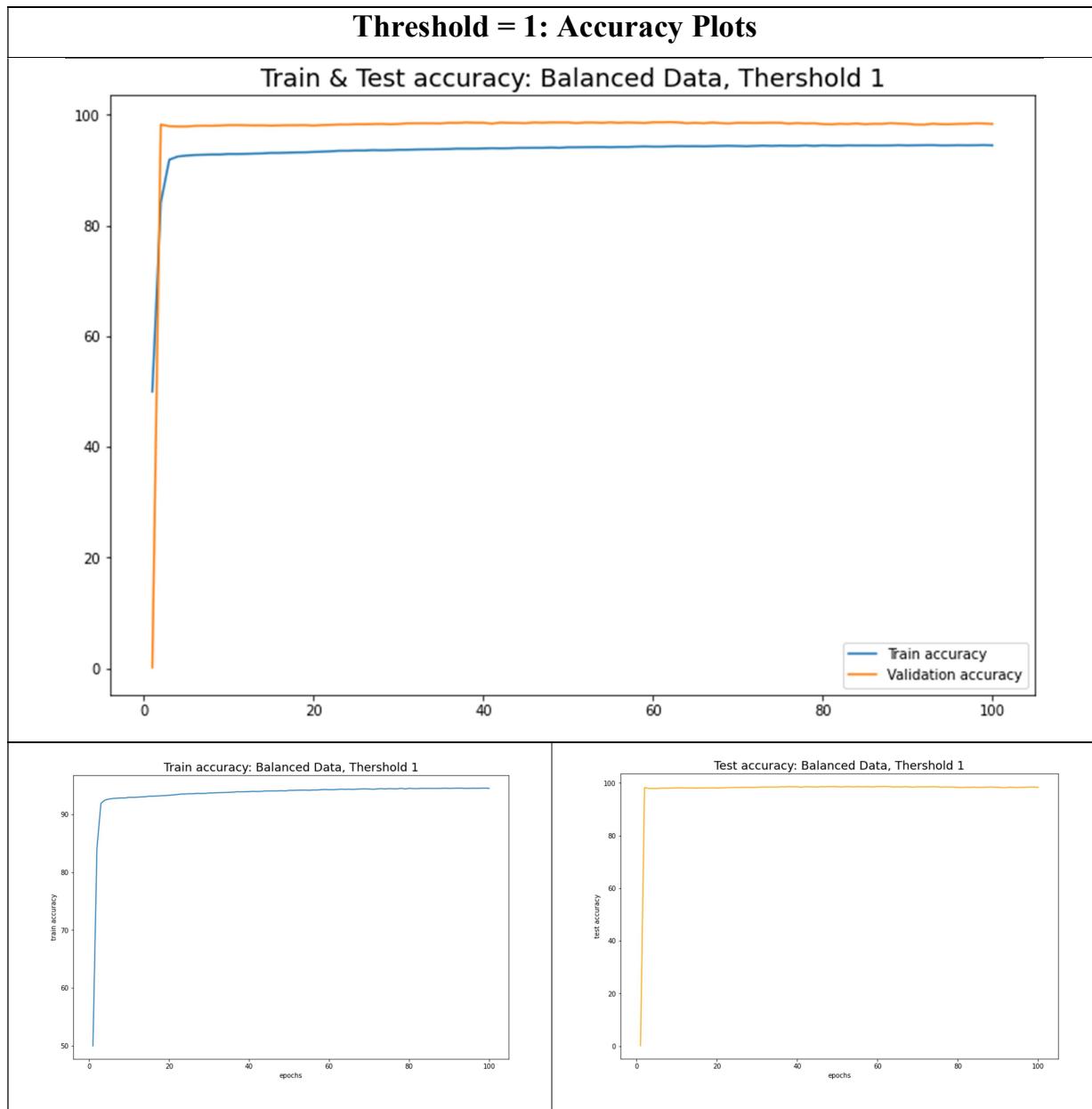


Train Loss: Balanced Data, Thersholt 1



Test Loss: Balanced Data, Thersholt 1





شکل ۵. نمودار loss و accuracy بدهست آمده برای داده های آموزش و validation با threshold های مختلف

