

# به نام خدا



دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر



درس تحلیل و طراحی شبکه های عصبی عمیق

تمرین شماره ۳

نام و نام خانوادگی :

ماجده رضائی - ۸۱۰۶۰۱۰۷۳

محمد اخلاقی - ۸۱۰۶۰۱۰۲۷

آذر ماه ۱۴۰۲

## فهرست

۳	.....	مقدمه
۳	سوال اول : آشنایی با ساختار شبکه های عصبی	
۳	الف) آموزش شبکه VGG11 به صورت کلی	
۶	ب) آموزش لایه های شبکه	
۱۳	ج) آموزش لایه لایه بدون فریز کردن	
۲۱	د) داده تقویت نشده در مراحل آخر	
۲۵	و) لایه های اول با داده با افزونه کمتر	
۳۳	جمع بندی و بحث	
۳۵	سوال دوم : شبکه بخش بندی تصاویر دو بعدی	
۳۵	الف) معرفی دو شبکه UNet و pspnet	
۳۵	..... unet(۱)	
۳۶	..... PSPNet(۲)	
۳۷	ب) پیادهسازی شبکه Unet	
۳۷	(۱) توضیحات اولیه پیادهسازی	
۳۷	(۲) آماده سازی دیتا ست	
۳۸	(۳) ساخت شبکه Unet	
۴۰	ب) آموزش مدل Unet	
۴۱	ج) ارزیابی مدل UNET	
۴۲	ج) پیاده سازی شبکه PSPNET	

## مقدمه

در این تکلیف در سوال اول با یادگیری لایه به لایه یک شبکه و در سوال دوم با شبکه بخش بندی تصاویر دو بعدی آشنا می‌شویم و مدل‌هایی را برای آنان آموزش می‌دهیم. نتایج حاصل از پیاده‌سازی و بررسی هر مدل در ادامه آمده است. همچنین فایل کدهای اجرا شده به همراه این گزارش ارسال می‌گردد.

در سوال یک قسمت‌های الف و ب در یک کد، قسمت‌های ج و د در یک کد و قسمت و در یک کد قرار دارند.

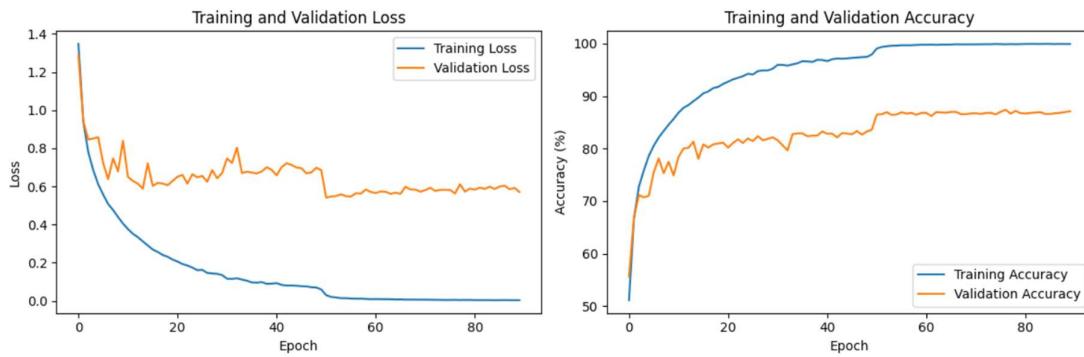
## سوال اول : آشنایی با ساختار شبکه‌های عصبی

### الف) آموزش شبکه VGG11 به صورت کلی

در ابتدا معماری شبکه VGG11 گفته شده پیاده‌سازی شد. این کار در کلاس VGG11 انجام شد.

سپس برای ایجاد دیتاست، مجموعه داده از Cifar10 دریافت شد. برای این مجموعه داده از transform های RandomHorizontalFlip ، RandomRotation ، ColorJitter برای ایجاد افزونگی روی دادگان آموزشی انجام شد و نرمالسازی نیز انجام شد. سپس دادگان ولیدیشن از دادگان آموزش جدا شد. برای توزیع بالانس دادگان ازتابع \_balance\_val\_split استفاده شد و ده درصد از دادگان آموزشی جدا شد. این تابع از [لينك](#) استخراج و صحت سنجی شد. سپس دادگان تست دانلود و تنسفورم نرمال سازی روی آن انجام شد. سپس برای هر سه دیتاست آموزش، آزمایش و اعتبارسنجی DataLoader به صورت مجزا ساخته شده و سایز batch = 256 و shuffle=True قرارداده شد.

برای اینکه در صورت وجود GPU بتوانیم از آن استفاده کنیم device را if torch.device("cuda") else "cpu" قرار گرفت و مدل را به آن انتقال یافت. تابع هزینه را CrossEntropyLoss و تابع بهینه‌ساز را SGD با نرخ یادگیری ۰,۰۱ و مومنتوم ۰,۹ و کاهش وزن نیز  $5 \times 10^{-4}$  قرار گرفت. حال این مدل با دیتاست توضیح داده شده را در یک حلقه‌ی for در ۹۰ گام آموزش می‌دهیم. قبل از این کار در متغیرهایی مانند train\_losses ایجاد می‌شوند تا دقت و خطاهای آموزش و ارزیابی را در هر دوره آموزش ذخیره کنند و در نهایت این مقادیر برای رسم نمودار استفاده می‌شوند. در آخر نمودار هزینه و دقت نمایش داده شد. در ۹۰ دقت داده‌های ترین ۹۹,۹۳٪ و داده‌های ولیدیشن ۸۷,۱٪ است. در شکل زیر نمودارها مشاهده می‌شود.



شکل ۱: نمودارهای دقت و هزینه در حین آموزش شبکه **VGG11**

همانطور که مشخص است حدودا پس ۵۰ مرحله شبکه به دقت نهایی رسیده است.

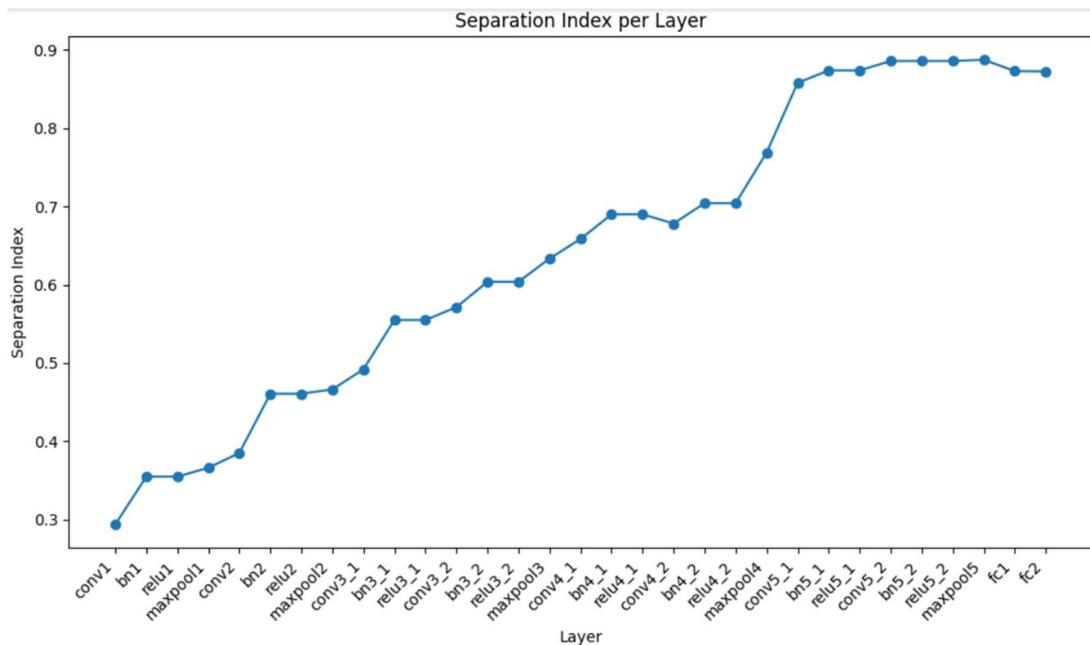
سپس شبکه ذخیره شد و دقت شبکه بر داده های تست هم محاسبه شد. مقدار دقت و هزینه نهایی به شرح زیر است.

Test Loss: 0.5791      Test Accuracy: 87.75%

برای محاسبه **CSI** تابع **Center SI** بازنویسی شد. دلیل این کار محدودیت GPU در محاسبه بود. در این تابع در حلقه **for** مرکز هر کلاس محاسبه میشود و سپس به کمک تابع **cdist** فاصله تمامی نقاط با این مرکز دسته محاسبه میشود. مرکز دارای کمترین فاصله برای هر داده به عنوان لیبل جدید ذخیره و با مقایسه لیبل جدید و لیبل اصلی CSI محاسبه میشود.

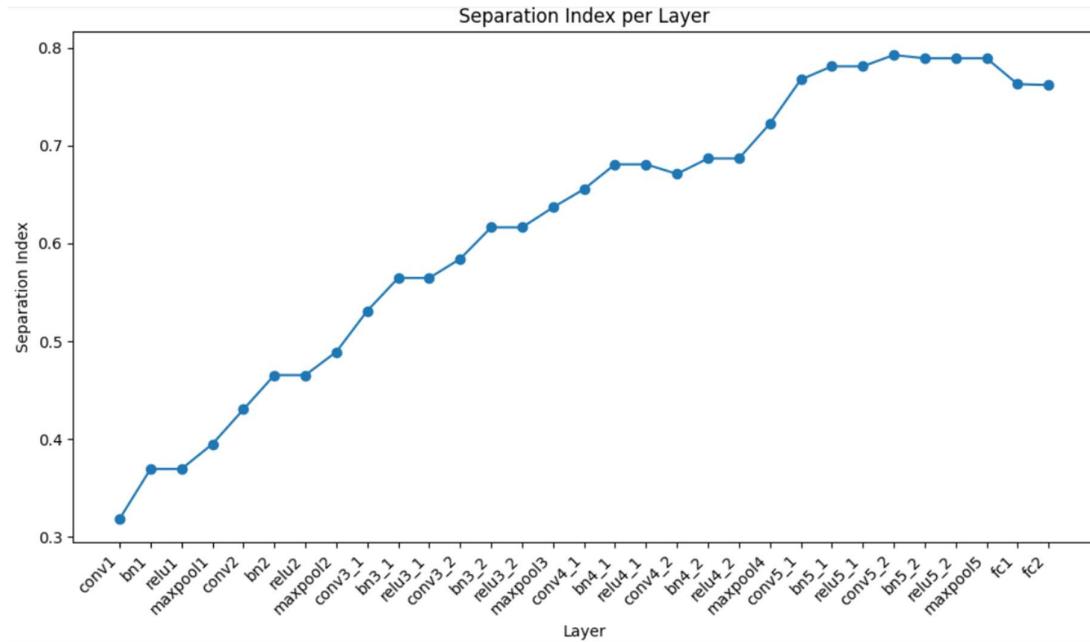
در این بخش با همان دیتاهای قبلی و به کمک تابع `_balance_val_split` درصد از دادگان **train** (بدون **cuda**) جداسازی شد. سپس دادگان و لیبل ها جدا شدند. در ادامه پس فراخوانی شبکه ذخیره شده و وارد کردن آن به **CenterSI** GPU فراهم شد. سپس از تابع **hook** که در تمارین قبلی نیز استفاده شده بود برای محاسبه فیچرهای هر لایه استفاده شد و در حلقه **for** لایه به کمک تابع **center\_si\_cal** محاسبه و در لیست **CSI** ذخیره شد. پس از این کار نمودار **CSI** هر لایه رسم شد.

برای دادگان آموزش:



شکل ۲: نمودار center SI در هر لایه VGG11 network برای دادگان آموزشی

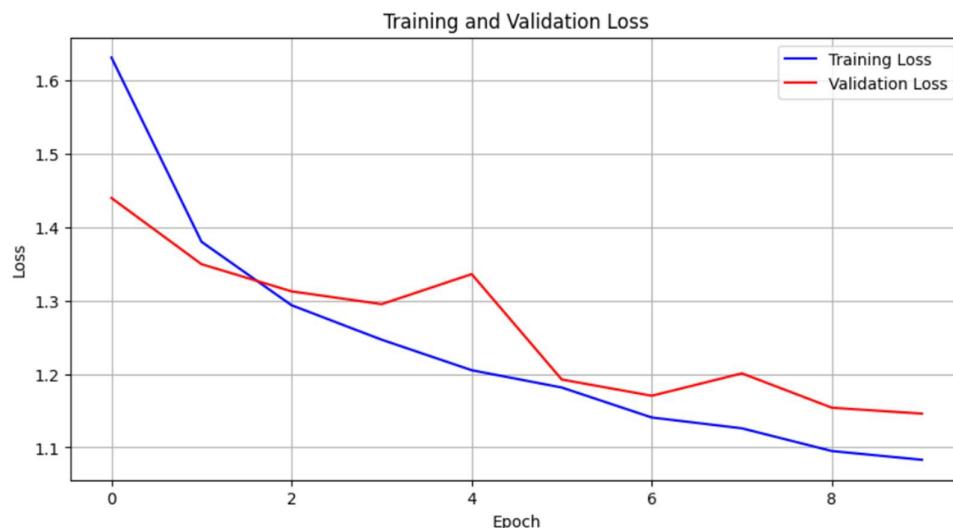
مشاهده میشود که SI در مجموع با عمیق تر شدن در شبکه افزایش یافته است. برای دادگان تست نیز انجام شد و نمودار زیر به دست آمد. به طور منطقی مشخص است مقادیر SI برای دادگان تست کمتر است.



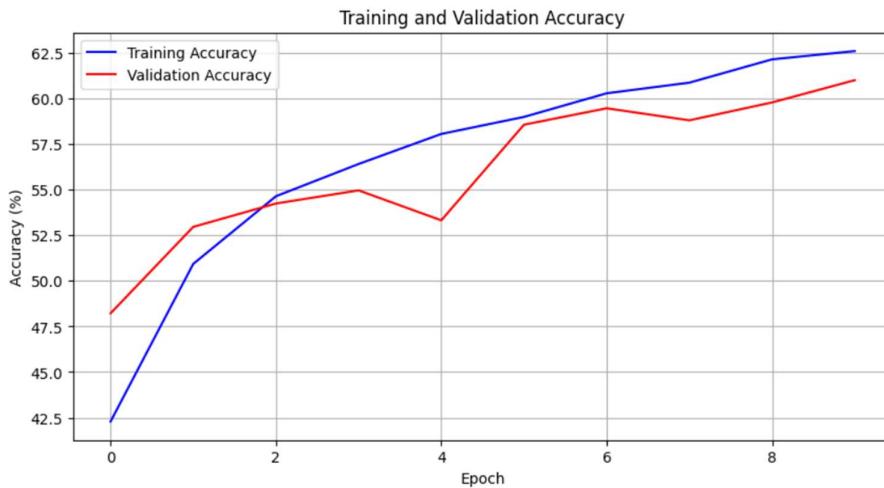
شکل ۳: نمودار center SI در هر لایه custom network برای دادگان تست

## ب) آموزش لایه‌ای شبکه

در این قسمت باید شبکه به صورت لایه به لایه تعریف شود و با اضافه شدن هر لایه آموزش داده شود. برای این کار از همان دیتاست قبلی استفاده شد و فرایند بخش الف بر روی دادگان پیاده شد. لایه‌های کانولوشنی نیازمند آموزش به صورت جداگانه هستند و جداسازی لایه‌های نرم‌الیزیشن و فعال‌ساز و **pooling** به تنها‌ی معنا ندارد. جداسازی سپس شبکه به صورت لایه اول( بلاک اول) تعریف شد و تنها در انتهای آن ۲ لایه **fully connected** قرار گرفت که لایه اول سایز خروجی بلاک است. مشابه بخش اول با هایپرپارامترهای گفته شده در صورت سوال آموزش داده شد. تنها به دلیل کمبود GPU و زمان زیاد آموزش به ۱۰ ایپاک برای این لایه بسته کردیم. نمودار دقیق و هزینه دادگان **validation** در شکل زیر آمده است.



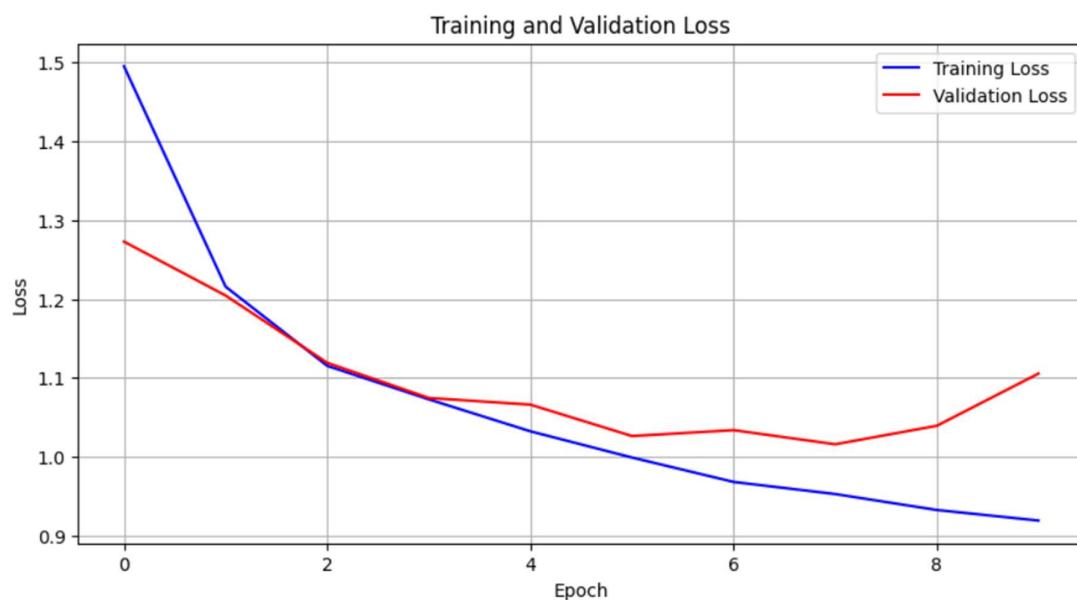
شکل ۴: نمودار هزینه در هنگام آموزش بلاک اول شبکه



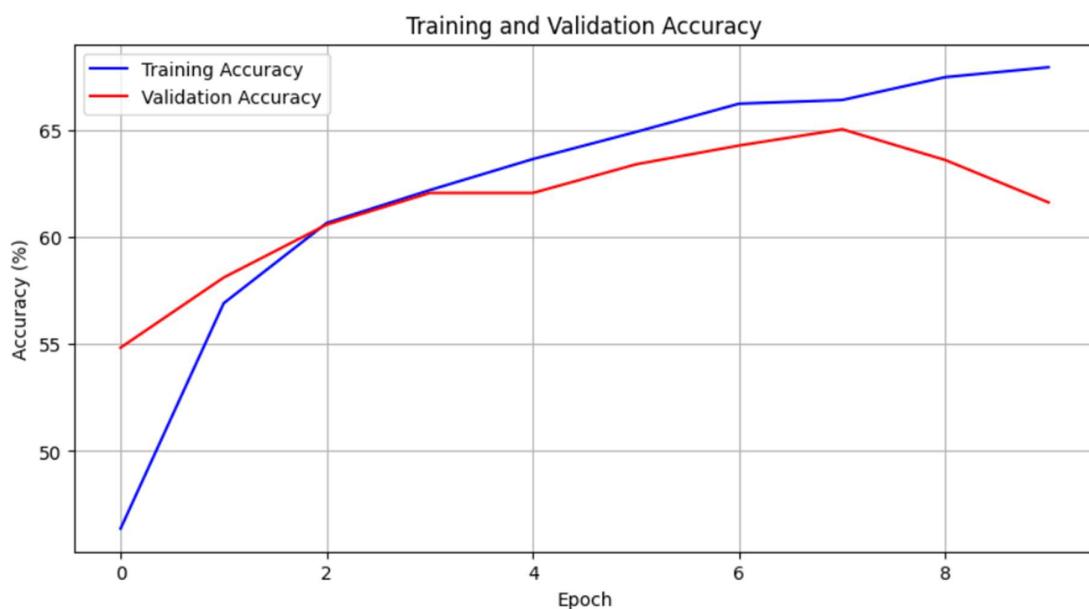
شکل ۵: نمودار دقت در هنگام آموزش بلاک اول شبکه

سپس اوزن بلاک ۱ ذخیره شد تا در مرحله بعد استفاده شود. (لایه **maxPooling** وزن خاصی ندارد و تنها سایز فیچر را بر اساس کرنل خود تغییر میدهد).

