به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین امتیازی

نام و نام خانوادگی	محمد ناصری – مریم عباس‌زاده
شماره دانشجویی	810100406 - 810100486
تاریخ ارسال گزارش	۱۴۰۱.۰۹.۲۸

فهرست

شکل‌ها

جدول‌ها

پاسخ ۱. تشخیص تقلب با استفاده از شبکه عمیق

۱-۱. بزرگترین چالش و راه حل‌ها ؟

چالش‌های متعددی با شناسایی کلاهبرداری با استفاده از کارت اعتباری مرتبط هستند. به طور کلی نمایه رفتار متقلبانه پویا است، یعنی تراکنش‌های متقلبانه شبیه تراکنش‌های قانونی هستند. مجموعه داده های تراکنش کارت اعتباری به ندرت در دسترس و بسیار نامتعادل (یا skewed) هستند. انتخاب بهینه ویژگی (متغیرها) برای مدل ها از دیگر چالش‌هاست همینطور انتخاب معیار مناسب برای ارزیابی عملکرد تکنیک ها بر روی داده های تقلب کارت اعتباری چالش پیش روی دیگر است. عملکرد شناسایی تقلب در کارت اعتباری تا حد زیادی تحت تأثیر نوع روش نمونه گیری مورد استفاده، انتخاب متغیرها و تکنیک (های) شناسایی مورد استفاده قرار می گیرد.

این مقاله به دنبال پیاده سازی تشخیص تقلب در کارت اعتباری با استفاده از Autoencoder حذف نویز و Oversampling است.

Autoencoder یک شبکه عصبی مصنوعی است که برای یادگیری بدون نظارت استفاده می شود. هدف این روش یادگیری بازنمایی برای بازسازی ویژگی‌ها برای مجموعه‌ای از داده‌ها، معمولاً به منظور کاهش ابعاد است. ساده ترین شکل Autoencoder یک شبکه عصبی پیشخور و non-recurrent است که شبیه پرسپترون چند لایه است.

در کنار این موارد مجموعه داده نامتعادل یک مشکل رایج در یادگیری ماشین است، زیرا اکثر مدل‌های طبقه‌بندی یادگیری ماشین سنتی نمی‌توانند مجموعه داده‌های نامتعادل را مدیریت کنند. هزینه طبقه‌بندی اشتباه بالا اغلب در کلاس اقلیت اتفاق می‌افتد، زیرا مدل طبقه‌بندی سعی می‌کند تمام نمونه داده‌ها را به کلاس اکثریت طبقه‌بندی کند.

Oversampling تکنیکی است که برای مقابله با مجموعه داده های نامتعادل استفاده می شود، موضوع آن برای ایجاد نمونه کلاس خاصی است تا توزیع کلاس مجموعه داده اصلی متعادل شود.

۱-۲ معماری شبکه ؟

ابتدا از oversampling برای تبدیل مجموعه داده نامتعادل به مجموعه داده متعادل استفاده میشود. سپس از Autoencoder حذف نویز برای دریافت مجموعه داده حذف نویز شده استفاده شده است. در نهایت با استفاده از مدل شبکه عصبی کاملا متصل عمیق برای طبقه بندی نهایی مدل شده است.

یک Autoencoder ۷ لایه برای فرآیند حذف نویز داده طراحی شده است. پس از اینکه مجموعه داده آموزشی متعادلی را از oversampling بدست آوردیم، نویز گاوسی را به مجموعه داده آموزشی اضافه می‌کنیم، سپس مجموعه داده آموزشی را به این Autoencoder حذف شده وارد می کنیم. پس از آموزش این مدل Autoencoder حذف‌شده، این Autoencoder قابلیت حذف داده‌های آزمایشی در فرآیند پیش‌بینی را دارد.

جدول 1 - مدل برای **Autoencoder** حذف شده

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

یک Autoencoder 6 لایه برای فرآیند حذف صدای داده طراحی شده است. پس از اینکه مجموعه داده آموزشی حذف نویز را از Autoencoder حذف نویز دریافت شد، مجموعه داده آموزشی را به این طبقه‌بندی‌کننده شبکه عصبی کاملا متصل عمیق وارد می‌کنیم. در پایان، از SoftMax با آنتروپی متقابل به عنوان تابع ضرر برای طبقه بندی نهایی استفاده می‌شود.