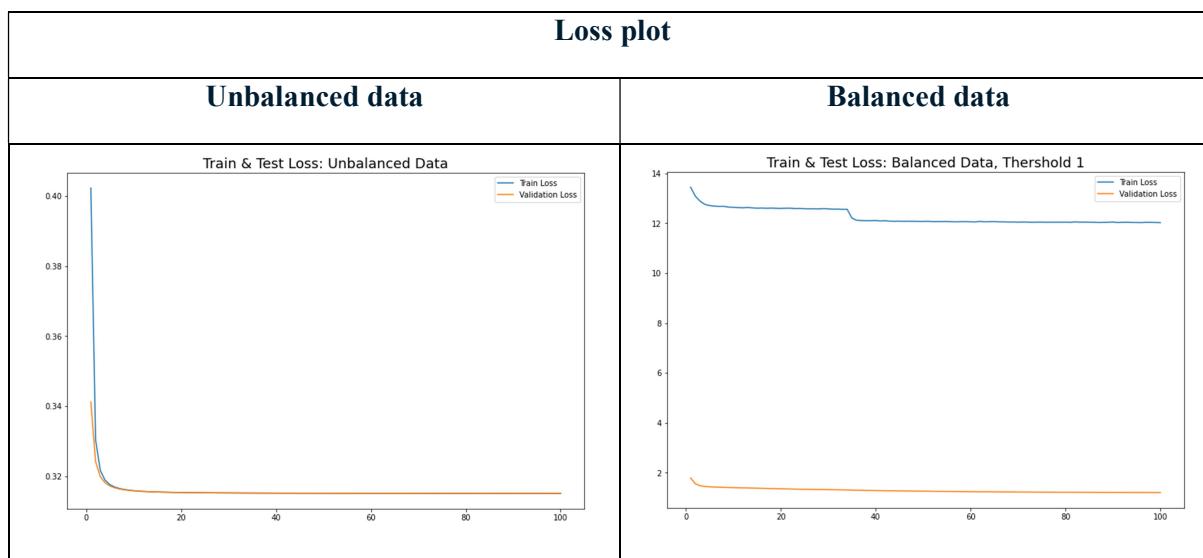
## .۱-۷

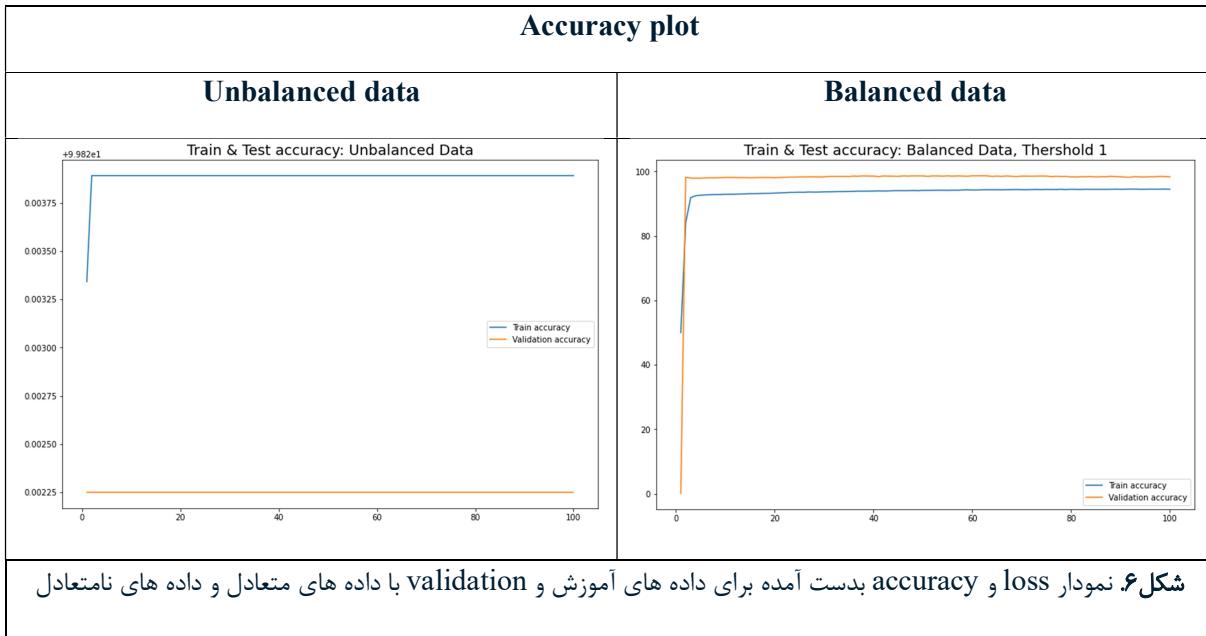
در این بخش از سوال خواسته شده که داده ها را unbalanced و بانویز استفاده کنیم، در نتیجه از oversampling برای balance کردن داده ها و همچنین از denoise برای autoencoder استفاده نشده و داده های نامتعادل را به طبقه بند داده شده است. خروجی دریافت شده به شرح زیر می باشد:

جدول ۷. نتایج شبکه برای داده های تست با استفاده از داده های **unbalanced** و **balanced**

| Classification result | Unbalanced data | Balanced data |
|-----------------------|-----------------|---------------|
| Test Accuracy         | 0.998           | 0.989         |
| Recall score          | <u>0.5</u>      | <u>0.950</u>  |
| F1-score              | 0.499           | 0.605         |
| Precision             | 0.498           | 0.561         |

همانطور که در جدول ۷ دیده می شود، از مقایسه نتایج مدل در حالتی که از داده های متعادل با حالتی که از داده های نامتعادل استفاده شده پیداست که نمی توان به دقت برای ارزیابی کارایی مدل تکیه کرد. در هر دو حالت تقریباً دقت مدل برابر با ۹۹ درصد بوده است هرچند با متعادل کردن داده ها می بینیم که recall score بسیار ارتقا یافته، یعنی مدل می تواند داده های ناهنجار بیشتری را تشخیص دهد.





## پاسخ ۳ - عنوان پرسش سوم به فارسی

### ۱-۳. تفاوت بین شبکه‌های CNN و DCNN

شبکه‌های Deep CNN از تعداد لایه‌های بیشتری استفاده می‌کند که باعث می‌شود توانایی آن برای استخراج ویژگی بسیار بیشتر شود در نتیجه دقت مدل هم برای خیلی از کاربردها بیشتر است. البته در این شبکه‌ها باید مراقب بود که مدل overfit نشود. در اکثر کاربردها نیاز است که از DCNN استفاده کرد و از چند ده لایه استفاده کرد تا به خروجی دلخواه رسید.