سپس بلاک دوم متشکل از کانلوشن و مشتقات اضافه شد و مشابه قبل با سایز **fully-connected** مناسب انتهای آن قرار گرفت. تابع **load\_block1\_weights** برای جایگذاری وزن های ذخیره شده در قسمت قبل در شبکه جدید تعریف شد. پس از تعریف مدل و جایگذاری وزن ها، لایه های **Batch normalization** و **convolution** فریز شدند. و مانند قسمت قبل شبکه دو بلاکه ترین شد. نمودار آموزش و ولیدیشن در شکل زیر قرار دارد.



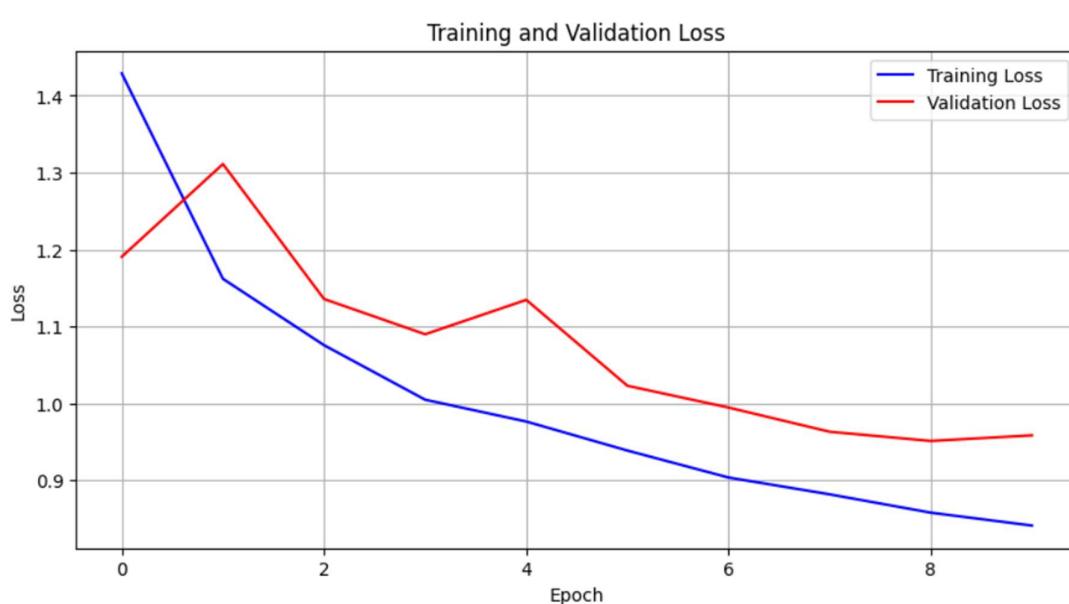
شکل ۶: نمودار هزینه در هنگام آموزش بلاک دوم شبکه



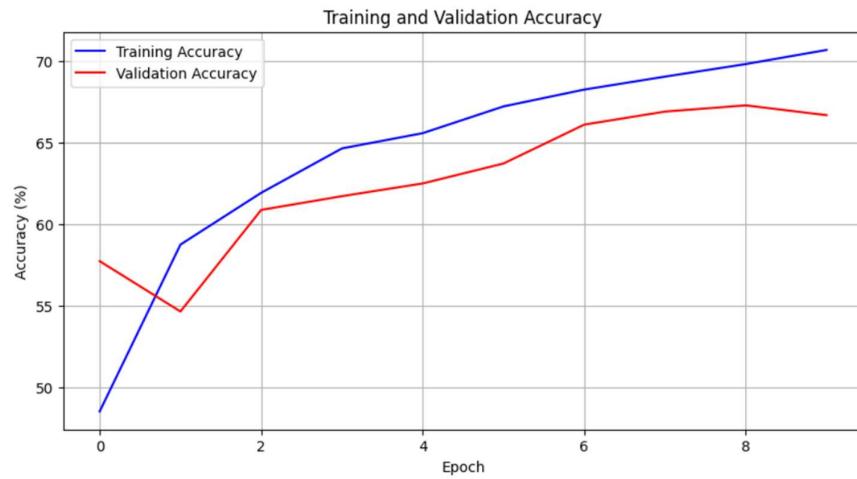
شکل ۷: نمودار هزینه در هنگام آموزش بلاک اول شبکه

مشابه قسمت قبل مراحل تکرار شد و در هر مرحله یک لایه (بلاک) اضافه شد.

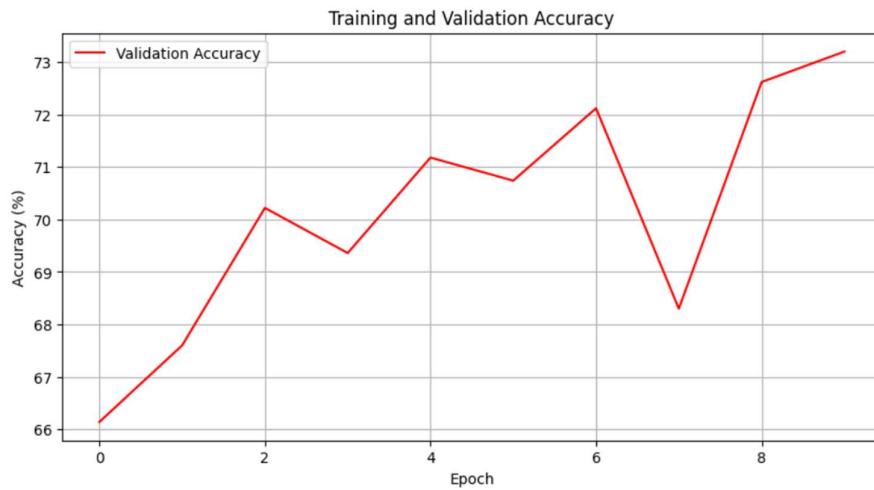
در ادامه نمودار آموزش تا لایه سوم را مشاهده میکنیم.



شکل ۸: نمودار هزینه در هنگام آموزش بلاک سوم شبکه

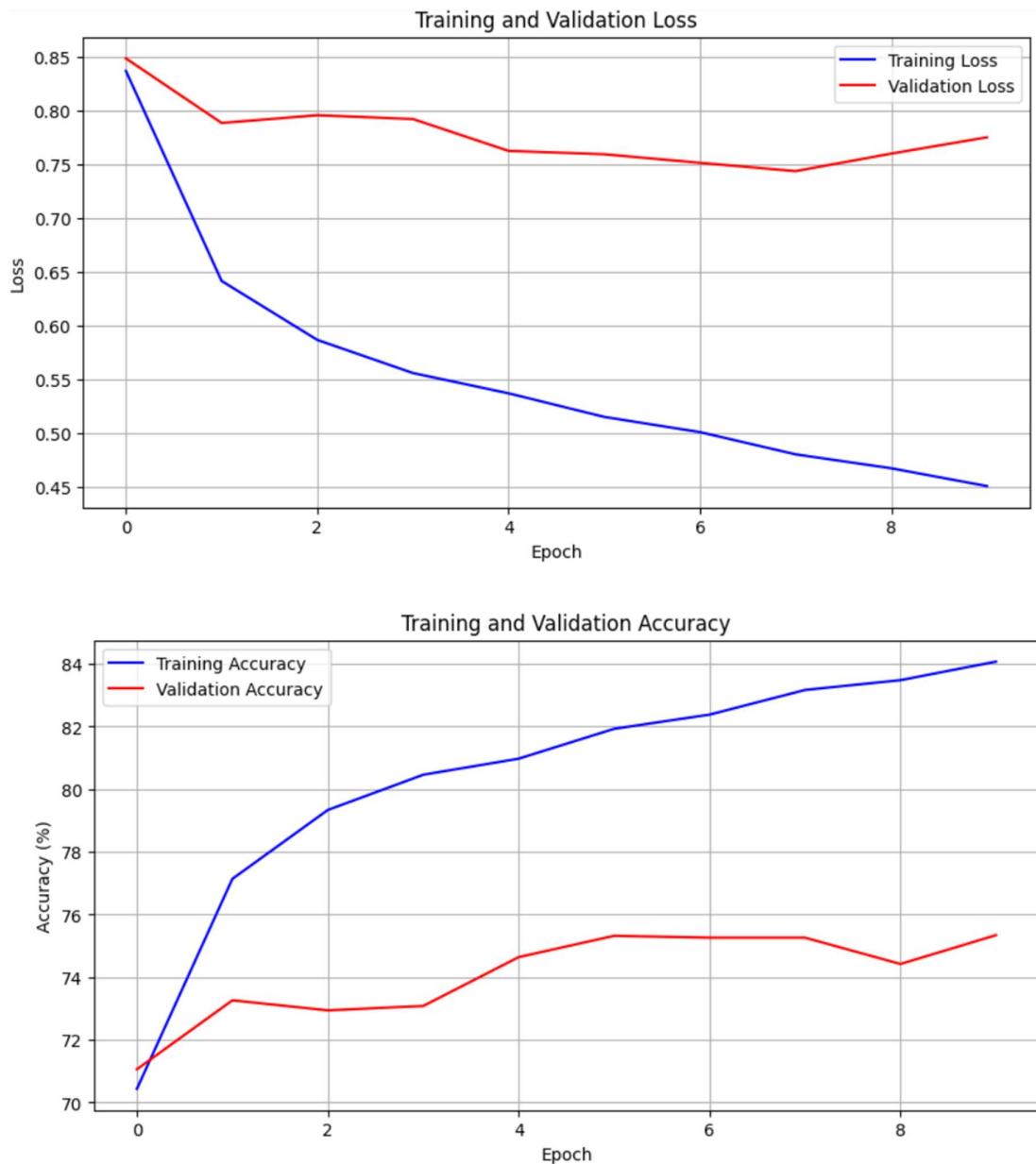


شکل ۹: نمودار نتیجه در هنگام آموزش بلاک سوم شبکه  
در ادامه نمودار آموزش تا لایه چهارم را مشاهده میکنیم.



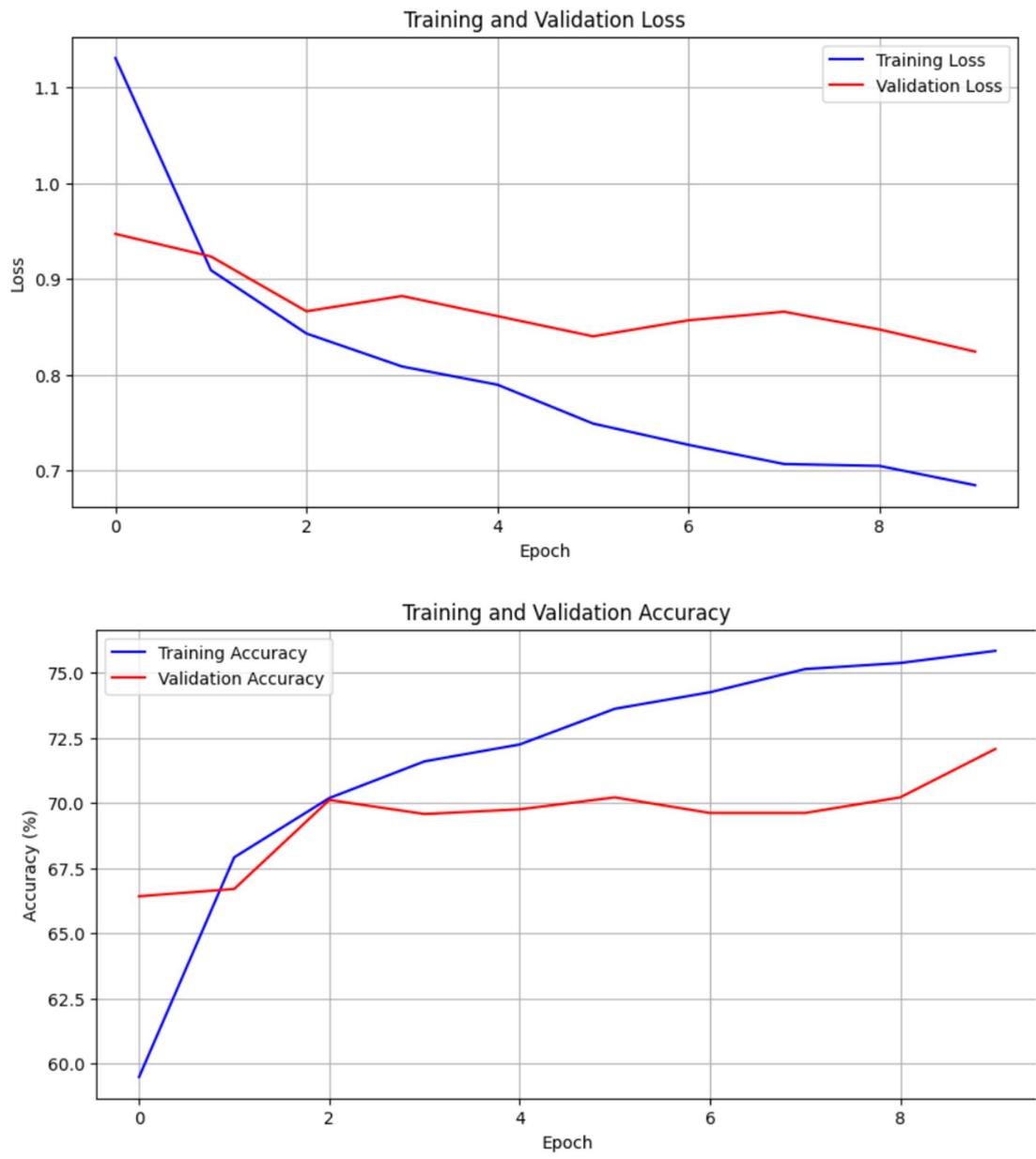
شکل ۱۰: نمودارهای هزینه و دقت در لایه چهارم

در ادامه نمودار آموزش تا لایه پنجم را مشاهده میکنیم.



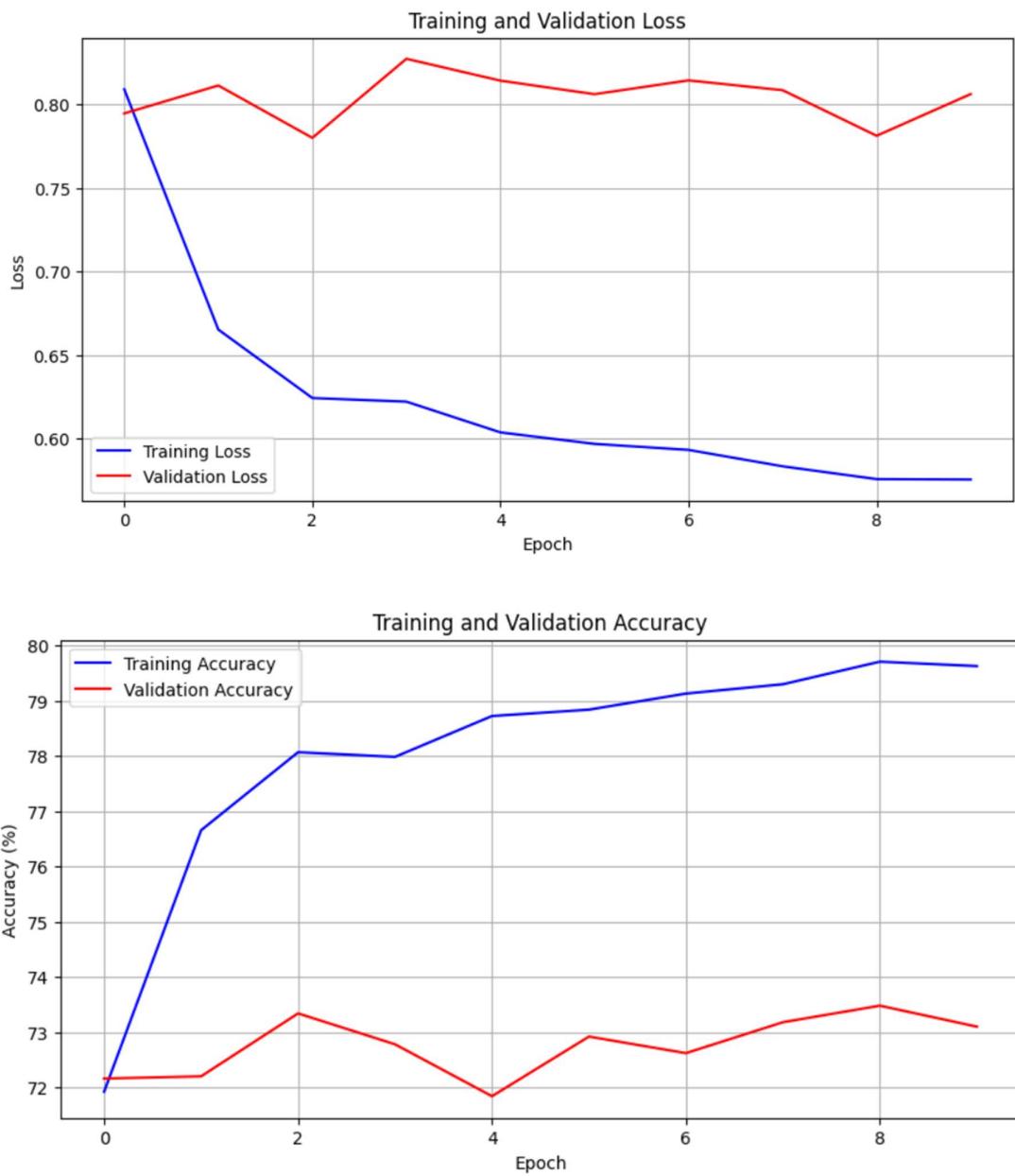
شکل ۱۱: نمودارهای هزینه و دقت در لایه پنجم

