İSTANBUL MEDENİYET ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BİL485 DERİN ÖĞRENME

Arzu Kalkışım

DERİN ÖĞRENME PROJE ÖDEVİ

Ayşenur YÖRÜR 22120205384

İSTANBUL, 2024

Ödevin İsterleri ve Açıklaması

Ödevin amacı, mahsul hastalıklarının sınıflandırılması için derin öğrenme mimarilerini eğitmek ve test etmektir.

Veri kümesi, 5 ayrı sınıftan oluşmaktadır. https://www.kaggle.com/datasets/mexwell/crop-diseases-classification/data

- Projede, Alex-Net, VGG-net, Resnet, ve GoogleNet mimarilerinin hepsini uygulamanız beklenmektedir. Bu mimarilerin doğrudan tüm katmanları (layerları) kodun içinde tanımlı olmalıdır. (Hazır kısa fonksiyon çağrılarına dayalı olarak mimari çalıştırılmamalıdır.)
- Tasarlayacağınız model için öğrenme hızı, kernel boyutu, padding vb. farklı parametre ayarlamaları yapmanız beklenmektedir.

Şunları içeren bir rapor hazırlayın:

- Doğruluk, Kesinlik, Geri Çağırma, F-ölçüsüne dayalı olarak sonuçları içeren modellerin performansı için bir karşılaştırma tablosu oluşturun.
- Geliştirdiğiniz mimariyi detaylıca anlatınız.
- En iyi performansı elde ettiğiniz parametre değerlerini sununuz.

Ödev ile ilgili Bağlantılar

Colab bağlantısı: https://colab.research.google.com/#fileId=https%3A//storage.googleapis.com/kaggle-colab-exported-notebooks/deep-learning-dev-b3daab0e-7dd3-4f47-b486-3e6085c7d0bb.ipynb%3FX-Goog-Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-161607.iam.gserviceaccount.com/20240528/auto/storage/goog4_request%26X-Goog-Date%3D20240528T195705Z%26X-Goog-Expires%3D259200%26X-Goog-SignedHeaders%3Dhost%26X-Goog-Signature%3D6ea0df62c366400b3280033cb27d906a59eda05535594c2402cfb22e4780cfbeeef1a80dad21ba77348fd6073faa92d9c19362732cc5f91d0988a3b8b6d51cb10954c304226cc2434ecb377db7104c76849979bbcf5398a6ccc9fef4332535824dcce87c65d63afa6030895f97e173b4bad782524e85464a3a6d3fc78be835af8a72ac27989b860a40b9843a96bb6bd3df2c2be3aa6354427b8a0d2f2f3c247b18c1cd1b5945eb5762be0fe4b3c44e88c473ac59226b2b67c72ebc7639245a1bb0b4167e3e68d695184eaf18151c9028c2c8009ec0ccc8e4b43a86e674b34175763d196dd7ad88bdb586a916ad7433f66f2a3d972cf0db6ab7b8074a032bddfe

1. Giriş:

Bu projede https://www.kaggle.com/datasets/mexwell/crop-diseases-classification/data adresindeki veri seti ve etiketleri kullanılarak birden fazla derin öğrenme modelini geliştirmeyi ve geliştirilen derin öğrenme modellerini karşılaştırmak amaçlanmıştır. Modellerin performansı karşılaştırılırken proje isterleri gereği aşağıdaki değerlendirmeler kullanılmıştır.

- Doğruluk (Accuracy):
- Kesinlik (Precision)
- Geri Çağırma (Recall)
- F-1 Skor (F1- Score)

2. Verilerin Hazırlanması:

Veri kümesi, çeşitli bitki hastalıklarını içeren görüntülerden oluşmaktadır. Görüntülerin etiketleri, *label_num_to_disease_map.json* dosyasından alınmıştır. Veriler, özel bir *CustomDataset* sınıfı kullanılarak yüklenmiş ve *DataLoader* ile eğitim sürecine hazırlanmıştır.