### ۲-۳. سه روش بهینه سازی

• Momentum: مانند SGD ساده است فقط به تابع گرادیان آن تکانه اضافه می‌کنیم. یعنی گرادیان فعلی به گرادیان‌های قبلیش وابسته باشد. این باعث می‌شود شبکه سریع‌تر از SGD معمولی همگرا شود. در شکل زیر تفاوت همگرایی آن‌ها مشهود است:

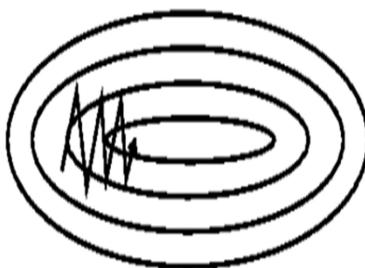


Image 2: SGD without momentum

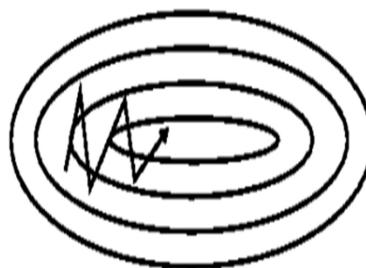


Image 3: SGD with momentum

#### ۱. تفاوت همگرایی Momentum و SGD

و فرمول آپدیت وزن و بایاس آن به صورت زیر است.

$$w_t = w_{t-1} - \eta V_{dw_t}$$

$$\text{where } V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta) \frac{\partial L}{\partial w_{t-1}}$$

$$b_t = b_{t-1} - \eta V_{db_t}$$

$$\text{where } V_{db_t} = \beta V_{db_{t-1}} + (1-\beta) \frac{\partial L}{\partial b_{t-1}}$$

• Adadelta: بهینه سازی Adadelta یک روش نزولی گرادیان تصادفی است که بر اساس

نرخ یادگیری تطبیقی در هر بعد برای رفع دو اشکال است:

الف. کاهش مداوم نرخ یادگیری در طول آموزش.

ب. نیاز به نرخ یادگیری جهانی انتخاب شده به صورت دستی.

Adadelt یک توسعه قوی تر از Adagrad است که به جای انباشته کردن همه گرادیان های گذشته، نرخ های یادگیری را بر اساس یک پنجره متحرک به روز رسانی گرادیان تطبیق می دهد. به این ترتیب، Adadelta حتی زمانی که به روز رسانی های زیادی انجام شده است، به یادگیری ادامه می دهد. در مقایسه با Adagrad، در نسخه اصلی Adadelta شما نیازی به تعیین نرخ یادگیری اولیه ندارید.

$$\eta_t' = \frac{\eta}{\sqrt{S_{dw_t} + \epsilon}} \text{ where } S_{dw_t} = \beta S_{dw_{t-1}} + (1-\beta) \left( \frac{\partial L}{\partial w_{t-1}} \right)^2$$

$\epsilon$  is a small + ve number to avoid divisibilty by 0

• Adams: الگوریتم آدام را می توان به عنوان ترکیبی از RMSprop و گرادیان نزولی

تصادفی با تکانه (Momentum) در نظر گرفت. در الگوریتم بهینه سازی آدام ، هم

مزیت های الگوریتم AdaGrad و هم نقاط قوت الگوریتم RMSProp به کار گرفته

شده اند. در آدام به جای انطباق نرخ های یادگیری پارامترها تنها بر اساس میانگین

گشتاور اول (یعنی Mean) مانند آنچه در الگوریتم RMSProp انجام شده است، از

میانگین گشتاور دوم گرادیان ها (واریانس غیر مرکزی) هم استفاده می شود. الگوریتم

آدام منحصراً میانگین متحرک نمایی گرادیان و گرادیان مربعات را محاسبه می کند و

پارامترهای beta1 و beta2 نرخ فروپاشی (خرابی) این میانگین های متحرک را کنترل

می کنند. داری چهار پارامتر alpha, beta1,beta2,epsilon است که آلفا نرخ یادگیری،

بta یک نرخ فروپاشی نمایی برای تخمین تکانه اول، و بتا دو نرخ فروپاشی نمایی برای

تخمین تکانه دوم است. اپسیلون هم برای جلوگیری از تقسیم بر صفر استفاده می شود.

میانگین وزن نمایی برای گرادیان های قبلی:

$$V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta) \frac{\partial L}{\partial w_{t-1}}$$

$$V_{db_t} = \beta V_{db_{t-1}} + (1-\beta) \frac{\partial L}{\partial b_{t-1}}$$

میانگین ورن دار نمایی مجدور گرادیان های قبلی:

$$S_{dw_t} = \gamma S_{dw_{t-1}} + (1-\gamma) \left( \frac{\partial L}{\partial w_{t-1}} \right)^2$$

$$S_{db_t} = \gamma S_{db_{t-1}} + (1-\gamma) \left( \frac{\partial L}{\partial b_{t-1}} \right)^2$$

از معادلات بالا برای آپدیت وزن و بایاس استفاده می کنیم:

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{S_{dw_t} - \varepsilon}} * V_{dw_t}$$

$$b_t = b_{t-1} - \frac{\eta}{\sqrt{S_{db_t} - \varepsilon}} * V_{db_t}$$

مزیت ها: پیاده سازی ساده، حافظه کم مورد نیاز، مناسب برای مسائل نویزی و تنک، تنظیم راحت های پرپارامترها به خاطر بصری بودن

### ۳-۳. پیاده سازی معماری DCNN

از پایتوج در محیط kaggle استفاده شد. برای اجرا از GPU-T4 استفاده کردیم. دیتاست هم در کگل با نام Persian-numbers موجود بود که از همان استفاده کردیم. برای هر بهینه ساز ۲۰ ایپاک آموزش می دهیم.

