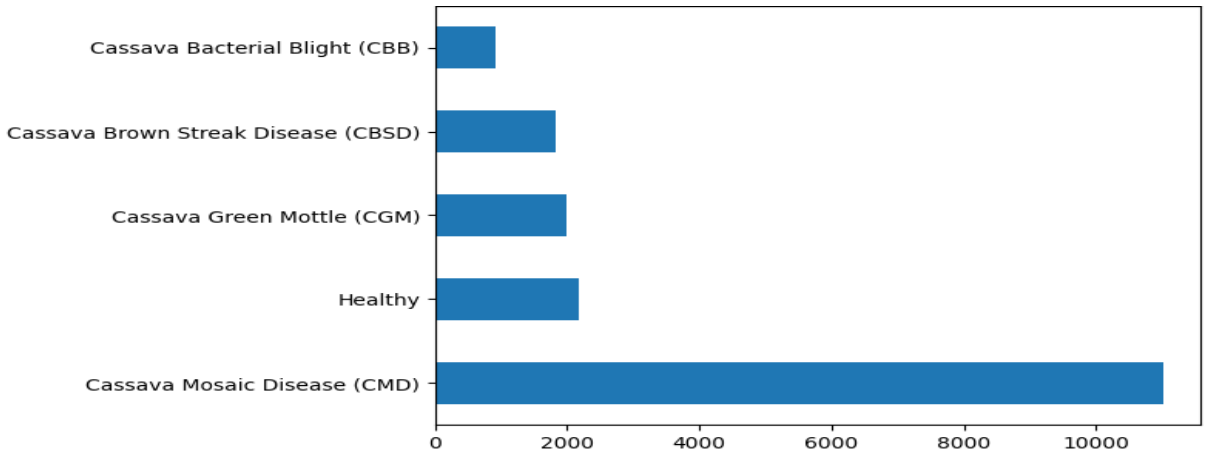
Ahmet Tütüncüoğlu
21120205057

Adım 1 - Probleme yaklaşım ve veri setini inceleme

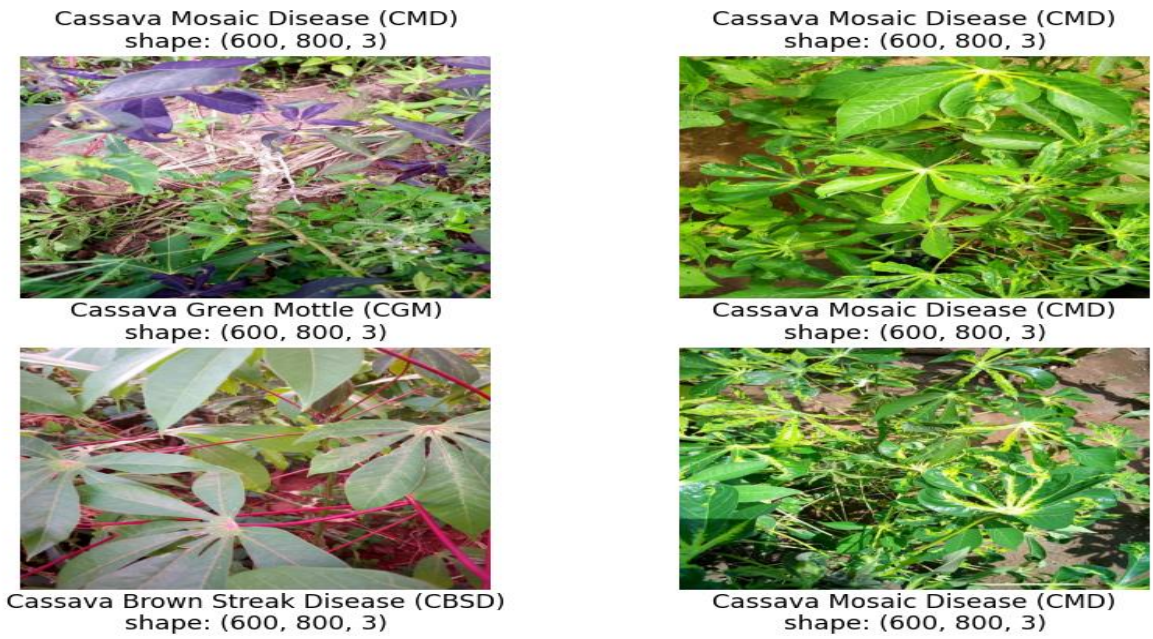
Kaynak veri seti incelendiğinde 21397 veri varken etiketli olan sadece 17.938 veri bulunmaktadır. Literatür ve kaynaklar tarandığında eksik olan verilerin de etiketlendiği veri setine [1] Kaggle platformu üzerinden erişilmiştir.

