پروژه۴

بنیامین اسدی

۴۰۰۱۲۶۲۴۴۴

شاهین فائزی

۴۰۰۱۲۶۲۴۰۰

۱-فاز ۱:

CNNModel2(

(conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))

(maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

(conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))

(maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

(conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))

(maxpool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

(conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))

(maxpool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

(global_avg_pool): AdaptiveAvgPool2d(output_size=1)

(fc1): Linear(in_features=256, out_features=512, bias=True)

(fc2): Linear(in_features=512, out_features=10, bias=True)

)

Total Parameters: 525130

ساختار مدل

۱. **ورودی:**
تصاویر ورودی دارای ۳ کانال) مثل تصاویر رنگی (RGB هستند.

۲. **لایه‌های کانولوشن:(Convolutional Layers)**

- **لایه ۱** 32 **:**فیلتر با اندازه .3×33 \times 3×33
- **لایه ۲** 64 **:**فیلتر با اندازه .3×33 \times 3×33
- **لایه ۳** 128 **:**فیلتر با اندازه .3×33 \times 3×33
- **لایه ٤** 256 **:**فیلتر با اندازه .3×33 \times 3×33

۳. **لایه‌های ماکس‌پولینگ:(MaxPooling)**
بعد از هر لایه کانولوشنی، یک لایه ماکس‌پولینگ با اندازه 2×22 \times 22×2اضافه شده است تا اندازه ویژگی‌ها کوچک‌تر شود و محاسبات کاهش یابد.

۴. **پولینگ متوسط جهانی:(Global Average Pooling)**

در انتهای لایه‌های کانولوشن، از یک لایه "پولینگ متوسط جهانی" استفاده شده است که تمام ویژگی‌ها را به یک مقدار میانگین کاهش می‌دهد. این باعث کاهش تعداد ویژگی‌ها و ساده‌تر شدن مدل می‌شود.

۵. **لایه‌های کاملاً متصل:(Fully Connected)**

- **fc1:** یک لایه با ۵۱۲ نورون که ویژگی‌های خروجی را به صورت خطی ترکیب می‌کند.
- **fc2:** یک لایه نهایی با ۱۰ نورون (به تعداد کلاس‌ها) برای پیش‌بینی دسته نهایی.

۶. **توابع فعال‌سازی:(Activation Functions)**

در تمامی لایه‌ها از تابع ReLU استفاده شده است که به مدل کمک می‌کند غیرخطی باشد و بهتر یاد بگیرد.

۷. **خروجی:**

خروجی مدل شامل احتمال تعلق تصویر به هر یک از ۱۰ کلاس است.

بهینه‌سازی و معیار خطا

- **بهینه‌ساز :(Optimizer)** از Adam استفاده شده که یک الگوریتم پیشرفته برای تنظیم وزن‌ها است.
- **تابع خطا :(Loss Function)** از CrossEntropyLoss استفاده شده است که مخصوص مسائل دسته‌بندی است.

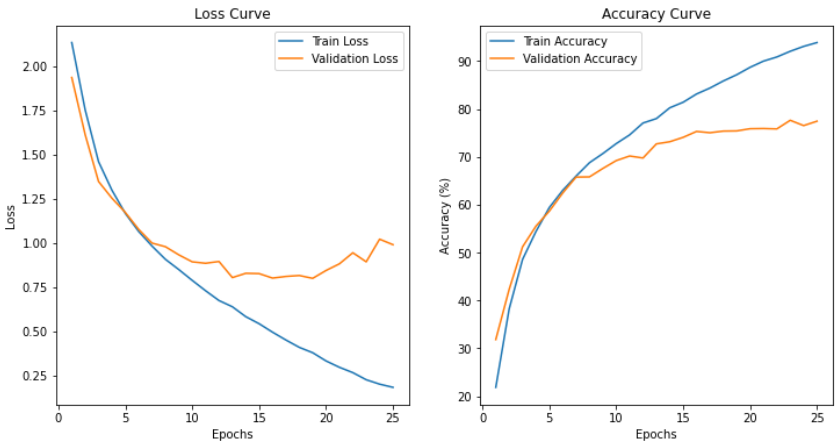
عملکرد مدل:

این مدل را با پارامترهای epochs = 25 batch_size = 32 learning_rate = 0.001 آموزش می‌دهیم.

نتیجه آخرین epoch:

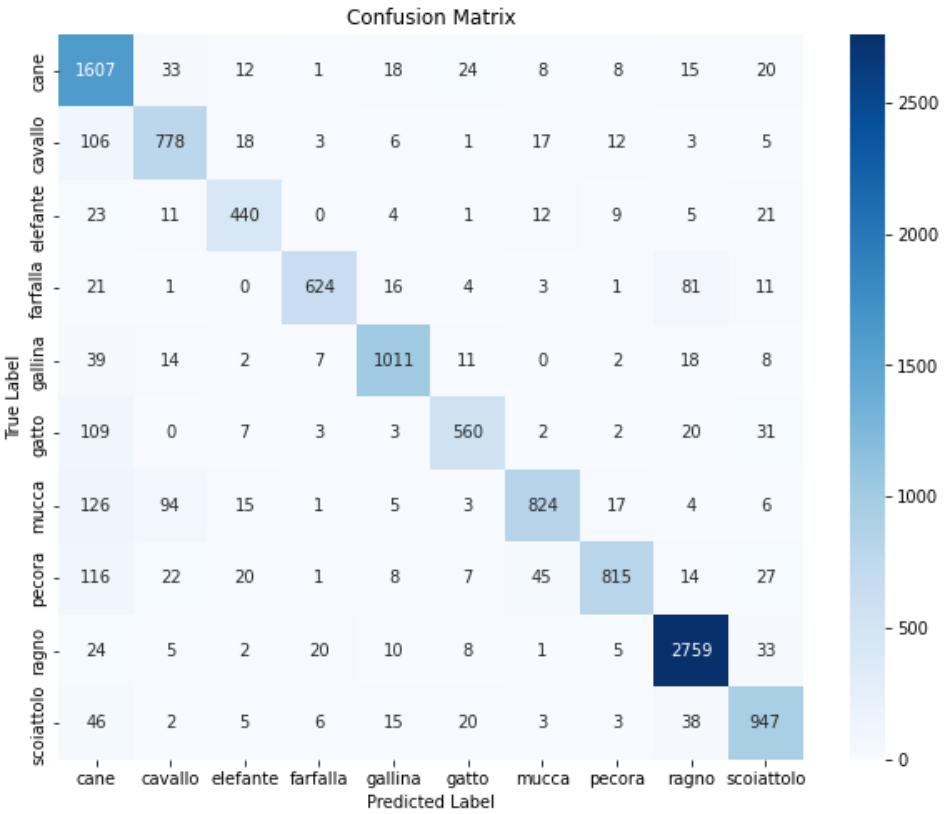
Epoch [25/25], Loss: 0.1819, Accuracy: 93.85%

Validation Loss: 0.9898, Validation Accuracy: 77.43%



با توجه به نمودار مقابل

مدل کاملا overfit شده است.



