به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین دوم

نام و نام خانوادگی	مریم دادخواه – جواد سراج
شماره دانشجویی	۸۱۰۱۰۱۱۵۱ - ۸۱۰۱۰۱۱۸۶
تاریخ ارسال گزارش	۱۴۰۲.۰۱.۱۶

فهرست

شکل‌ها

جدول‌ها

پاسخ ۱. شبکه‌ی عصبی پیچشی کم‌عمق برای طبقه‌بندی تصاویر

استفاده از شبکه‌های عصبی عمیق برای طبقه بندی تصاویر پیشرفت چشمگیری داشته است. اما این شبکه‌ها معمولا عمق زیادی دارند و استفاده از آنها نیاز به قدرت پردازشی و حافظه سیستمی زیادی دارد. و همچنین عمیق بودن شبکه‌ها باعث می‌شود که زمان آموزش آنها بسیار بالا باشد. در این سوال شبکه‌ای طراحی می‌گردد که با عمق کم‌تر بتواند با شبکه‌های عمیق‌تر در طبقه‌بندی تصاویر رقابت کند.

۱-۱. پیش پردازش داده‌ها

در مقاله اشاره شد در داده Fashion Mnist با احتمال ۵۰٪ داده‌های آموزش و ارزیابی با یکدیگر جابجا شدند و به عنوان داده‌های آموزش و ارزیابی قرار گرفتند. برای پیش‌پردازش داده‌ها ابتدا تصاویر هر سه دیتاست را پس از لود کردن نرمالایز کردیم. این کار با تقسیم تمامی پیکسل‌های تصاویر به ۲۵۵ و تبدیل astype(np.float32) انجام شد. بنابراین هر تصویر تنسوری است که کوچک‌ترین مقدار آن ۰ و بزرگ ترین مقدار آن ۱ است. این روش نرمالایز کردن در شبکه‌های عصبی از این جهت مفید است که رنج اعداد بین ۰و ۱ قرار می‌گیرد و سرعت محاسباتی بهتر می‌شود. پس از پردازش تصاویر لیبل‌ها از اعداد دسیمال به onehot کد شدند. ده کلاس داده داریم بنابراین هر لیبل یک ارایه به طول ۱۰ است که یکی از داریه‌های آن ۱ و بقیه صفر است. همچنین برای پردازش تصاویر می‌توانسنیم تصاویر را به میانگین صفر برسانیم و مقادیر بین ۱ و ۱- باشند.

۱-۲. توضیح لایه‌های شبکه

لایه‌هایی که در معماری شبکهٔ عصبی استفاده شده‌اند در ادامه معرفی شده اند. همچنین تمرکز اصلی این مقاله بر روی لایه Batch Normalization است که در انتهای این بخش به طور مفصل به معرفی و دلیل استفاده از این لایه پرداخته شده است. لایه Batch Normalization قدرت تعمیم و سرعت همگرایی شبکه عصبی را افزایش می‌دهد و از این جهت می‌تواند برای افزایش دقت طبقه‌بندی مفید باشد.

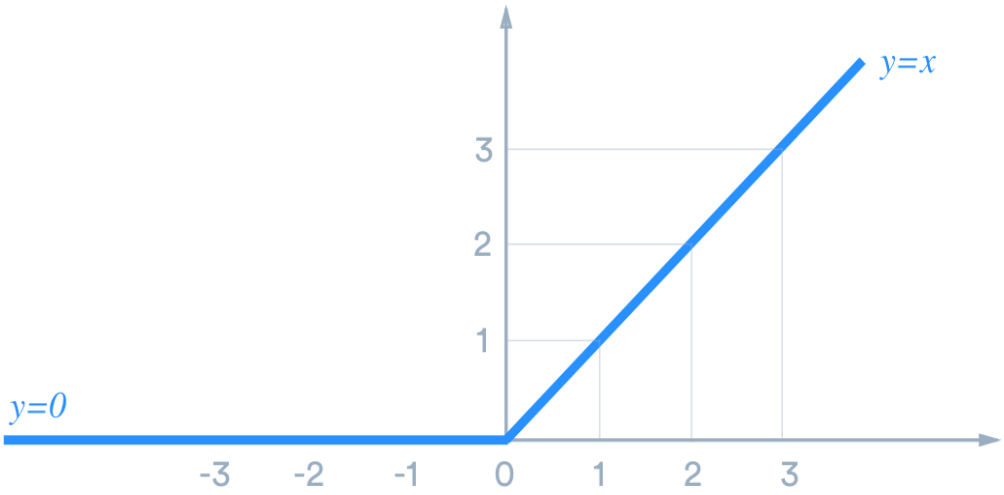
لایه‌های convolution : لایه‌های کانولوشنی وظیفه اصلی استخراج ویژگی از تصویر را بر عهده دارند. کانولوشن‌ها با کرنل‌های مختلف ویژگی‌های مختلفی از داده ورودی استخراج می‌کنند. پارامترهایی که برای لایه کانولوشن قابل تنظیم کردن هستد عبارت اند از:

- اندازه کرنل
- Stride
- تعداد کانال‌های خروجی
- Initial weights

لایه max pooling : این لایه وظیفه کاهش ابعاد را پس از استخراج ویژگی‌ها بر عهده دارد. این لایه از طریق down-sampling کاهش افزونگی بردار ویژگی (تنسور ویژگی) را انجام خواهد داد. این لایه افزونگی و ریسک over-fitting را کاهش می‌دهد و سرعت آموزش را افزایش می‌دهد. موارد مشابه Maxpooling، Mean pooling است که به جای ماکسیمم گیری، میانگین‌گیری میکند.

Fully Connected : این لایه در انتهای شبکه (یعنی بعد از لایه‌های کانولوشنی) قرار میگیرد. تمامی نورون‌های این لایه با تمامی نورون‌های لایه قبلی‌اش متصل است. در لایه‌های fully connected در طبقه‌بندی، آخرین لایه fully معمولا به تعداد کلاس‌هایی که میخواهیم طبقه بندی انجام دهیم نورون دارد. برای مثال برای دیتاست‌های ما آخرین لایه Fully connected دارای ۱۰ نورون است.

تابع فعالساز ReLU : این تابع فعالساز به عنوان فعالساز غیر خطی شناخته می‌شود. ReLU می‌تواند بر روی داده‌ها هر تبدیل و تابع پیچیده بین ورودی و خروجی را بسازد. در شکل زیر فعالساز ReLU نمایش داده شده است.



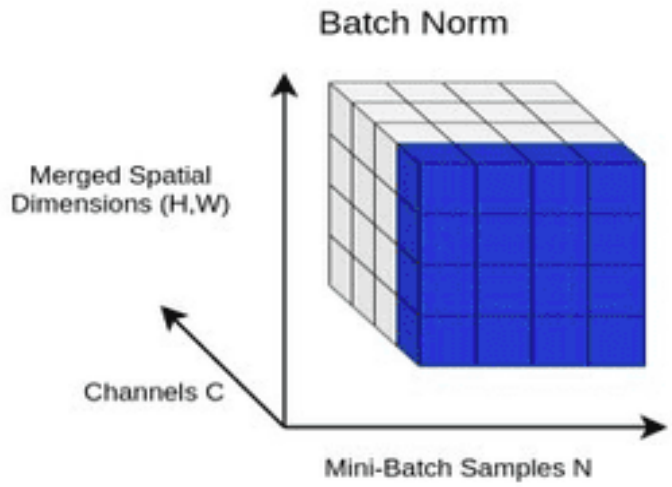
شکل ۱. تابع فعالساز **ReLU**

SoftMax : در طبقه‌بندی چند کلاسه، SoftMax یکی از بهترین روش‌های نسبت دادن احتمال به هر کلاس است. این فانکشن که آخرین لایه از طبقه‌بند است و به صورت زیر برای هر کلاس محاسبه می‌گردد. هر خروجی فانکشن softmax نشان دهنده احتمال تعلق ورودی به آن کلاس است.

$$softmax(y)_i = \frac{e^{y_i}}{\sum e^{y_i}}$$

لایه Batch Normalization : مهم‌ترین لایه ارائه شده در مقاله، لایه batch Normalization است. پس از لایه‌های کانولوشن، لایه‌های batch normalization برای نرمالیز کردن فیچرمپ‌های خروجی از

کانولوشن قرار داده می‌شوند. این لایه به این دلیل به کار میرود که minibatch داده‌های ورودی به لایه‌های میانی شبکه ممکن است از توزیع‌های مختلفی آمده باشند و این امر باعث می‌شوند فرایند آموزش پارامترها سخت و کندتر شود. لایه batch normalization قبل از تابع فعالساز قرار می‌گیرد تا ورودی به فعالساز را نرمال کند. این عمل باعث می‌شود که ورودی به فعالساز در بازه حساسیت فعالساز قرار بگیرد و این عمل باعث کاهش احتمال رخداد gradient vanishing می‌شود. تکنیک batch normalization بر روی افزایش سرعت همگرایی شبکه عصبی موثر است و قدرت تعمیم‌پذیری شبکهٔ عصبی را افزایش میدهد. در شکل زیر جهت نرمالسازی در batch normalization نشان داده شده است. به علت اینکه فیچرمپ بالاتر از سه بعد (در اینجا ۴ بعد) است. ابعاد یک فیچرمپ (مانند طول و عرض تصویر) در یک جهت ادغام شده اند تا در تصویر زیر قابل نمایش باشد.



شکل ۲. تصویری از جهت **Batchnormalization** بر روی تنسور ویژگی‌ها

۳-۱. پیاده سازی شبکه

با توجه به معماری بیان شده در مقاله، شبکه عصبی را طراحی می‌کنیم. طبق مقاله، سه شبکه برای آموزش در نظر گرفته شده است. شبکه‌های SCNNB ، SCNNB-a و SCNNB-b. برای تمامی لایه‌های کانولوشن در هر سه شبکه، مطابق مقاله kernel_size = 3 و stride = 1 در نظر گرفته شد. شبکه پیاده‌سازی شده در زیر نمایش داده شده است. تفاوت سه شبکه SCNNB ، SCNNB-a و SCNNB-b در لایه‌های Batch Normalization است؛ به این صورت که در شبکه SCNNB بعد از هر لایه کانولوشن، یک لایه Batch Normalization آمده است. در SCNNB-a لایه Batch Norm پس از از اولین لایه کانولوشن حذف شده است و در شبکه SCNNB-b لایه Batch Norm پس از هر دو لایه کانولوشن حذف شده است. این سه شبکه در بقیه لایه‌ها دقیقا مشابه همدیگر هستند. در تصویر شبکه طراحی شده را مشاهده می‌کنید.