Veri seti incelendiğinde sınıfların dengesiz (unbalanced) dağılımı tespit edilmiş ve [Resim 1] sınıfların bar grafiği çizdirilmiş olup tüm süreç boyunca araştırılan ve uygulunan yöntemler bu dengesizlik göze alınarak yapılmıştır.



Resim 1

Son olarak veri setindeki resimlerin boyutunu ve genel bitki yapısını gözlemlemek amacıyla 4 adet sample alınıp görselleştirilmiştir [Resim 2].

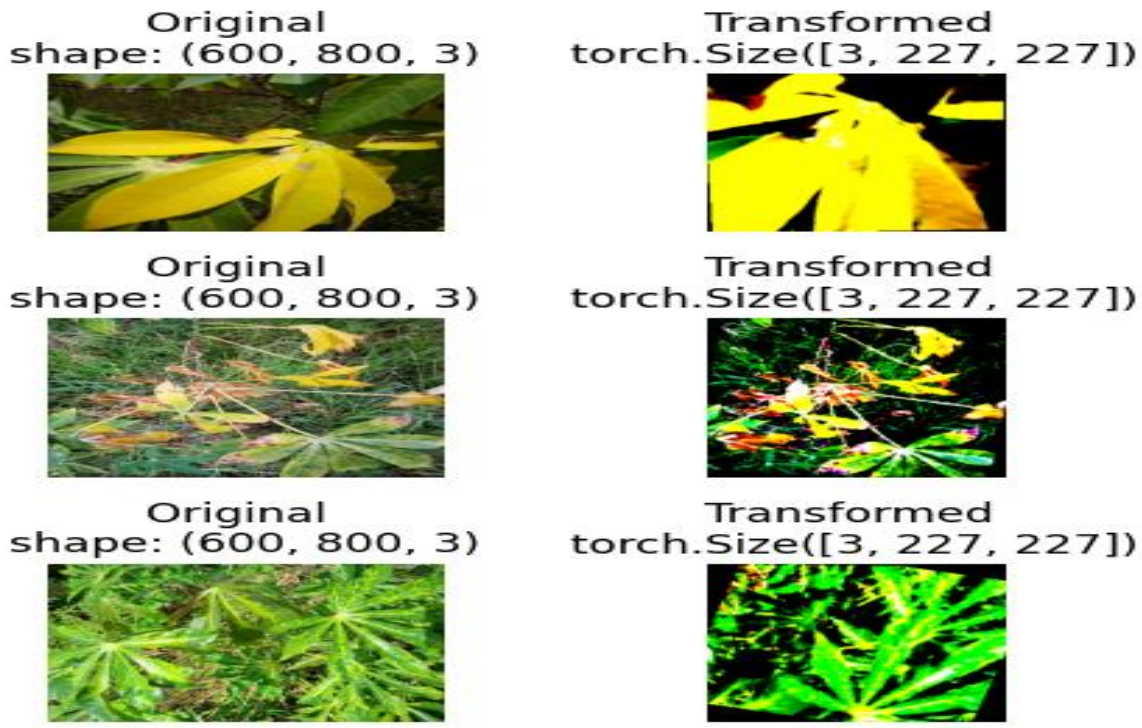


Resim 2

Adım 2 – Probleme dair literatür taraması

Araştırmalarım sonucu probleme ait birçok çözüm içeren makale ve uygulama bulsam da çözüm noktasında ele aldığım temel kaynaklardan birisinde [2] probleme herhangi başka data eklemeden Cutmix, Cutout ve Mixup preproccess yöntemleriyle eğitilen modellerin accuracy skorunda belirli bir artış elde edildiği gözlemleniyor. Bu sebeple ben de modellerimde bu yöntemlerden Cutout ve Cutmix işlemine yer verdim.