نتایج در داده تست:

Precision: 0.8746,

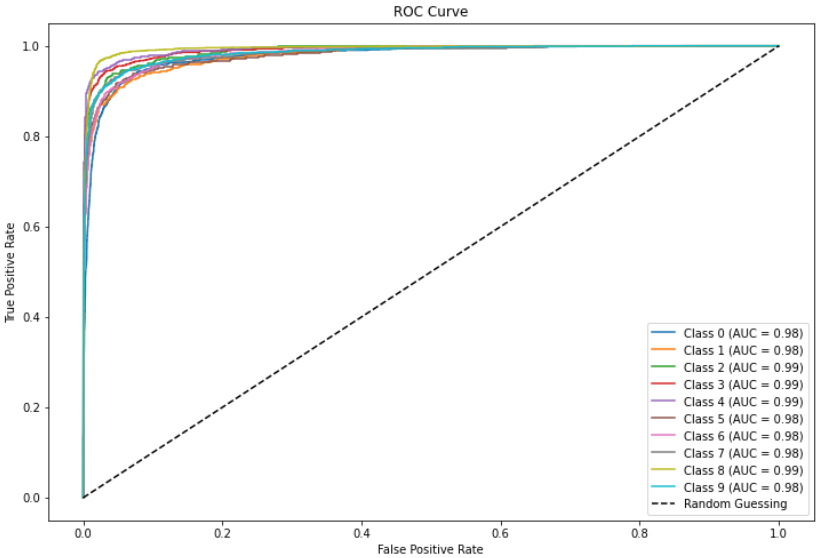
Recall: 0.8671,

F1 Score: 0.8671

Test Accuracy: 86.71%

مدل با توجه به overfit شدن نتیجه نسبتا مطلوبی در داده های تست دارد و نشانه این است که به مقدار زیادی داده های تست را حفظ نکرده و کمی قابلیت generalization دارد(اما هم چنان میزان قابل توجهی خطا نسبت به داده های train دارد)

همانطور که از پیشبینی ها پیداست عملکرد مدل با درصد دقت train فاصله واضحی دارد.



تمامی کلاس‌ها دارای مقادیر AUC بین **0.98تا ۰.۹۹** هستند که نشان‌دهنده عملکرد بسیار خوب مدل برای تمامی کلاس‌ها است.

منحنی‌ها به‌صورت فشرده در نزدیکی گوشه بالا–چپ قرار دارند که نشان‌دهنده حساسیت و اختصاصی بودن بالا است.

از آنجایی که مقادیر AUC برای تمامی کلاس‌ها نزدیک به ۱ هستند، مدل در تفکیک کلاس‌ها حتی در تنظیم آستانه‌های مختلف، عملکرد بسیار خوبی دارد با اینکهoverfit شده است.

۲–فاز۲:

```
OptimizedCNNModel(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (maxpool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
```

(maxpool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

(global_avg_pool): AdaptiveAvgPool2d(output_size=1)

(fc1): Linear(in_features=128, out_features=256, bias=True)

(fc2): Linear(in_features=256, out_features=10, bias=True)

)

Total Parameters: 133034

مدل اول، یعنی CNNModel2، به مدل دوم، یعنی OptimizedCNNModel، تبدیل شده است، عمدتاً با کاهش تعداد نقشه‌های ویژگی (feature maps) در هر لایه‌ی کانولوشن و لایه‌های کاملاً متصل .(FC) این تغییر باعث کاهش قابل‌توجه در تعداد کل پارامترها شده است. در ادامه، توضیحات دقیق‌تر ارائه شده است:

۱ .کاهش نقشه‌های ویژگی در لایه‌های کانولوشن

- **conv1:**تعداد خروجی کانال‌ها از ۳۲ به ۱۶ کاهش یافته است.
- **conv2:**تعداد خروجی کانال‌ها از ۶۴ به ۳۲ کاهش یافته است.
- **conv3:**تعداد خروجی کانال‌ها از ۱۲۸ به ۶۴ کاهش یافته است.
- **conv4:**تعداد خروجی کانال‌ها از ۲۵۶ به ۱۲۸ کاهش یافته است.
- با نصف کردن تعداد نقشه‌های ویژگی در هر لایه، پیچیدگی محاسباتی و مصرف حافظه کاهش یافته است، در حالی که اندازه‌ی کرنل‌ها، گام‌ها(stride) ، و سایر پارامترها بدون تغییر باقی مانده‌اند.

۲ .کاهش تعداد ویژگی‌ها در لایه‌های کاملاً متصل(FC)

- **fc1:**تعداد ویژگی‌های ورودی برای تطبیق با خروجی کاهش‌یافته از لایه‌های کانولوشن از ۲۵۶ به ۱۲۸ کاهش یافته است و تعداد ویژگی‌های خروجی از ۵۱۲ به ۲۵۶ کاهش یافته است.
- **fc2:**تعداد ویژگی‌های ورودی از ۵۱۲ به ۲۵۶ کاهش یافته است، در حالی که تعداد کلاس‌های خروجی (۱۰) بدون تغییر باقی مانده است.

۳ .حفظ ساختار کلی مدل

- معماری کلی مدل بدون تغییر باقی مانده است، شامل ترتیب لایه‌های کانولوشن، pooling، و کاملاً متصل.
- استفاده از لایه‌های MaxPool2dو AdaptiveAvgPool2dباعث شده است که ابعاد مکانی (spatial) نقشه‌های ویژگی به تدریج کاهش یابد و pooling میانگین جهانی همچنان یک بردار ویژگی با اندازه‌ی ثابت برای ورودی به لایه‌های FC تولید کند.