لایه ششم:



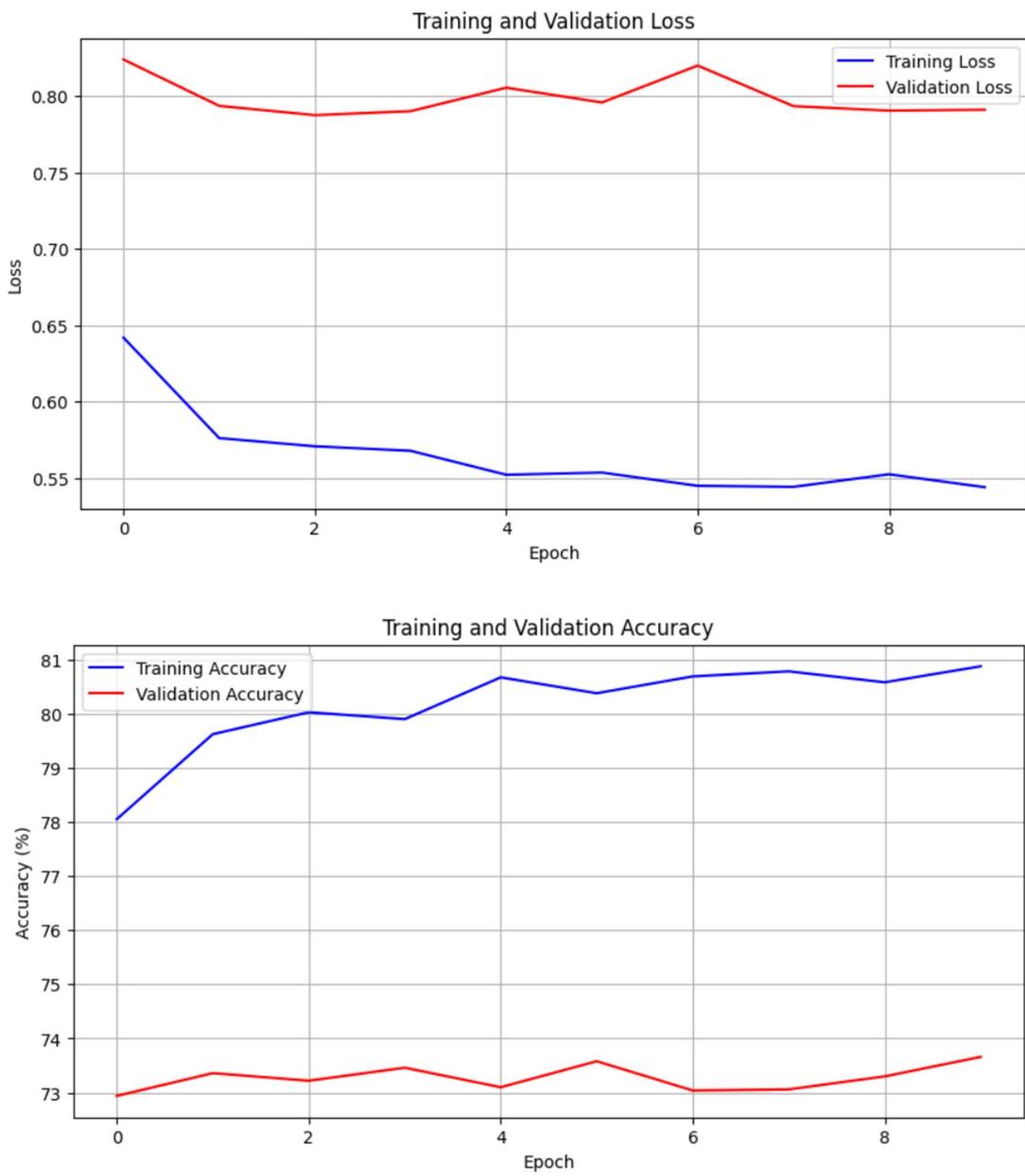
شکل ۱۲: نمودارهای هزینه و دقت در لایه ششم

لایه ۷:



شکل ۱۳: نمودارهای هزینه و دقت در لایه هفتم

لایه ۸:



شکل ۱۴: نمودارهای هزینه و دقت در لایه هشتم

عملکرد روی دادگان تست:

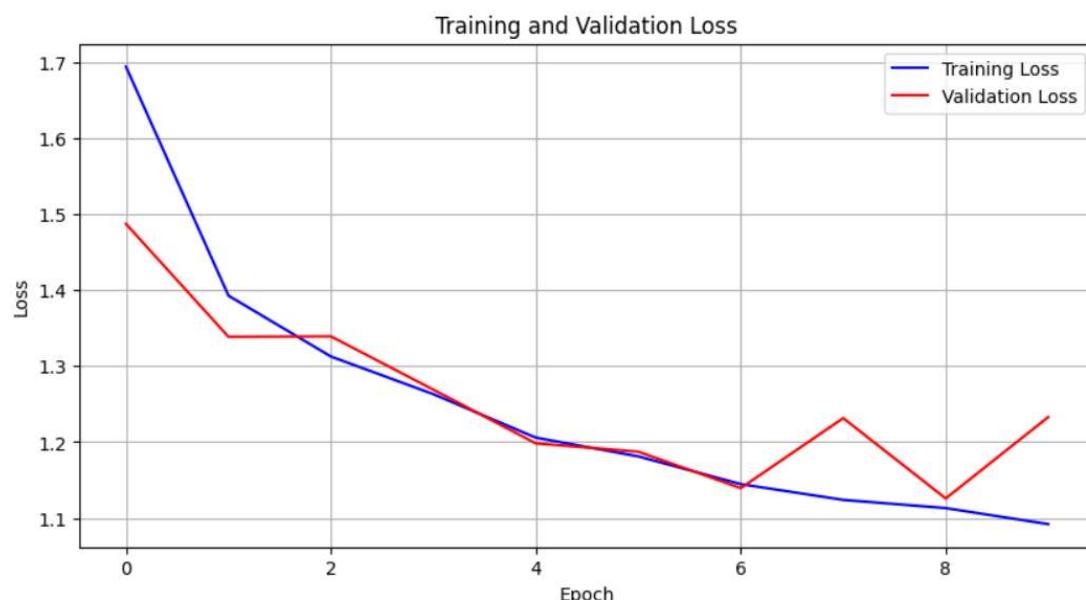
Test Loss: 0.8584

Test Accuracy: 72.20%

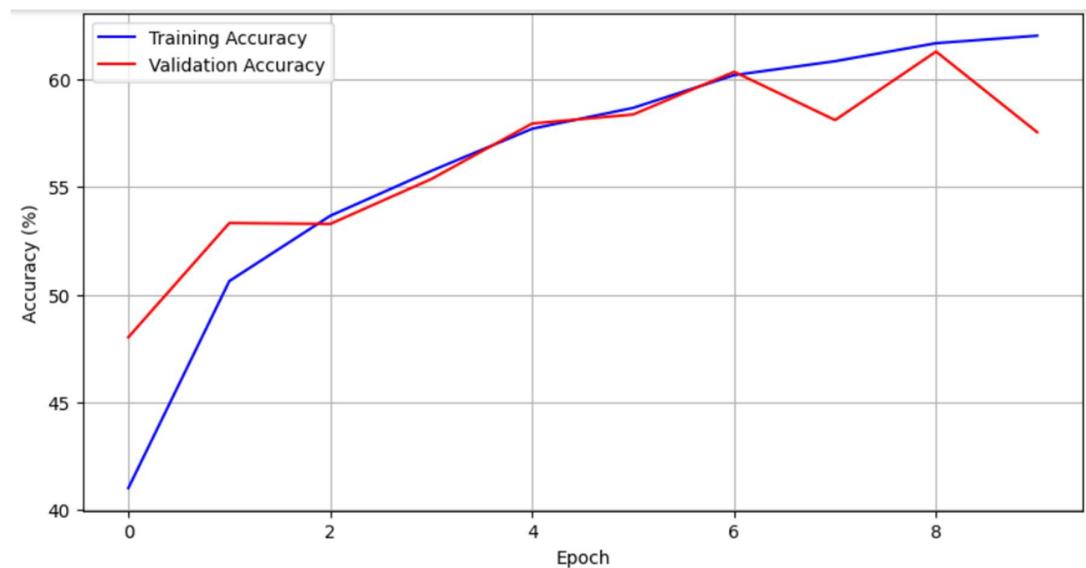
ج) آموزش لایه لایه بدون فریز کردن

مشابه قسمت قبل عمل میکنیم تنها در قسمت های فریز وزن ها نیاز به آموزش را **True** میکنیم. سپس آموزش لایه به لایه مشابه قسمت الف انجام میشود. در اینجا تنها نمودارهای آموزش نمایش داده میشود.

نمودار آموزش و ولیدیشن در شکل زیر قرار دارد.



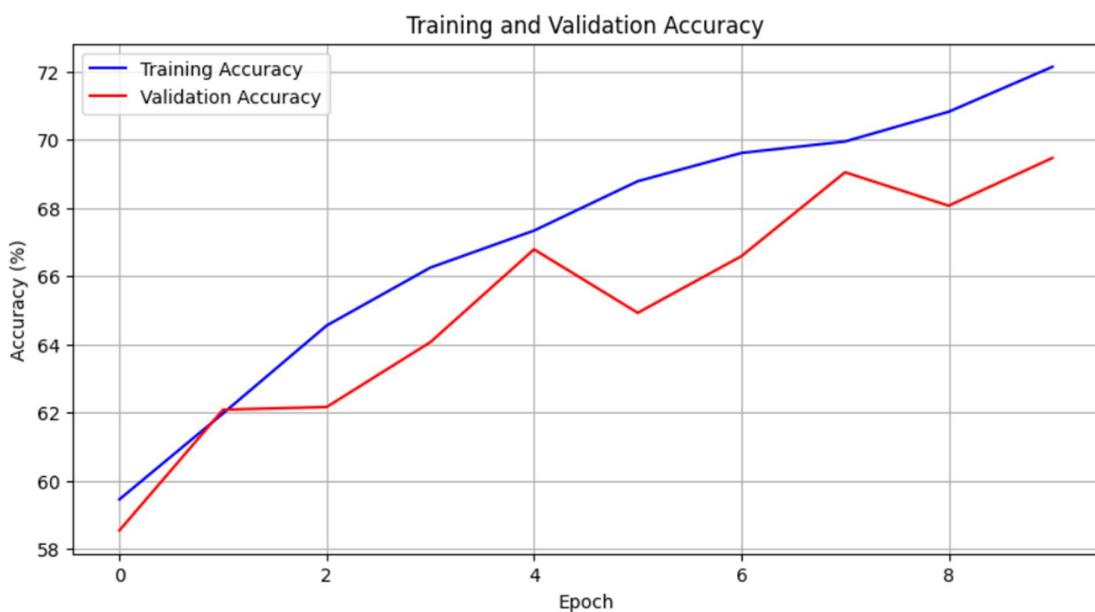
شکل ۱۵: نمودار هزینه در هنگام آموزش بلاک اول شبکه



شکل ۱۶: نمودار هزینه در هنگام آموزش بلاک اول شبکه



شکل ۱۷: نمودار هزینه در هنگام آموزش بلاک دوم شبکه



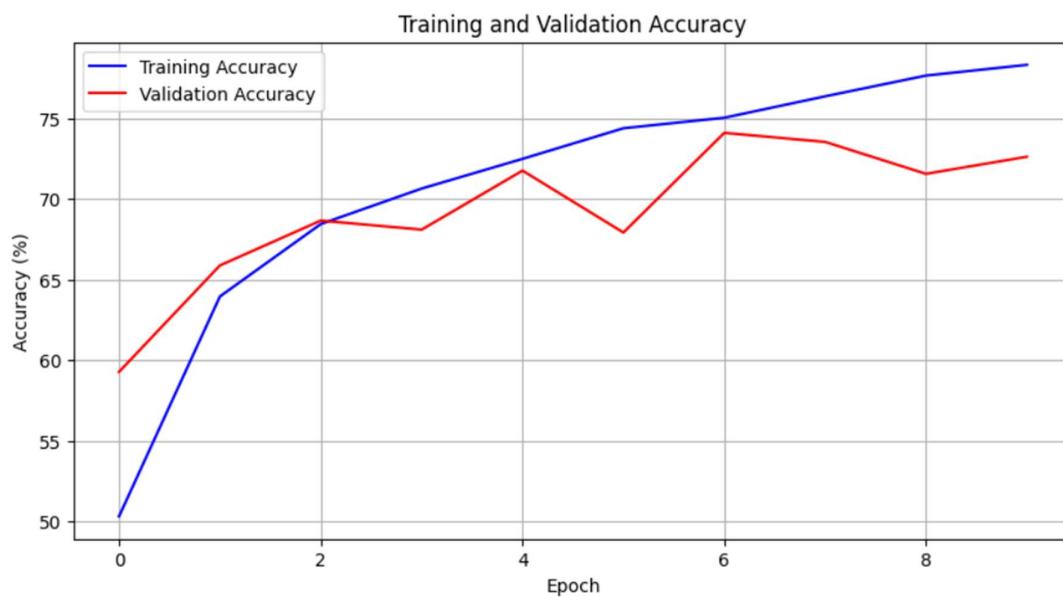
شکل ۱۸ نمودار هزینه در هنگام آموزش بلاک دوم شبکه

مشابه قسمت قبل تکرار شد و در هر مرحله یک لایه(بلاک) اضافه شد.

در ادامه نمودار آموزش تا لایه سوم را مشاهده میکنیم.