## پیش پردازش‌ها و نرم‌السازی‌ها:

- تغییر سایز عکس به  $40 \times 40$ : تا بتوان مدل را روی آن به درستی اجرا کرد اگه سایز کوچتر باشد به دلیل maxpooling اوسط مدل سایز عکس صفر می‌شود و خطای دهد.
- نرم‌ال کردن مقدار پیکسل‌ها: باید مقادیر پیکسل‌ها بین ۰ تا ۲۵۵ قرار گیرد.
- باینری کردن عکس: پردازش عکس خاکستری زمان زیادی نیاز دارد برای همین عکس را با توجه به یک آستانه مثل ۰.۵ به سیاه و سفید تبدیل می‌کنیم.
- نرم‌ال سازی بجها: برای سریع‌تر شدن آموزش بعد از هر لایه کانولوشنال از آن استفاده می‌کنیم.

خلاصه معماری در جدول زیر آورده شده است.

| Layer (type:depth-idx) | Output Shape       | Param #   |
|------------------------|--------------------|-----------|
| Sequential: 1-1        | $[1, 64, 19, 19]$  | --        |
| └ Conv2d: 2-1          | $[1, 64, 40, 40]$  | 640       |
| └ ReLU: 2-2            | $[1, 64, 40, 40]$  | --        |
| └ BatchNorm2d: 2-3     | $[1, 64, 40, 40]$  | 128       |
| └ MaxPool2d: 2-4       | $[1, 64, 19, 19]$  | --        |
| └ Dropout: 2-5         | $[1, 64, 19, 19]$  | --        |
| Sequential: 1-2        | $[1, 128, 9, 9]$   | --        |
| └ Conv2d: 2-6          | $[1, 128, 19, 19]$ | 73,856    |
| └ ReLU: 2-7            | $[1, 128, 19, 19]$ | --        |
| └ BatchNorm2d: 2-8     | $[1, 128, 19, 19]$ | 256       |
| └ Conv2d: 2-9          | $[1, 128, 19, 19]$ | 147,584   |
| └ ReLU: 2-10           | $[1, 128, 19, 19]$ | --        |
| └ BatchNorm2d: 2-11    | $[1, 128, 19, 19]$ | 256       |
| └ MaxPool2d: 2-12      | $[1, 128, 9, 9]$   | --        |
| └ Dropout: 2-13        | $[1, 128, 9, 9]$   | --        |
| Sequential: 1-3        | $[1, 256, 4, 4]$   | --        |
| └ Conv2d: 2-14         | $[1, 256, 9, 9]$   | 295,168   |
| └ ReLU: 2-15           | $[1, 256, 9, 9]$   | --        |
| └ BatchNorm2d: 2-16    | $[1, 256, 9, 9]$   | 512       |
| └ Conv2d: 2-17         | $[1, 256, 9, 9]$   | 590,080   |
| └ ReLU: 2-18           | $[1, 256, 9, 9]$   | --        |
| └ BatchNorm2d: 2-19    | $[1, 256, 9, 9]$   | 512       |
| └ MaxPool2d: 2-20      | $[1, 256, 4, 4]$   | --        |
| └ Dropout: 2-21        | $[1, 256, 4, 4]$   | --        |
| Sequential: 1-4        | $[1, 512, 1, 1]$   | --        |
| └ Conv2d: 2-22         | $[1, 512, 4, 4]$   | 1,180,160 |
| └ ReLU: 2-23           | $[1, 512, 4, 4]$   | --        |
| └ BatchNorm2d: 2-24    | $[1, 512, 4, 4]$   | 1,024     |
| └ Conv2d: 2-25         | $[1, 512, 4, 4]$   | 2,359,808 |
| └ BatchNorm2d: 2-26    | $[1, 512, 4, 4]$   | 1,024     |
| └ MaxPool2d: 2-27      | $[1, 512, 1, 1]$   | --        |

|                     |                   |         |
|---------------------|-------------------|---------|
| └ Dropout: 2-28     | [ -1, 512, 1, 1 ] | --      |
| └ Sequential: 1-5   | [ -1, 10 ]        | --      |
| └ Linear: 2-29      | [ -1, 1024 ]      | 525,312 |
| └ ReLU: 2-30        | [ -1, 1024 ]      | --      |
| └ BatchNorm1d: 2-31 | [ -1, 1024 ]      | 2,048   |
| └ Dropout: 2-32     | [ -1, 1024 ]      | --      |
| └ Linear: 2-33      | [ -1, 10 ]        | 10,250  |
| └ Softmax: 2-34     | [ -1, 10 ]        | --      |

Total params: 5,188,618  
Trainable params: 5,188,618  
Non-trainable params: 0  
Total mult-adds (M): 214.78

شبکه از ۴ بلاک کانولوشنی و یک بلاک فولی کانکت تشكیل شده است.

هر بلاک کانولوشنی شامل یک یا دو لایه کانولوشن و یک لایه max-pooling است. که مجموعاً ۱۱ لایه کانولوشن و max-pool استفاده شده است و دو لایه فولی کانکت.

لایه‌های کانولوشن: وظیفه استخراج ویژگی از ورودی داده شده را دارند روی ورودی فیلتر می‌زنن و آن را با توجه به مقدار stride روی ورودی جابه‌جا می‌کنند. هر چه این لایه کانولوشنال در لایه‌های آخر تر باشد ویژگی‌های کلی‌تر استخراج می‌کند.