٤ .کاهش تعداد کل پارامترها

- مدل اصلی (CNNModel2) دارای **۵۲۵,۱۳۰ پارامتر** بود.
- مدل بهینه‌سازی شده (OptimizedCNNModel) این تعداد را به **۱۳۳,۰۳٤ پارامتر** کاهش داده است.
- این کاهش عمدتاً به دلیل نصف شدن تعداد نقشه‌های ویژگی در هر لایه‌ی کانولوشن و کاهش اندازه‌ی لایه‌های کاملاً متصل بوده است.

مزایای این تغییر

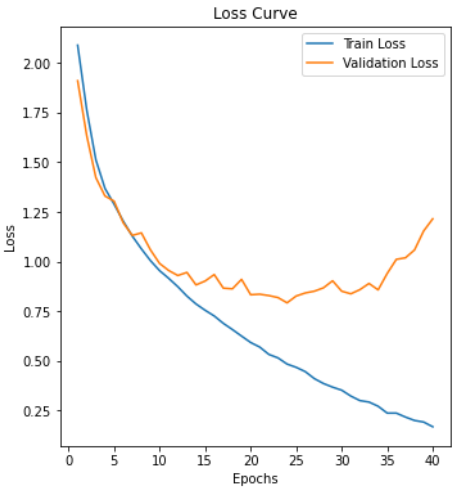
- **کارایی :**مدل بهینه‌سازی شده سبک‌تر و سریع‌تر برای آموزش است و برای محیط‌هایی با منابع محاسباتی محدود مناسب‌تر است.
- **کاهش ریسک بیش‌برازش :(Overfitting)** کاهش تعداد پارامترها ریسک بیش‌برازش را به خصوص هنگام کار با مجموعه داده‌های کوچک کاهش می‌دهد.
- **حفظ معماری :**با وجود کاهش پارامترها، معماری و عملکرد اصلی مدل حفظ شده است.

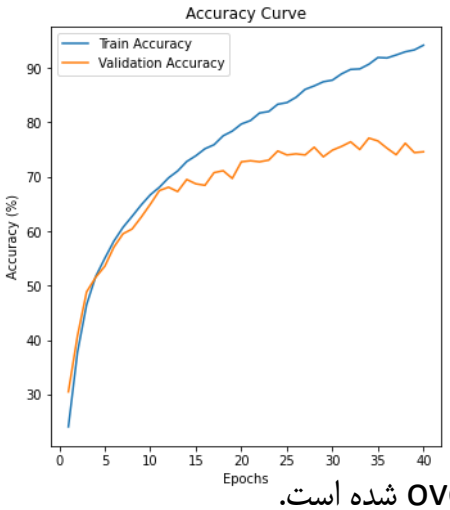
این تغییر بازتابی از تعادل بین پیچیدگی مدل و کارایی محاسباتی است.

عملکرد مدل:

این مدل را با پارامترهای epochs = 25 batch_size = 32 learning_rate = 0.001

آموزش می‌دهیم.





نتیجه آخرین epoch:

Epoch [40/40],

Loss: 0.1690,

Accuracy: 94.19%

Validation Loss: 1.2157

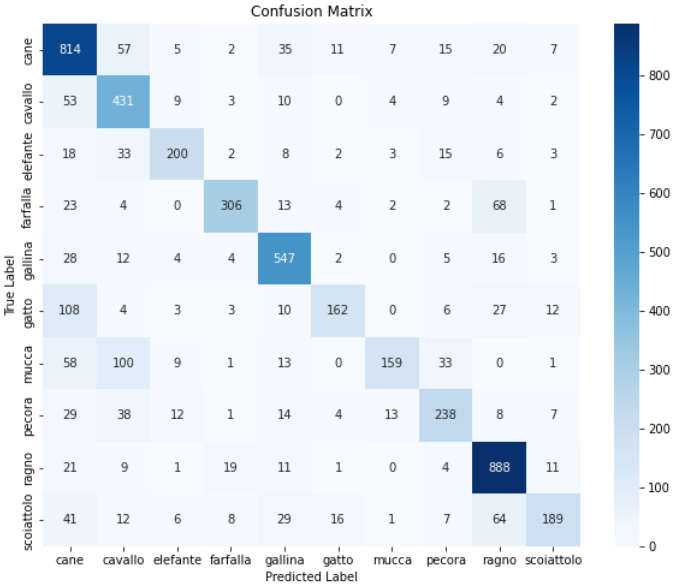
, Validation Accuracy: 74.60%

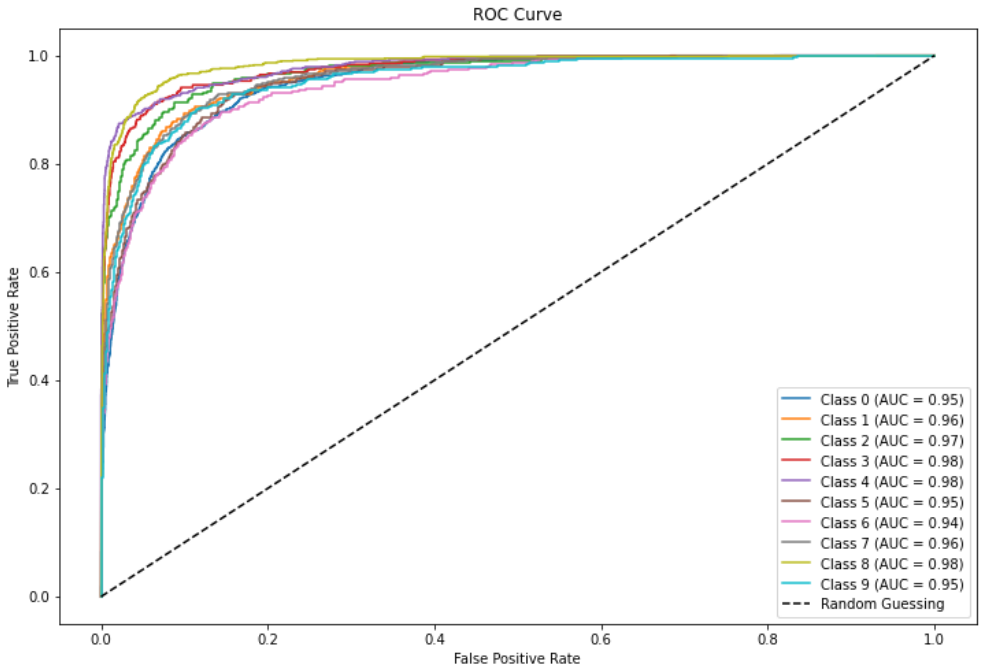
با توجه به نتایج این مدل نیز overfit شده است.