و در شکل۴. خلاصه‌ای از معماری شبکه و تعداد پارامترهای آن آورده شده است. شبکه پیشنهادی مقاله از ۲ بلوک کانولوشن ساخته شده است که هر بلوک یک لایه کانولوشن ۲ بعدی و یک لایه batch normalization (ساختار SCNNB) قرار داده شده و از تابع فعال ساز relu عبور داده شده است. سپس از maxpooling برای کاهش بعد استفاده شده است. پس از ۲ بلوک کانولوشن، یک لایه fully connected با ۱۲۸۰ نورون، با تابع فعالساز Relu قرار داده شده است. سپس، dropout با p=50% قرار داده شده است که آموزش شبکه را سریع تر و از overfitting جلوگیری کند. در نهایت خروجی با ۱۰ نورون از فعالساز SoftMax عبور داده خواهد شد.

```python
class SCNNB(torch.nn.Module):
    def __init__(self, inpSize, inp_channel):
        super(SCNNB, self).__init__()
        self.convBlock1 = nn.Sequential(
        nn.Conv2d(in_channels = inp_channel, out_channels =32, kernel_size=3, stride=1, padding ="same"),
        nn.BatchNorm2d(32),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size = 2))
        if inpSize == 32:
            self.convBlock2 = nn.Sequential(
            nn.Conv2d(in_channels = 32, out_channels =64, kernel_size=3, stride=1, padding = 'valid'),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2))
        elif inpSize == 28:
            self.convBlock2 = nn.Sequential(
            nn.Conv2d(in_channels = 32, out_channels =64, kernel_size=3, stride=1, padding = 'same'),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2))
        self.linear1 = nn.Sequential(
        nn.Linear(64*7*7, out_features=1280),
        nn.ReLU(),
        nn.Dropout(p = 0.5),
        nn.Linear(1280,out_features = 10)
        )

    def forward(self, x):
        x = self.convBlock1(x)
        x = self.convBlock2(x)
        x = x.view(-1, x.shape[1]*x.shape[2]*x.shape[3])
        x = self.linear1(x)
        x = F.softmax(input =x, dim = 1)
        return x
```