تابع فعال ساز: از آن برای غیر خطی کردن خروجی استفاده می‌کنیم تا قدرت و انعطاف پذیری شبکه بیشتر شود اگر استفاده نکنیم مدل ما با یک مدل خطی فرقی نمی‌کند چون کنار هم گذاشتن چند مدل خطی کنار هم یک مدل خطی می‌دهد.

لایه‌های max\_pool: از یک پنجره بزرگترین مقدار را انتخاب می‌کند. در واقع این کار feature selection است. این باعث می‌شود تعداد پارامترها کمتر شود و شبکه سریع‌تر آموزش ببیند همچنین از overfitting جلوگیری می‌کند.

لایه‌های فولی کانکت: وظیفه آن‌ها Classification ورودی‌ها است.

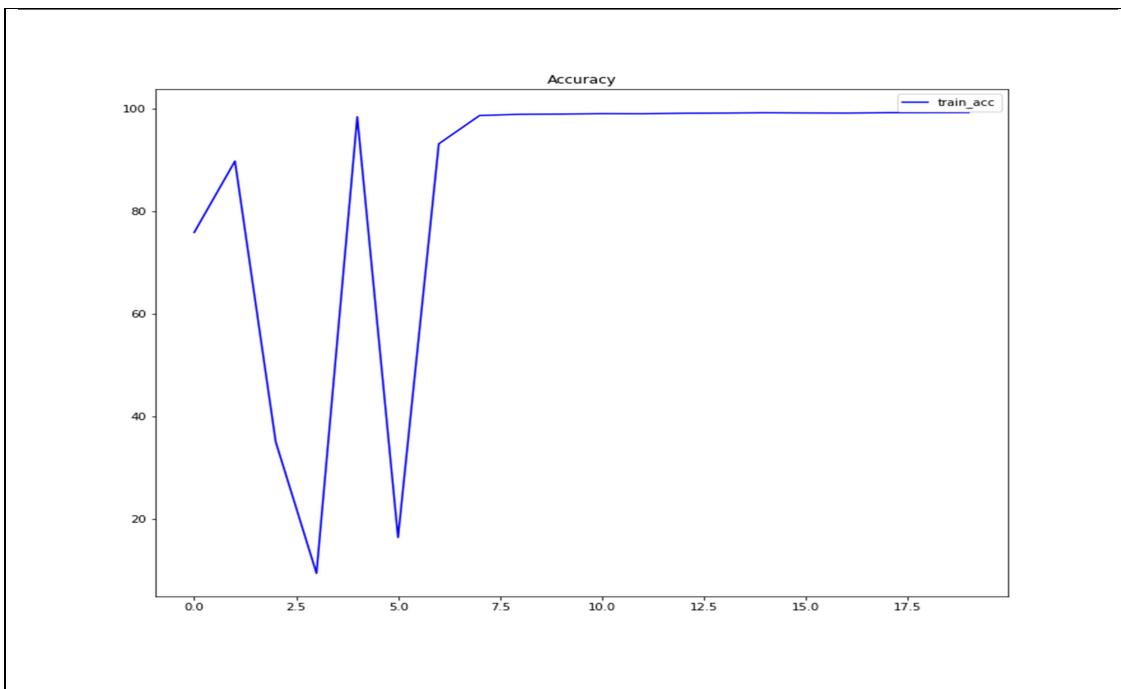
جلوگیری از overfitting: دو کار مهم انجام داده شده است. اول استفاده از dropout که خیلی از نورون‌ها یا خونه‌های شبکه را غیر فعال می‌کند و همین باعث می‌شود که شبکه وابسته به نورن یا خونه خاصی نباشد و از طرقی از قدرت شبکه کمی می‌کاهد که این باعث می‌شود مدل کمتر نویز رو مدل کنه.

راه دیگر هم استفاده از max\_pooling از feature selection هم جلوگیری می‌کند چون عملاً از یکسری دیتا استفاده نمی‌کند و این هم قدرت مدل را در مدل کردن نویز کاهش می‌دهد.