جدول 2 - مدل برای طبقه بند

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 انواع روش‌های Resampling ؟

مشکل طبقه بندی نامتعادل همان چیزی است که وقتی یک انحراف شدید در توزیع کلاسی داده های آموزشی ما وجود دارد با آن مواجه می شویم. بسیار خوب، انحراف ممکن است بسیار شدید نباشد (می‌تواند متفاوت باشد)، اما دلیل اینکه طبقه‌بندی نامتعادل را به عنوان یک مشکل تشخیص می‌دهیم این است که می‌تواند بر عملکرد تأثیر بگذارد.

کم نمونه برداری از اکثریت (Undersampling the Majority)

کم نمونه برداری را می توان به عنوان کاهش تعداد طبقه اکثریت تعریف کرد. این تکنیک به بهترین وجه برای داده هایی استفاده می شود که در آن هزاران یا میلیون ها نقطه داده دارید. به طور معمول، شما نمی‌خواهید مقدار داده ای را که با آن کار می کنید کاهش دهید، اما اگر بتوانید برخی از داده های آموزشی را قربانی کنید، این تکنیک مفید خواهد بود.

استفاده از این رویکرد در شرایطی مؤثر است که طبقه اقلیت علیرغم عدم تعادل شدید، نمونه های کافی داشته باشد. از سوی دیگر، همیشه مهم است که چشم‌انداز حذف اطلاعات ارزشمند را در نظر بگیریم، زیرا به‌طور تصادفی آنها را از مجموعه داده‌های خود حذف می‌کنیم، زیرا هیچ راهی برای شناسایی یا حفظ نمونه‌هایی که در طبقه اکثریت اطلاعات غنی هستند، نداریم.

نمونه برداری تصادفی بیش از حد (Random Oversampling)

Oversampling تصادفی شامل انتخاب نمونه های تصادفی از کلاس اقلیت با جایگزینی و تکمیل داده های آموزشی با چندین نسخه از این نمونه است، از این رو ممکن است یک نمونه واحد چندین بار انتخاب شود.

هرچند که تنظیم توزیع کلاس هدف در بسیاری از سناریوها توصیه می‌شود، زیرا جستجوی توزیع متعادل برای مجموعه داده‌های به شدت نامتعادل می‌تواند منجر به overfit الگوریتم با کلاس اقلیت شود و در نتیجه منجر به افزایش خطای generalization ما شود.

نکته دیگری که باید از آن آگاه باشیم افزایش هزینه محاسباتی است. افزایش تعداد نمونه‌ها در کلاس اقلیت (مخصوصاً برای مجموعه داده‌های با انحراف شدید) ممکن است منجر به افزایش محاسباتی و زمانی که مدل خود را آموزش می‌دهیم، بشود و با توجه به اینکه مدل چندین بار نمونه‌های مشابه را می‌بیند، این چیز خوبی نیست. با این وجود، Oversampling یک راه حل بسیار مناسب است و باید آزمایش شود.

SMOTE (تکنیک نمونه برداری بیش از حد اقلیت مصنوعی)

SMOTE نقاط داده را از مجموعه موجود کلاس اقلیت ترکیب می کند و آنها را به مجموعه داده اضافه می کند. این تکنیک با ایجاد نقاط داده جدید و نادیده برای آموزش مدل، نشت داده بسیار کمی را تضمین می‌کند.