از آنجایی که تعداد پارامتر ها نزدیکتر به تعداد داده های train هستند مدل راحتتر converge می‌شود و بهتر داده هارا حفظ می‌کند که در نتایج بعدی واضح هستند.

Precision: 0.7635, Recall: 0.7503, F1 Score: 0.7424

Test Accuracy: 75.03%





همانطور که از ROC curve ها نیز پیداست مقادیر کمتری نسبت به مدل اول دارند وضعیفتر کلاس هارا ازهم جدا می کنند.

۳-فاز۳:

مدل اول(با پارامترهای بیشتر):

```
CNNModel(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (batchnorm1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (batchnorm2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (batchnorm3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (maxpool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (batchnorm4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (maxpool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (global_avg_pool): AdaptiveAvgPool2d(output_size=1)
  (fc1): Linear(in_features=256, out_features=512, bias=True)
  (dropout1): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=512, out_features=10, bias=True)
)
Total Parameters: 526090
```

در مدل اول(مدل CNN اولیه) یک شبکه عصبی کانولوشنی (CNN) ساده تعریف شده است که شامل چهار لایه کانولوشن، هر کدام با یک لایه pooling و در نهایت یک لایه کاملاً متصل است که به دسته‌بندی انجام می‌دهد. در این مدل، هیچ نوع نرمال‌سازی یا دراپاوت وجود ندارد.

مدل دوم) مدل CNN با بهبودها (تغییراتی دارد که شامل موارد زیر است:

۱. **افزودن نرمال‌سازی دسته‌ای :(Batch Normalization)** در مدل دوم، بعد از هر لایه کانولوشن، یک لایه نرمال‌سازی دسته‌ای (BatchNorm2d) اضافه شده است. این لایه به کاهش تغییرات داخلی توزیع داده‌ها کمک می‌کند و سرعت یادگیری را بهبود می‌بخشد.

۲. **استفاده از :Dropout** در مدل دوم، بعد از لایه‌های کاملاً متصل (fully connected) از دراپاوت (Dropout) استفاده شده است. این تکنیک به جلوگیری از overfitting کمک می‌کند و به‌ویژه در هنگام آموزش روی داده‌های با تنوع زیاد مفید است.

۳. **استفاده از ReLU قبل از هر لایه کانولوشن :**در مدل دوم، برای هر لایه کانولوشن، ابتدا از نرمال‌سازی دسته‌ای استفاده شده، سپس از تابع فعال‌سازی ReLU استفاده می‌شود. این تغییر به مدل کمک می‌کند تا بهتر از ویژگی‌های ورودی خود بهره‌برداری کند.

۴. **پوشش لایه‌های پیشرفته‌تر :**مدل دوم، از یک لایه Global Average Pooling استفاده می‌کند تا اندازه ویژگی‌ها را کاهش داده و تعداد پارامترها را کاهش دهد. این تغییر به ویژه برای کاهش پیچیدگی مدل و جلوگیری از overfitting مفید است.

این تغییرات در مدل دوم باعث می‌شوند که مدل عملکرد بهتری داشته باشد و قابلیت تعمیم بالاتری پیدا کند.

تغییراتی در پیش پردازش داده ها نیز انجام میدهیم:

```
train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),
    transforms.RandomAffine(degrees=0, translate=None, scale=(0.8, 1.2), shear=None),  # Zoom via scaling
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # Example for ImageNet normalization])
```

کد ارائه شده، مجموعه‌ای از تغییرات (transforms) را برای پیش‌پردازش داده‌های ورودی تصویر در فرآیند آموزش مدل در PyTorch تعریف می‌کند. هر یک از این تغییرات به منظور افزایش تنوع داده‌های آموزشی و بهبود عملکرد مدل انجام می‌شود. در اینجا توضیح هر مرحله آورده شده است:

۱. :**transforms.Resize((224, 224))**این تغییر اندازه تصویر را به ابعاد ۲۲۴x۲۲۴ پیکسل انجام می‌دهد. این اندازه معمولاً برای مدل‌هایی که بر اساس دیتاست‌های معروف مانند ImageNet آموزش داده شده‌اند، استفاده می‌شود.

۲. :**transforms.RandomHorizontalFlip()**این تغییر به صورت تصادفی تصویر را به صورت افقی می‌چرخاند (flip می‌کند .(این تکنیک به مدل کمک می‌کند تا ویژگی‌های متقارن مانند حیوانات یا اشیاء را بهتر یاد بگیرد.

۳. :**transforms.RandomRotation(20)**این تغییر به صورت تصادفی تصویر را تا ۲۰ درجه می‌چرخاند. این کار باعث می‌شود که مدل نسبت به چرخش‌های مختلف تصویر مقاوم‌تر شود.

۴. :**transforms.RandomAffine(degrees=0, translate=None, scale=(0.8, 1.2), shear=None)**
این تغییر به صورت تصادفی تصویر را از نظر مقیاس (zoom) تغییر می‌دهد. مقیاس‌ها به مقدار تصادفی بین ۰.۸ و ۱.۲ تغییر می‌کنند که باعث بزرگ‌نمایی یا کوچک‌نمایی تصویر می‌شود. این کار به مدل کمک می‌کند تا ویژگی‌ها را در مقیاس‌های مختلف یاد بگیرد.

۵. **transforms.ToTensor()**: این تغییر تصویر را از فرمت PIL یا Numpy به تنسور تبدیل می‌کند. در PyTorch ، تصاویر باید به صورت تنسور برای آموزش مدل‌ها استفاده شوند.

۶. **transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])**: این تغییر به نرمال‌سازی پیکسل‌های تصویر می‌پردازد. میانگین و انحراف معیار برای هر یک از کانال‌های رنگی (قرمز، سبز و آبی) در دیتاست ImageNet ارائه شده است. این نرمال‌سازی کمک می‌کند تا مقیاس داده‌ها برای آموزش مدل متعادل شود و سرعت یادگیری بهبود یابد.

در کل، این تغییرات به منظور بهبود کیفیت و تنوع داده‌های ورودی برای آموزش بهتر مدل استفاده می‌شوند.