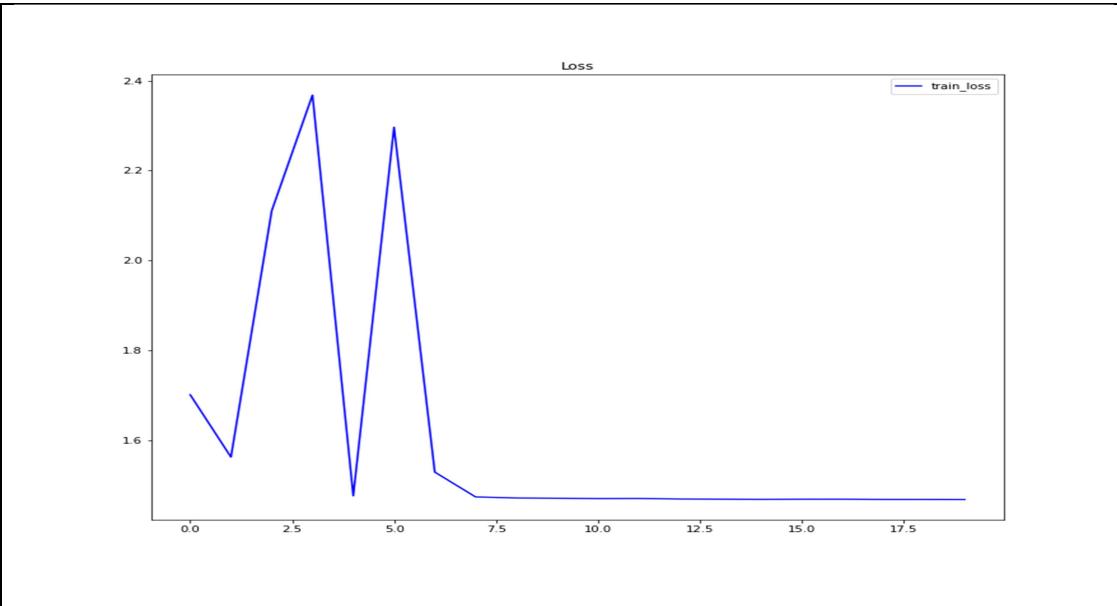
#### ۴-۳. رسم نمودارها و معیارها

الف. Adam

:Accuracy نمودار



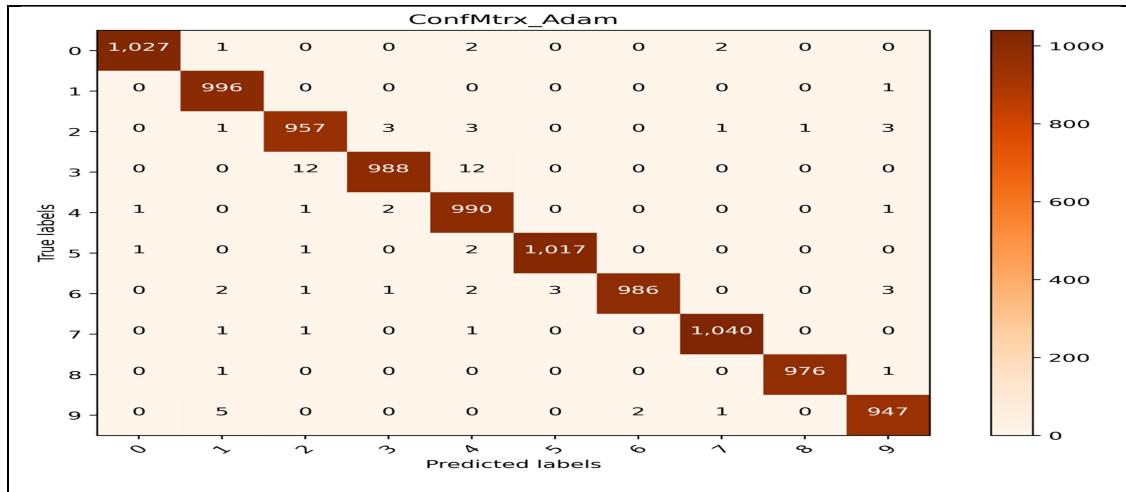
:loss نمودار



:f1-score, Recall, Precision مقادیر

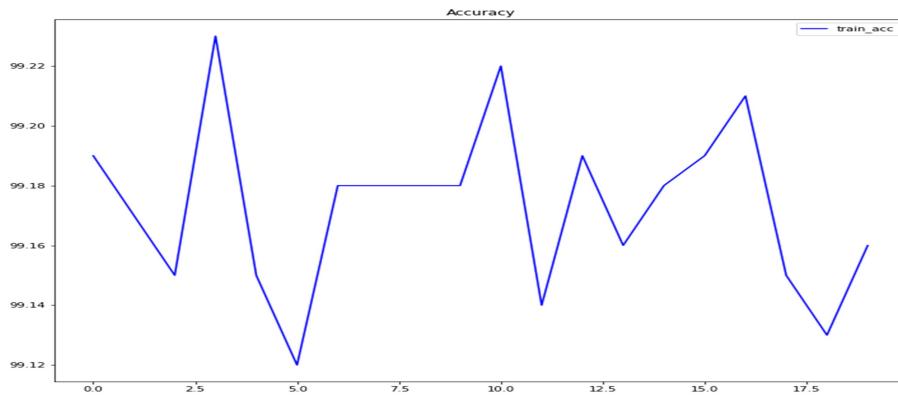
|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 1.00      | 1.00   | 1.00     | 1029    |
| 1            | 1.00      | 0.99   | 0.99     | 1007    |
| 2            | 0.99      | 0.98   | 0.99     | 973     |
| 3            | 0.98      | 0.99   | 0.99     | 994     |
| 4            | 0.99      | 0.98   | 0.99     | 1012    |
| 5            | 1.00      | 1.00   | 1.00     | 1020    |
| 6            | 0.99      | 1.00   | 0.99     | 988     |
| 7            | 1.00      | 1.00   | 1.00     | 1044    |
| 8            | 1.00      | 1.00   | 1.00     | 977     |
| 9            | 0.99      | 0.99   | 0.99     | 956     |
| accuracy     |           |        | 0.99     | 10000   |
| macro avg    | 0.99      | 0.99   | 0.99     | 10000   |
| weighted avg | 0.99      | 0.99   | 0.99     | 10000   |

:confusion matrix نمودار

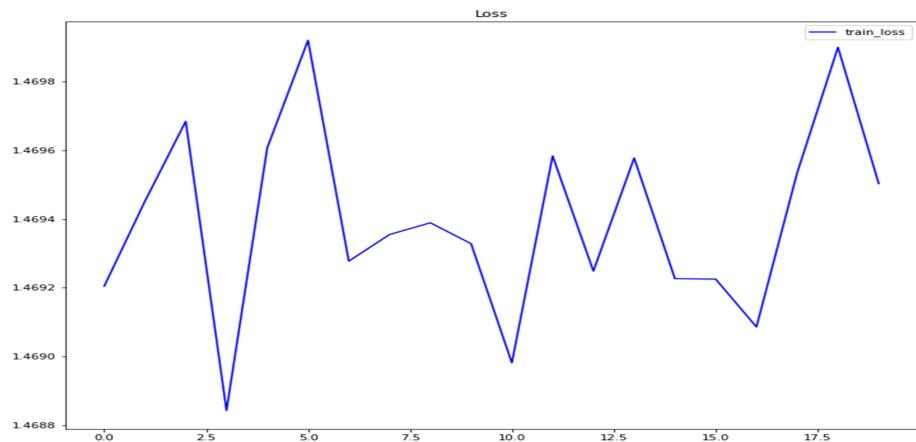


• Momentum .