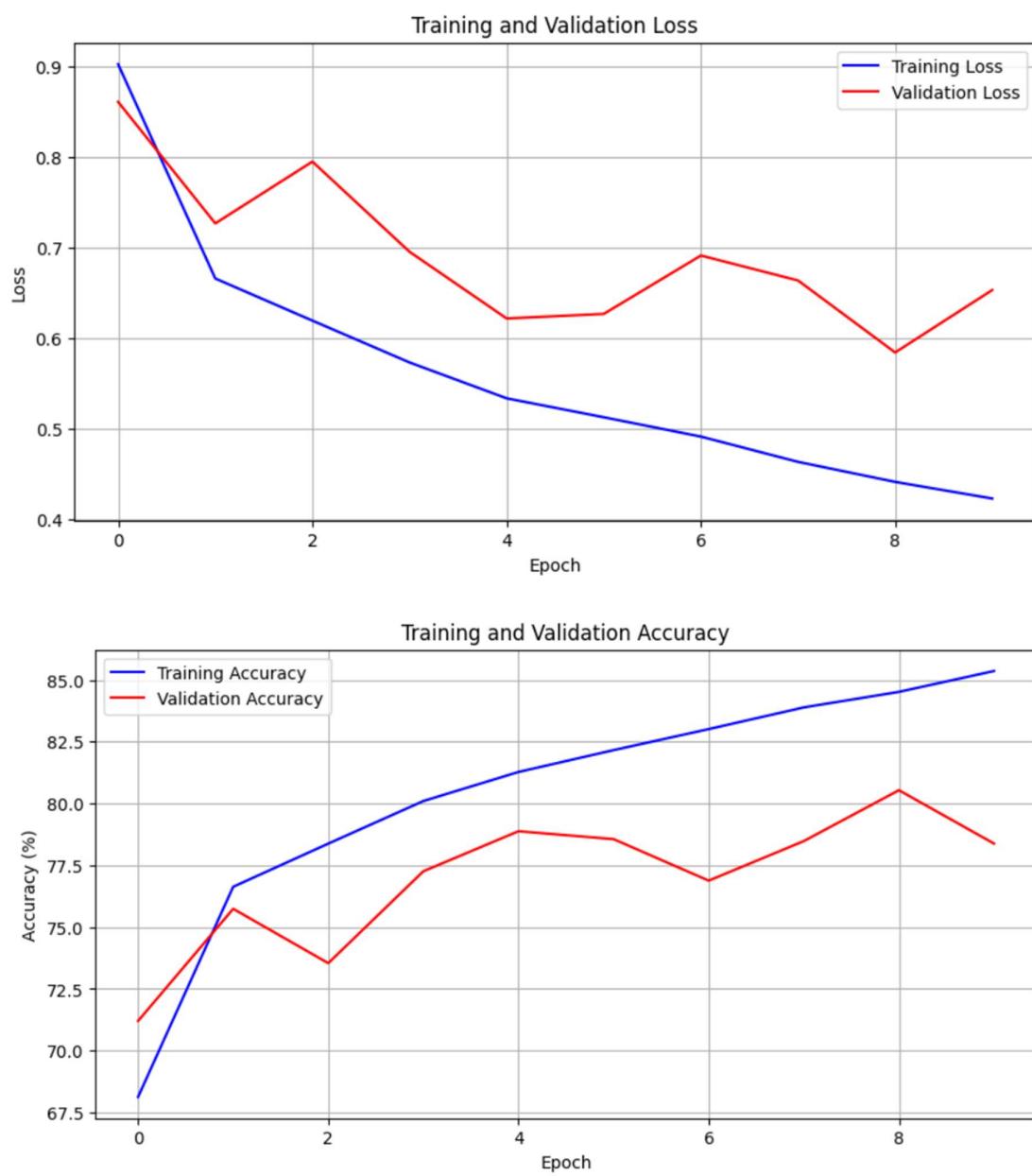


شکل ۱۹: نمودار هزینه در هنگام آموزش بلاک سوم شبکه



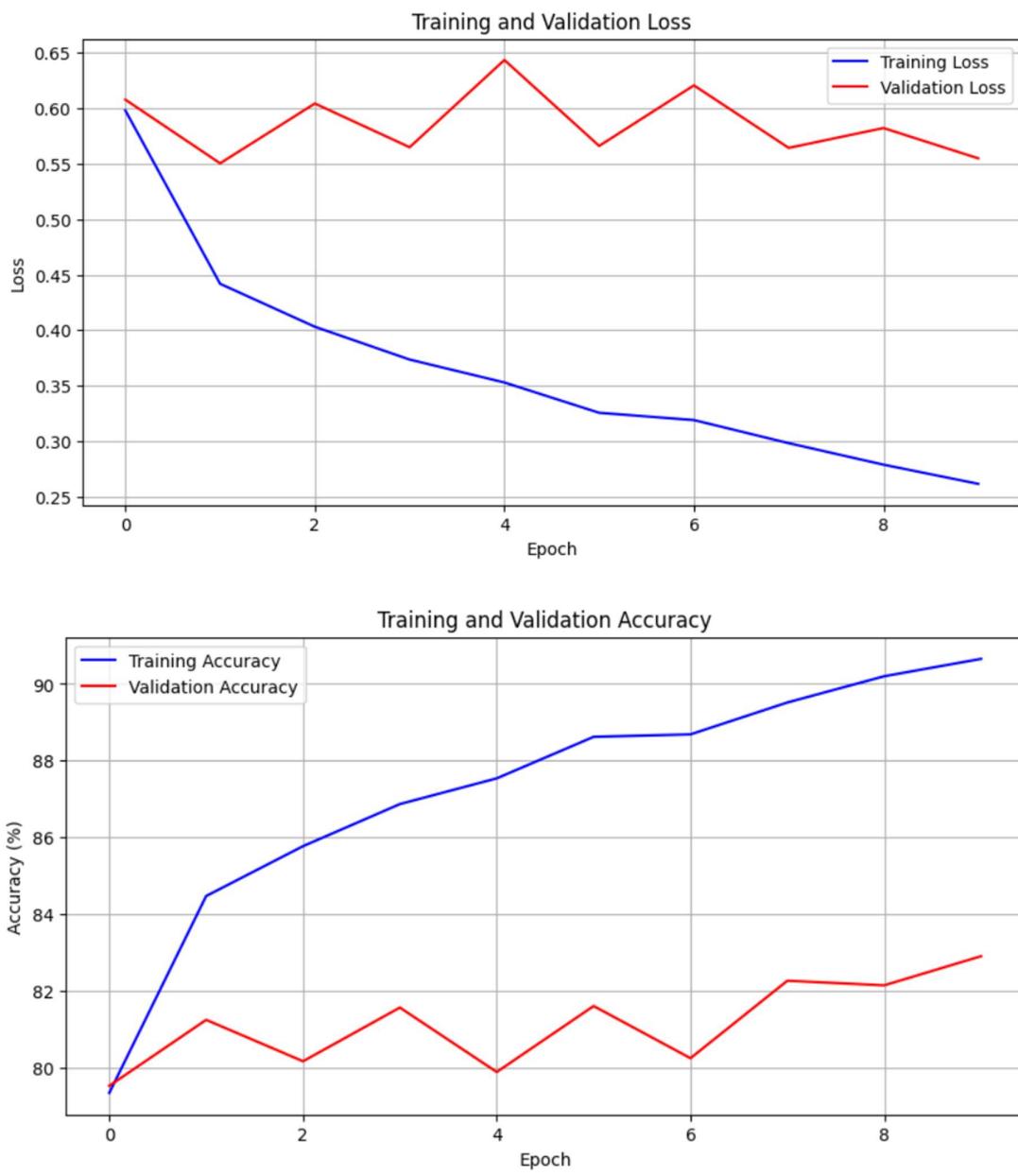
شکل ۲۰: نمودار نتیجه در هنگام آموزش بلاک سوم شبکه

لایه چهارم:



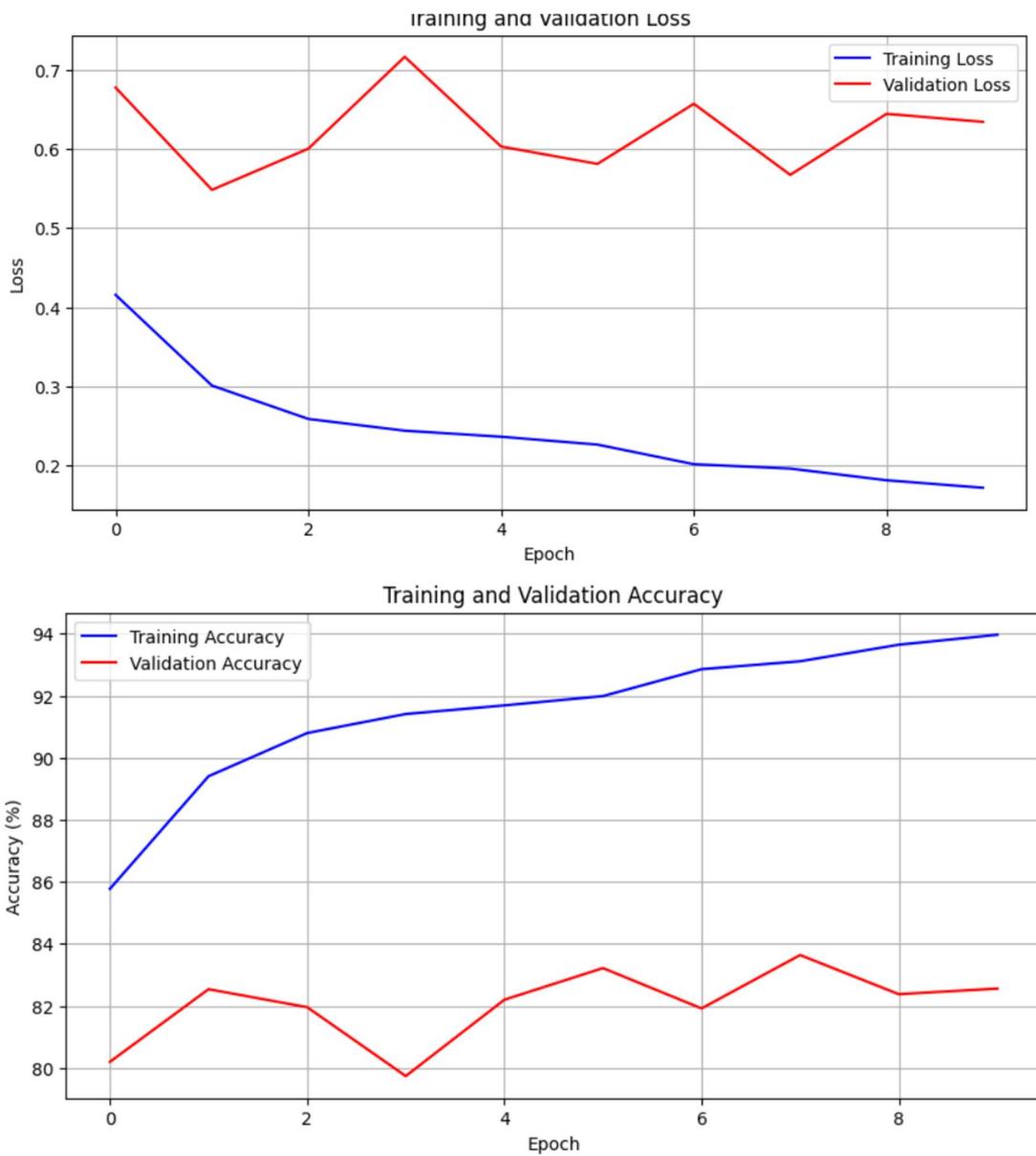
شکل ۲۱: نمودارهای هزینه و دقت لایه چهارم

لایه پنجم:



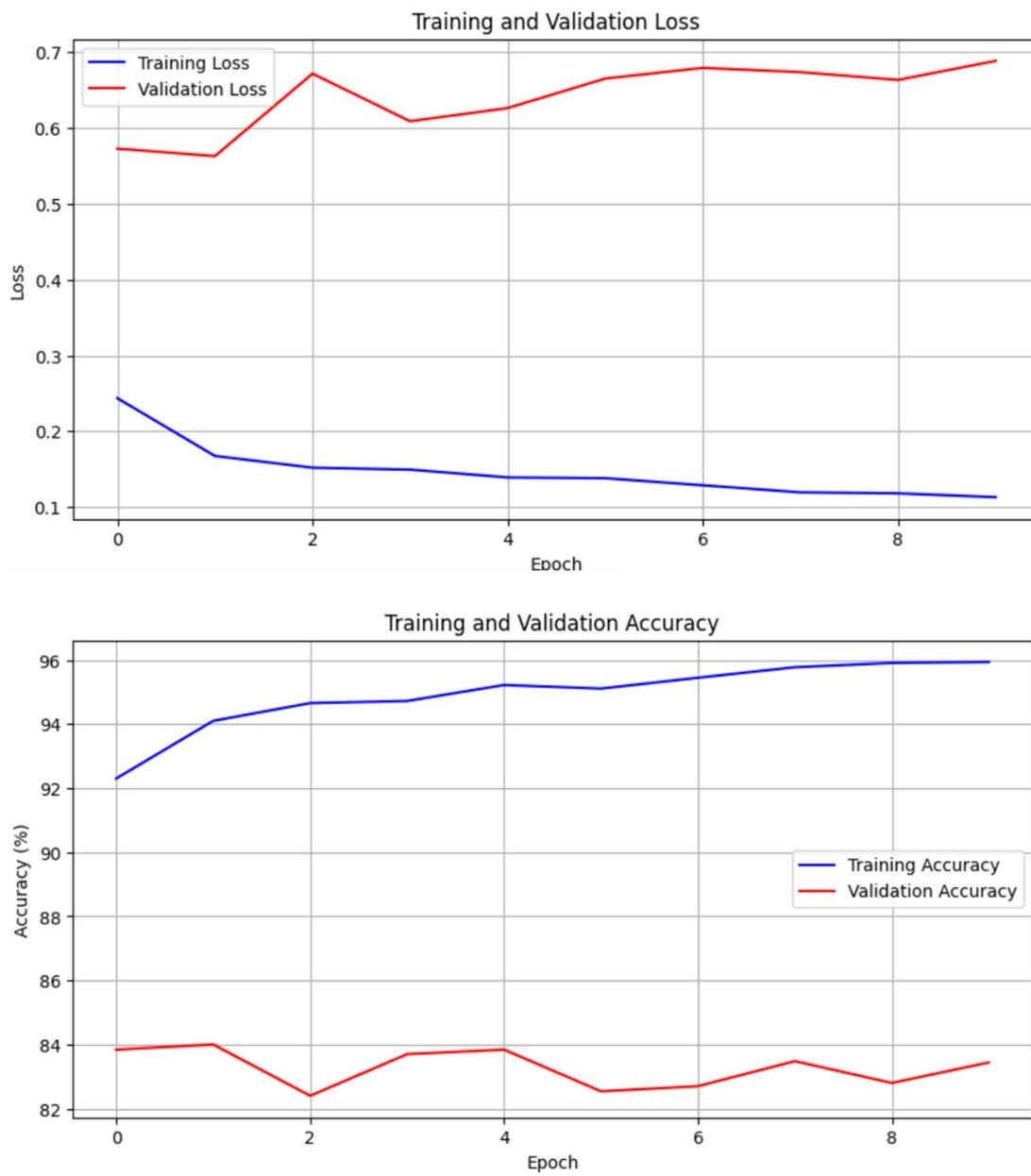
شکل ۲۲: نمودارهای هزینه و دقت لایه پنجم

لایه ۶



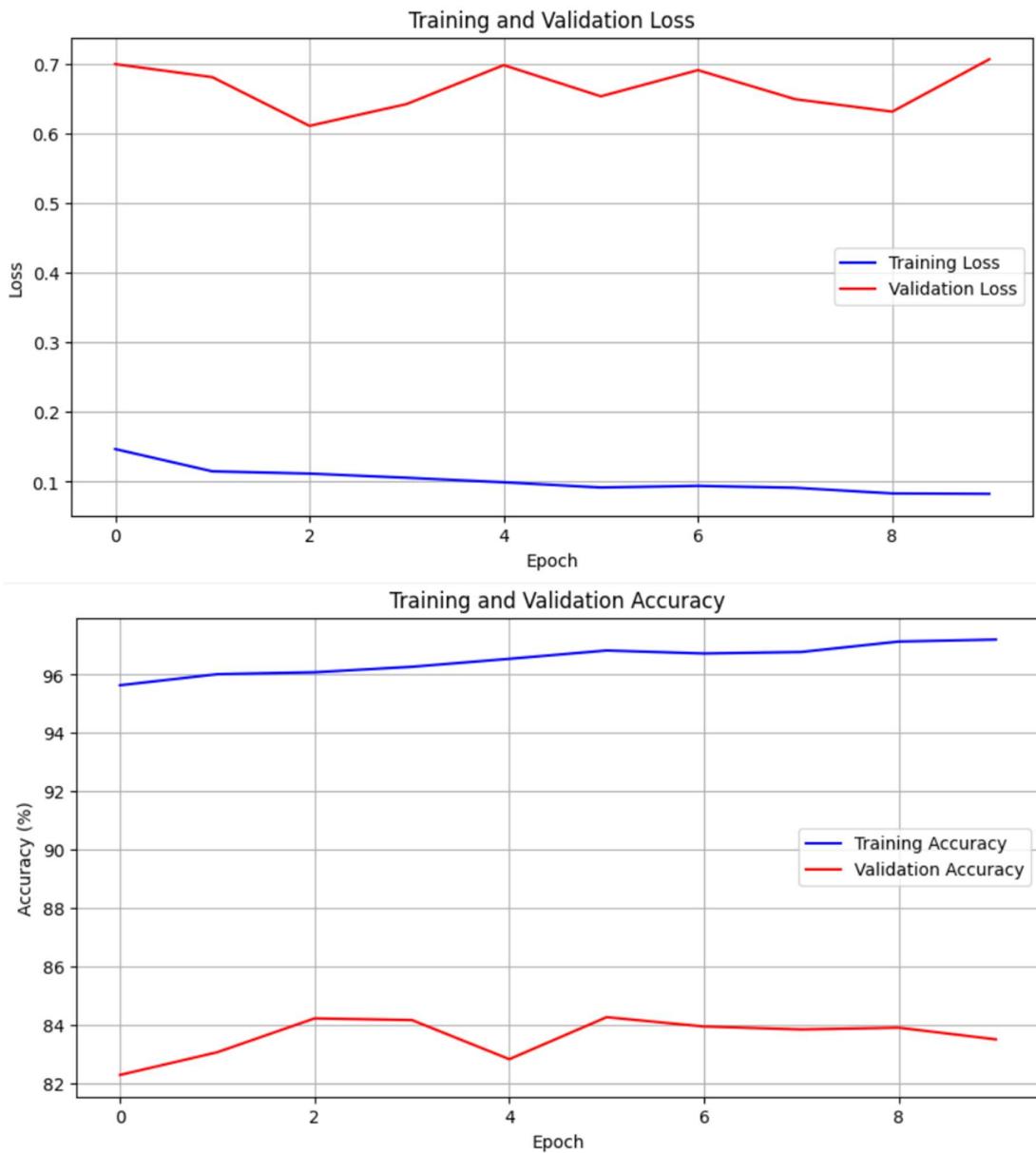
شکل ۲۳: نمودارهای هزینه و دقت لایه ششم

لایه هفت:



شکل ۴: نمودارهای هزینه و دقت لایه هفتم

لایه هشتم:



شکل ۲۵: نمودارهای هزینه و دقت لایه آخر

دقت بر روی دادگان تست:

Test Loss: 0.6474

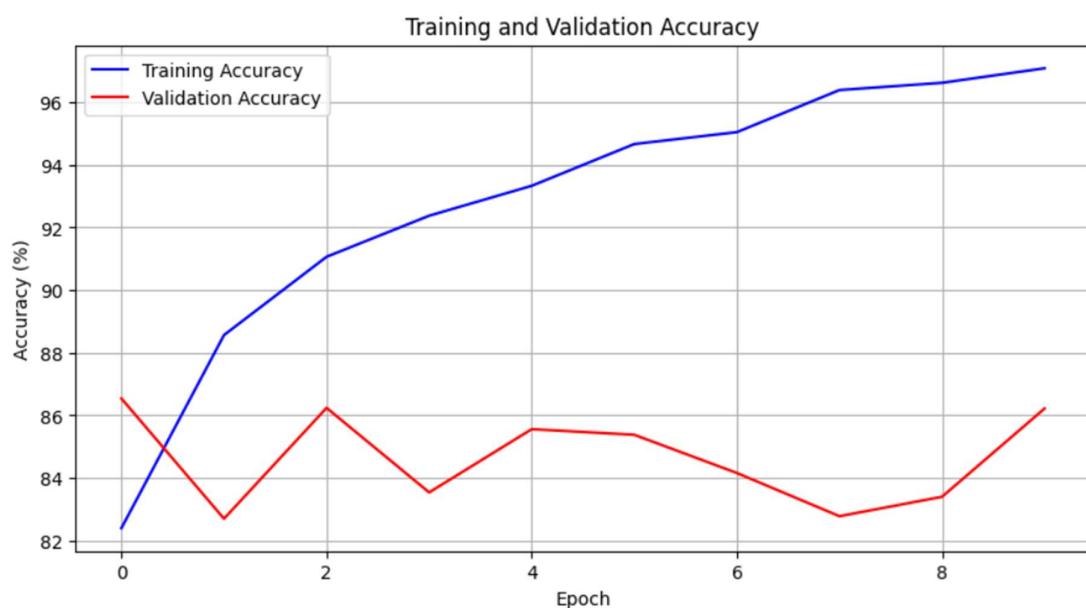
Test Accuracy: 85.40%

#### د) داده تقویت نشده در مراحل آخر

مانند قسمت ج عمل کردیم فقط پس از لایه چهار یک بار دیگر دیتابست با افزونه کمتر (فقط flip) تعریف شد. از همان ضرایب ذخیره شده در قسمت ج و لایه ۴ استفاده شد و از تکرار ۴ لایه ابتدایی اجتناب شد.