نتایج مدل:

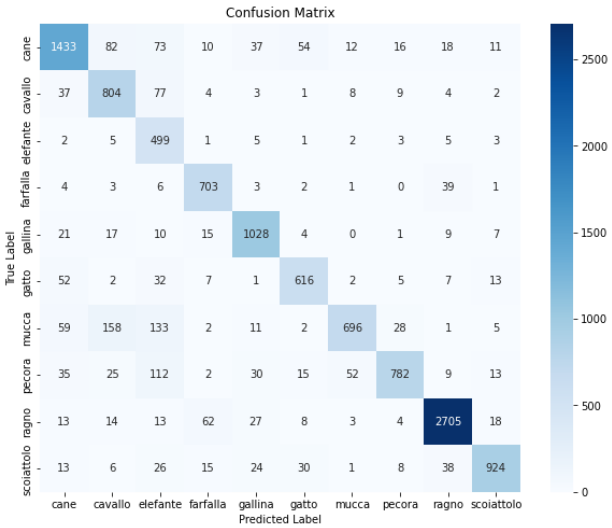
مدل را با پارامتر های مقابل آموزش میدهیم batch_size = 64 learning_rate = 0.005

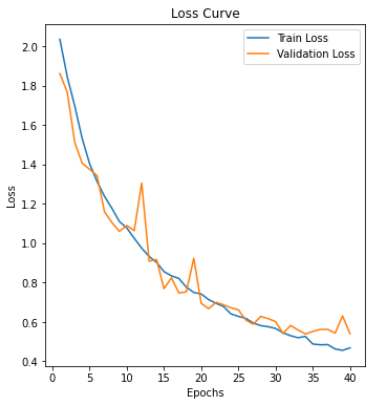
epochs = 40

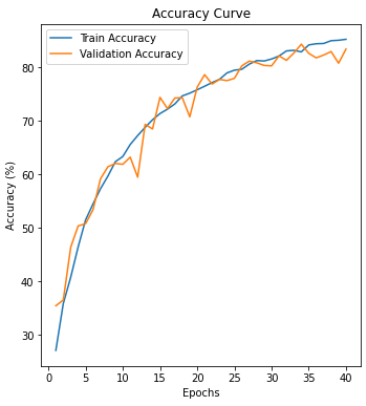
نتیجه epoch آخر:

Epoch [40/40], Loss: 0.4689, Accuracy: 85.30%

Validation Loss: 0.5410, Validation Accuracy: 83.47%



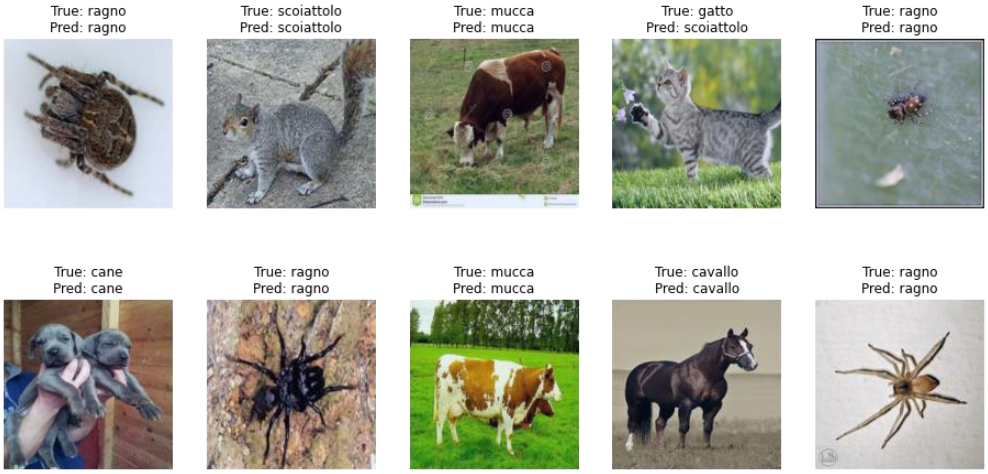


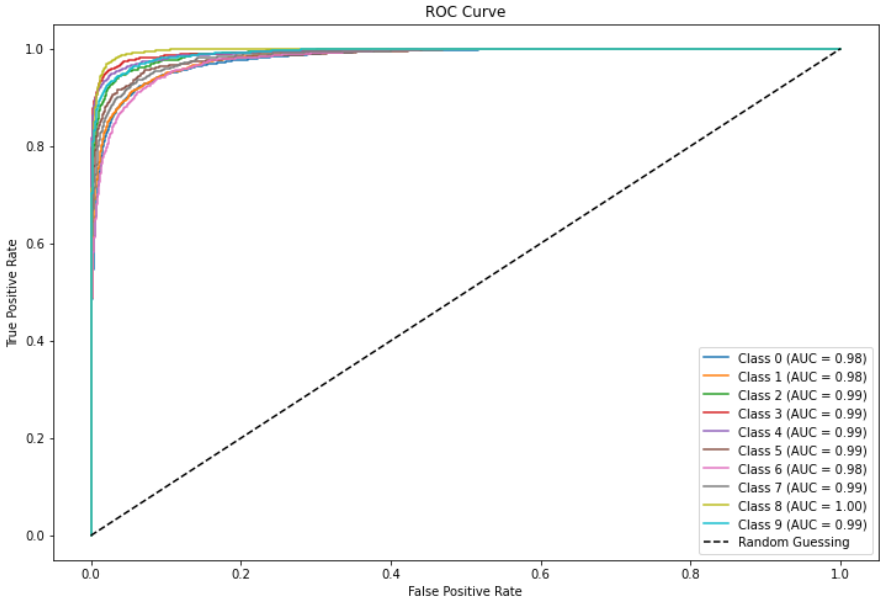


Precision: 0.8703, Recall: 0.8524, F1 Score: 0.8547

Test Accuracy: 85.24%

با توجه نتایج مدل دیگر overfit نیست و اقدامات انجام شده به خوبی مدل را از overfit شدن باز می‌دارند.





و با توجه به ROC curve و پیش بینی های مدل مشخص است که به خوبی عمل میکند و کلاس هارا به خوبی ازهم جدا می‌کند.