شکل ۳. معماری شبکه عصبی **SCNNB**

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 28, 28]             320
       BatchNorm2d-2          [-1, 32, 28, 28]              64
              ReLU-3          [-1, 32, 28, 28]               0
         MaxPool2d-4          [-1, 32, 14, 14]               0
            Conv2d-5          [-1, 64, 14, 14]          18,496
       BatchNorm2d-6          [-1, 64, 14, 14]             128
              ReLU-7          [-1, 64, 14, 14]               0
         MaxPool2d-8            [-1, 64, 7, 7]               0
            Linear-9                 [-1, 1280]       4,015,360
             ReLU-10                 [-1, 1280]               0
          Dropout-11                 [-1, 1280]               0
           Linear-12                   [-1, 10]          12,810
================================================================
Total params: 4,047,178
Trainable params: 4,047,178
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.96
Params size (MB): 15.44
Estimated Total Size (MB): 16.40
----------------------------------------------------------------
```

شکل ۴. لایه‌ها و تعداد پارامترهای قابل آموزش هر لایه

باید اشاره کنیم برای آنکه پارامترها دقیقا مشابه با مقاله باشد باید padding ها را در مدل پیاده‌سازی شده بر روی کانولوشن‌ها اعمال کنیم که خروجی دومین لایه کانولوشن (دقیقا قبل از ورود به لایه Fully connected) برابر با ۷×۷×۶۴ باشد. برای دیتاست با تصاویر ورودی به ابعاد ۱×۲۸×۲۸ same=padding قرار داده شد و برای دیتاست با تصاویر ورودی ۳×۳۲×۳۲ برای کانولوشن اول padding=same و برای کانولوشن دوم padding=valid قرار داده شد. با تنظیم کردن padding ساختار شبکه دقیقا مشابه مقاله ایجاد شد.

هایپرپارامترهای آموزش شبکه

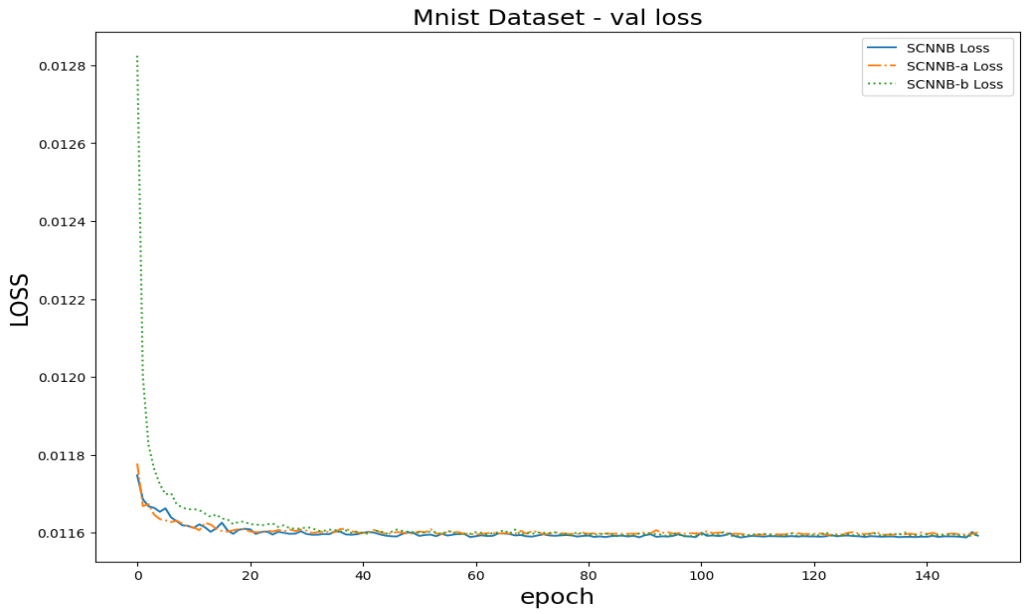
همان طور که در مقاله اشاره شد از بهینه‌ساز SGD با نرخ یادگیری 0.02 و مومنتوم 0.9 برای آموزش شبکه استفاده شد. با توجه به مقاله از regularization weight هم استفاده شده که برای این کار، پارامتر weight_decay=0.000005 را روی بهینه‌ساز قرار دادیم. هم‌چنین از تابع هزینه CrossEntropy برای آموزش مدل‌ها استفاده شد. لایه Dropout با Dropout rate=0.5 در قسمت Fully Connected به کار گرفته شد. Dropout برای جلوگیری از overfitting و افزایش سرعت آموزش موثر است.

۴-۱. نتایج پیاده سازی

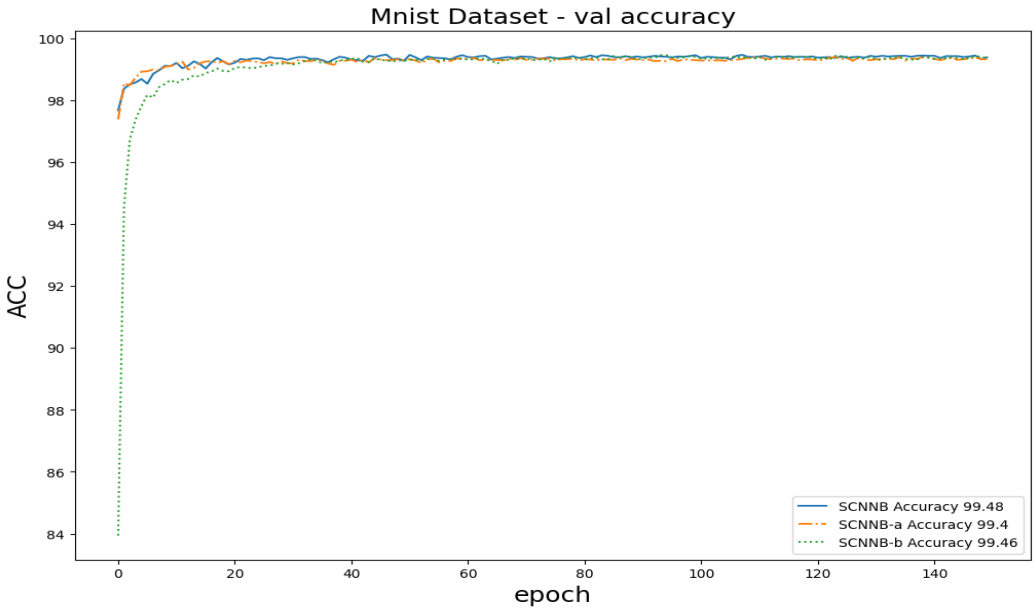
سه شبکه عصبی نام برده شده در مقاله (SCNNB ، SCNNB-a و SCNNB-b) بر روی سه دیتاست Mnist و Fashion Mnist و CFAR10 آموزش داده شده‌اند. نمودار خطا و دقت بر روی داده‌های ارزیابی و داده‌های آموزشی به ترتیب در ادامه آورده شده است.

الف) نمودار خطا و دقت برای دادگان ارزیابی

دیتاست Mnist

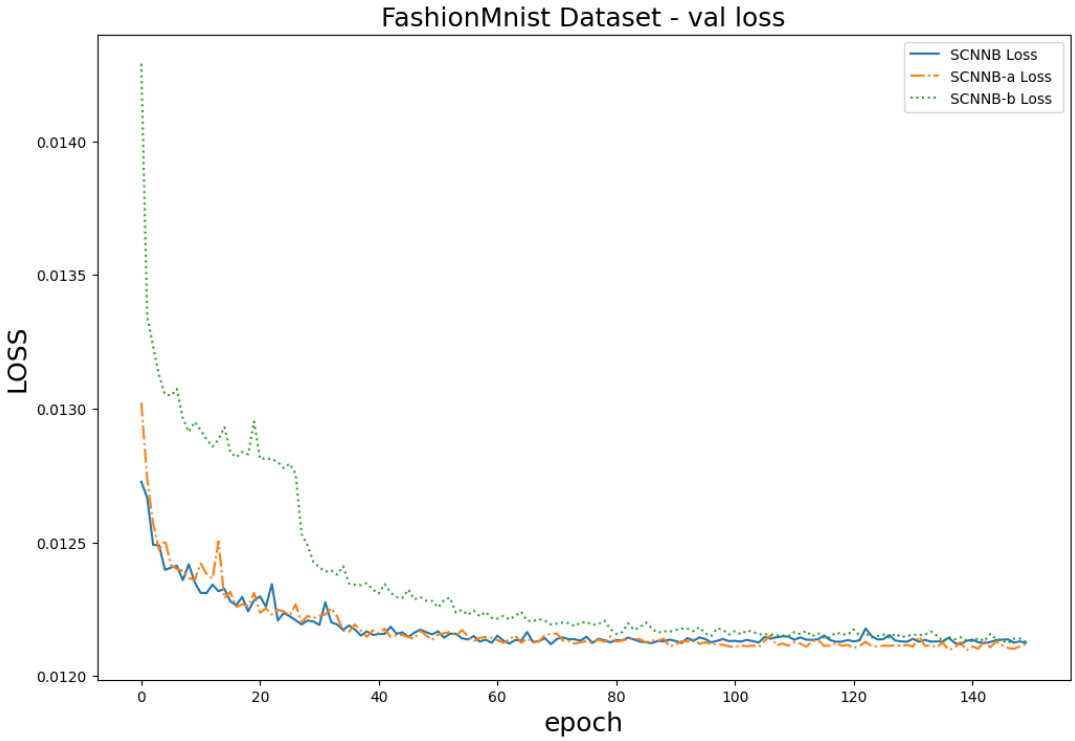


شکل ۵. تابع هزینه داده‌های ارزیابی - دیتاست **Mnist**

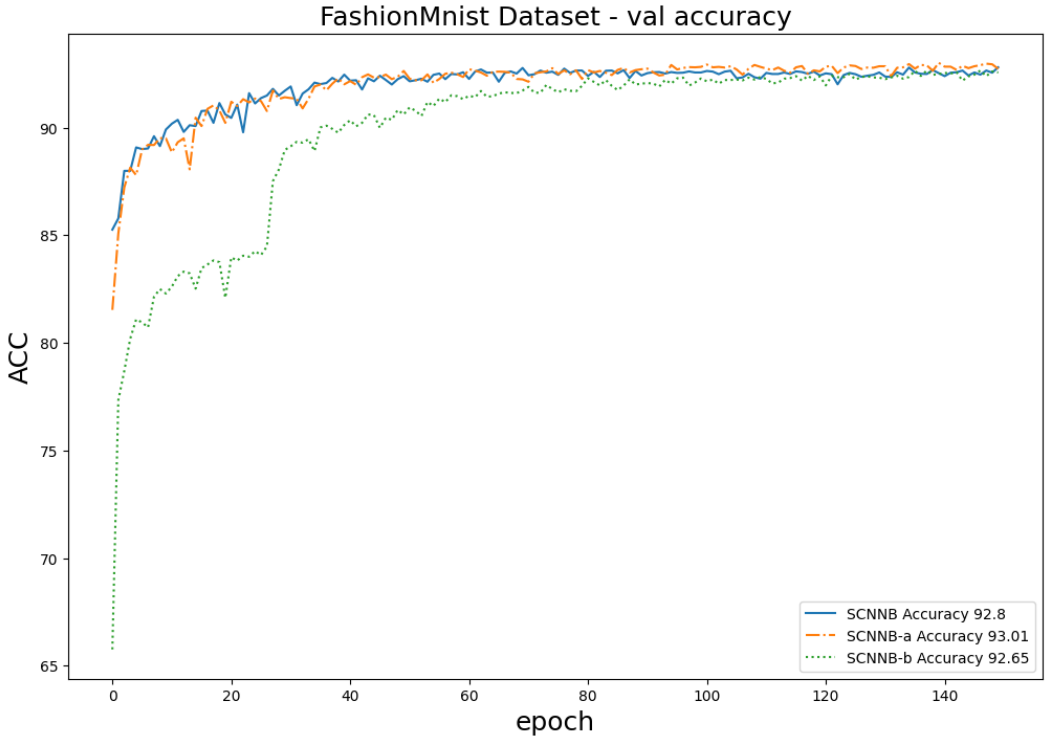


شکل ۶. دقت شبکه‌های عصبی بر روی دادگان ارزیابی- دیتاست **Mnist**

دیتاست Fashion Mnist

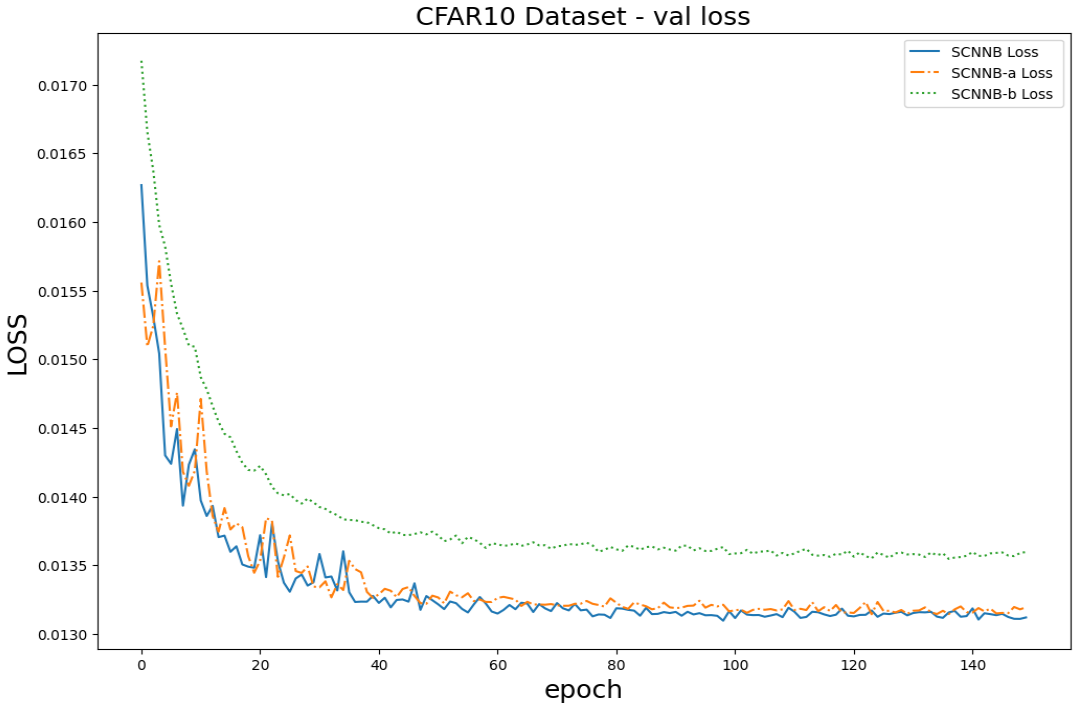


شکل ۷. تابع هزینه داده‌های ارزیابی - دیتاست **Fashion Mnist**

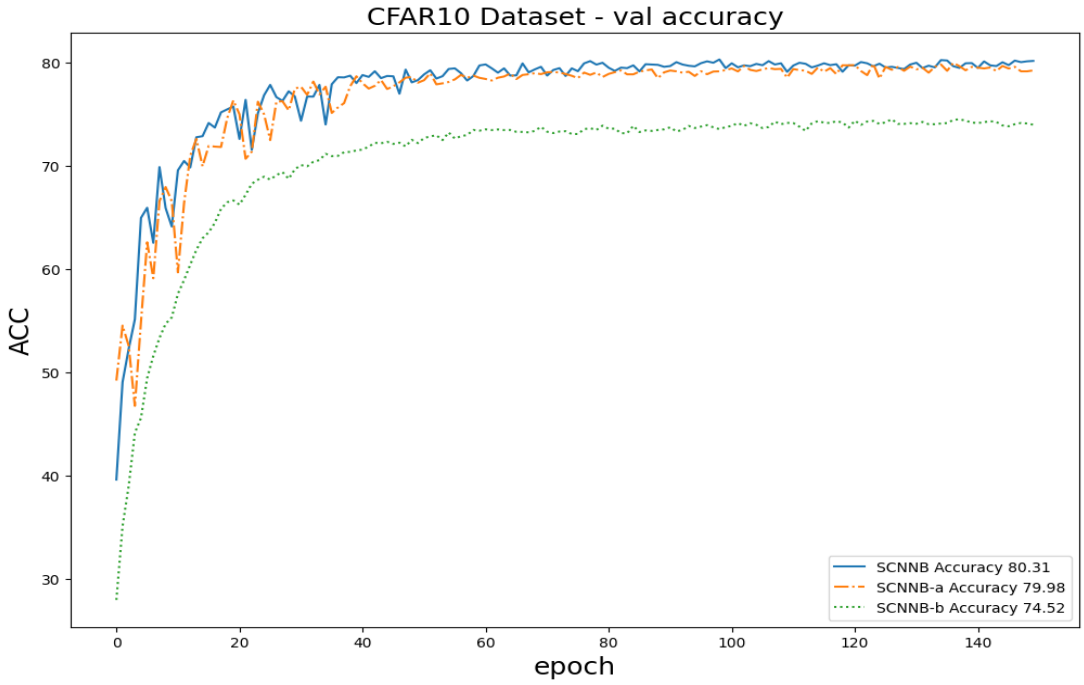


شکل ۸. دقت شبکه‌های عصبی بر روی دادگان ارزیابی- دیتاست **Fashion Mnist**

دیتاست CFAR10



شکل ۹. تابع هزینه داده‌های ارزیابی - دیتاست **CFAR10**



شکل ۱۰. دقت شبکه‌های عصبی بر روی دادگان ارزیابی- دیتاست **CFAR10**

علاوه بر اینکه دقت‌های شبکه‌های پیشنهادی مقاله در شکل‌های بالا آورده شده است، در جدول زیر مشابه جدول درون مقاله، دقت‌ها روی داده ارزیابی آورده شده است.

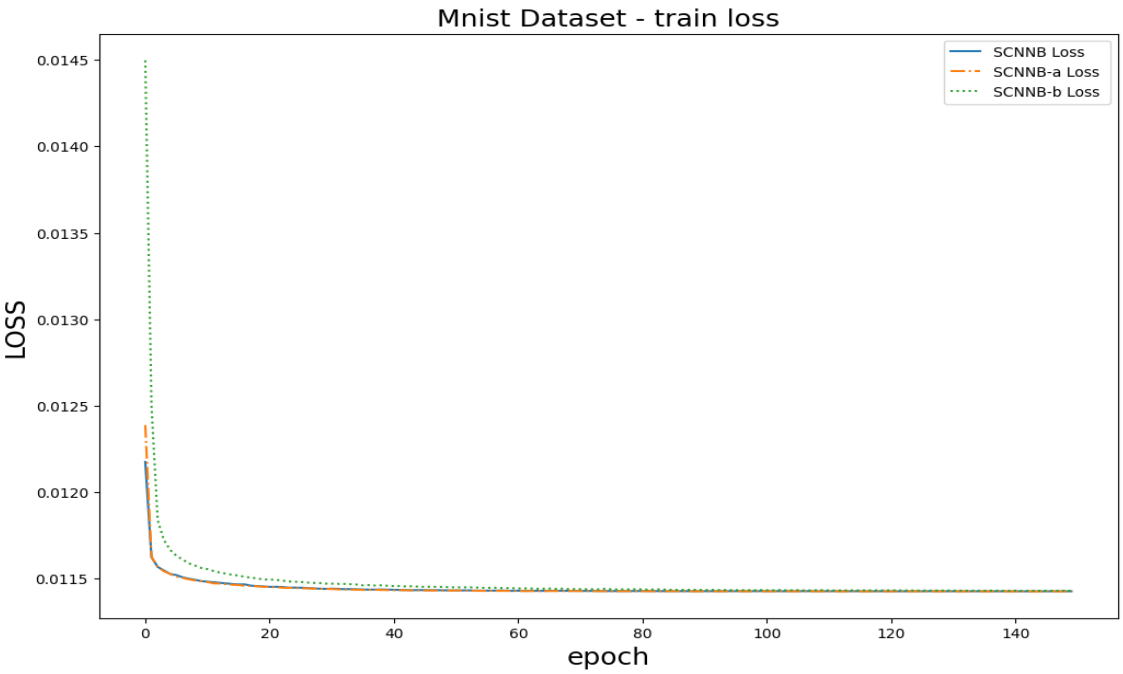
جدول ۱. دقت شبکه‌ها بر روی داده‌های ارزیابی

Our Models	MNIST test accuracy (%)	Fashion-MNIST test accuracy (%)	CIFAR10 test accuracy (%)
SCNNB	99.48	92.8	80.31
SCNNB-a	99.40	93.01	79.98
SCNNB-b	99.46	92.65	74.52

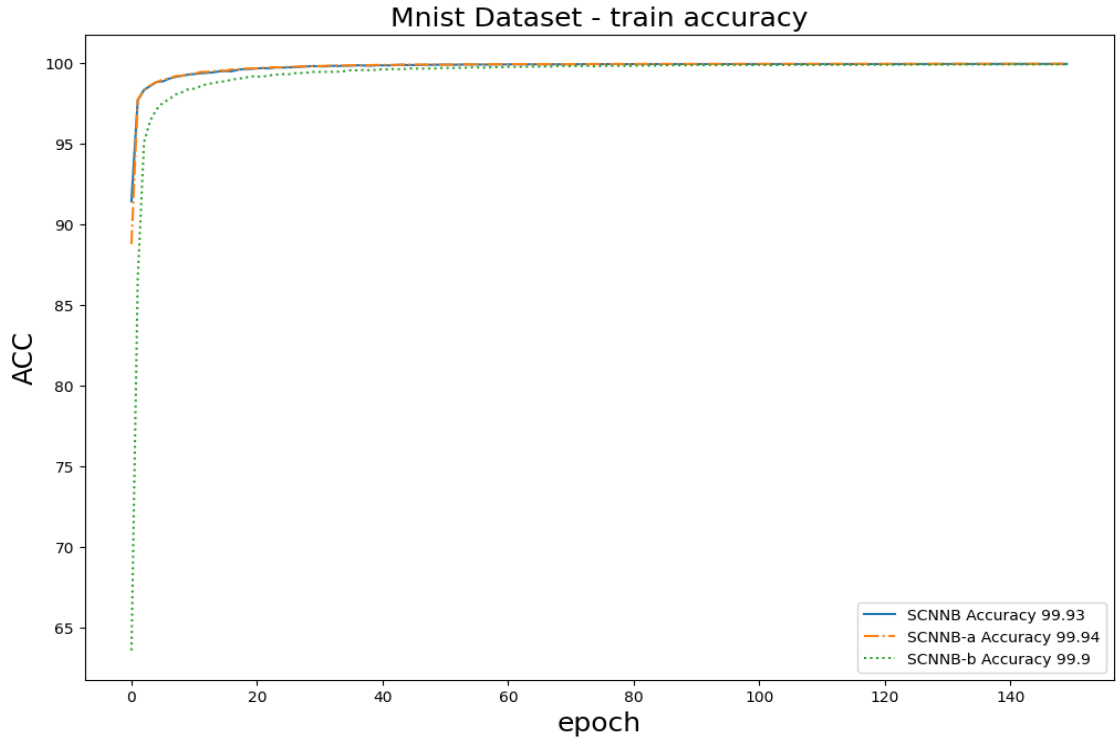
ب) نمودار خطا و دقت برای دادگان آموزشی

در نمودارهای زیر، دقت و تابع هزینه برای داده های آموزشی در طول فرآیند آموزش رسم شده است.

دیتاست Mnist

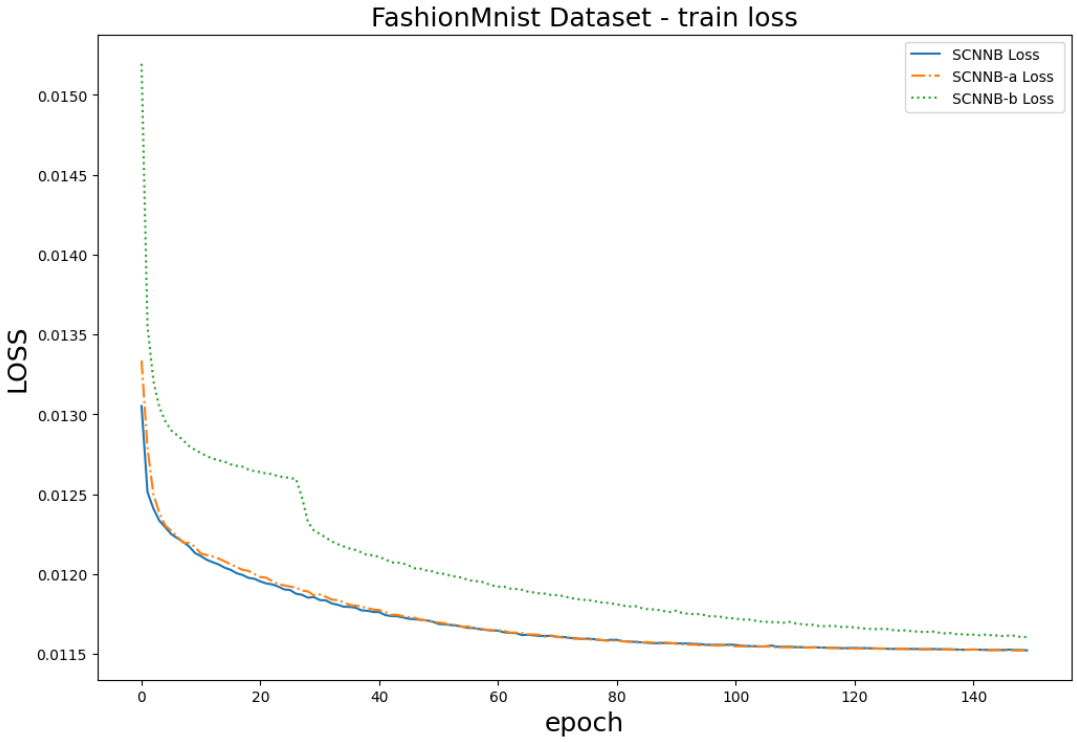


شکل ۱۱. تابع هزینه داده‌های آموزشی - دیتاست **Mnist**

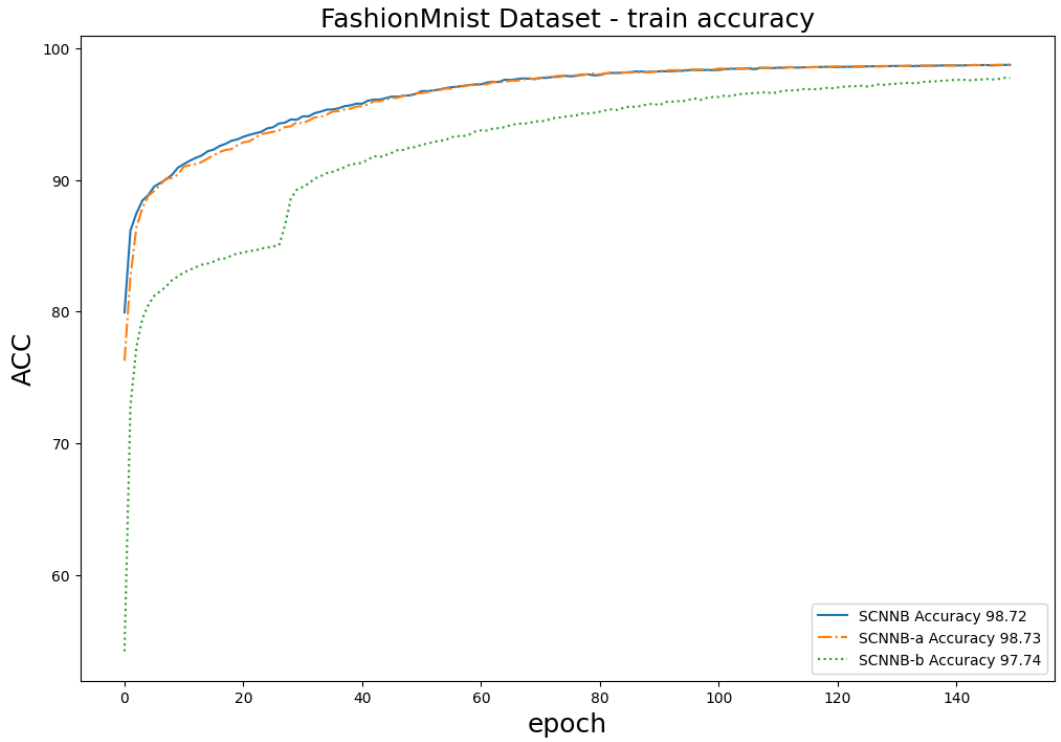


شکل ۱۲. دقت شبکه‌های عصبی بر روی دادگان آموزشی- دیتاست **Mnist**

دیتاست Fashion Mnist

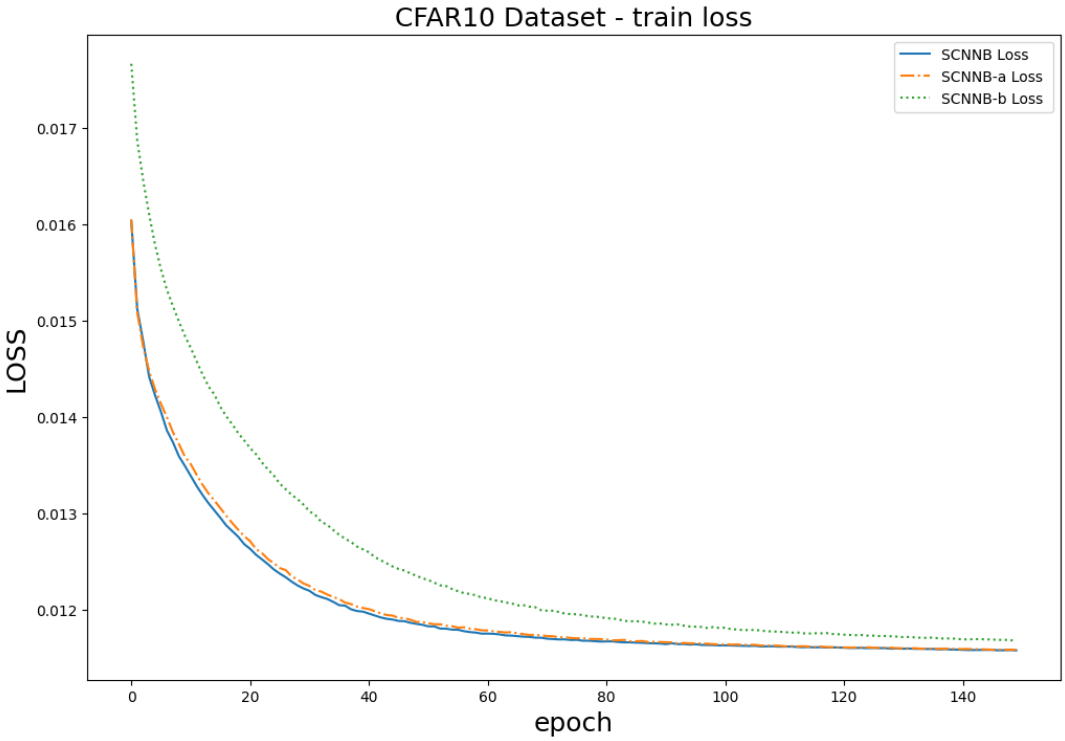


شکل ۱۳. تابع هزینه داده‌های آموزشی - دیتاست **Fashion Mnist**

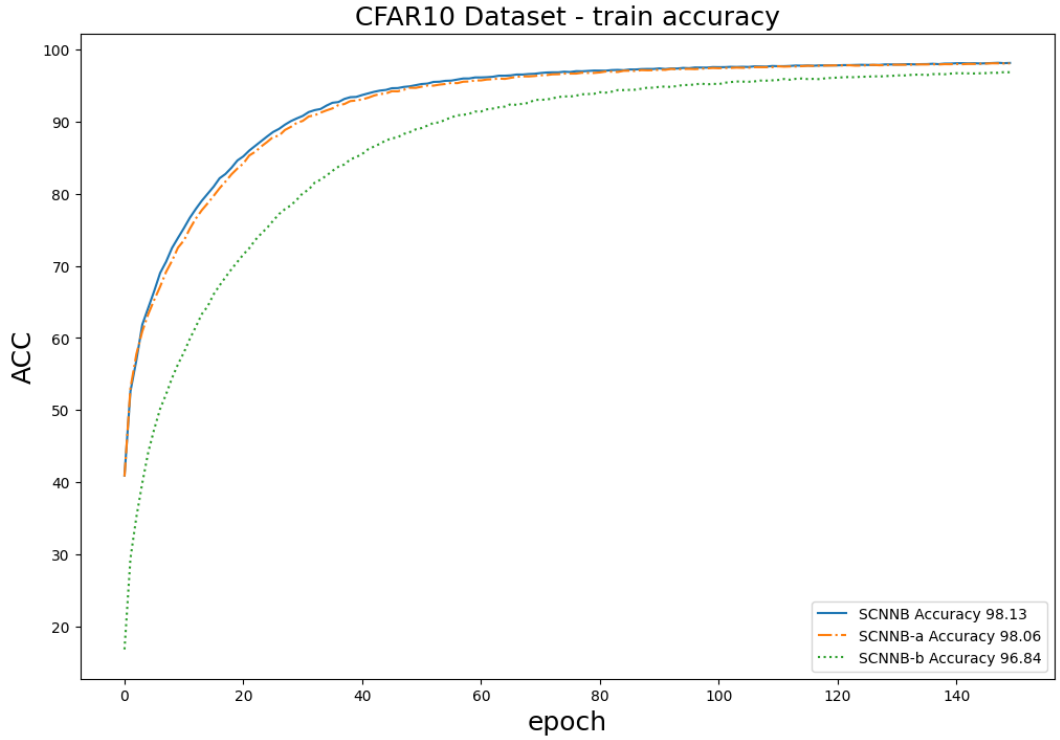


شکل ۱۴. دقت شبکه‌های عصبی بر روی دادگان آموزشی - دیتاست **Fashion Mnist**

دیتاست CFAR10



شکل ۱۵. تابع هزینه داده‌های آموزشی - دیتاست **CFAR10**



شکل ۱۶. دقت شبکه‌های عصبی بر روی دادگان آموزشی- دیتاست **CFAR10**

ج) تحلیل نتایج

در این بخش با توجه به نتایج به دست آمده از قسمت‌های قبل، به تحلیل نتایج بر روی دیتاست‌های Mnist و Fashion Mnist و CFAR10 می‌پردازیم.

جدول ۲. دقت شبکه‌های عصبی- دیتاست **Mnist**

	دقت روی داده آموزشی	دقت روی داده ارزیابی
شبکه SCNNB	۹۹.۹۳	۹۹.۴۸
شبکه SCNNB-a	۹۹.۹۴	۹۹.۴۰
شبکه SCNNB-b	۹۹.۹۰	۹۹.۴۶