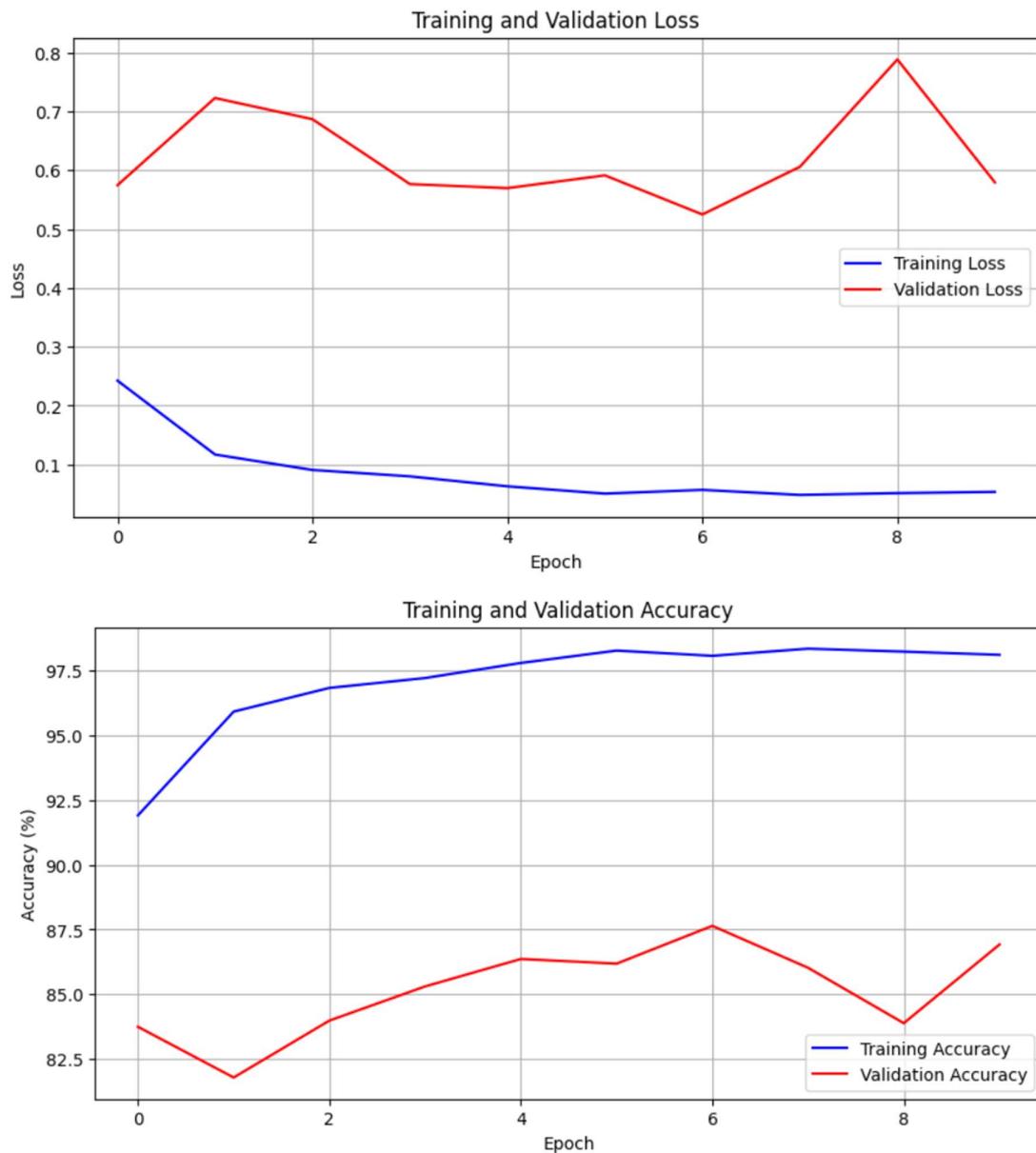
نمودار چهار لایه آخر به شکل زیر است:

لایه ۵



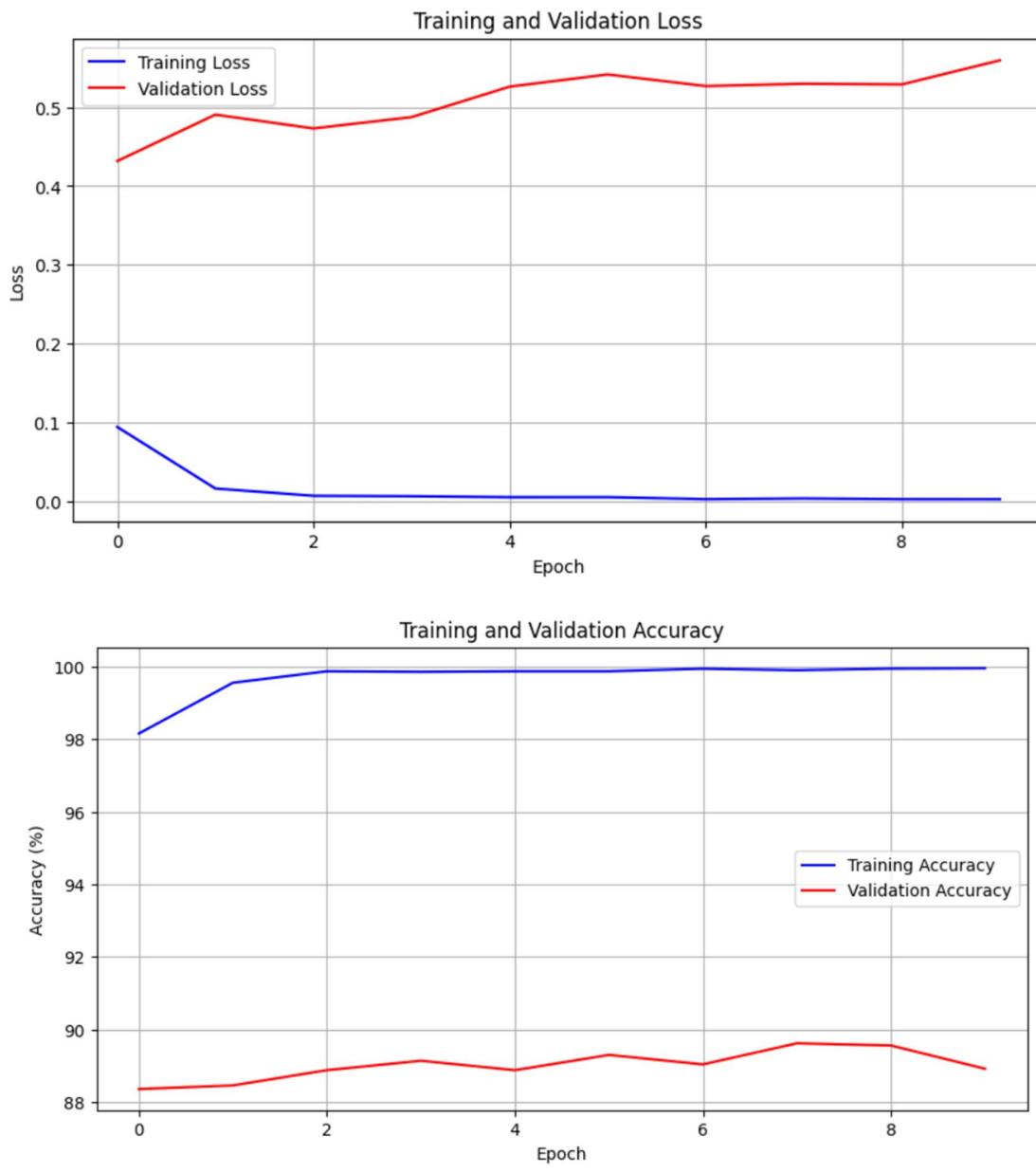
شکل ۲۶: نمودارهای هزینه و دقت لایه پنجم

لایه ششم:



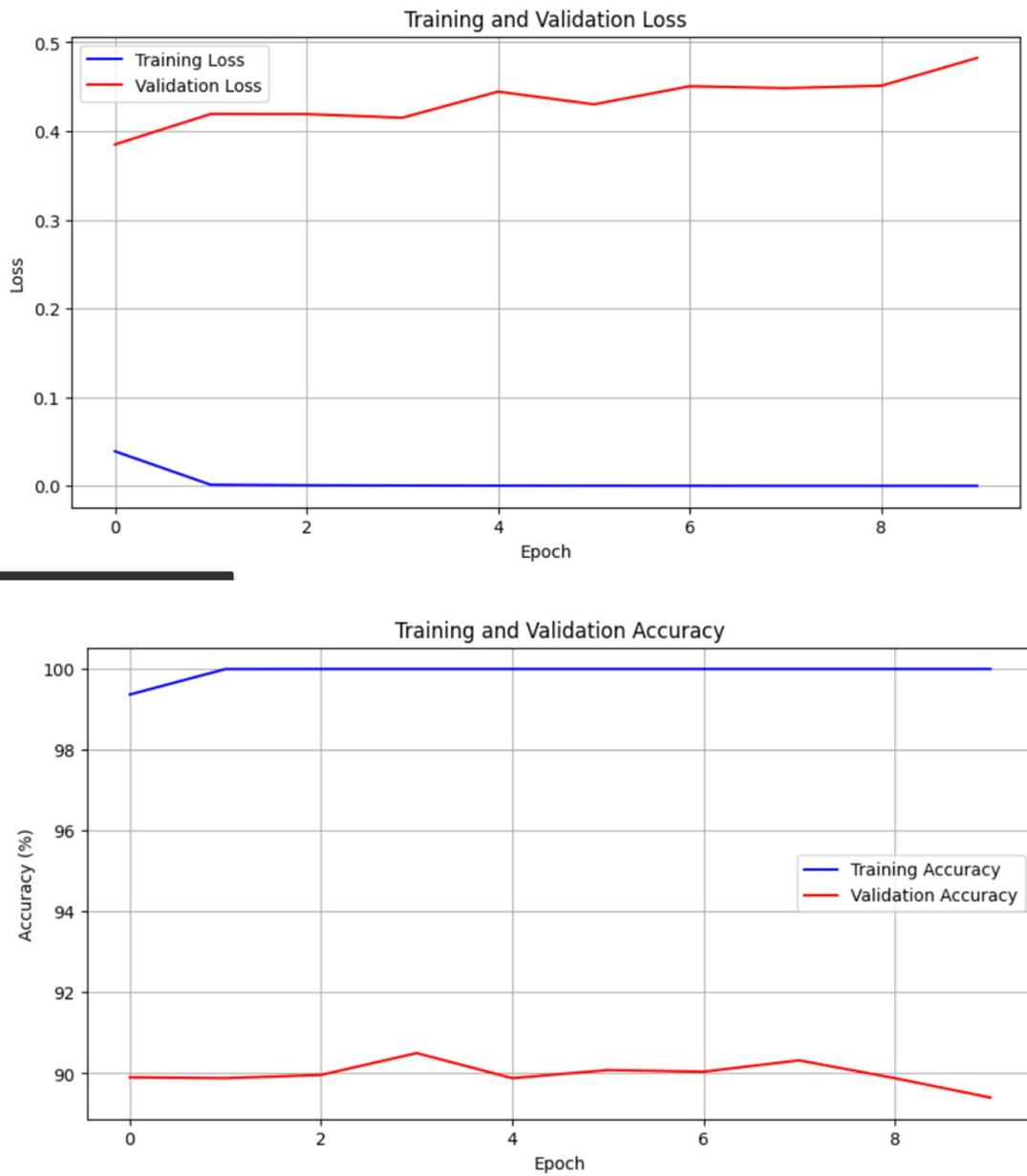
شکل ۲۷: نمودارهای هزینه و دقت لایه ششم

لایه هفت:



شکل ۲۸ : نمودارهای هزینه و دقت لایه هفتم

لایه آخر:



شکل ۲۹: نمودارهای هزینه و دقت لایه آخر

دقت بر روی دادگان تست:

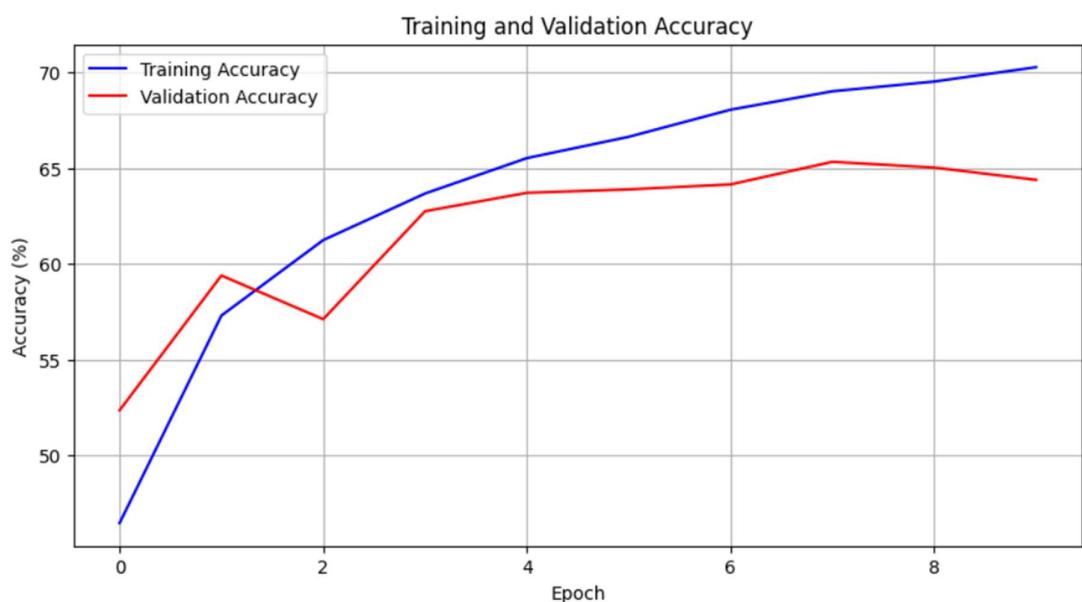
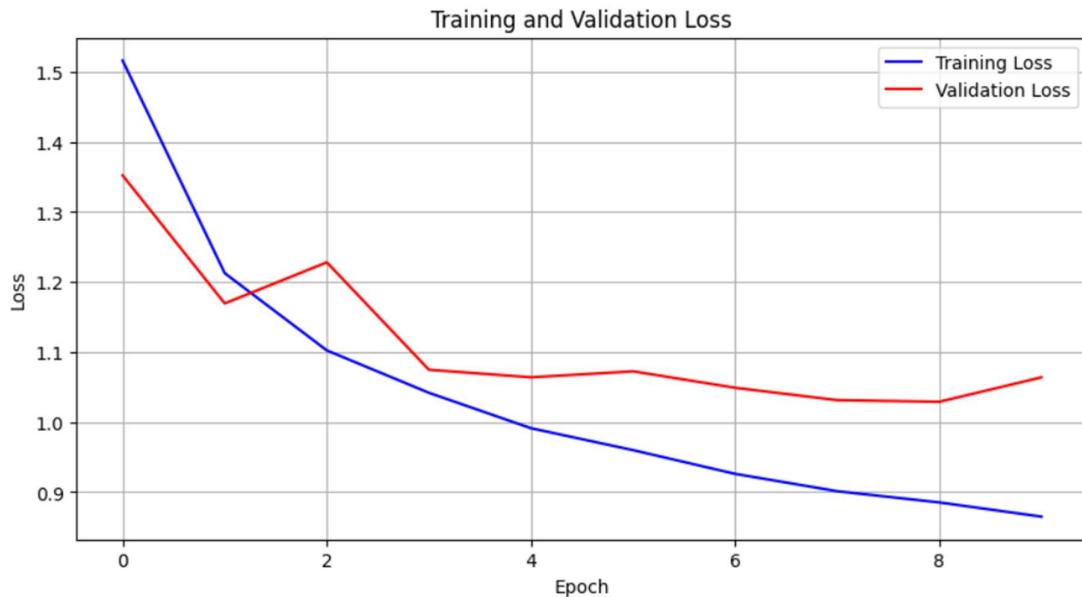
Test Loss: 0.6402  
Test Accuracy: 87.34%

و) لایه های اول با داده با افزونه کمتر

مانند قسمت ج عمل کردیم ابتدا دادگان را فقط با افزونه flip تعریف کردیم. فقط پس از لایه چهار یک بار دیگر دیتابست با افزونه کامل تعریف شد.

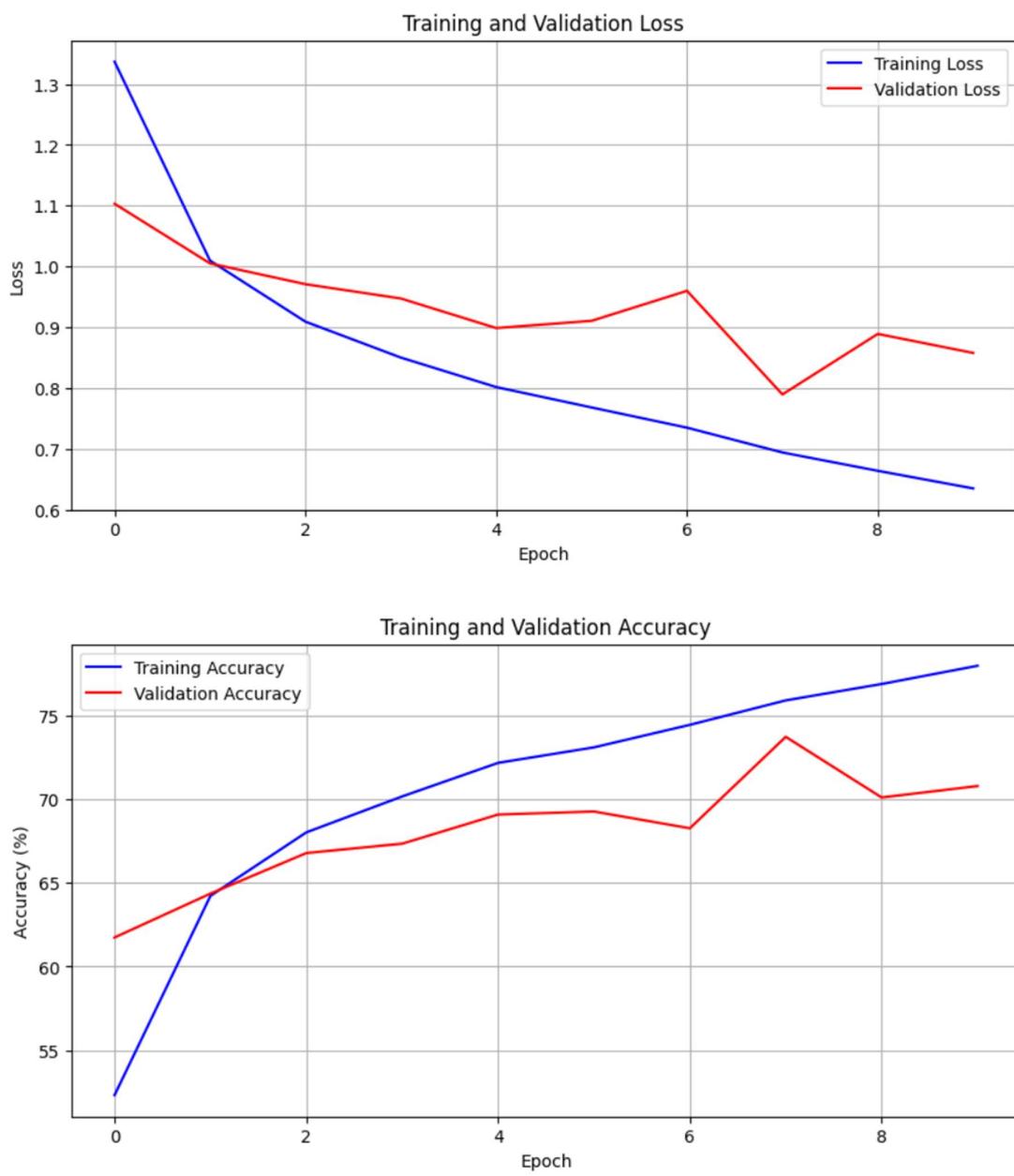
نمودار ۸ لایه به صورت زیر است.