Ayrıca veri arttırma noktasında (data augmentation) Torchvision kütüphanesinin transforms metodu probleme dair birçok uygulama ve makalede yer aldığı için projemde ben de bu kütüphaneyi kullandım. Bu metodla birlikte veriye randomflip, randomrotation, gaussianblur, colorjitter, randomresizedcrop ve ImageNet veri setindeki RGB değerlerin ortalaması ve standart sapması normalize fonksiyonu kullanılarak uygulanmıştır [Resim 3].



Resim 3

Adım 3 - Eğitim için gerekli fonksiyonlar ve dosyaların hazırlanması

Kullanacağım yöntemde verileri dataframe üzerinde pathleri ekleyip kullanmak yerine her class için train ve test adlı 2 ayrı klasör oluşturup Shutil kütüphanesiyle bu klasörlere classlara ait verileri ekledim. Böylece daha nizami ve belirli bir formata uygun veri seti elde etmiş oldum.

Ardından torch.utils kütüphanesi içerisinde bulunan Dataset class'ını ImageFolderCustom class'ım içerisinde inherit ederek yine aynı kütüphane içerisinde DataLoader class'ı formatına uygun ve transform fonksiyonu uygulanabilecek bir düzene getirdim.

Tüm bu yöntemler sayesinde veri setimi batchlere ayırıp hız, hafıza verimliliği ve paralel çalışma noktasında avantaj elde ettim.

Adım 4 – GoogleNet, AlexNet, VGGNet ve RestNet mimarilerinin araştırılması

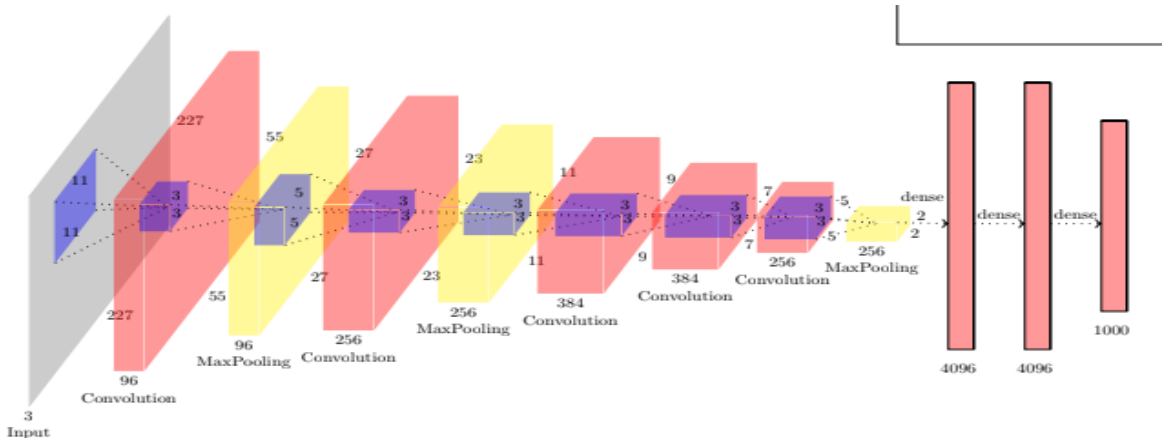
GoogleNet için literatür taraması yapılıp orijinal makalesi [3] bulunmuş olup bu makaleye uygun mimari, modelde birebir uygulanmıştır [Resim 4].

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

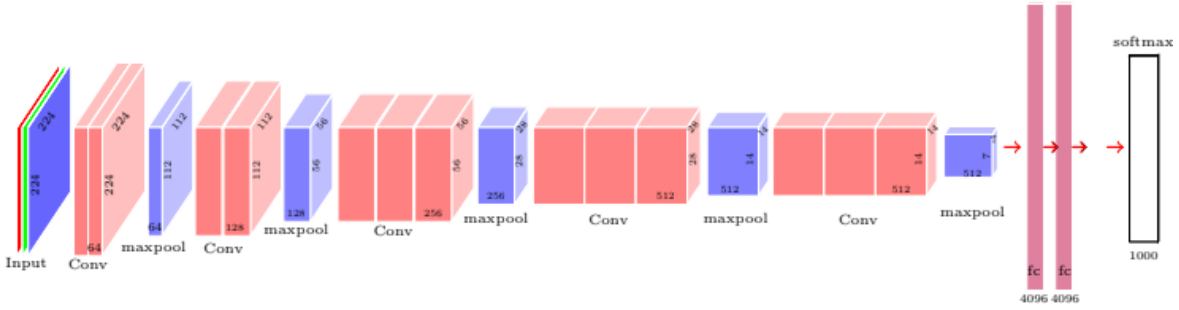
Resim 4

AlexNet için kullanılan mimari ders kaynaklarından alınmış olup modelde birebir uygulanmıştır [Resim 5]