بر روی داده‌های Mnist دقت مدلی که آموزش دادیم بر روی داده‌های ارزیابی به دقت‌های گزارش شده در مقاله رسید. همان طور که در شکل-۵، مشخص است، Batch Normalization تاثیر مثبت بر روی آموزش شبکه و افزایش دقت بر روی دادگان ارزیابی داشته است. همچنین با توجه به جدول-۲، دقت مدل بر روی داده‌های ارزیابی و آموزش بسیار به هم نزدیک است و overfitting رخ نداده است. سرعت آموزش شبکه نیز بسیار بالا بوده و در ایپاک‌های ابتدایی به دقت‌های مطلوب رسیدیم.

جدول ۳. دقت شبکه‌های عصبی- دیتاست **Fashion Mnist**

	دقت روی داده آموزشی	دقت روی داده ارزیابی
شبکه SCNNB	۹۸.۷۲	۹۲.۸
شبکه SCNNB-a	۹۸.۷۳	۹۳.۰۱
شبکه SCNNB-b	۹۷.۷۴	۹۲.۶۵

بر روی داده‌های Fashion Mnist نیز توانستیم به دقت‌های مقاله دست یابیم. با توجه به جدول-۳، دقت مدل بر روی داده‌های آموزشی و ارزیابی حدود ۶٪ اختلاف دارد و البته می‌توان دید که لایه Batch-Normalization نیز در اینجا تاثیر مثبتی بر روی آموزش شبکه داشته و اختلاف دقت مدل بر روی داده‌های ارزیابی و آموزش را کاهش داده است. همچنین مشاهده می‌شود استفاده از Batch-Normalization باعث افزایش سرعت همگرایی شبکه شده است مدل SCNNB نسبت به SCNNB-b در ایپاک‌های پایین تر به دقت‌های بالاتر دست پیدا کرده است.

جدول ۴. دقت شبکه‌های عصبی- دیتاست **CFAR10**

	دقت روی داده آموزشی	دقت روی داده ارزیابی
شبکه SCNNB	۹۸.۱۳	۸۰.۳۱
شبکه SCNNB-a	۹۸.۰۶	۷۹.۹۸
شبکه SCNNB-b	۹۶.۸۴	۷۴.۵۲

بر روی داده‌های CFAR10 همان طور که از شکل-۱۶ و شکل-۱۰ قابل مشاهده است مدل SCNNB که در آن از لایه Batch-Normalization استفاده شده است نسبت به مدل SCNNB-b و SCNNN-a به نتایج بهتری بر روی داده‌های ارزیابی رسیده و همچنین به شکل قابل توجه‌ای سرعت همگرایی بالاتری است نسبت به مدل SCNNB-b داشته است. شکل-۹ و شکل-۱۵ نیز تایید می‌کنند که همگرایی شبکه با لایه‌های batch-normalization سریع‌تر است. با توجه به جدول-۴میتوان دریافت که در این دیتاست، اختلاف دقت بر روی داده‌های آموزش و ارزیابی کمی افزایش داشته است. همچنین تاثیر مثبت Batch-Normalization قابل مشاهده است و این لایه باعث کاهش اختلاف دقت بین داده آموزش و ارزیابی شده است و دقت را روی داده‌های ارزیابی افزایش داده است و برای جلوگیری از Overfitting موثر بوده است.

پاسخ ۲ . طبقه بندی تصاویر اشعه ایکس قفسه سینه

۲-۱. آماده سازی و پیش پردازش داده ها

الف.

۱. **Re-scale**

rescaling یکی از روش های پر استفاده برای افزایش داده به خصوص در پردازش تصویر و بینایی کامپیوتری است. این روش شامل تغییر اندازه تصویر با افزایش یا کاهش ابعاد آن و حفظ aspect ratio تصویر است. این فرآیند تغییر مقیاس تصویر شامل اعمال یک تبدیل ریاضی است که تعداد پیکسل های تصویر را تغییر می دهد، که می تواند به افزایش اندازه مجموعه داده و ایجاد تنوع بیشتر در داده ها کمک کند. rescaling به دو صورت انجام می شود:Upscaling و Downsizing

۲. **Shear**

shear یک تکنیک افزایش داده ها در پردازش تصویر است که شامل جابجایی پیکسل های یک تصویر در امتداد یک جهت خاص و در عین حال حفظ موقعیت نسبی آنها است. اعمال این تکنیک به این صورت است که ، تصویر با اعمال یک shear matrix به مختصات پیکسل هایش، به یک تصویر جدید تبدیل می شود. shear matrix یک ماتریس ۲×۲ است که میزان shear را که باید در امتداد محور x و محور y تصویر اعمال شود، تعیین می‌کند. میزان shear اعمال شده در امتداد هر محور را می توان به طور جداگانه کنترل .

۳. **Width Shift**

در width shift، تصویر به صورت افقی به اندازه تعداد معینی پیکسل، به چپ یا راست انتقال داده می شود تا نمونه های جدیدی از همان تصویر ایجاد شود. مثلاً فرض کنید تصویری از یک گربه داریم. با اعمال width shift، می‌توانیم تصاویر جدیدی از گربه با جابجایی تصویر به چپ یا راست با تعداد مشخصی پیکسل ایجاد کنیم. تصاویر به دست آمده دارای موقعیت های کمی متفاوت از گربه در قاب خواهند بود که می تولند به آموزش مدل برای تشخیص گربه ها در موقعیت های مختلف کمک کند.

۴. **Height Shift**

دقیقاً مانند width shift، با این تفاوت که تصـویر به جای افقی به صـورت عمودی جابجا می شود.

۵. **Rotation**

rotation شـامل چرخش تصـویر با یک زاویه خاص اسـت که معمولاً با اعمال یک ماتریس چرخشی بر روی پیکسل های تصویر انجام می شود و شامل یک سری عملیات ریاضی برای تبدیل تصـویر به یک orientation جدید اسـت. rotation می تواند به ویژه در شـرایطی مفید باشـد که شـیء مورد نظر می تواند در زوایای مختلف ظاهر شـود، مانند صـورت انسـان یا حیوانات.

۶. **Horizontal Flip**

این تکنیک معمولاً با انعکاس پیکسـل های تصـویر در امتداد محور عمودی انجام می شـود، به طوری که سمت چپ تصویر به سمت راست تبدیل می شود و بالعکس. این روش می تواند به ویژه در شرایطی مفید باشد که direction شی برای طبقه بندی مهم نیست.

۷. **Zooming**

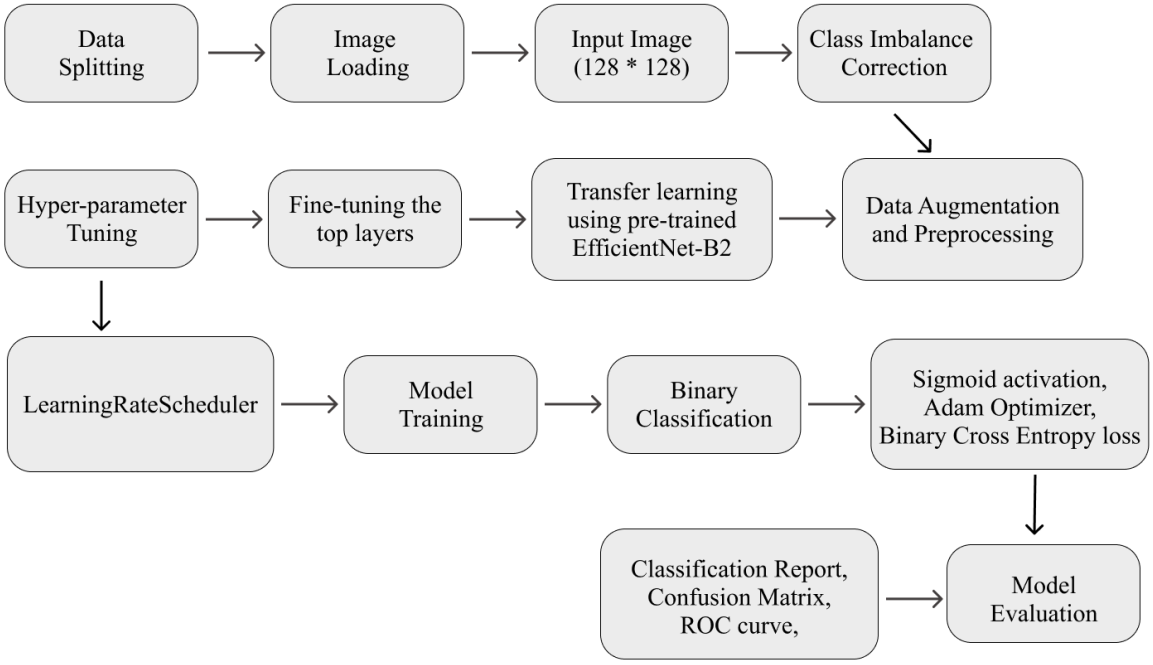
در این روش تصـویر اصـلی بزرگنمایی یا کوچکنمایی می شـود و به خصـوص در شـرایطی که اندازه یا فاصله جسم برای طبقه بندی مهم نیست ، می تواند مفید باشد.

ب.

برای به دست آوردن دیتای مربوطه همانطور که در شکل زیر نشان داده شده است از Kaggle API اسـتفاده کردیم اما مشـکلی که وجود داشـت این بود که نسـبت تقسـیم بندی داده ها به train, test, validation بـا نسـبتی کـه صـورت سـوال خواسـتـه یکسـان نبود، لـذا مطـابق آنچـه در فـایـل HW02_Q02_dataset.ipynb وجود دارد ، دیتا ها را با نسـبت خواسـته شـده تقسـیم بنده کرده و در پوشه splitted-data ذخیره کردیم .