٤-فاز٤:

٤_١:

بهترین مدل فاز ٣ را load می‌کنیم و تمام لایه های آنرا بجز لایه های fully connected آن freeze میکنیم و این لایه هارا با پارامتر های مقابل آموزش میدهیم:

epochs = 40 batch_size = 32 learning_rate = 0.017

نتایج:

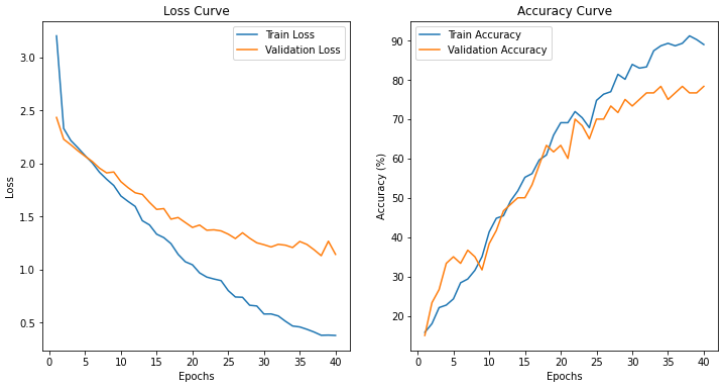
Epoch 40/40:

Train Loss: 0.3773, Train Accuracy: 88.96%

Val Loss: 1.1418, Val Accuracy: 78.33%

Precision: 0.8261, Recall: 0.8071, F1 Score: 0.8054

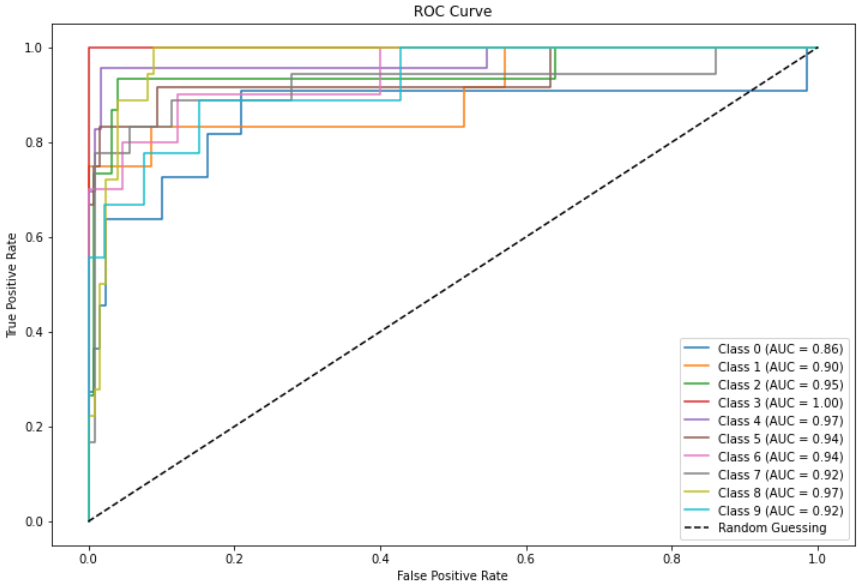
Test Accuracy: 80.71%



با اینکه در این فاز نیز از dropout و data augmentation استفاده شده مدل کمی overfit شده است که به دو دلیل زیر ممکن است اتفاق بیوفتد.

اندازه کوچک دیتاست :وقتی دیتاست هدف کوچک است، حتی با استفاده از یادگیری انتقالی، مدل ممکن است اورفیت شود زیرا داده کافی برای آموزش مدل وجود ندارد. وزن‌های پیش‌آموزش ممکن است برای تعمیم به تکلیف جدید کافی نباشند، به ویژه اگر دیتاست جدید تنوع زیادی داشته باشد.

عدم تطابق مدل پیش‌آموزش :اگر مدل پیش‌آموزش بر روی یک دیتاست که تفاوت زیادی با دیتاست هدف دارد آموزش دیده باشد، ویژگی‌های یادگرفته شده ممکن است به خوبی تعمیم نیابند و منجر به اورفیتینگ شوند. به عنوان مثال، مدلی که بر روی ImageNet آموزش دیده، ممکن است روی دیتاست‌های کوچک‌تر یا خاص با ویژگی‌های متمایز خوب عمل نکند.



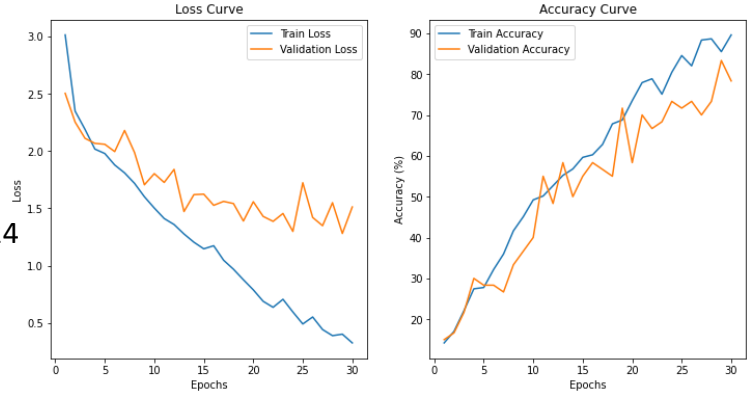
همانطور که از نتایج ROC curve و prediction مدل پیداست مدل به خوبی فرق کلاس هارا تشخیص نمیدهد و حتی برخی کلاس ها که دارای عکس‌های بی کیفیتر هستند، مدل در شناسایی آنها عملکرد بدتری داشته است.

حال اگر لایه convolution آخر را نیز از freeze خارج کنیم نتایج زیر را داریم.

Epoch 30/30

Train Loss: 0.3249, Train Accuracy: 89.59%

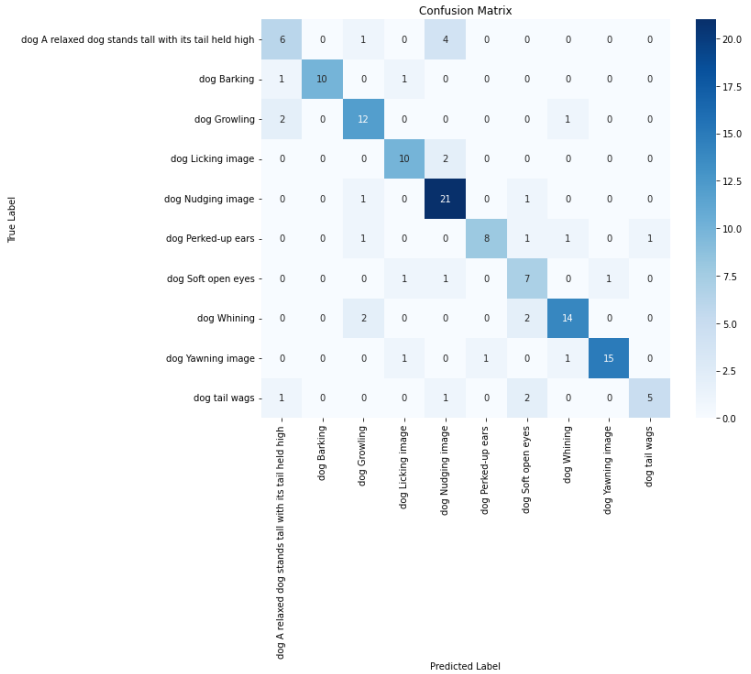
Val Loss: 1.5114, Val Accuracy: 78.33%



Precision: 0.7880, Recall: 0.7714, F1 Score: 0.7724

Test Accuracy: 77.14%

این مدل نیز overfitمیشود رفتار مشابهی مانند حالت قبل دارد.