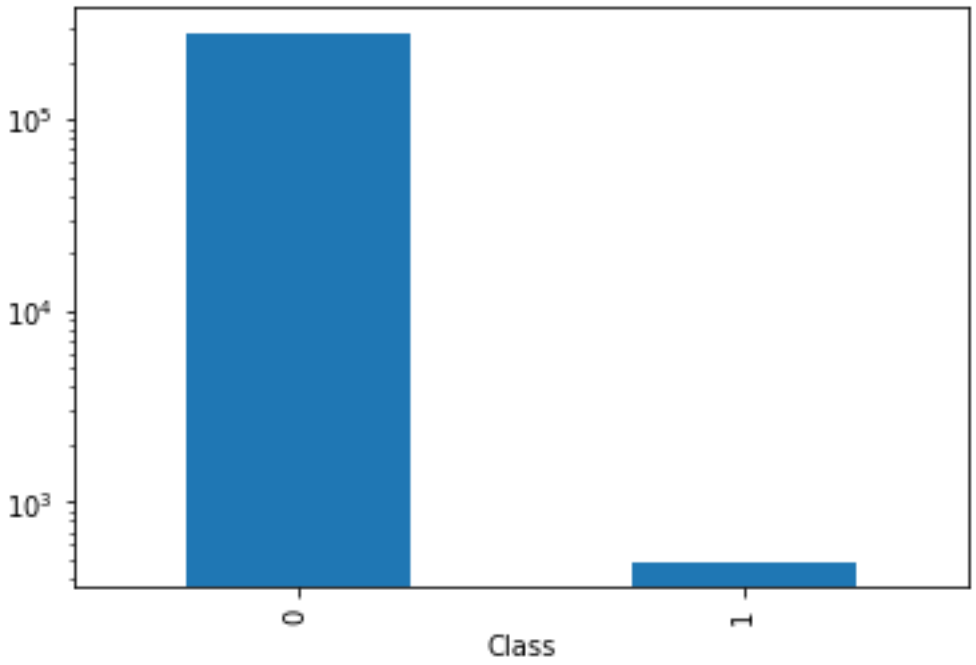
ترکیب هر دو روش نمونه گیری تصادفی

ترکیب هر دو روش نمونه‌گیری تصادفی می‌تواند گاهی منجر به بهبود کلی عملکرد در مقایسه با روش‌هایی شود که به صورت مجزا انجام می‌شوند. مفهوم این است که ما می‌توانیم مقدار کمی از نمونه‌گیری بیش‌ازحد

را برای کلاس اقلیت اعمال کنیم، که سوگیری را برای نمونه‌های کلاس اقلیت بهبود می‌بخشد، در حالی که مقدار کمی از undersampling را روی کلاس اکثریت برای کاهش bias در نمونه‌های کلاس اکثریت انجام می‌دهیم.

۱-۴ پیاده‌سازی مدل

داده مورد نظر با توجه به نمودار زیر دارای تعداد بسیار زیادی از کلاس صفر(معمولی) و تعداد بسیار کمی از کلاس یک (تقلب) میباشد.



شکل 1 - توزیع کلاس‌ها در مجموعه داده کارت بانکی

۱-۵ ارزیابی مدل

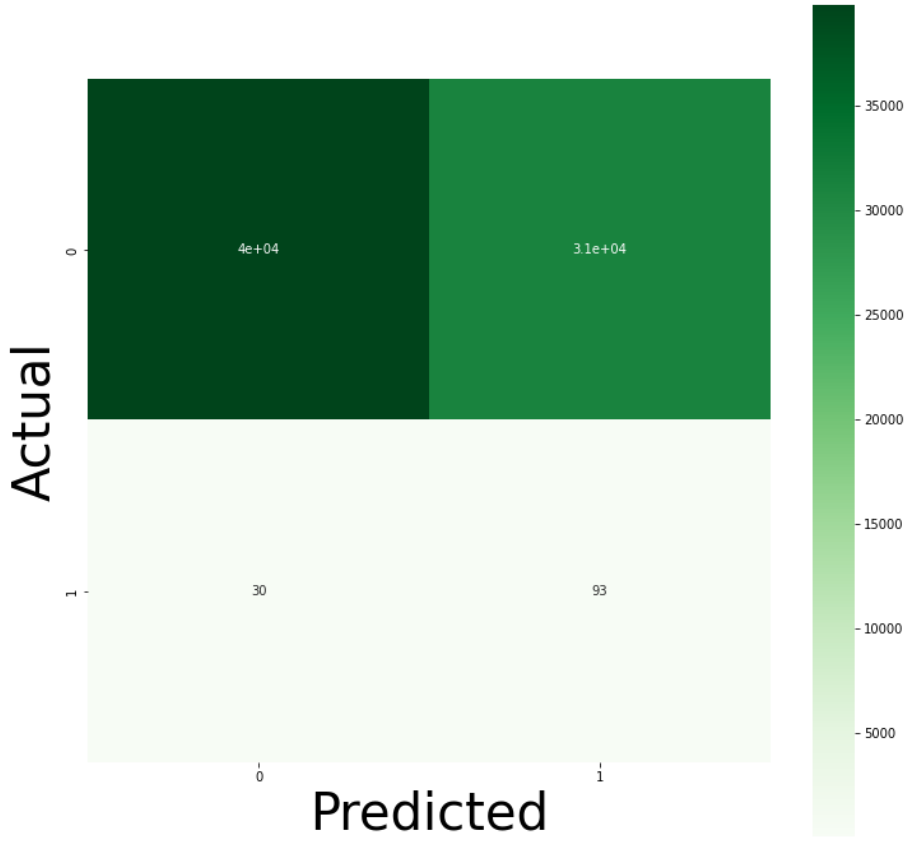
دقت طبقه‌بندی پرکاربردترین معیار برای ارزیابی مدل‌های طبقه‌بندی است. دلیل استفاده گسترده از آن به دلیل محاسبه آسان، تفسیر آسان و یک عدد واحد برای خلاصه کردن قابلیت مدل است.