```
!kaggle datasets download paultimothymooney/chest-xray-pneumonia
```

۲-۲. توضیح لایه های مختلف معماری شبکه



شکل ۱۷. **Workflow**

در شکل بالا workflow کلی مربوط به این سوال نشان داده شده است که در ادامه به بررسی مراحل مختلف آن می پردازیم:

1. **Data Splitting:**

در این مرحله مطابق آنچه در قسمت ۱-۲-ب توضیح داده شده، داده ها را با نسبت 60، 20، 20 به ترتیب به train، test، validation تقسیم بندی کردیم.

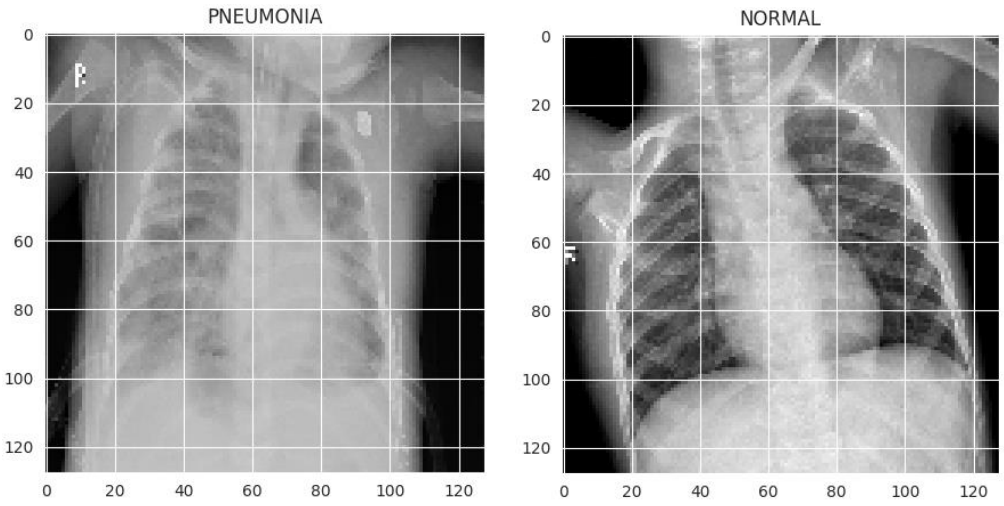
2. **Image Loading**

برای load کردن تصاویر از directory مربوطه، تابعی که در شکل ۱۸ نشان داده شده است را تعریف کردیم. این تابع تصاویر را از directory مشخص شده دریافت کرده و یک آرایه شامل تصاویر resized شده به اندازه 128*128 به همراه label آن تصویر، بر می گرداند.