:Accuracy نمودار



:loss نمودار



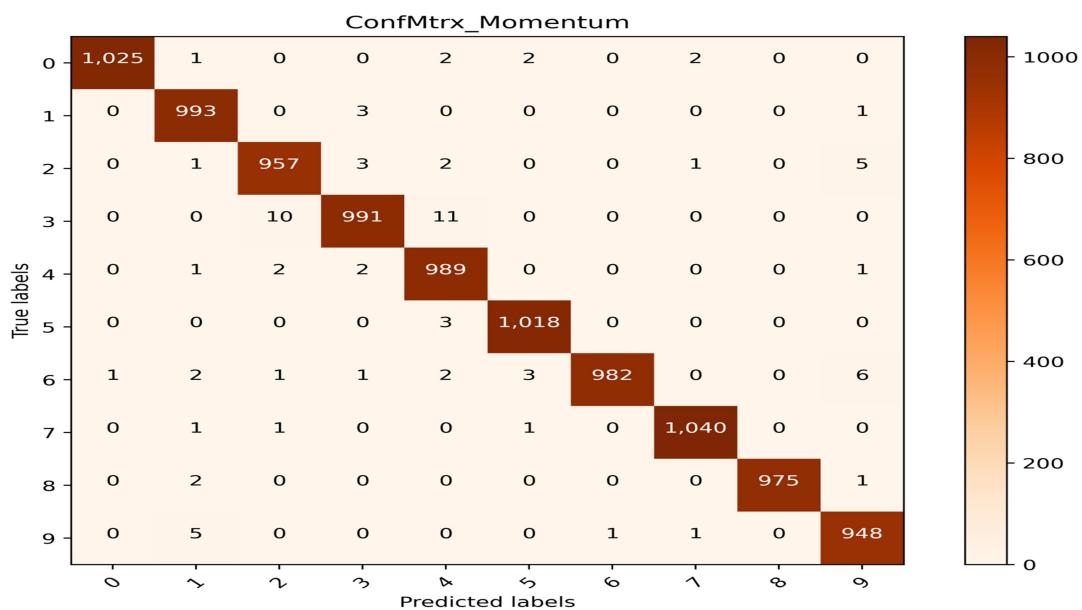
:f1-score, Recall, Precison مقادیر

precision recall f1-score support

|   | precision | recall | f1-score | support |
|---|-----------|--------|----------|---------|
| 0 | 0.99      | 1.00   | 1.00     | 1026    |
| 1 | 1.00      | 0.99   | 0.99     | 1006    |
| 2 | 0.99      | 0.99   | 0.99     | 971     |
| 3 | 0.98      | 0.99   | 0.99     | 1000    |
| 4 | 0.99      | 0.98   | 0.99     | 1009    |
| 5 | 1.00      | 0.99   | 1.00     | 1024    |
| 6 | 0.98      | 1.00   | 0.99     | 983     |
| 7 | 1.00      | 1.00   | 1.00     | 1044    |
| 8 | 1.00      | 1.00   | 1.00     | 975     |
| 9 | 0.99      | 0.99   | 0.99     | 962     |

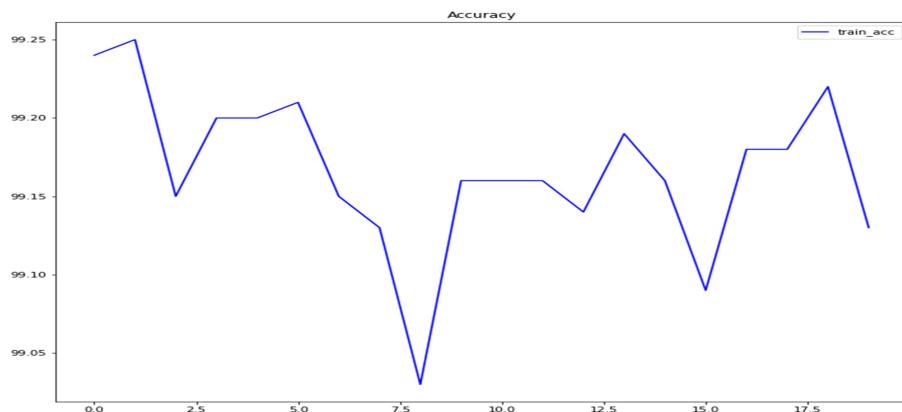
|              |      |      |       |       |
|--------------|------|------|-------|-------|
| accuracy     | 0.99 | 0.99 | 10000 |       |
| macro avg    | 0.99 | 0.99 | 0.99  | 10000 |
| weighted avg | 0.99 | 0.99 | 0.99  | 10000 |