به این ترتیب، استفاده از آن در مسائل طبقه بندی نامتعادل، که در آن توزیع نمونه ها در مجموعه داده آموزشی در بین کلاس ها برابر نیست، طبیعی است.هنگامی که توزیع کلاس کمی منحرف است، دقت هنوز می تواند یک معیار مفید باشد. هنگامی که انحراف در توزیع های کلاس شدید باشد، Accuracy می تواند به یک معیار غیر قابل اعتماد برای عملکرد مدل تبدیل شود. بسیاری از مدل‌های یادگیری ماشین بر اساس فرض توزیع کلاس متوازن طراحی شده‌اند و اغلب قوانین ساده (صریح یا غیرمستقیم) را مانند

همیشه پیش‌بینی کلاس اکثریت می‌آموزند، که باعث می‌شود به دقت 99 درصد دست یابند، اگرچه در عمل عملکردی بهتر از افراد غیر ماهر ندارند. به همین خاطر از معیارهای دیگر مانند precision, recall, f1 که ترکیبی از دو مورد اول است برای ارزیابی استفاده میشود.

	precision	recall	f1-score	support
0	1.00	0.56	0.72	71079
1	0.00	0.76	0.01	123
accuracy			0.56	71202
macro avg	0.50	0.66	0.36	71202
weighted avg	1.00	0.56	0.72	71202