```python
LABEL_NAMES = ['PNEUMONIA', 'NORMAL']
IMAGE_SIZE = 128
IMAGE_MODE = cv2.IMREAD_GRAYSCALE

def get_data(data_dir):
    """
    Load the image data from the specified directory.

    Args:
        data_dir (str): The directory containing the image data.

    Returns:
        data (numpy.ndarray): An array of image data, where each row contains a
            resized image and its corresponding label.
    """
    data = []
    for label in LABEL_NAMES:
        path = os.path.join(data_dir, label)
        label_num = LABEL_NAMES.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img), IMAGE_MODE)
                resized_arr = cv2.resize(img_arr, (IMAGE_SIZE, IMAGE_SIZE))
                data.append([resized_arr, label_num])
            except Exception as e:
                print(e)
    return np.array(data)
```

شکل ۱۸. تابع خواندن تصاویر از **directory** مربوط

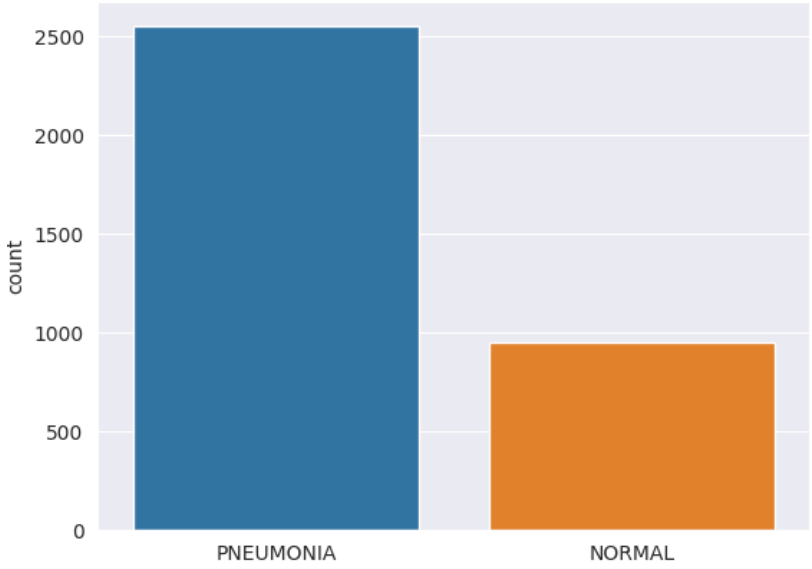
در شکل 19 یک نمونه از تصویر مربوط به دو کلاس PNEUMONIA و NORMAL نشان داده شده است.



شکل ۱۹. نمونه ای از تصاویر مربوط به دو کلاس **Normal** و **Pneumonia**

3. Class Imbalance Correction

یکی از مشکلاتی که در این دیتاست وجود دارد و در شکل 20 نیز نمایش داده شده است، بالانس نبودن اندازه دو کلاس است. همانطور که در تصویر نیز مشاهده می شود، تعداد تصاویر مربوط به کلاس PNEUMONIA بسیار بیشتر است از تعداد تصاویر کلاس دیگر. این مسئله می تواند مشکلات زیادی از قبیل Biased prediction، Poor Generalization، Reduced Model Performance و دیگر مشکلات از این دست ایجاد کند. لذا برای رفع آن مطابق آنچه در شکل 21 نشان داده شده است عمل کرده و با تعریف class weights و بعد بکار گیری آن در مرحله fit کردن مدل ، این مشکل را حل کردیم. این روش سعی می کند مشکل بالانس نبودن کلاس ها را با اختصاص دادن وزنی به هر کلاس حل کند. همانطور که در تصویر مشخص است برای کلاس PNEUMONIA که تعداد تصاویر آن بیشتر است وزن 0.68 و برای کلاس NORMAL با تعداد تصاویر کمتر وزن 1.84، اختصاص داده شده است.



شکل ۲۰. نسبت تعداد تصاویر دو کلاس

```
# dealing with class imbalance
class_weights = class_weight.compute_class_weight(
                                        class_weight = "balanced",
                                        classes = np.unique(y_train),
                                        y = y_train
                                    )
class_weights = dict(zip(np.unique(y_train), class_weights))
class_weights
```

```
{0: 0.6861514319340918, 1: 1.8429926238145415}
```

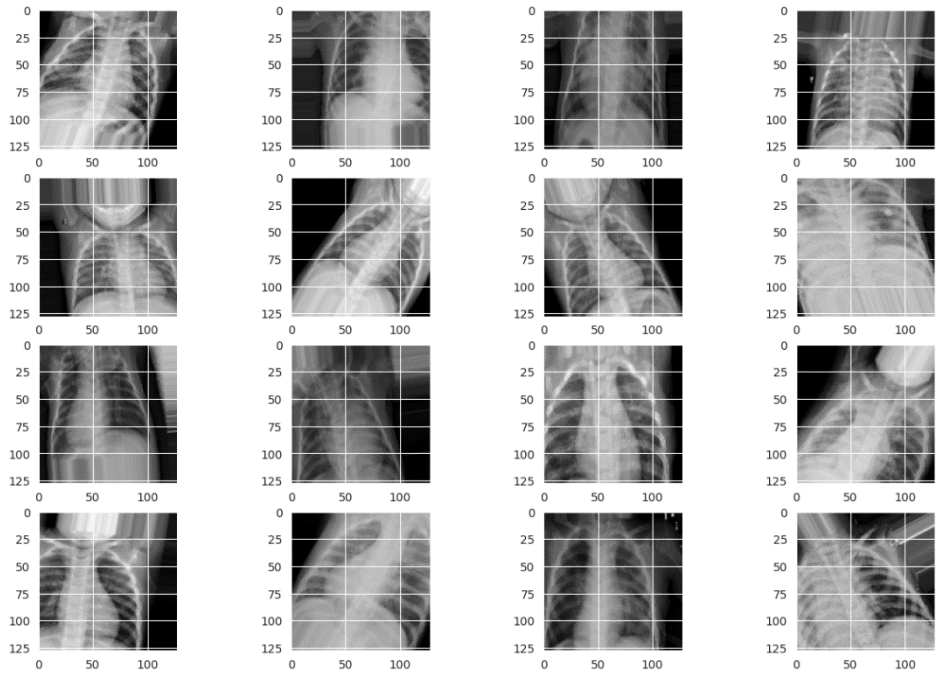
شکل ۲۱. نمونه کد استفاده از **class-weights**

4. Image Augmentation and Preprocessing

به طور کلی هدف اصلی از image augmentation کمک به بهبود مدل از طریق generalizable کردن آن است. این نکته به خصوص زمانی که اندازه دیتاست زیاد نباشد بسیار مهم است چرا که به مدل اجازه می دهد از مجموعه ای بزرگتر و متنوع تری از تصاویر train شود. همچنین مدل را در برابر تغیراتی همچون تغییر در نور، viewpoint و از این دست تغییرات robust می کند. مراحل مختلف image augmentation در شکل 22 نشان داده شده است . این مراحل به همراه پارامتر های آن با توجه به مقادیری که در مقاله[1] آورده شده است، طراحی شده است. دقت شود که برخلاف آنچه در مقاله ذکر شده است rescale کردن تصاویر در image augmentation انجام نشده است و در مرحله preprocessing که در ادامه به آن می پردازیم انجام شده است.

```
img_augmentation = ImageDataGenerator(
        rotation_range = 30,  # randomly rotate images in the range
        zoom_range = 0.2, # Randomly zoom image
        shear_range=0.2,# Randomly shear image
        width_shift_range=0.2,  # randomly shift images horizontally
        height_shift_range=0.2,  # randomly shift images vertically
        horizontal_flip = True  # randomly flip images
        )
```

شکل ۲۲. استفاده از کلاس **ImageDataGenerator** برای **augment** کردن تصاویر



شکل ۲۳. نمونه ای از تصاویر **augment** شده

[1] Automated Diagnosis of Pneumonia from Classification of Chest X-Ray Images using EfficientNet

در مرحله preprocessing ابتدا همانطور که در شکل 24 نشان داده شده است، آرایه های مربوط به داده ها را resize کردیم به گونه ای که برای استفاده در شبکه عصبی مناسب شوند.