لایه ۱:



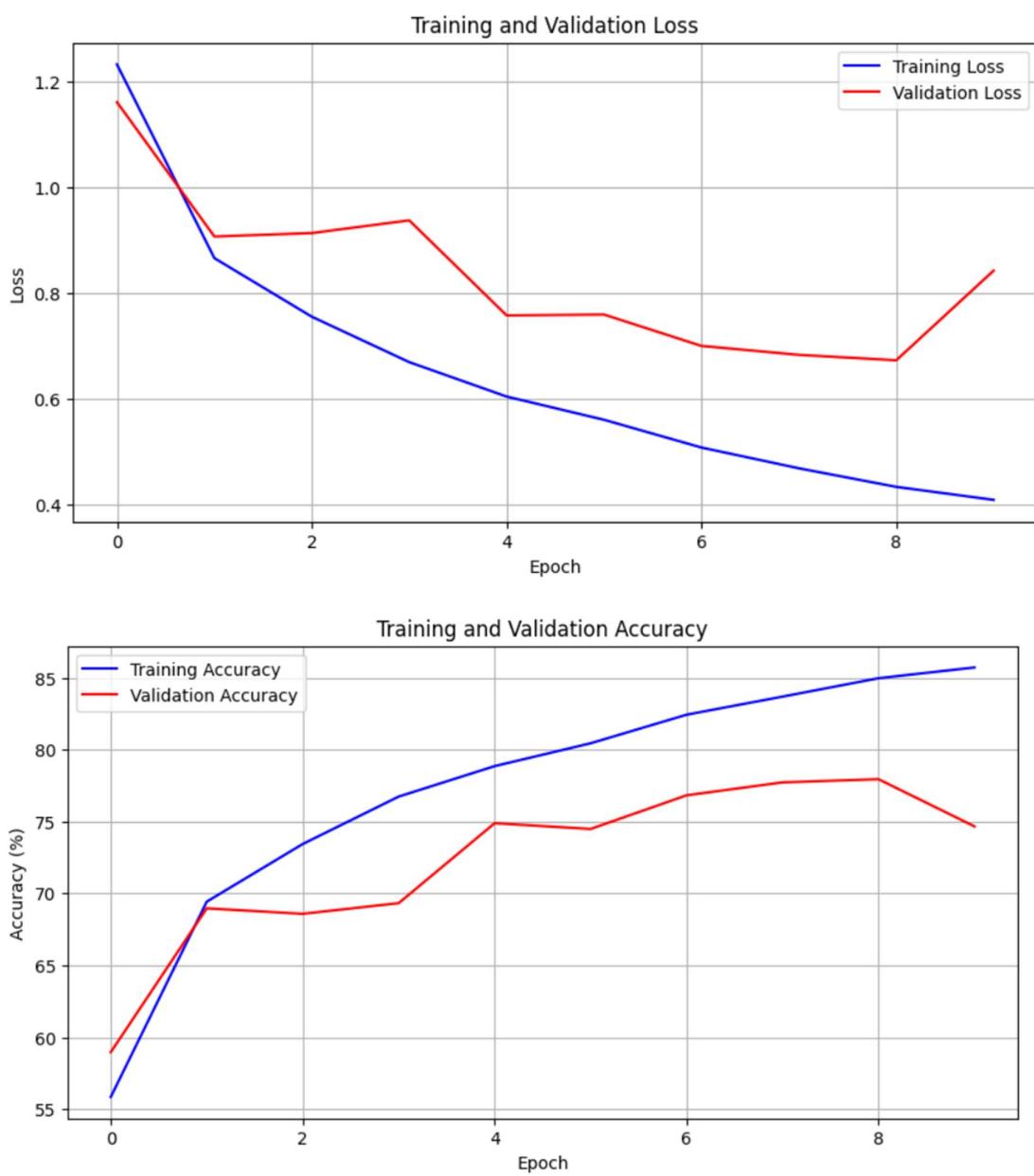
شکل ۳: نمودارهای هزینه و دقت لایه اول

لایه ۲

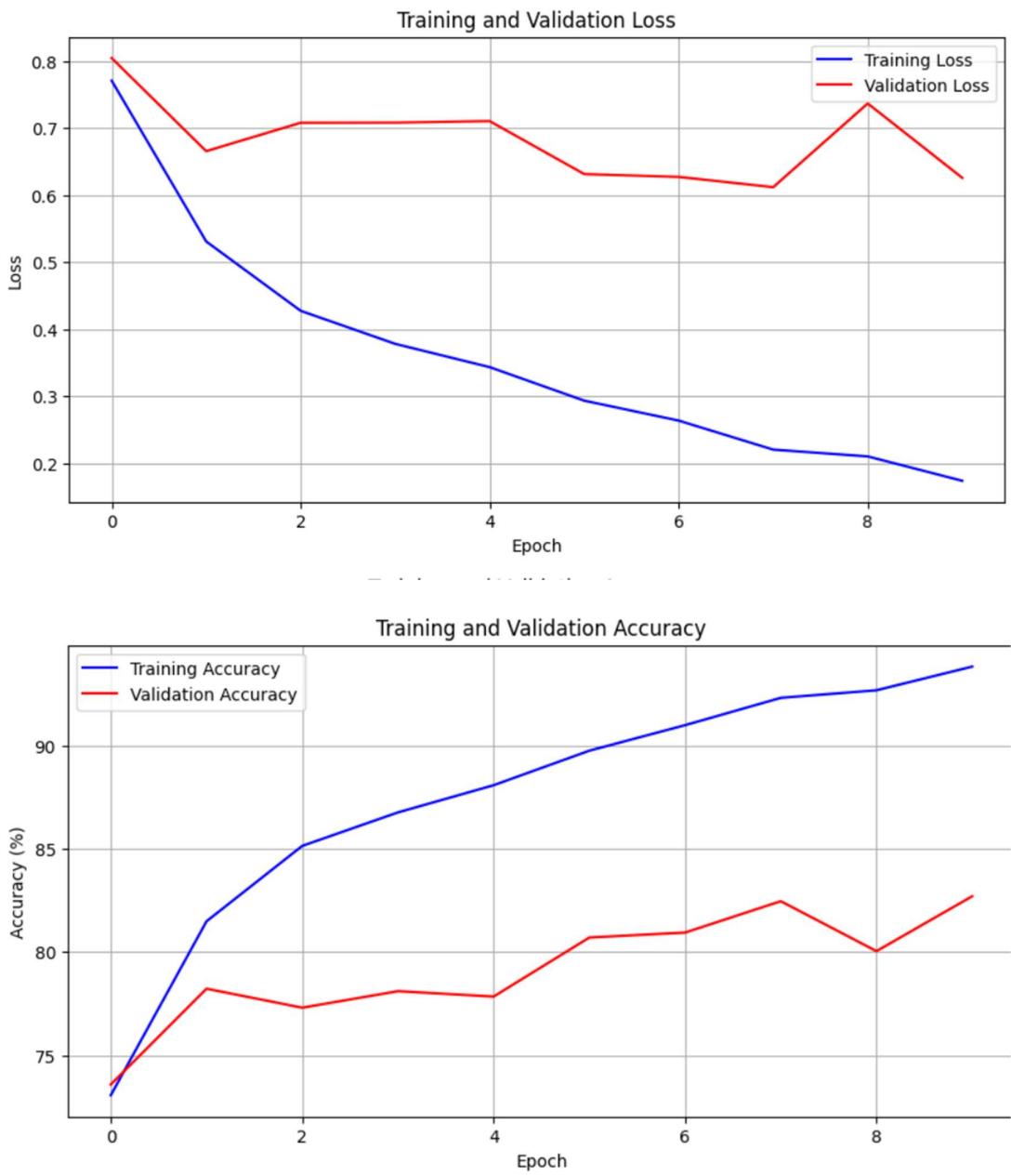


شکل ۳۱: نمودارهای هزینه و دقت لایه دوم

لایه ۳:

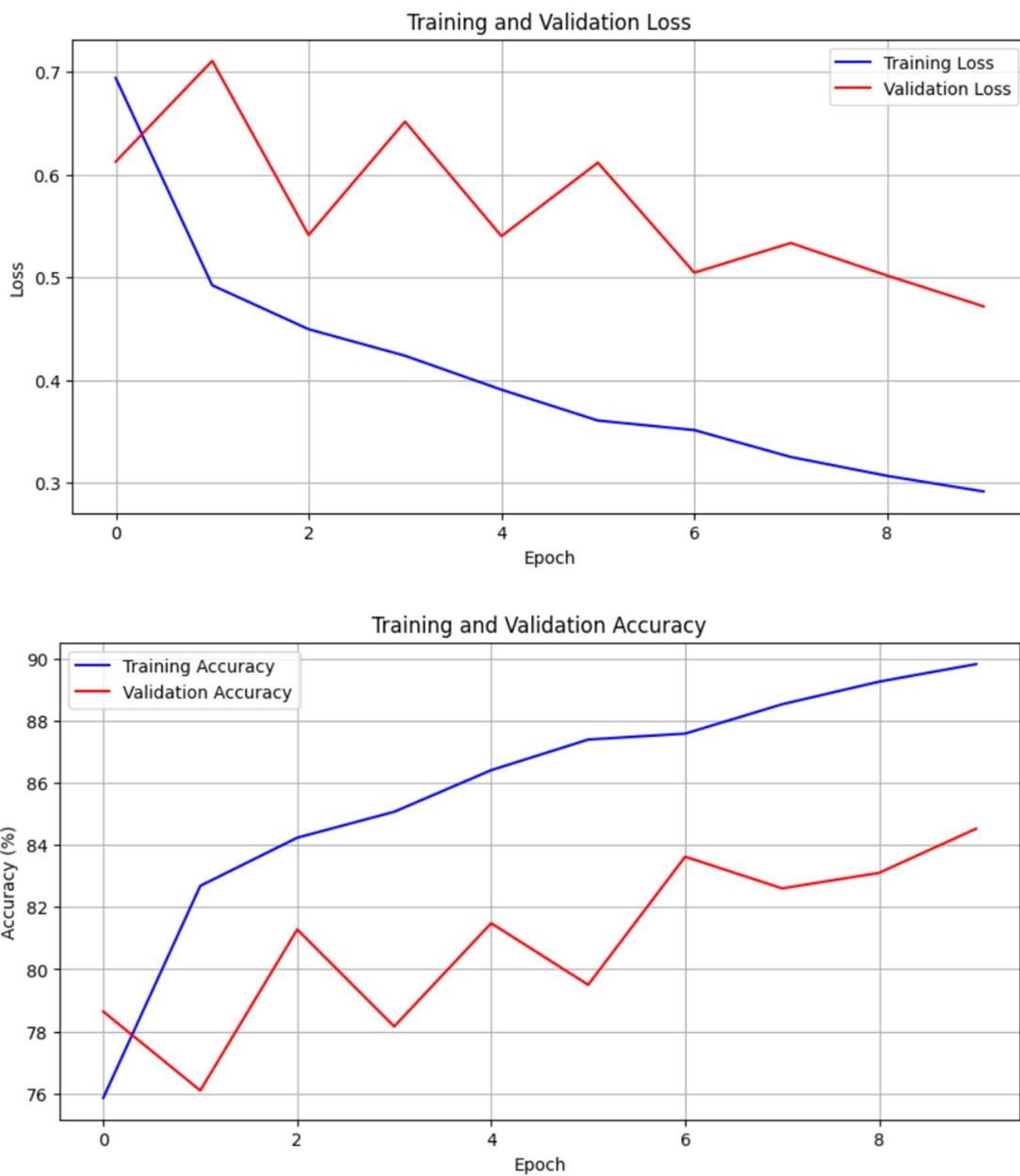


۱۴

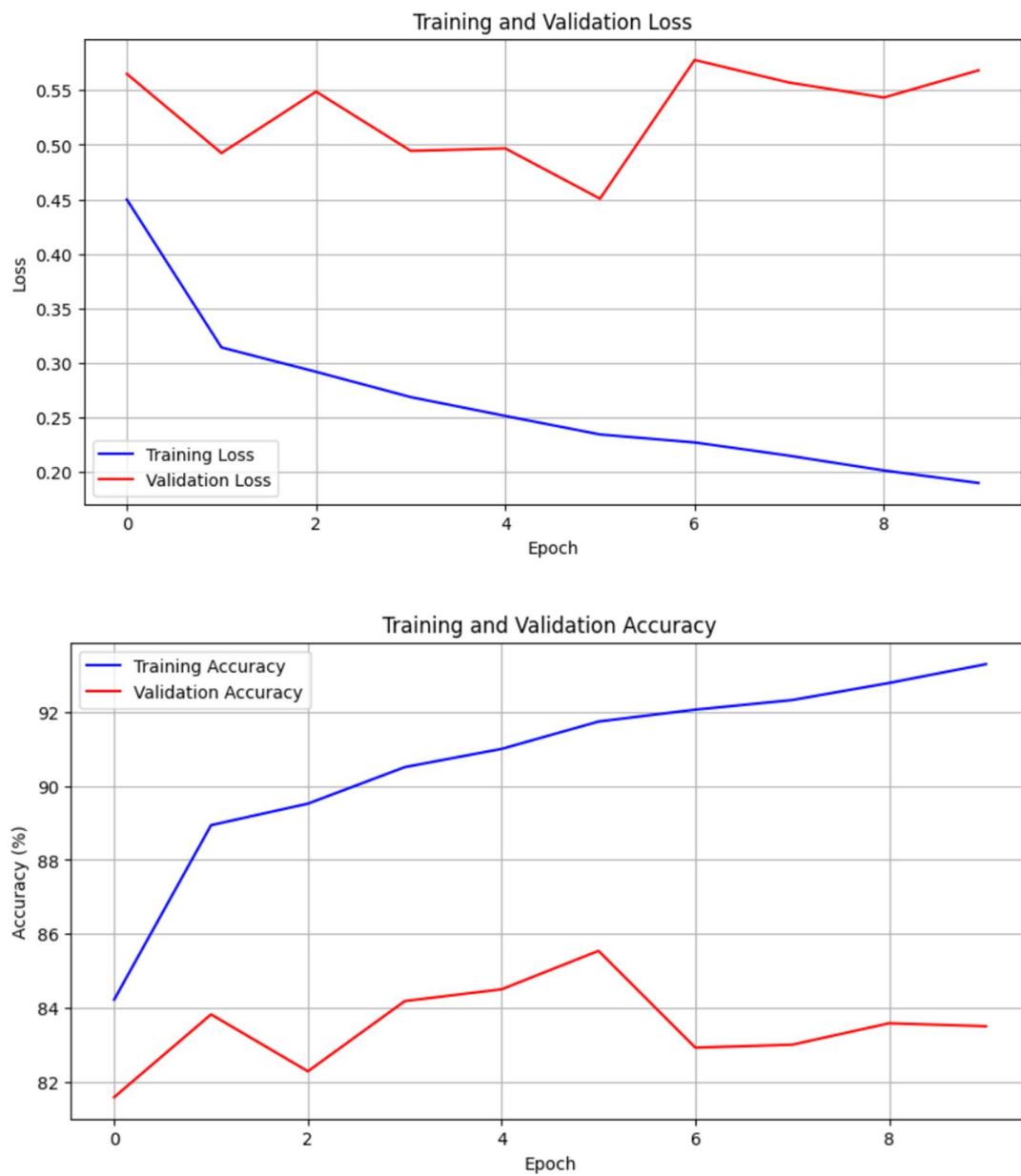


شکل ۳۲: نمودارهای هزینه و دقت لایه چهارم

: لایه ۵

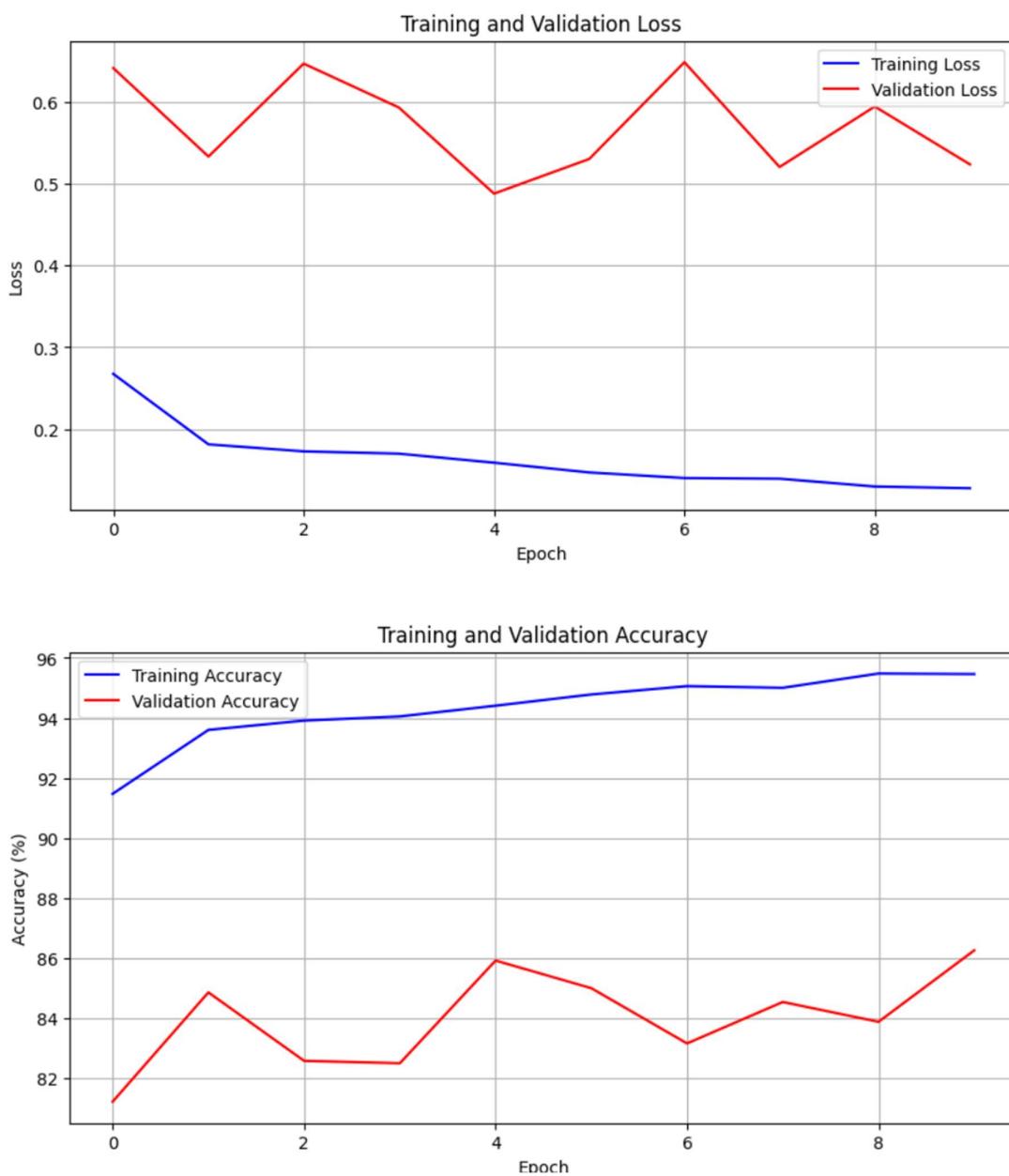


شکل ۳۳: نمودارهای هزینه و دقت لایه پنجم



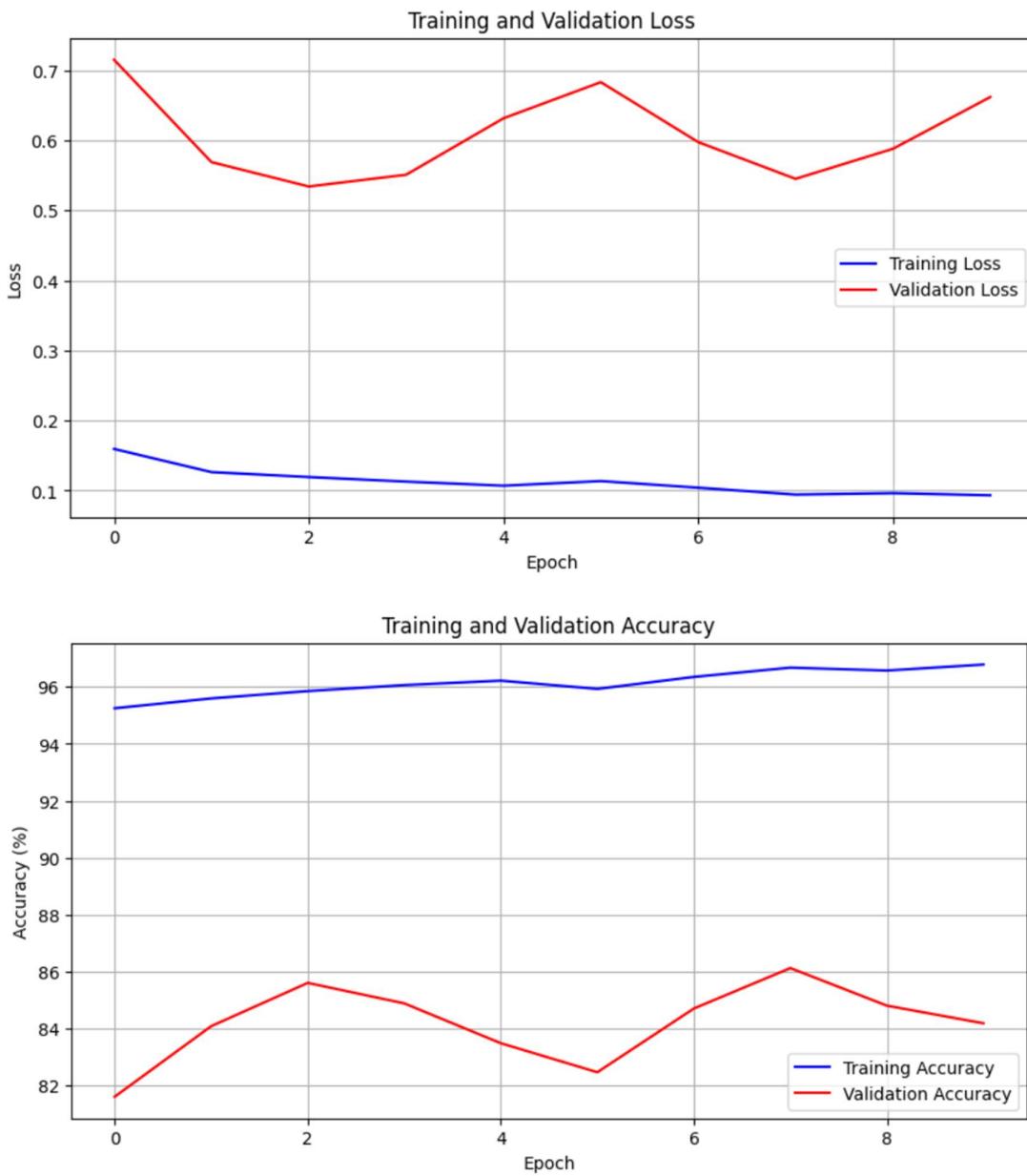
شکل ۴: نمودارهای هزینه و دقت لایه ششم

لایه ۷:



شکل ۳۵: نمودارهای هزینه و دقت لایه هفتم

لایه آخر:



شکل ۳۶: نمودارهای هزینه و دقت لایه آخر

عملکرد روی دادگان تست:

Test Loss: 0.6568

Test Accuracy: 85.56%

### جمع‌بندی و بحث

دقت روی دادگان تست در نمودار زیر آمده است:

جدول ۱: دقت بر دادگان تست در هر قسمت

روش	دقت
الف	۸۷.۷۵
ب	۷۲.۲
ج	۸۵.۴
د	۸۷.۳۴
و	۸۵.۵۶

همچنین در این تمرین از ۹۰ ایپاک در بخش end to end و مجموعاً از ۸۰ ایپاک برای بخش-layer-wise استفاده شد.

دقت در بخش ب در اکثر لایه‌ها با پیشروی بیشتر می‌شد، اما گاهی اندکی افت در دقت دادگان validation وجود داشت. اما در قسمت‌های ج، د، و با توجه به آزاد بودن وزن‌ها با افزودن لایه دقت یا افزایش میابد یا تغییر چندانی ندارد.

## سوال دوم : شبکه بخش بندی تصاویر دو بعدی

در این سوال قصد داریم دو شبکه‌ی UNet و PSPNet را باسازیم با استفاده از دیتاست SUIIM آن دو را آموزش دهیم.

### الف) معرفی دو شبکه‌ی UNET و PSPNET

#### UNET(۱)

در ابتدا به معرفی شبکه‌ی UNet می‌پردازیم.

این شبکه شامل دو بخش اصلی فشرده سازی(Contracting) و گستردۀ سازی(Expanding) است. مسیر فشرده‌سازی شامل چندین لایه کانولوشنی و پولینگ است که به تدریج اندازه تصاویر را کاهش می‌دهند. هر لایه کانولوشنی دارای فیلترهای کانولوشنی است که الگوها و ویژگی‌های تصویر را استخراج می‌کنند. (جزئیات مربوط به مشخصات این لایه‌ها در قسمت (ب) آورده می‌شود.)

مسیر گستردۀ سازی شامل لایه‌هایی است که با استفاده از عملیات upsampling و ادغام concatenation با اطلاعات مسیر فشرده‌سازی، اندازه تصاویر را افزایش می‌دهند. این بخش کمک می‌کند تا اطلاعات مکمل از مراحل قبلی و همچنین جزئیات دقیق‌تر را بازیابی کند. در این قسمت از لایه‌های Convolutional Transpose استفاده می‌شود. این لایه‌ها برای افزایش ابعاد تصویر و بازیابی جزئیات دقیق‌تر استفاده می‌شوند. (جزئیات مربوط به مشخصات این لایه‌ها در قسمت (ب) آورده می‌شود.)

یکی از ویژگی‌های مهم UNet، استفاده از ارتباطات مستقیم بین لایه‌های مختلف دو بخش فشرده سازی و گستردۀ سازی است که به عنوان Skip Connections شناخته می‌شوند. این ارتباطات اطمینان می‌دهند که اطلاعات از لایه‌های اولیه به لایه‌های بالاتر و از لایه‌های فشرده به لایه‌های گستردۀ منتقل می‌شوند که به بهبود دقت و بازیابی اطلاعات کمک می‌کند.

در نهایت پس از این دو بخش، یک لایه Convolutional ابعاد مورد نظر برای تصویر سگمنت نهایی خروجی را ایجاد می‌کند.

به طور خلاصه می‌توان گفت این شبکه در یک بخش اطلاعات را از تصاویر استخراج و ویژگی‌های را استخراج می‌کند و در بخش بعدی این اطلاعات را استفاده می‌کنند تا تصویر خروجی با اندازه‌ی مدنظر را تولید کند.

## PSPNET(۲)

این شبکه در ابتدا با استفاده از یک مدل CNN به عنوان شبکه Backbone ویژگی‌ها را از تصاویر ورودی استخراج می‌کند.

شبکه Backbone یک شبکه عمیق عمومی مانند MobileNet، VGG یا ResNet است که با استفاده از لایه‌های کانولوشنی و پولینگی، ویژگی‌های تصویر را استخراج می‌کند.

این ویژگی‌ها شامل جزئیات مختلفی هستند که درک صحنه‌های پیچیده را تسهیل می‌کنند.

این فضای ویژگی‌ها با داشتن اطلاعات معنادارتر از تصویر، به بخش‌های دیگر شبکه PSPNet (مانند ماژول‌های Pyramid Pooling) منتقل می‌شوند تا در فرایند تقسیم‌بندی دقیق‌تر و بهتر صحنه‌های تصاویر شرکت کنند. این شبکه وظیفه اصلی خود، یعنی استخراج ویژگی‌های اولیه را انجام می‌دهد و سپس این ویژگی‌ها به لایه‌های بالاتری از شبکه انتقال داده می‌شوند تا با استفاده از Skip Connections و ارتباطات مستقیم، ویژگی‌های مختلف از تصویر ترکیب و یکپارچه شوند.

اطلاعات خروجی این شبکه‌ی پایه در یک شاخه به عنوان ورودی Pyramid Pooling Module استفاده و در یک شاخه‌ی دیگر با خروجی Pyramid Pooling Module، ادغام می‌شود.

اطلاعات خروجی این شبکه‌ی پایه در یک شاخه به عنوان ورودی Pyramid Pooling Module استفاده می‌کند. هر لایه Pooling اطلاعات را از یک ناحیه با مقیاس و ابعاد خاص تصویر جمع‌آوری می‌کند. به عبارت دیگر، هر لایه Pooling برای جمع‌آوری ویژگی‌های مختلف از تصویر با اندازه‌ها و ابعاد متفاوت مسئول است.

سپس ویژگی منحصر به‌فرد از هر لایه Pooling به یک اندازه مشخص (معمولًاً با اندازه خروجی دلخواه) تغییر اندازه داده می‌شود. و در نهایت این ویژگی‌های مختلف از لایه‌های Pooling با هم ترکیب می‌شوند.

در نهایت خروجی این ماژول با ویژگی‌های خروجی شبکه‌ی پایه ادغام می‌شوند (به نوعی Skip connection است). و پس از عبور از یک لایه‌ی کانولوشنی به ابعاد مدنظر خروجی می‌رسد.

## ب) پیاده‌سازی شبکه UNET

### ۱) توضیحات اولیه‌ی پیاده‌سازی

در این قسمت قصد داریم شبکه‌ی UNet را بسازیم و با استفاده از دیتابست SUIIM آن را آموزش دهیم.

برای پیاده‌سازی این مدل از چارچوب پایتورج و از محیط گوگل کولب استفاده می‌کنیم. دیتابست را از مرجع داده شده دانلود کرده و برای استفاده آن در گوگل کولب از گوگل درایو استفاده می‌کنیم و با استفاده از کتابخانه‌ی zipfile آن را از حالت فشرده خارج می‌کنیم.

در ابتدا کتابخانه‌های لازم را برای ساخت و ساخت و آموزش مدل را فراخوانی می‌کنیم.

### ۲) آماده‌سازی دیتابست

یک کلاس به نام Dataset\_generator را تعریف می‌کنیم که برای ایجاد داده‌های آموزش و آزمایش از آن استفاده کنیم. ورودی‌های این کلاس تصاویر و ماسک‌هایشان و دو ترانسفورمر است. در این کلاس در ابتدا تصاویر و ماسک‌ها را توسط Image.open می‌خوانیم. سپس ترانسفور اول را برای تبدیل فرمت آن‌ها به تنسور و تغییر سایز آن‌ها به  $160 \times 160$  بر روی تصاویر و ماسک‌ها اعمال می‌کنیم. درون این کلاس یکتابع تعریف می‌کنیم که ماسک‌های ۳ کanalه را می‌گیرد و به عنوان خروجی ماسک‌های ۸ کanalه ایجاد می‌کند به نحوی که هر رنگ در یک کanal قرار داده می‌شود یعنی به ازای هر کلاس یک کanal ایجاد می‌شود.

در ادامه در این کلاس قصد داریم تقویت دادگان را انجام دهیم. تغییر تصاویر جهت تقویت دادگان را با احتمال ۳۰ درصد انجام می‌دهیم. برای این کار از پیکسل‌ها به راست و چپ یا بالا و پایین (با احتمال ۴۰ درصد از ۳۰ درصد)، آینه گردان در جهت افق یا عمودی (با احتمال ۴۰ درصد از ۳۰ درصد) و دوران ۱۸۰ درجه (با احتمال ۲۰ درصد از ۳۰ درصد) استفاده می‌کنیم. این عمیات را بر روی تصاویر و ماسک‌هایشان به یک میزان انجام می‌دهیم.

حال قصد داریم با استفاده از این کلاس ساخته شده، مجموعه دادگان آموزش، آزمایش و ارزیابی را ایجاد کنیم.

برای ساخت دیتاست آموزش و ارزیابی به صورت متوازن، در یک حلقه‌ی `For` از هر دسته‌ی کلی دادگان که با پیشوند مشخصی نامگذاری شده‌اند( $f, w, d, n$ ): در هر دسته تصویر شامل کلاس‌های مشخصی به طور ویژه است) داده‌ها را با استفاده از `train_test_split` به نسبت ۱۰ و ۹۰ به ارزیابی و آموزش اختصاص می‌دهیم. حال با استفاده از کلاس ایجاد شده و ترانسفورهای مناسب برای نرمال و استانداردسازی و تغییر سایز داده‌ها، دیتاست آموزش و ارزیابی را می‌سازیم و سپس با استفاده از `batch_size=16, shuffle=True` و با قرار دادن `torch.utils.data.DataLoader` برای هر دیتاست، دیتالودر ایجاد می‌کنیم.

عملیات فوق را برای ساخت دیتالودر آزمایش هم با استفاده از تصاویر و ماسک‌های آزمایش تکرار می‌کنیم.

### ۳) ساخت شبکه UNET

برای ساخت مدل `Unet` از یک کلاس استفاده می‌کنیم. در این کلاس در ابتدا لایه‌های مورد نیاز را می‌سازیم و سپس در قالب تابع `forward` از آن‌ها استفاده می‌کنیم.

ساختار نهایی این مدل به این صورت است که ورودی در ابتدا به بخش اول فشرده سازی(انکودر) وارد می‌شود. این بخش شامل دو لایه `Conv2d` است که سایز ورودی و خروجی `Conv2d` اول ۳ و ۶۴ و `Conv2d` دوم ۶۴ و ۶۴ است. (سایز کرنل و `padding` همه‌ی لایه‌های `Conv2d` استفاده شده در این مدل به جز آخرین `Conv2d`، به ترتیب ۳\*۳ و ۱ است و بعد از همه‌ی آن‌ها به جز آخرین، یک لایه‌ی `ReLU` می‌آید). و سپس در ادامه‌ی آن یک لایه `maxpool` با گام ۲ و سایز کرنل ۲\*۲ می‌آید. خروجی `Conv2d` این بخش وارد لایه‌هایی مشابه بخش اول انکودر می‌شود با این تفاوت که سایز ورودی و خروجی `Conv2d` اول ۶۴ و ۱۲۸ و `Conv2d` دوم ۱۲۸ و ۲۵۶ است و سپس یک لایه `maxpool` با گام ۲ و سایز کرنل ۲\*۲ می‌آید.

در ادامه نیز دو لایه‌ی کانولوشن دیگر(با مشخصات و نکات ذکر شده) با سایز ورودی و خروجی ۱۲۸ و ۲۵۶ برای `Conv2d` اول و سایز ۲۵۶ و ۲۵۶ برای `Conv2d` و سپس لایه `maxpool` می‌آید.

پس از این چهارمین بلوک انکودر نیز همانند بلوک‌های قبلی ولی با سایز ورودی و خروجی ۲۵۶ و ۵۱۲ برای `Conv2d` اول و سایز ۵۱۲ و ۵۱۲ برای `Conv2d` می‌آید.

پس از این ۴ بلوک مشابه ولی با سایز ورودی و خروجی متفاوت، یک بلوک دیگر نیز می‌آید که سایز ورودی و خروجی Conv2d اول ۵۱۲ و ۱۰۲۴ و Conv2d دوم ۱۰۲۴ و ۱۰۲۴ است. ولی پس از آن دیگر لایه‌ی maxpool نمی‌آید.

تا این مرحله انکود کردن انجام می‌شود و ویژگی‌های خروجی به دست می‌آید.

این خروجی را در ابتدا وارد یک لایه‌ی ConvTranspose2d با سایز ورودی ۵۱۲ و سایز خروجی ۲۵۶ و kernel\_size=2, stride=2 می‌کنیم. خروجی این لایه و خروجی چهارمین بلوک انکودر قبل از maxpool را باهم ادغام می‌کنیم. حال خروجی حاصل از این ادغام را در یک بلوک شامل دو لایه‌ی کانولوشن که سایز ورودی و خروجی Conv2d اول ۵۱۲ و ۱۰۲۴ و Conv2d دوم ۱۰۲۴ و ۵۱۲ است و دو لایه‌ی Relu وارد می‌کنیم.

خروجی آن را وارد لایه‌ی ConvTranspose2d دیگری با سایز ورودی ۱۲۸ و سایز خروجی ۶۴ می‌کنیم. (با اندازه گام و کرنل لایه‌ی ConvTranspose2d قبلی) خروجی این لایه مشابه قبل با خروجی سومین بلوک انکودر قبل از maxpool باهم ادغام می‌شوند و خروجی آن در یک بلوک شامل دو لایه‌ی کانولوشن که سایز ورودی و خروجی Conv2d اول ۲۵۶ و ۵۱۲ و Conv2d دوم ۲۵۶ و ۱۲۸ است و دو لایه‌ی Relu وارد می‌شود.

این روند برای سومین لایه‌ی ConvTranspose2d با سایز ورودی ۲۵۶ و سایز خروجی ۱۲۸ و لایه‌های کانولوشن با سایزهای ورودی و خروجی به ترتیب ۲۵۶ و ۱۲۸ و ۱۲۸ و ۶۴ نیز تکرار می‌شود. (نتیجه‌ی لایه‌ی ConvTranspose2d با دومین لایه‌ی انکودر ادغام می‌شود).

در نهایت چهارمین بخش دیکودر نیز در ابتدا یک لایه‌ی ConvTranspose2d با سایز ورودی ۱۲۸ و سایز خروجی ۶۴ و لایه‌های کانولوشن با سایزهای ورودی و خروجی به ترتیب ۱۲۸ و ۶۴ و ۶۴ و ۶۴ نیز تکرار می‌شود. (نتیجه‌ی لایه‌ی ConvTranspose2d با اولین لایه‌ی انکودر ادغام می‌شود).