شکل 2 - نتایج ارزیابی مدل



شکل 3 - ماتریس کانفیوژن مدل

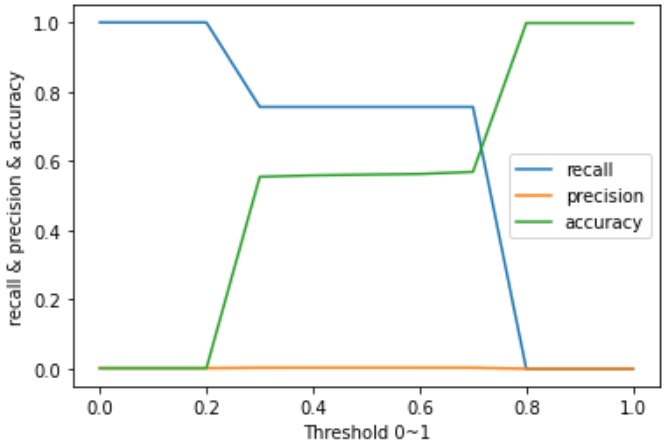
۱-۶ مقایسه آستانه‌های متفاوت

طبق گفته و پیاده‌سازی مقاله برای این کار آستانه‌های متفاوتی را برای محاسبه احتمال assign کردن کلاس‌ها در نظر میگیریم و به همین نسبت مقادیر TP, FP, TN, FN را محاسبه و نمودار مدنظر را رسم میکنیم.

```
evaluate_results = []
for threshold in np.arange(0, 1.01, 0.1):
    TP, FN, FP, TN = 0, 0, 0, 0
    for i in tqdm(range(len(y_test))):
        prediction = predictions[i]
        actual = y_test_cat[i]
        if prediction[1] >= threshold and actual[1] == 1:
            TP += 1
        elif prediction[1] >= threshold and actual[1] == 0:
            FP += 1
        elif prediction[1] < threshold and actual[1] == 1:
            FN += 1
        elif prediction[1] < threshold and actual[1] == 0:
            TN += 1
    result = dict()
    result['threshold'] = threshold
    result['TP'] = TP
    result['FP'] = FP
    result['FN'] = FN
    result['TN'] = TN
    result['recall'] = 0 if TP + FN == 0  else TP / (TP + FN)
    result['precision'] = 0 if TP + FP == 0 else TP / (TP + FP)
    result['accuracy'] = (TP + TN) / (TP + FN + FP + TN)
    evaluate_results.append(result)
    print(result)
```

شکل 4 - پیاده سازی مقایسه آستانه

نتایج به صورت زیر بودند: (علت تفاوت شکل در نظر گرفتن گام‌های ۰.۱ به جای ۰.۰۱ مقاله برای مقادیر آستانه احتمال از ۰ تا ۱ است به دلیل کمبود منابع و زمان اجرا ولی روند کلی نمودار درست است)



شکل 5 - نمودار **recall, precision** برای آستانه‌های متفاوت

پاسخ ۳ – تشخیص کاراکتر نوری

۳-۱. تفاوت CNN و DCNN

تفاوت بین CNN و Deep CNN فقط در تعداد لایه ها است. یک مدل یادگیری عمیق زمانی عمیق تر نامیده می شود که تعداد لایه های بیشتری داشته باشید. بنابراین، Deep CNN اساساً CNN با لایه های عمیق تر است. در CNN معمولی، معمولاً 5 تا 10 لایه وجود دارد، در حالی که بیشتر معماری‌های CNN مدرن 30 تا 100 لایه عمق دارند.

.

۳-۲. مقایسه optimizer

Momentum

مشکل SGD این است که در حالی که به دلیل نوسان زیاد سعی می کند به حداقل برسد، نمی توانیم نرخ یادگیری را افزایش دهیم. بنابراین برای همگرایی زمان می برد. در این الگوریتم، ما از میانگین وزنی نمایی برای محاسبه گرادیان استفاده می کنیم و از این گرادیان برای به روز رسانی پارامتر استفاده می کنیم. در SGD با تکانه، ما تکانه را در تابع گرادیان اضافه کرده ایم. منظور من از این است که گرادیان فعلی به گرادیان قبلی خود و غیره وابسته است. این باعث تسریع SGD برای همگرایی سریعتر و کاهش نوسان می شود.

Adadelta

Adadelta توسعه ای از Adagrad است که تلاش می کند تا نرخ یادگیری را به شدت کاهش دهد. ایده پشت Adadelta این است که به‌جای جمع‌بندی تمام گرادیان‌های مجذور گذشته از گام‌های زمانی ۱ تا «t»، چه می‌شود اگر بتوانیم اندازه پنجره را محدود کنیم. به عنوان مثال، محاسبه گرادیان مجذور 10 گرادیان گذشته و میانگین. این را می توان با استفاده از میانگین وزنی نمایی روی گرادیان به دست آورد.

Adam

بهینه ساز Adam یکی از محبوب ترین بهینه سازها است. ایده پشت بهینه ساز Adam استفاده از مفهوم حرکت از "SGD با تکانه" و نرخ یادگیری تطبیقی از "Ada delta" است.

مزیت استفاده از بهینه ساز ADAM:

- ساده برای پیاده سازی
- کارآمد از نظر محاسباتی
- نیاز کم حافظه
- مناسب برای مشکلات با گرادیان های noisy
- فراپارامترها تفسیر بصری دارند و معمولاً نیاز به تنظیم کمی دارند

۳-۳. پیاده‌سازی مدل

مدل مربوطه با توجه به مقاله پیاده شده و گام‌های طی شده به صورت زیر هستند:

- گام پیش پردازش
  - تبدیل تصاویر به عدد
  - تبدیل مقادیر ۲۵۵ به ۱
  - تبدیل سایز تصاویر به ۴۰*۴۰
  - تغییر shape برای آماده سازی ورودی شبکه
- تعریف مدل
  - لایه ورودی: تصاویر ورودی در ابتدا در اولین لایه که به عنوان لایه ورودی شناخته می شود ذخیره می شوند. ابعاد (ارتفاع، عرض) داده های ورودی و تعداد کانال ها (اطلاعات RGB) را مشخص می کند.
  - لایه Conv: لایه کانولوشن از چندین فیلتر برای استخراج ویژگی های اساسی یک تصویر استفاده می کند. حاصل ضرب نقطه ای نورون ورودی m*m و فیلتر n*n که از طریق تصویر ورودی پیچیده شده است با استفاده از فرمول (m-n+1).(m-n+1) محاسبه می شود. تابع فعال سازی ReLU که در لایه پیچیدگی استفاده می شود. تصاویر اغلب متناقض هستند. برای حذف این تناقضات، از تابع ReLU برای تشکیل یک مجموعه منسجم با تابع اجرا (عملیات کانولوشن) استفاده می شود.
  - لایه Pooling: لایه ادغام به عنوان لایه نمونه برداری فرعی نیز شناخته می شود. بین دو لایه کانولوشن قرار می گیرد تا ابعاد تصویر کاهش یابد. علاوه بر این، عملیات ادغام پیچیدگی محاسباتی شبکه را به حداقل می رساند.
  - لایه fully connected: به آن لایه طبقه بندی نیز می گویند که به عنوان آخرین لایه در معماری DCNN اعمال می شود. بردار ورودی به دست آمده پس از تبدیل و ادغام متوالی

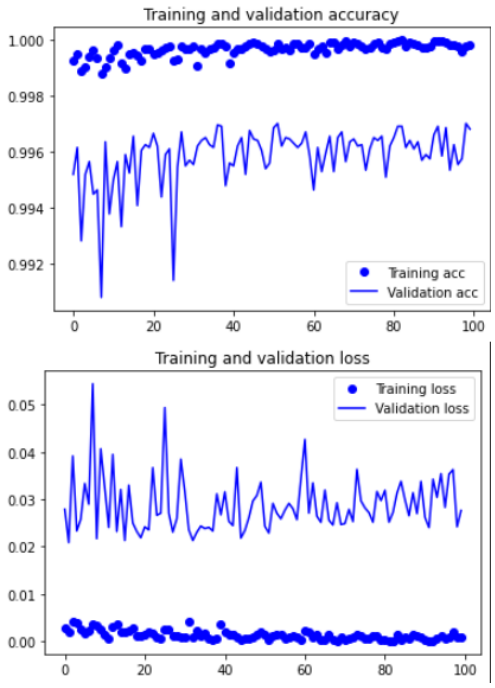
در بخش پایانی کاملا متصل برای طبقه بندی اعداد دست نویس ارائه شده است. تابع Softmax کلاس های پیش بینی شده را با تشخیص تصویر ورودی محاسبه می کند که با یکپارچه سازی تمام ویژگی های استخراج شده توسط لایه های قبلی تکمیل می شود. لایه طبقه‌بندی تابع فعال‌سازی Softmax را برای شناسایی ویژگی‌های ده کلاس تولید شده بر اساس داده‌های آموزشی در کار ما اعمال می‌کند.

- لایه خروجی: لایه خروجی، همچنین به عنوان لایه متراکم شناخته می شود که به ویژه برای پیش بینی استفاده می شود. با توجه به مجموعه داده HODA، تعداد کل کلاس های خروجی 10 است، به این معنی که ما ده نورون خروجی داریم که نشان دهنده یک رقم است.

برای کاهش overfitting ، پس از ادغام لایه‌ها در هر بلوک، از یک لایه حذفی استفاده شد. به طور مشابه، 3 بلوک لایه های پیچیدگی دیگر برای تکمیل پیکربندی مدل انباشته شده اند.لایه pooling همچنین از فرآیند انتخاب ویژگی و کاهش overfitting پشتیبانی می کند.