Resim 5

VGGNet için birçok alternatif versiyon araştırmalarım sonucu bulunduysa da kullandığım mimari versiyonu ders kaynaklarından alınmıştır [Resim 6]



Resim 6

RestNet mimarisi için literatür taraması sonucu orijinal makaleye [4] ulaşılmış olup makelede yer alan block tipine göre çeşitli RestNet varyasyonlarından basic block ve bottleneck block kullanılarak sırasıyla modelde RestNet34 ve RestNet152 varyasyonları kullanılmıştır [Resim 7].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Resim 7

Adım 5 – Model, eğitim ve test fonksiyonlarının oluşturulması

Her model pytorch temelli torch.nn modülünü inherit ederek belirtilen CNN mimarilerini oluşturmaktadır. Bazı modellerde ek olarak batch normalization, dropout kullanılarak model, eğitim ve test verimliliği sağlanmıştır. Her model için Adam optimizer ve learning rate değerini güncellemek için ise ReduceLROnPlateau scheduler kullanılmıştır.

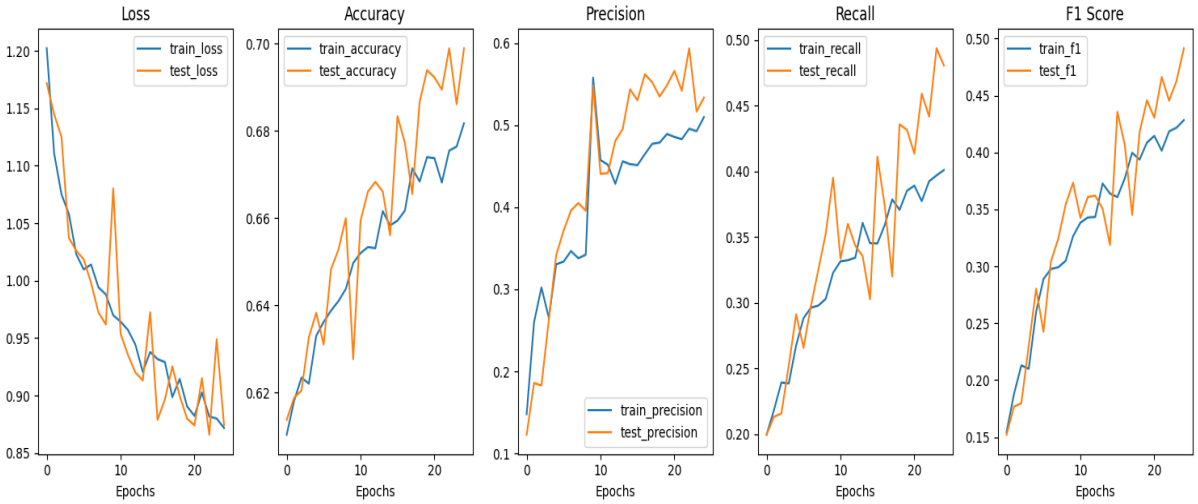
Train ve test aşaması için oluşturduğum train_step ve test_step fonksiyonları ile modelin eğitimi sırasında elde edilen verileri kaydedip görsel olarak sunumu için hazırlık yapılıyor. Ayrıca train_step fonksiyonunda önceden oluşturduğum cutout ya da cutmix fonksiyonlarından birisi 0.5 ihtimalle veriye uygulanıyor. Tüm bunlar dışında her iki fonksiyon da standart olarak şu işlemleri yapar

1. Modeli kullanarak raw tahminleri al
2. Gerçek değerler ve raw tahminleri CrossEntropyLoss kullanarak kıyasla
3. Optimizer türevini sıfırla
4. Backpropagation uygula
5. Optimizer'ı uygula