```
def resize_data(x, y, image_size):
    x_resized = np.array(x).reshape(-1, image_size, image_size, 1)
    y_resized = np.array(y)
    return x_resized, y_resized
```

```
# resize data for deep learning
x_train, y_train = resize_data(x_train, y_train, IMAGE_SIZE)
x_val, y_val = resize_data(x_val, y_val, IMAGE_SIZE)
x_test, y_test = resize_data(x_test, y_test, IMAGE_SIZE)
```

شکل ۲۴. تابع مربوط به **resize** کردن آرایه تصاویر

سپس همانطور که در شکل 25 نشان داده شده است، آرایه تصاویر که به صورت 1D channel بودند را به 3D channel تبدیل کردیم. ضرورت این کار به این خاطر است که در طراحی شبکه عصبی به صورت transfer learning، وزن اولیه شبکه را به جای وزن های رندوم، وزن های Imagenet قرار می دهیم و برای اینکار لازم است تصاویر ورودی به صورت RGB باشند.

```
def convert1chTo3ch(x):
    x3ch = np.zeros((x.shape[0], x.shape[1], x.shape[2], 3))
    x3ch[:,:,:,0] = x.squeeze()
    x3ch[:,:,:,1] = x.squeeze()
    x3ch[:,:,:,2] = x.squeeze()
    return x3ch.astype(np.uint8)
```

```
xVal3ch = convert1chTo3ch(x_val)
xTest3ch = convert1chTo3ch(x_test)
xTrain3ch = convert1chTo3ch(x_train)
xVal3ch.shape,xTrain3ch.shape,xTest3ch.shape
```

```
((1160, 128, 128, 3), (3498, 128, 128, 3), (1156, 128, 128, 3))
```

شکل ۲۵. کد مربوط به تبدیل آرایه تصاویر از **1D Channel** به **3D Channel**

خروجی مراحل گفته شده در در شکل 26 نشان داده شده است. همانطور که مشخص است Normalize کردن تصاویر و rescale کردن آنها بین مقادیر [255 0] نیز انجام شده است و داده های برای استفاده در شبکه عصبی آماده هستند.

```
x_train :  (3498, 128, 128, 3) , range: [ 0   255 ]
y_train :  (3498,) , values:  [0 1]
**************************
x_test :  (1156, 128, 128, 3) , range: [ 0   255 ]
y_test :  (1156,) , values:  [0 1]
**************************
x_val :  (1160, 128, 128, 3) , range: [ 0   255 ]
y_val :  (1160,) , values:  [0 1]
```

شکل ۲۶. خروجی حاصل از مرحله **Image Augmentation & Preprocessing**

5. Transfer Learning Using Pre-trained EfficientNet-B2

به طور کلی transfer learning در فرآیندی است که در آن از دانش به دست آمده از یک مدل از پیش آموزش دیده برای بهبود عملکرد یک مدل جدید در یک کار متفاوت اما مرتبط، استفاده می شود. در واقع به جای شروع از صفر، این امکان را می دهد از دانش مدل های موجود برای حل یک مشکل جدید با داده ها و منابع محاسباتی کمتر استفاده کنید. از مزایای استفاده از این روش می توان به موارد زیر اشاره کرد:

- Faster training
- Improved performance
- Better generalization
- Less data required
- More accurate results

برای استفاده از این روش ، مدل شبکه را به دو قسمت base-model و top-model تقسیم کردیم.

base-model همان شبکه EfficientNet-B2 است که برای وزن های ابتدایی آن به روش transfer learning از وزن های Imagenet استفاده کردیم. top-model را نیز با توجه به معماری که در مقاله مذکور آورده شده است، طراحی کردیم.

در این قسمت همچنین قصد داریم علت استفاده از شبکه EfficientNet را بیان کنیم. EfficientNet نوعی معماری شبکه عصبی عمیق است که به گونه ای طراحی شده است که از نظر منابع محاسباتی بسیار کارآمد است. این روش با استفاده از یک ضریب ترکیبی، تمام ابعاد عمق/عرض/رزولیشن را به‌طور یکنواخت مقیاس‌بندی می‌کند. همچنین استفاده از EfficientNet با داده‌های کوچک می‌تواند منجر به مدل‌های بسیار دقیق و با منابع کارآمد شود که می‌توانند به راحتی با استفاده از transfer learning با task جدید سازگار شوند. در این مثال نیز داده های ما به نسبت سایز کوچکی دارند و استفاده از EfficientNet برای رسیدن به دقت بالا بسیار کارآمد است. از دیگر مزایای آن می توان به موارد زیر اشاره کرد:

- Improved accuracy:

این مدل توانسته با تعداد پارامتر و توان محاسباتی کمتر در image classification task های مختلفی ، دقت بالا به دست آورد.

- Scalability:

مقیاس پذیر بودن این مدل از آن جهت است که می توان تصاویر ورودی با اندازه های مختلف را با کمترین نیاز به تنظیمات به شبکه داد و به دقت های بالایی نیز دست یافت.

- Efficiency:

از نظر محاسبات، حافظه و مصرف انرژی بسیار کارآمد است و آن را برای استفاده در دستگاه هایی با منابع محدود مانند تلفن های همراه و سیستم های تعبیه شده مناسب می کند.

- Transfer learning:

آن را می توان به عنوان یک مدل از پیش آموزش دیده و از طریق transfer learning بر روی انواع وظایف بینایی کامپیوتری استفاده کرد که نیاز به مجموعه داده های بزرگ برچسب گذاری شده را کاهش می دهد و روند آموزش را سرعت می بخشد.

ما در این سوال به پیشنهاد مقاله مذکور، از EfficientNet-B2 استفاده کردیم .

۲-۳. پیاده سازی شبکه

6. Fine-tuning the Top Layers

برای fine-tune کردن وزن لایه های top-model، ابتدا base-model را اصـــطلاحا freeze کردیم به این معنی که در فرایند training وزن های آن آپدیت نمی شـوند، و سـپس مدل را train کردیم. در طی فرایند training تنها وزن های مربوط به لایه های top-model آپدیت می شـوند. شـکل 27 معماری کلی شبکه طراحی شده را نشان می دهد.

```
img_input = tf.keras.layers.Input(shape=(IMAGE_SIZE, IMAGE_SIZE,3))
base_model = EfficientNetB2(include_top=False,  weights="imagenet", input_tensor=img_input)

# Freeze the pretrained weights
base_model.trainable = False

# Rebuild top
x = layers.GlobalAveragePooling2D(name="avg_pool")(base_model.output)
x = layers.Dense(128, activation="relu", name="dense_1")(x)
x = layers.Dropout(0.3, name="dropout_1")(x)
x = layers.Dense(64, activation="relu", name="dense_2")(x)
x = layers.Dropout(0.2, name="dropout_2")(x)

output = layers.Dense(1, activation="sigmoid", name="pred")(x)

# Compile
model = keras.Model(inputs = img_input, outputs =output , name="EfficientNet")
optimizer = keras.optimizers.Adam(learning_rate=5e-5)
model.compile(
    optimizer=optimizer,loss= keras.losses.BinaryCrossentropy(),
    metrics=["accuracy"]
    )
```

شکل ۲۷. معماری کلی شبکه طراحی شده

با توجه به آنچه در شکل 28 نشان داده شده است، مدل freeze شده در طی 29 epoch آموزش داده شد. در طی فرایند آموزش، از تصاویر Augment شده با batch-size = 128 به عنوان ورودی استفاده می شود. این بدین معنی است که در هر epoch، 128 تا از تصاویر ورودی به صورت رندوم انتخاب شده و Augment می شوند و در فرایند training به همراه تصاویر original مورد استفاده قرار می گیرند.

```
epochs = 29
history = model.fit(img_augmentation.flow(xTrain3ch,y_train, batch_size = 128),
                    epochs=epochs,
                    validation_data=(xVal3ch, y_val),
                    class_weight=class_weights,callbacks = [learning_rate_reduction])
```

شکل ۲۸. کد مربوط به فیت کردن مدل **freeze** شده

نتیجه حاصـل از این مرحله در شـکل 29 نشـان داده شـده اسـت. مدل تنها با fine-tune کردن top-model به دقت 90% بر روی داده های train و دقت 85% بر روی داده های validation رسیده است.

شکل ۲۹. دقت مدل **freeze** شده بر روی داده های **train**و **validation**

7. Hyper-parameter Tuning

تعدادی از hyper-parameter های شـــبکه از جمله تعداد epoch، learning rate (initialization)، batch-size، تعداد نورون های لایه های top-model، مقادیر لایه dropout، سـایز تصـاویر(128*128) و پارامتر های کلاس ImageDataGenerator، با توجه به مقادیر پیشنهادی در مقاله تعیین شدند.

8. LearningRateScheduler

Learning-rate از hyper-parameter هایی اسـت که مقدارش برای initialization با توجه به مقدار مقاله تعیین شده است اما با تعریف یک تابع scheduler طبق شکل زیر در طول فرایند training با توجه به مقدار val-accuracy مدل در هر ایپاک، مقدارش آپدیت می شـود. لازم به ذکر اسـت که این تابع تنها زمانی به عنوان callback function اسـتفاده می شـود که مدل freeze را آموزش می دهیم. زمانی که base-model را unfreeze کرده و مدل را دوباره آموزش می دهیم ، با توجه به پیشـنهاد اکثر مقاله ها ، مقداری بسـیار کم به learning-rate نسـبت می دهیم و نیازی به scheduler نیسـت. علت نسـبت دادن مقدار بسیار کم نیز این است که میخواهیم مقادیر وزن های base-model که در این مسئله همان مقادیر Imagenet هستند، زیاد تغییر نکند و مدل overfit نشود.

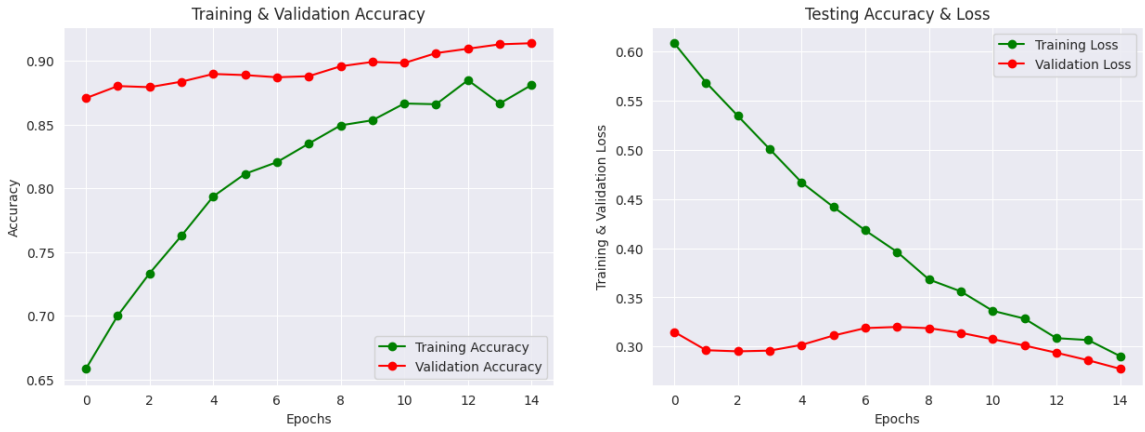
```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy')
```

9. Model Training

نهایتا طبق آنچه در شکل 30 نشان داده شده است، base-model را unfreeze کرده و دوباره در طی 15 ایپاک مدل را train می کنیم. در طی این فرایند با یک learning-rate بسیار کم، 5e-6، وزن های هر دو مدل base-model و top-model را آپدیت می کنیم. نتایج حاصل از این مرحله در شکل 31 نشان داده شده است. همانطور که مشاهده می شود دقت مدل بر روی داده های train و validation با شیب تقریبا مثبت افزایش یافته و در نهایت به دقت 91% برای داده های validation و 88% برای داده های train رسیده است. همچنین نمودار loss برای هر دو دیتاست نیز ترسیم شده است که آن هم روند نزولی داشته در طی 15 ایپاک و در نهایت به مقدار 0.27 برای داده های validation و 0.29 بر روی داده های train رسیده است.