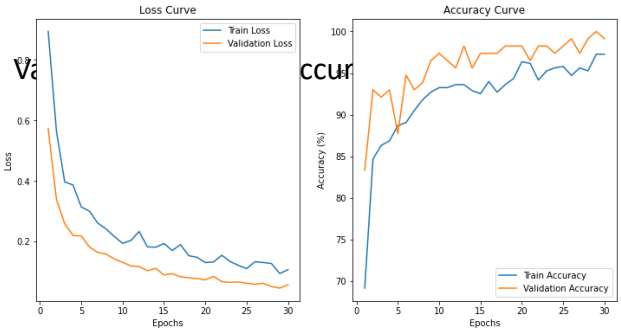
۲_۴:

ابتدا خروجی مدل را به ۳ تغییر میدهیم و سپس با فریز کردن تمامی لایه ها بجز لایه های fully connected و hyperparameter های زیر + data augmentation مدل را آموزش میدهیم.

Batch size = 32 epoch = 30 learning rate = 0.005

نتایج:

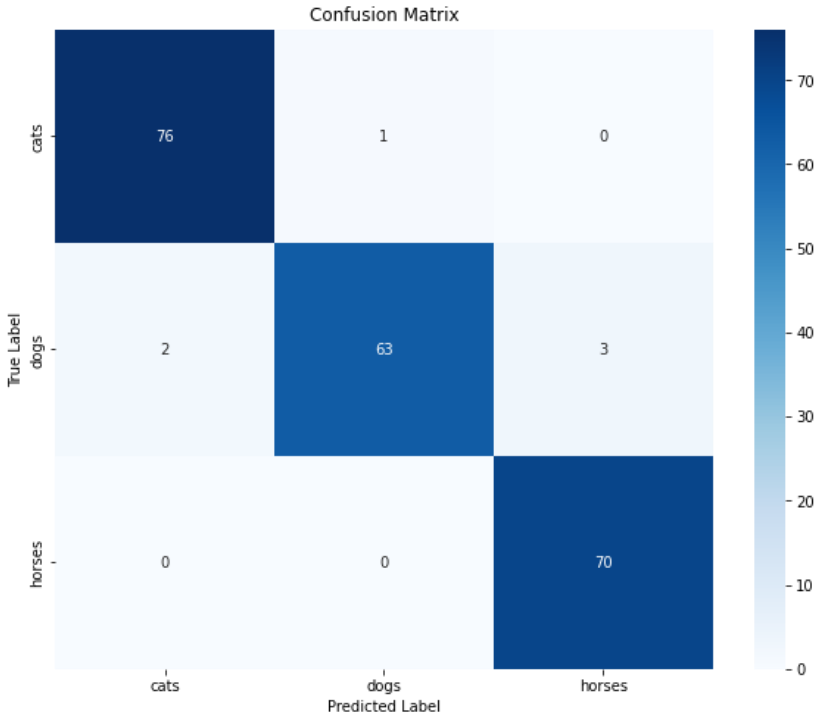
Epoch 30/30: Train Loss: 0.1057, Train Accuracy: 97.26% Va ccur



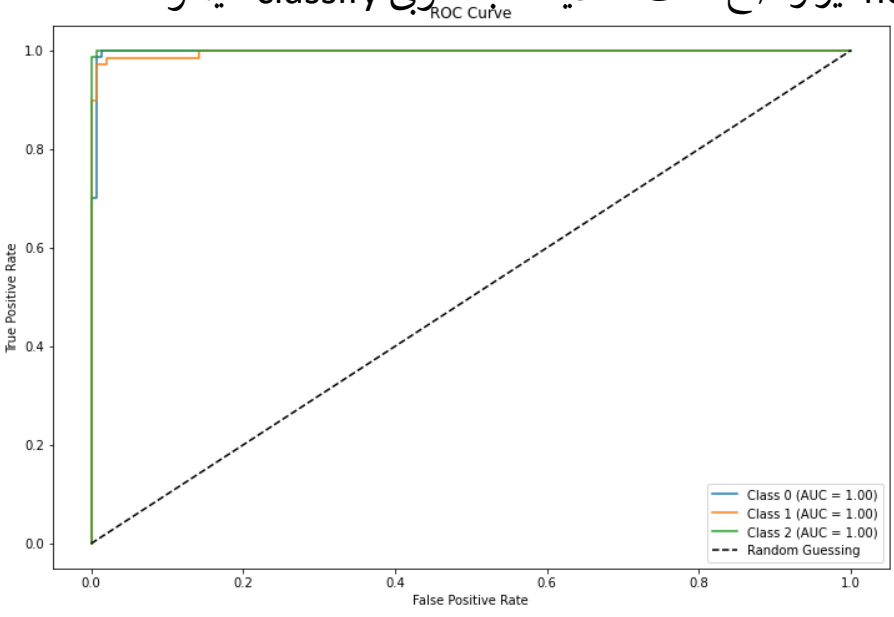
Precision: 0.9725, Recall: 0.9721, F1 Score: 0.9719

Test Accuracy: 97.21%

مدل به خوبی در این دیتاست کار کرده و به بهترین نتیجه میرسد.



از heatmap نیز واضح است که دیتاها به خوبی classify میشوند.



با توجه به مقادیر ROC curve میفهمیم به بهترین نحو کلاس ها ازهم تفکیک شده اند.