۳-۴. پیاده سازی مدل با بهیه‌سازهای متفاوت:

در ابتدا مدل را برای adam آموزش داده و نتایج به صورت زیر هستند:



شکل 6 - نمودار دقت و **loss** برای **adam**

مقادیر معیارهای ارزیابی نیز به شکل زیر هستند:

LOSS: 0.03787151724100113,

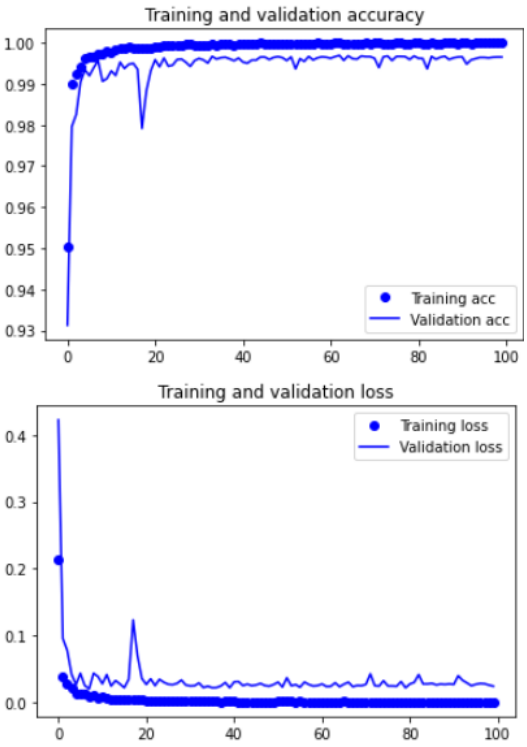
ACCURACY: 0.9950000047683716,

F1_SCORE: 0.9955357313156128,

PRECISION: 0.9955357313156128,

RECALL: 0.9955357313156128

سپس مدل را برای momentum آموزش دادیم ونتایج زیر حاصل شد:



شکل 7 - نمودار دقت و **loss** برای **momentum**

مقادیر معیارها برای داده تست نیز به شرح زیر هستند:

LOSS: 0.008223810233175755,

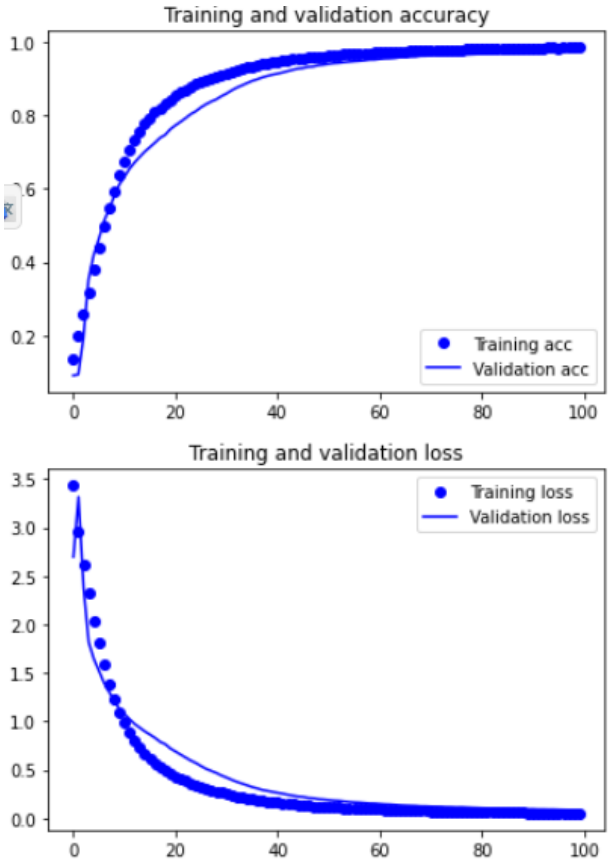
ACCURACY: 0.9950000047683716,

F1_SCORE: 0.9955357313156128,

PRECISION: 0.9955357313156128,

RECALL: 0.9955357313156128

در نهایت مدل برای adadelta آموزش داده میشود که نتایج زیر حاصل شدند:



شکل 8 - نمودار دقت و **Loss** برای **adadelta**

و مقادیر معیارهای ارزیابی برای داده تس به شرح زیر بودند:

LOSS: 0.02179643325507641

ACCURACY: 0.9900000095367432,

F1_SCORE: 0.9910714030265808,

PRECISION: 0.9910714030265808,

RECALL: 0.9910714030265808

و عملکرد بهتر را میتوان به **momentum** با توجه به نتایج به دست آمده نسبت داد.