:confusion matrix نمودار

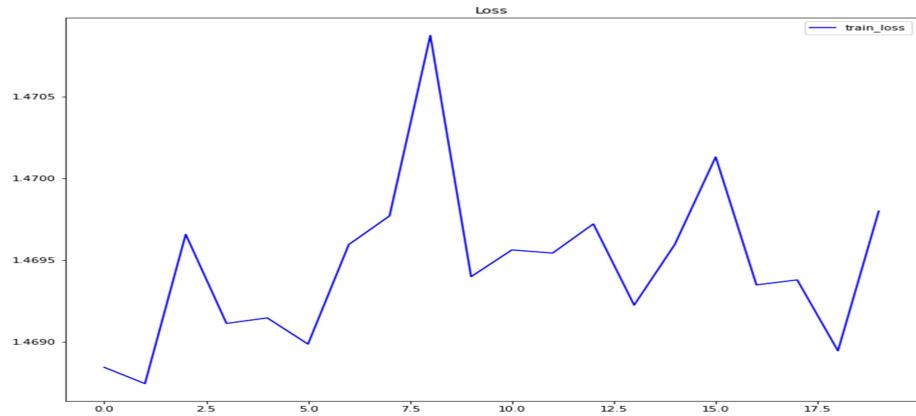


الف. Adaline

:Accuracy نمودار



:loss نمودار



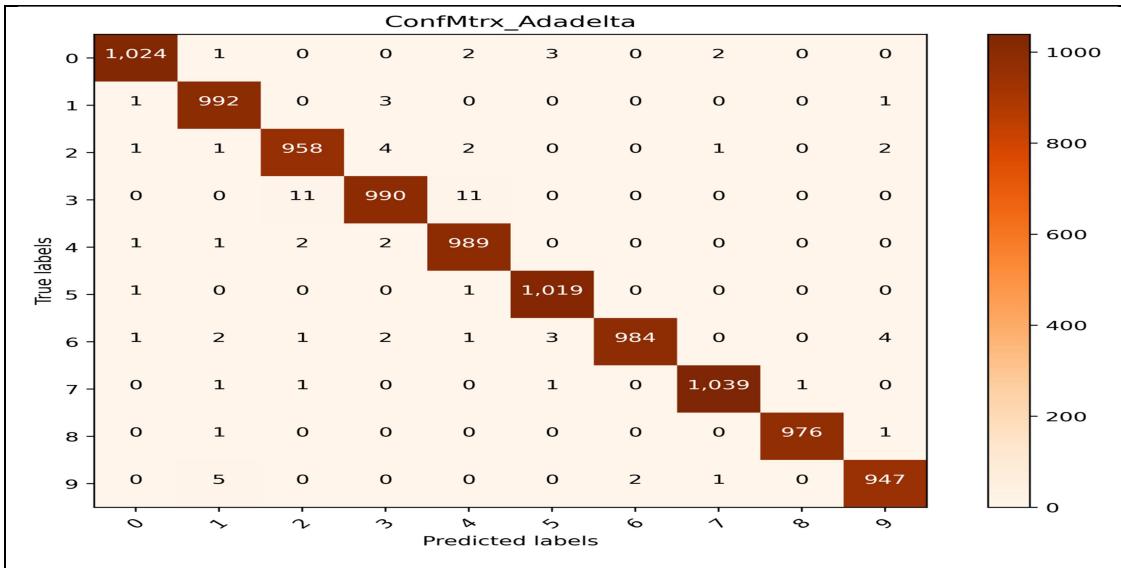
:f1-score, Recall, Precison مقادیر

precision recall f1-score support

|   |      |      |      |      |
|---|------|------|------|------|
| 0 | 0.99 | 1.00 | 0.99 | 1029 |
| 1 | 0.99 | 0.99 | 0.99 | 1004 |
| 2 | 0.99 | 0.98 | 0.99 | 973  |
| 3 | 0.98 | 0.99 | 0.98 | 1001 |
| 4 | 0.99 | 0.98 | 0.99 | 1006 |
| 5 | 1.00 | 0.99 | 1.00 | 1026 |
| 6 | 0.99 | 1.00 | 0.99 | 986  |
| 7 | 1.00 | 1.00 | 1.00 | 1043 |
| 8 | 1.00 | 1.00 | 1.00 | 977  |
| 9 | 0.99 | 0.99 | 0.99 | 955  |

|              |      |      |       |       |
|--------------|------|------|-------|-------|
| accuracy     |      | 0.99 | 10000 |       |
| macro avg    | 0.99 | 0.99 | 0.99  | 10000 |
| weighted avg | 0.99 | 0.99 | 0.99  | 10000 |

:confusion matrix نمودار



### ۵-۳. بهترین شبکه

اگه منظور بهترین بهینه ساز است. همه دقتشان در ۲۰ ایپاک ۹۹ درصد است. برای همین حداقل در این تعداد ایپاک فرقی ندارند. البته اگه تعداد ایپاک را هم بالا ببریم باز هم دقت همین است و بیشتر نخواهد شد. مگر اینکه ۱۰۰ درصد شود که بعید است!

در مقاله اما دقت بهینه ساز adam از همه بهتر است و بعد آن adadelta و بدترین آنها momentum است. ولی در پیاده سازی ما شاید به خاطر هایپر پارامترهای بهتر سریع به دقت بالا میرسیم و همه در یک سطح عمل می‌کنند.

و نکته دیگر اینکه برخلاف مقاله بهینه ساز adam از همه دیرتر به دقت بالا می‌رسد و در ابتداء نوسان آن زیاد است. البته این موضوع شاید متاثر از هایپر پارامترها باشد.

در کل به نظر مسئله آنقدر سخت نیست که بشه تفاوت آنها را درست سنجید. و تعداد ایپاک بیشتر هم فقط باعث overfitting می‌شود.