مدل های pretrained:

۱۰ لایه از این مدل هارا آموزش میدهیم داریم:

num_epochs = 10 batch_size = 32 learning_rate = 0.001

ResNet50

Epoch [10/10] - Train Loss: 0.0475, Train Acc: 98.18% - Val Loss: 0.0079, Val Acc: 100.00%

MobileNetV2

Epoch [10/10] - Train Loss: 0.0464, Train Acc: 98.54% - Val Loss: 0.0764, Val Acc: 98.25%

EfficientNetB0

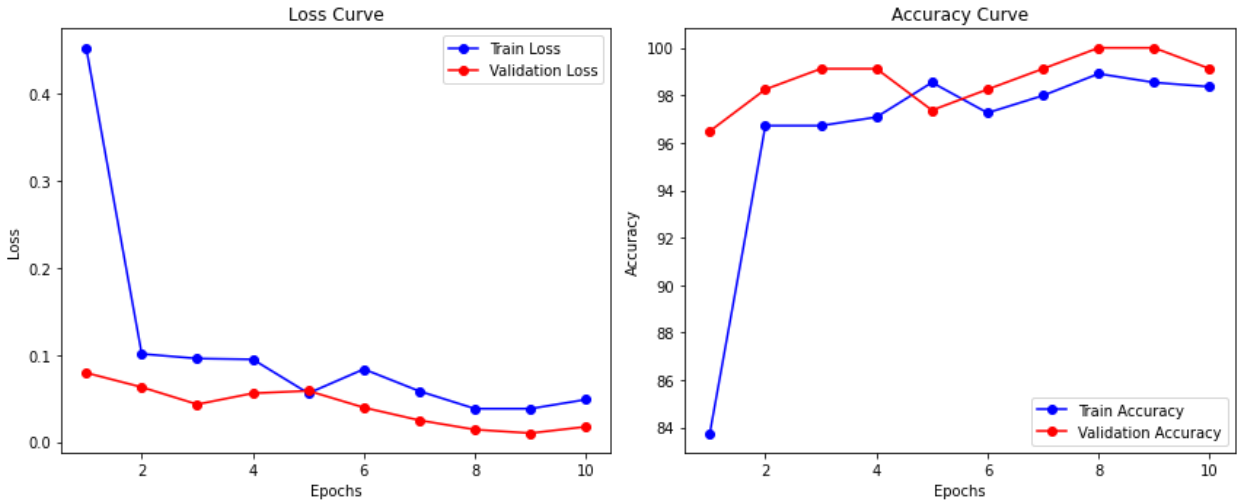
Epoch [10/10] - Train Loss: 0.0488, Train Acc: 98.36% - Val Loss: 0.0176, Val Acc: 99.12%

Final Results:

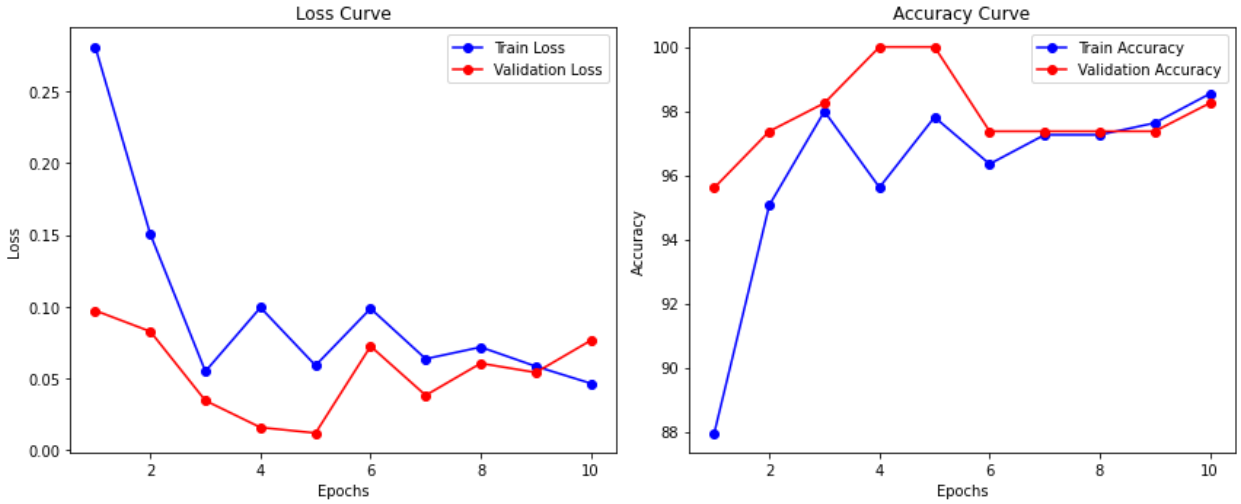
ResNet50: Best Validation Accuracy = 100.00%

MobileNetV2: Best Validation Accuracy = 100.00%

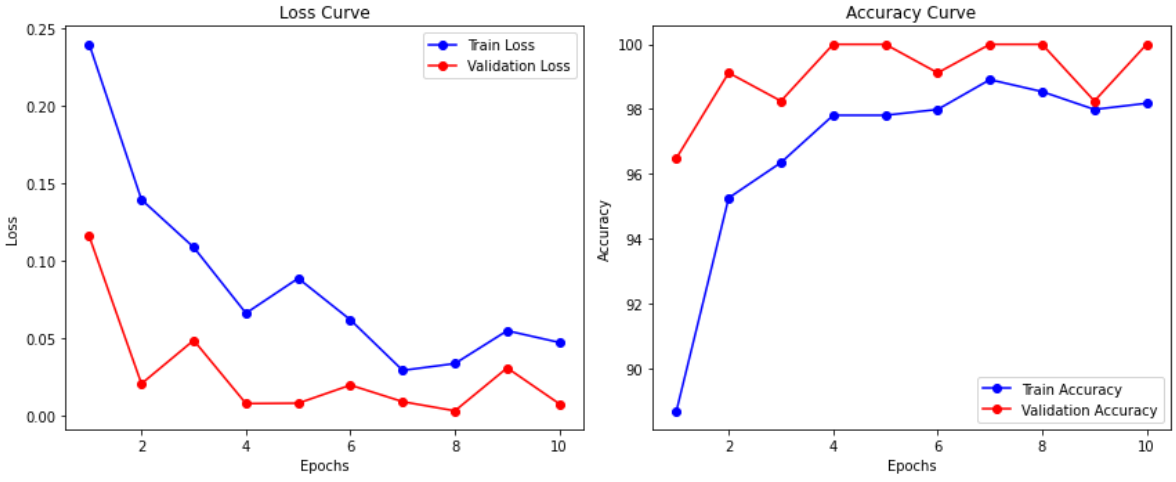
EfficientNetB0: Best Validation Accuracy = 100.00%



ResNet50



EfficientNetB0



MobileNet2

همانطور که پیداست این مدل ها نیز به بهترین نحو عمل کرده و به بهترین نحو داده هارا classify میکنند(نتایج دیگر نمودار ها مشابه بخش قبل).