در این مرحله نیز دیکودر مدل Unet ساخته می‌شود. حال در مرحله و گام آخر این خروجی را در یک لایه‌ی کانولوشن با ورودی ۶۴ (که خروجی آخرین لایه‌ی قبل از آن است می‌سازیم) و اندازه‌ی خروجی آن را برابر تعداد کلاس‌های مسئله‌ی semantic segmentation مد نظر (که در دیتابست مد نظر ۸ است) وارد می‌کنیم. سایز کرنل این لایه‌ی کانولوشن ۱ است.

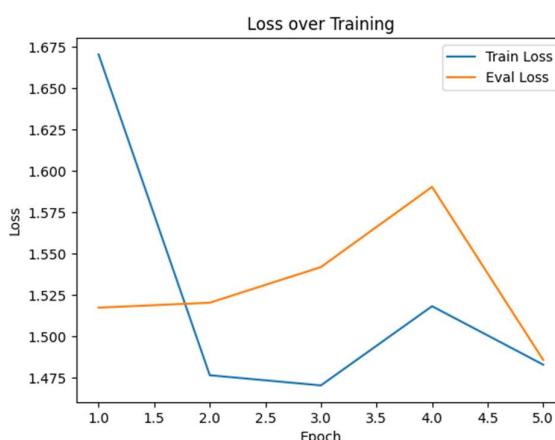
مدل ما شامل دو بخش انکودر و دیکودر ساخته شد.

## ب) آموزش مدل UNET

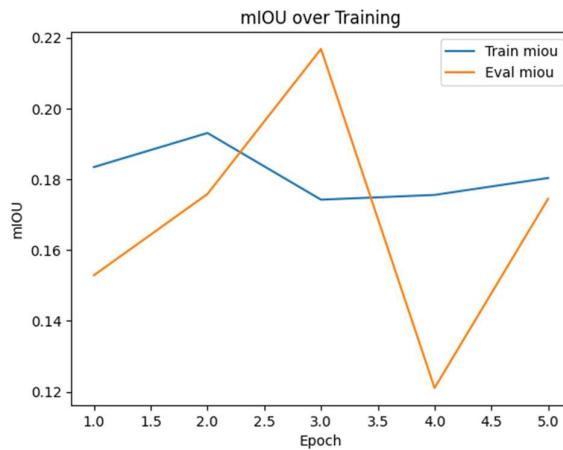
برای آموزش این مدل از تابع هزینه‌ی CrossEntropyLoss و تابع بهینه‌ساز SGD با قرار دادن momentum = 0.99 و decay = 0.01 و learning\_rate = 0.01 استفاده می‌کنیم.

قصد داریم مدل را به نحوی آموزش دهیم که در هر گام خطای mIoU را محاسبه و ذخیره کند که پس از آموزش بتوانیم آن‌ها را مشاهده کنیم. پس برای این کار یک تابع برای محاسبه‌ی mIoU می‌نویسیم که برای هر کanal ماسک IoU را محاسبه می‌کند و میانگین آن‌ها را به عنوان خروجی می‌دهد. (IoU حاصل تقسیم اشتراک دو ماسک بر اجتماع آن‌هاست).

حال یک حلقه for می‌نویسیم که در آن به ازای هر گام روند آموزش را تکرار کند. درون این حلقه مدل در حالت آموزش قرار می‌گیرد و با خواندن تصاویر و ماسک‌هایشان از دیتالودر آموزش، خروجی مدل را برای تصویر محاسبه می‌کند و با مقایسه آن با ماسک واقعی وزن‌های مدل را به روز می‌کند و سپس مقدار mIoU بین ماسک واقعی و ماسک پیش‌بینی را محاسبه و ذخیره می‌کند. درون این حلقه در گام بعد مدل را در حالت ارزیابی قرار می‌دهیم و خروجی مدل را برای دادگان ارزیابی محاسبه می‌کنیم و نتایج آن را ذخیره می‌کنیم. این مدل را در ۵ گام آموزش داده‌ایم. (این تعداد گام برای آموزش این مدل بسیار کم است و منجر به نتایج ضعیف می‌شود ولی با توجه به محدودیت‌های سیستم قادر به افزایش تعداد گام‌ها نبودیم). در تصاویر زیر نتایج آموزش مدل آمده‌اند:



شکل ۳۷- خطای آموزش و ارزیابی در حین روند آموزش مدل UNET



شکل ۳۸- مقدار mIoU آموزش و ارزیابی در حین روند آموزش و ارزیابی مدل UNET

توجه: با توجه به اینکه مدل باید ۸ کلاس را سگمنت و طبقه بندی کند و تصاویر دارای جزئیات هستند، مدل نیاز به آموزش دیدن در تعداد گام‌های بیشتری دارد. از تصاویر فوق مشخص است که هنوز روند آموزش مدل ثابت نشده است و دچار نوسان است که این در گام‌های بالاتر حل می‌شود و روند تغییرات خط‌نازولی و روند تغییرات mIoU صعودی می‌گردد و همگرا می‌شود. که ما متاسفانه به علت ضعف سیستم مورد استفاده قادر به آموزش در گام‌های بیشتری نشدیم اما با تست‌های گوناگون بهترین نرخ یادگیری انتخاب شد که باعث شود در تعداد گام‌های بالاتر مدل به دقت مناسبی برسد.

### ج) ارزیابی مدل UNET

در این مرحله قصد داریم مدل را ارزیابی کنیم (که با توجه به نتایج بخش قبل انتظار نداریم مدل به دقت مناسبی با این تعداد گام آموزش برسد).

برای این کار مدل را در حالت ارزیابی قرار می‌دهیم و با ورود تصاویر آزمایش به مدل ماسک‌های خروجی را دریافت و آن‌ها را با ماسک‌های واقعی مقایسه و mIoU را محاسبه می‌کنیم که مقدار آن ۰,۱۲۶۰ به دست می‌آید.(تصاویر، ماسک‌های واقعی و ماسک‌های پیش‌بینی شده توسط شبکه را در لیست‌های ذخیره می‌کنیم تا در مرحله‌ی بعد از آن استفاده کنیم.)

برای نمایش ۱۰ تصویر از تصاویر آزمایش، ۱۰ عدد رندوم تولید می‌کنیم.

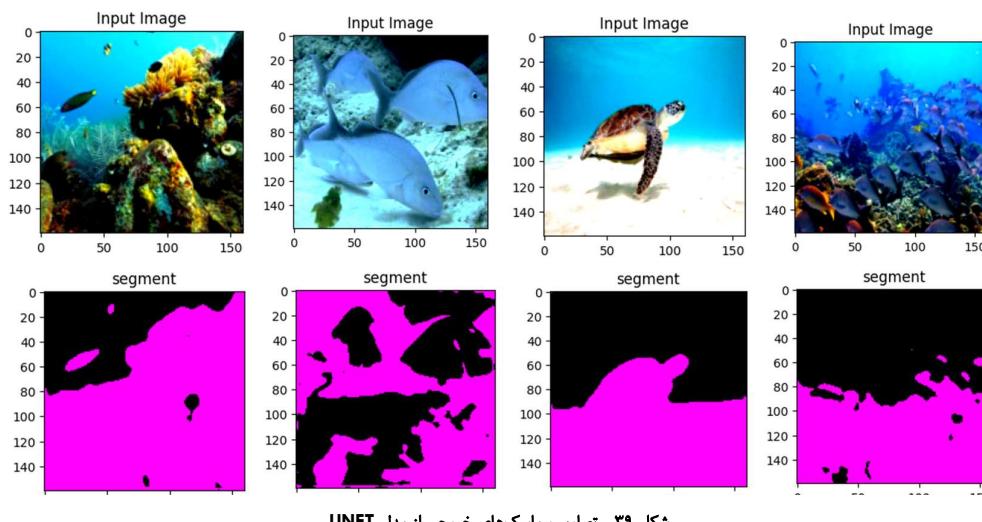
در یک حلقه‌ی **for** به ازای این ده عدد، تصویر متناظر از تصاویر آزمایش و ماسک شبکه را می‌خوانیم و آن‌ها را نمایش می‌دهیم.

در ابتدا مقادیر پیکسل‌های ماسک را به صفر یا یک تبدیل می‌کنیم.

برای نمایش ماسک، نیاز است آن را از ۸ کانال به ۳ کانال تبدیل کنیم. برای این کار در گام اول یک آرایه با ابعاد ماسک اصلی و ۳ کاناله با درایه‌های صفر می‌سازیم. سپس مقدار هر پیکسل در هر کانال را می‌خوانیم. اگر مقدارش ۱ بود، رنگ **RGB** متناظر آن کانال را در پیکسل متناظر در آرایه ساخته شده وارد می‌کنیم. حال با این کار ماسک را از ۸ کاناله به ۳ کاناله تبدیل کرده‌ایم و آن را نمایش می‌دهیم.

نتایج این مرحله در تصاویر زیر آمده است. (۶ تصویر دیگر مدل سگمنت نامناسبی داشته که در فایل گزارش آورده نشده است ولی در فایل اجرا شده کد موجود است.)

(مجدد ذکر می‌شود با توجه به این که مدل می‌بایست ۸ کانال و ۸ کلاس را برای هر پیکسل آموزش میدیده، تعداد ۵ گام برای آموزش مدل بسیار پایین است که این منجر به نتیجه ضعیف شده که در تصاویر خروجی مش هود است.)



شکل ۳۹ - تصاویر و ماسک‌های خروجی از مدل **UNET**

ج) پیاده‌سازی شبکه‌ی **PSPNET**

بخش‌های توضیحات اولیه‌ی پیاده‌سازی و آماده‌سازی دیتاست برای این قسمت مشابه بخش (ب) این سوال است و از توضیح مجدد آنچه ذکر شده خودداری می‌شود.

برای ساخت این مدل نیاز به ساخت سه بخش داریم. بخش اول استخراج ویژگی‌ها از شبکه **Backbone**. بخش دوم **pyramid pooling modul** و بخش سوم عبور خروجی نهایی از یک لایه‌ی کانولوشن  $1 \times 1$ .

پس در ابتدا یک کلاس برای تعریف **pyramid pooling module** می‌سازیم. در این کلاس، ورودی آن (keh ویژگی‌های استخراج شده از شبکه **backbone** است). به طور موازی ۴ بار وارد لایه‌ی **AdaptiveAvgPool2d** با سایزهای ۱۶×۱۶ و ۳۲×۳۲ می‌شود و خروجی هر کدام از یک لایه‌ی کانولوشن  $1 \times 1$  عبور می‌کند. سپس خروجی حاصل از این کانولوشن‌ها **upsample** می‌شوند تا خروجی همه‌ی پولینگ‌ها با اندازه‌های گوناگون به اندازه‌ی واحد و به اندازه‌ی ویژگی‌های ورودی دوباره تبدیل شوند.

در نهایت در آخر این کلاس، خروجی این پولینگ‌های **upsample** شده و ویژگی‌های شبکه **backbone** باهم ادغام می‌شوند.

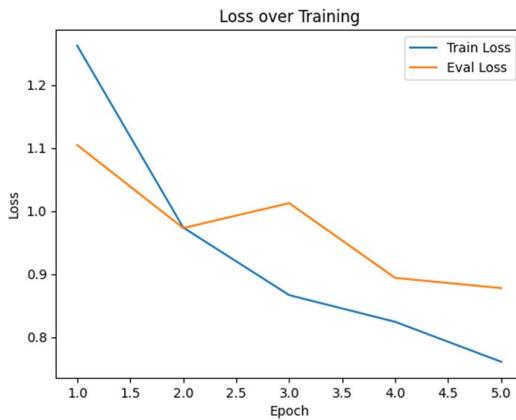
تا اینجا **PPModule** در کلاس **pyramid pooling modul** ساخته شد.

حال کلاس مدل اصلی یعنی **PSPNet** را می‌سازیم. در این مدل قصد داریم از شبکه‌ی **mobilenet\_v2** به عنوان **backbone** مدل برای استخراج ویژگی‌ها استفاده کنیم.

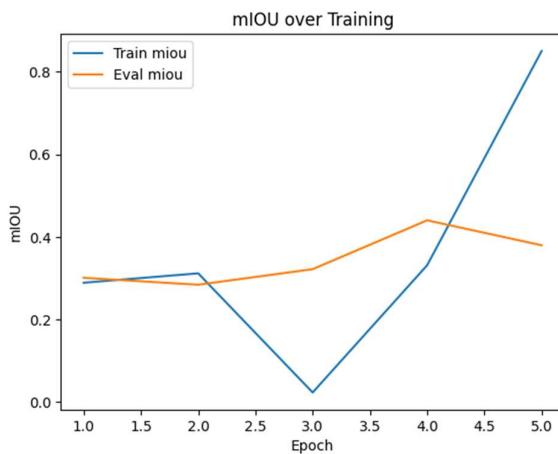
در کلاس **PSPNet** در ابتدا ویژگی‌های مدل **mobilenet\_v2** به ازای تصاویر ورودی را استخراج می‌کنیم. این ویژگی‌ها را به بلوک **PPModule** می‌دهیم پولینگ با ۴ سایز را به تصویر اعمال می‌کند و با ویژگی‌های استخراج شده از مدل **backbone** ادغام می‌کند(جزئیات در توضیحات کلاس **PPModule** ذکر شده) حال خروجی این لایه‌ی کانولوشن  $1 \times 1$  با سایز خروجی ۸ (تعداد کلاس‌ها) عبور می‌دهیم و در نهایت آن را **upsample** می‌کنیم تا به سایز ماسک‌های مد نظر تبدیل شوند.

مدل **PSPNet** ساخته شد.

رونده‌آموزش و ارزیابی همانند بخش (ب) همین سوال انجام می‌شود با این تفاوت که پارامترهای تابع بهینه‌ساز **SGD** به صورت  $lr=0.01$ ,  $momentum = 0.9$  تعریف می‌شوند. نتایج حاصل از آموزش و ارزیابی در ادامه آمده است.



شکل ۴۰- خطای آموزش و ارزیابی در حین روند آموزش مدل PSPNET

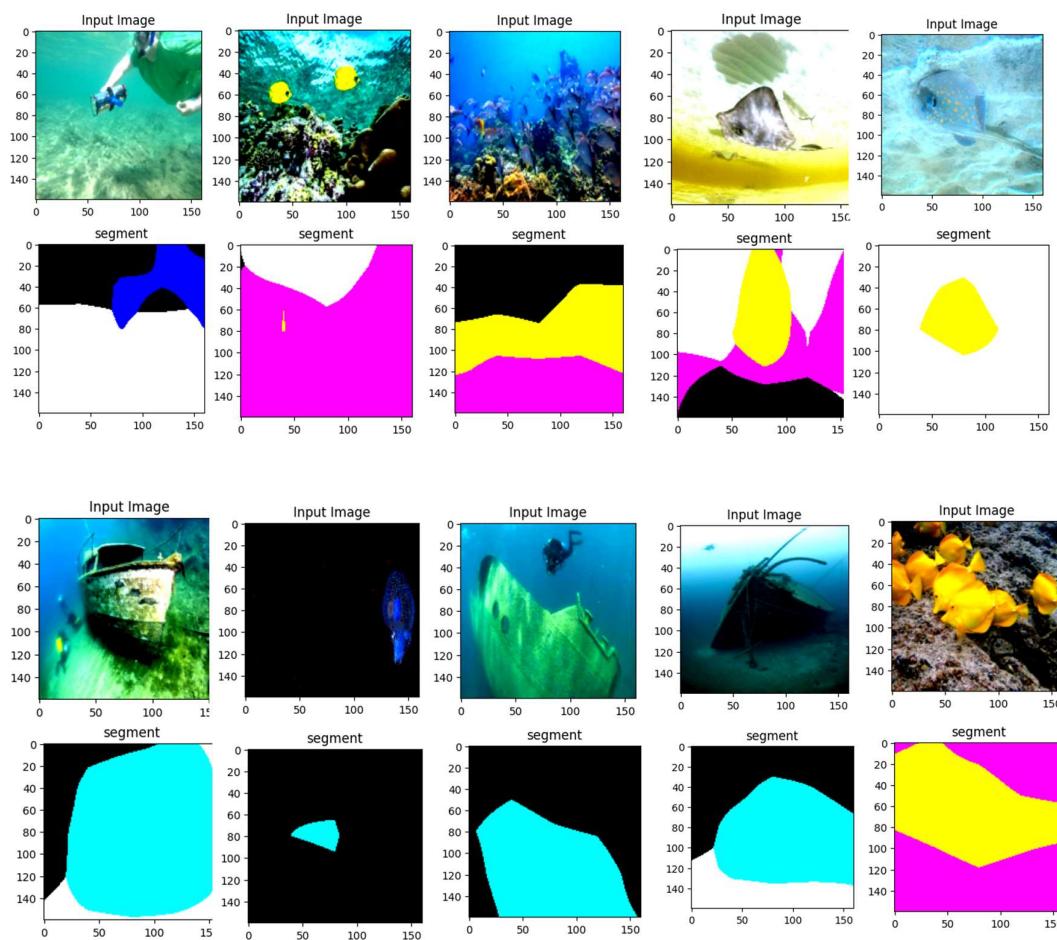


شکل ۴۱- مقدار mIoU آموزش و ارزیابی در حین روند آموزش و ارزیابی مدل PSPNet

۰,۳۳۴۴: PSPNet میانگین ماسکهای خروجی واقعی و پیش‌بینی شده توسط شبکه Miou بین

همانگونه که از نمودارهای فوق مشخص است، نمودارهای خطاهنوز در حال کاهش هستند و همگرایش نشده‌اند و نمودارهای mIoU نیز همچنان دارای نوسان کم و در حال افزایش هستند و هنوز همگرا نشده‌اند. پس این مدل نیاز به آ« وزش در گام‌های بالاتری دارد که بتواند عملکرد مناسبی از خود نشان دهد، که متاسفانه به دلیل ضعف سیستم قادر به چنین کاری نیستیم.

اما با مقایسه همین نتایج با نتایج مدل UNet نیز کاملا مشهود است که مدل PSPNet عملکرد بهتری داشته است و در همین گام‌های ابتدایی زودتر از نوسان خارج شده و به مقادیر بهتری رسیده و mIoU برای دادگان تست نیز بهبود قابل توجهی داشته است.



شکل ۴۲- خروجی شبکه PSPNet برای ۱۰ تصویر رندوم از بین تصاویر آزمایش

از تصاویر بالا و با مقایسه با نتایج مدل UNet کاملا مشهود است که در همین تعداد کم گام‌های آموزش و با توجه به ۸ کanalه بودن ماسک‌ها و جزئیات تصاویر، مدل PSPNet بسیار بهتر از مدل UNet در تشخیص کلاس هر پیسکل عمل کرده است و اگر تعداد گام‌ها را افزایش می‌دادیم به دقت بسیار خوبی می‌رسیدیم.