```
base_model.trainable = True

model.compile(
    optimizer=keras.optimizers.Adam(5e-6),  # Low learning rate
    loss= keras.losses.BinaryCrossentropy(),
    metrics=[keras.metrics.BinaryAccuracy()],
)


epochs = 15
hist = model.fit(img_augmentation.flow(xTrain3ch,y_train, batch_size = 128),
                    epochs=epochs,
                    validation_data=(xVal3ch, y_val),
                    class_weight=class_weights,
                )
```

شکل ۳۱. آموزش مدل **unfreeze** شده



شکل ۳۰. نمودار **loss** و **accuracy** مدل بر روی داده های **train** و **validation**

10. Binary Classification

مسئله ما از آنجایی که دو کلاس دارد یک مسئله binary است و خروجی آن نیز به صورت binary(0,1) است.

11. Sigmoid Activation, Adam Optimizer, Binary Cross Entropy loss

همچنین برای لایه output شبکه طراحی شده از تابع Sigmoid به عنوان activation function استفاده شده است و دلیل آن دو کلاسه بودن مسئله است. از Adam Optimizer و Binary Cross Entropy loss به عنوان تابع loss نیز در طراحی شبکه استفاده شده است.

۴-۲. نتایج پیاده سازی

12. Model Evaluation

برای آنکه عملکرد مدل طراحی شده بر روی unseen data ارزیابی شود مطابق آنچه در شکل 32 نمایش داده شده است، از مدل برای پیش بینی کلاس تصاویر test استفاده کردیم.

```
predictions = model.predict(xTest3ch)
predictions = predictions.reshape(1,-1)[0]
predictions =np.where(predictions < 0.5, 0, 1)
```

شکل ۳۲. **Model prediction**

13. Classification Report, Confusion Matrix, ROC curve

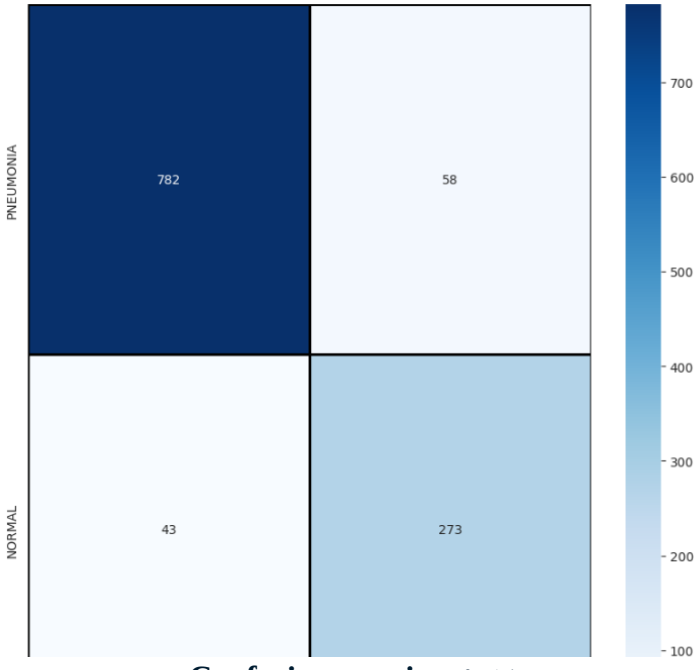
در این مرحله سعی برآن است که با استفاده از روش های مختلف دقت مدل طراحی شده ارزیابی شود. ابتدا به بررسی Classification Report که در شکل 33 نمایش داده شده است ، می پردازیم. همانطور که در شکل نشان داده شده است، مقدار precision برای کلاس Pneumonia به نسبت کلاس Normal بیشتر است. این بدین معناست که مدل در تشخیص تصاویر Pneumonia بهتر عمل کرده و از تعداد باری که تشخیص داده تصویر مربوط به کلاس Pneumonia است، تعداد کمتری را خطا پیش بینی کرده است. (fewer false positive) همچنین در خصوص recall نیز مدل برای کلاس Pneumonia بهتر عمل کرده است و از کل تصاویر Pneumonia تعداد بیشتری را توانسته به درستی تشخیص دهد. در کل و با توجه به مقدار f1-score میتوان دید که عملکرد کلی مدل برای کلاس Pneumonia بهتر بوده است . قابل ذکر

است اگرچه ما سعی کردیم با استفاده از class-weight اختلاف تعداد عکس های مربوط به دو کلاس را کمتر کنیم، اما با این وجود نتایج به دست آمده نشان می دهد نسبت بیشتر تصاویر مربوط به کلاس Pneumonia ، میتواند یکی از عومل موثر بر عملکرد بهتر مدل برای این کلاس باشد. همچنین دقت مدل بر روی داده های test به 91% رسیده که با توجه به آنکه دقت روی داده های train حدود 88% بود پس می توان نتیجه گرفت که مدل overfit نشده و قابلیت generalizable بودن و پیش بینی داده های unseen را دارد.

	precision	recall	f1-score	support
Pneumonia (Class 0)	0.95	0.93	0.94	840
Normal (Class 1)	0.82	0.86	0.84	316
accuracy			0.91	1156
macro avg	0.89	0.90	0.89	1156
weighted avg	0.91	0.91	0.91	1156

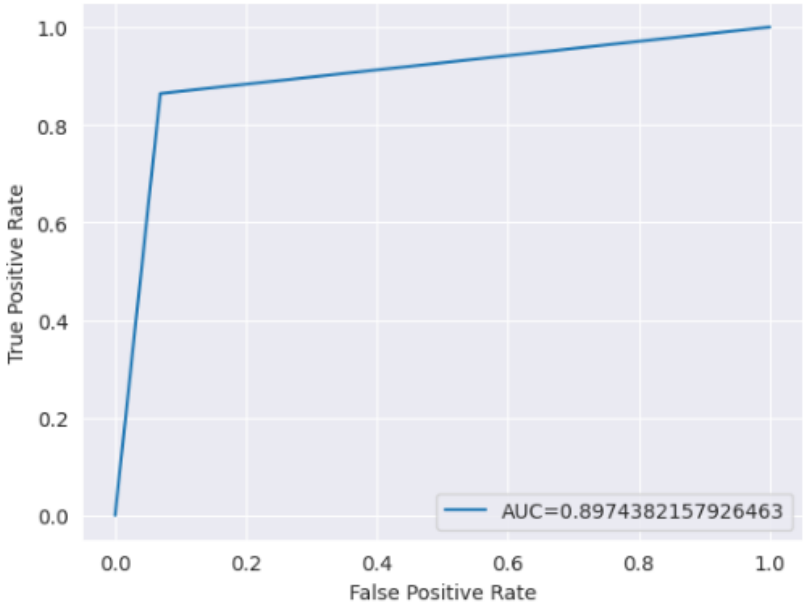
شکل ۳۳. **Classification report**

شکل34، confusion matrix مربوط به مدل را نشان می دهد. همانطور که در اینجا نیز مشخص است مدل توانسته تعداد بیشتری از تصاویر Pneumonia را درست تشخیص دهد. 58 تصویر به غلط Normal تشخیص داده شده است در حالی که تنها 43 تصویر به غلط برای کلاس Pneumonia پیش بینی شده است.



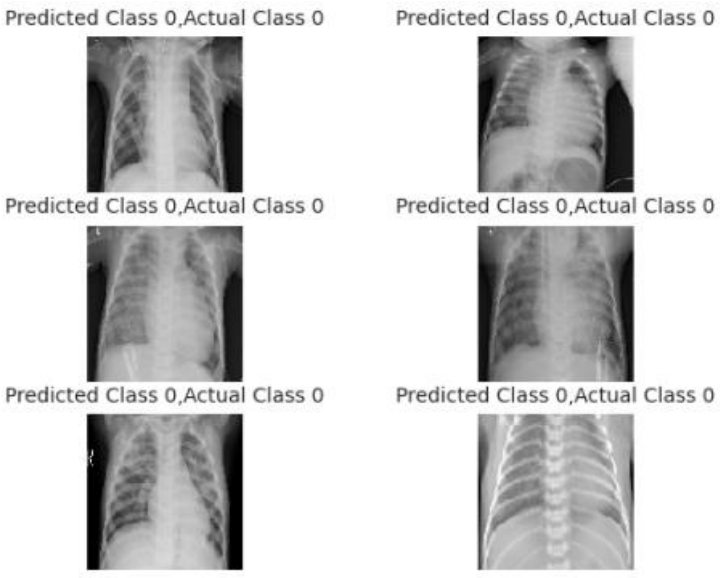
شکل ۳۴. **Confusion matrix**

به عنوان آخرین معیار برای ارزیابی model performance از ROC curve اســتفاده کردیم (شــکل 35). در این نمودار هرچقدر مســاحت زیر نمودار (AUC) به 1 نزدیک تر باشــد بدین معنی اســت که عملکرد مدل بهتر بوده. همانطور که نمایش داده شده است، این مساحت برای مدل ما 0.89 است .

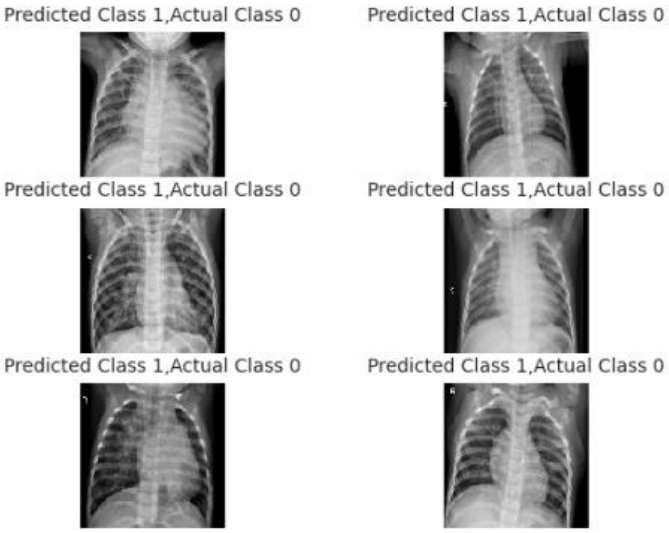


شکل ۳۵. **ROC Curve**

در نهایت برای داشتن دید بهتری از عملکرد مدل بر روی تصاویر مربوط به هر یک از دو کلاس، تعداد از تصاویر که به درستی یا به غلط توسط مدل پیش بینی شده اند را در شکل 36 و 37 نمایش داده ایم.



شکل ۳۶. نمونه ای از تصاویر درست پیش بینی شده توسط مدل



شکل ۳۷. نمونه ای از تصاویر غلط پیش بینی شده توسط مدل