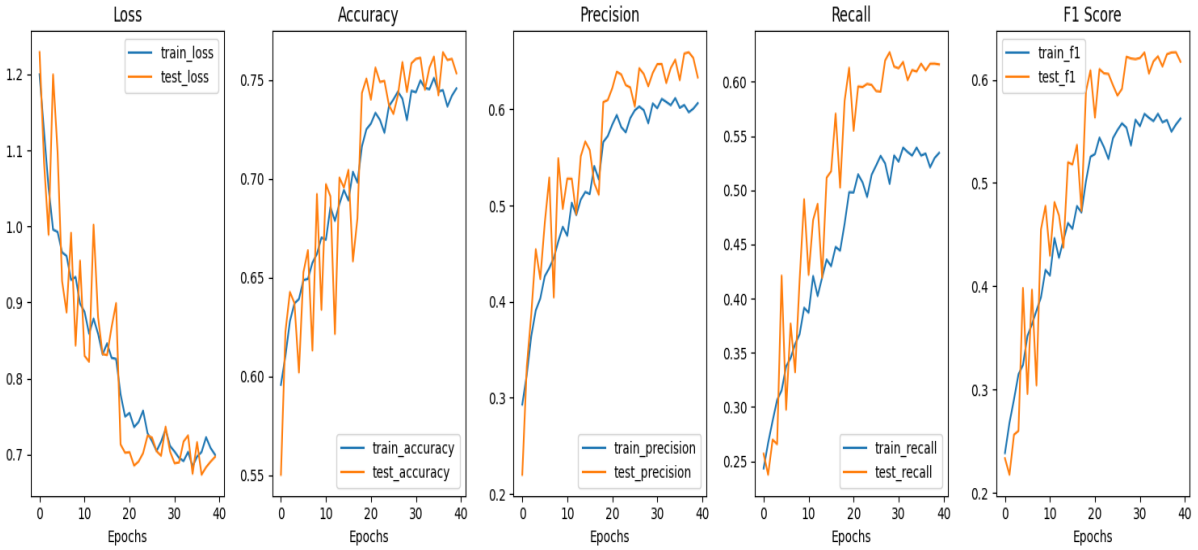
Son olarak train fonksiyonu belirtilen epoch sayısı kadar train_step ve test_step fonksiyonlarını kullanarak eğitim ve test işlemlerini yapıp belirtilen scheduler ile learning rate güncellenir ardından results değişkenine metrikleri kaydeder.

Adım 6 – Modeller ve ortalama metrikleri tablosu

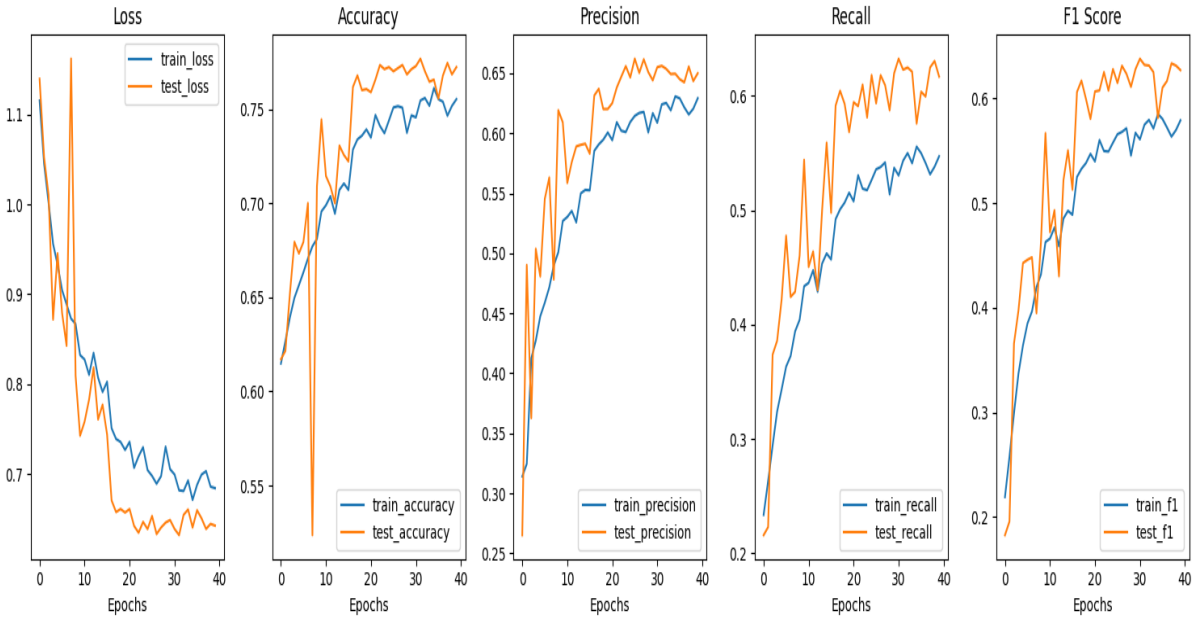
Model - Epoch	Loss	Accuracy	Precision	Recall	F1-Score
VGGNet - 35	0.71	0.74	0.61	0.58	0.57
AlexNet - 25	0.96	0.66	0.44	0.35	0.35
GoogleNet - 40	0.81	0.70	0.56	0.51	0.51
RestNet34 - 40	0.74	0.73	0.60	0.53	0.54
RestNet152 - 40	0.85	0.69	0.52	0.45	0.46



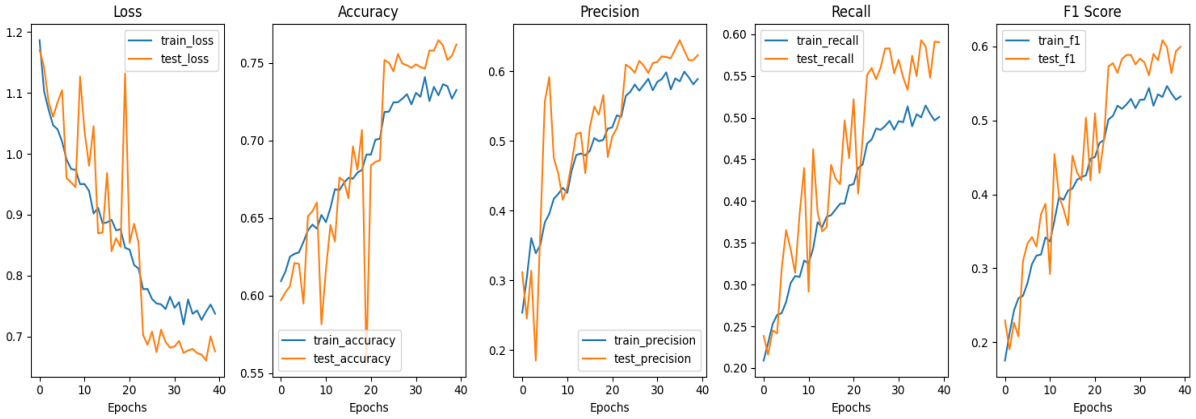
AlexNet



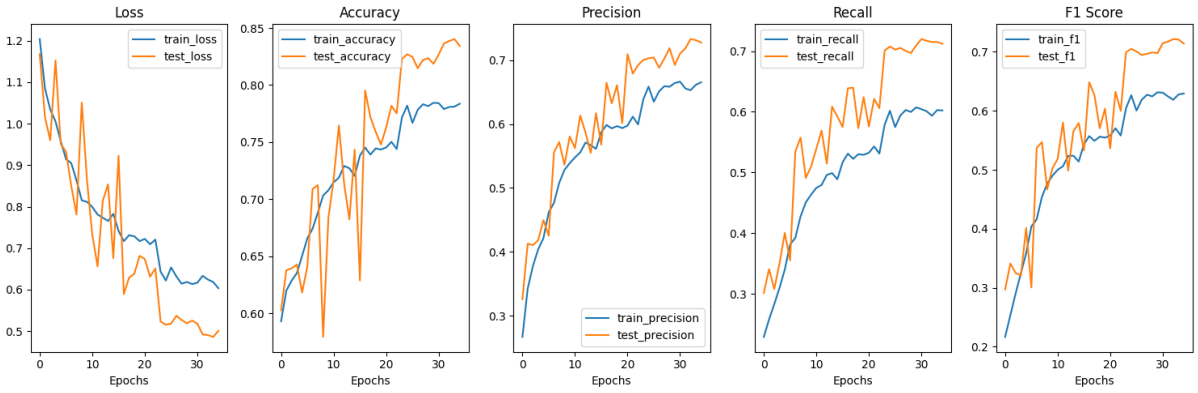
GoogleNet



RestNet34



RestNet152



VGGNet