Verilerin hazırlanması bölümünde diğer yapılan çalışmalar incelenmiştir ve buna bağlı olarak [1] [2] [3] çalışmalardan yararlanılmıştır.

Veri hazırlanması bölümünde test ve train verileri ayrı olmadığından, image verisi olmalarından, birden fazla label olmasından ve bazı labellar için görsel bulunmadığından ötürü zorluk yaşanmış ve bu problemlerin üstünden gelinmiştir.

x. fotoğrafta geçen kütüphaneler import edilmelidir. Bilgisayar yetersizliğinden ötürü Kaggle üzerinden model eğitimi aşamaları yapılmıştır. Böylelikle sınırlı süreliğine 2 GPU desteği ile daha hızlı bir biçimde işlemlerimizi gerçekleştirebildik ve bu eksikliğin önüne geçmiş olduk.

3. Modellerin Hazırlanması ve Açıklamaları:

Ders üzerinde keras ile model geliştirilmiştir ve LNet örneği yapılmıştır lakin ben PyTorch ile model geliştirmeyi tercih ettim. Resim x ve y de gösterilen PyTorch'un resmî sitesindeki Fotoğraf 1 ve Fotoğraf 2 deki verilen görseller ele alınarak ve ChatGPT'den yardım alınarak katmanların temel yazımları öğrenilmiştir.

```
class LeNet(torch.nn.Module):

    def __init__(self):
        super(LeNet, self).__init__()
        # 1 input image channel (black & white), 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = torch.nn.Conv2d(1, 6, 5)
        self.conv2 = torch.nn.Conv2d(6, 16, 3)
        # an affine operation: y = Wx + b
        self.fc1 = torch.nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
        self.fc2 = torch.nn.Linear(120, 84)
        self.fc3 = torch.nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

Fotoğraf 1. LeNet Örnek yazım

```
import torch

class TinyModel(torch.nn.Module):

    def __init__(self):
        super(TinyModel, self).__init__()

        self.linear1 = torch.nn.Linear(100, 200)
        self.activation = torch.nn.ReLU()
        self.linear2 = torch.nn.Linear(200, 10)
        self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.softmax(x)
        return x

tinymodel = TinyModel()
```

Fotoğraf 2. Küçük bir Model Örneği

3.1. Alex.Net:

Alex.Net ve PyTorch için github'dan çeşitli örnekler [4] incelenmiştir. Medium blogu [5] incelenmiş ve Tablo 1'deki katmanlar temel alınarak model geliştirilmiştir.

AlexNet Network - Structural Details													
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
						Total							**62,378,344**

AlexNet Parametre sayısı

Tablo 1. AlexNet Yapısal Detaylar

Geliştirilen modelin kodları fotoğraf 3'te, modelin özeti fotoğraf 4'te yer almaktadır.

```
class AlexNet(nn.Module):
    def __init__(self, num_classes=5):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=0),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=0),
            nn.Conv2d(96, 256, kernel_size=5, stride=1, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=0),
            nn.Conv2d(256, 384, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 384, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=0)
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

# Modeli oluşturun ve cihaza (device) taşıyın
model = AlexNet().to(device)

# Model özetini alın
summary(model, (3, 227, 227))
```

Fotoğraf 3. AlexNet Kod

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 55, 55]          34,944
              ReLU-2           [-1, 96, 55, 55]               0
         MaxPool2d-3           [-1, 96, 27, 27]               0
            Conv2d-4          [-1, 256, 27, 27]         614,656
              ReLU-5          [-1, 256, 27, 27]               0
         MaxPool2d-6          [-1, 256, 13, 13]               0
            Conv2d-7          [-1, 384, 13, 13]         885,120
              ReLU-8          [-1, 384, 13, 13]               0
            Conv2d-9          [-1, 384, 13, 13]       1,327,488
             ReLU-10          [-1, 384, 13, 13]               0
           Conv2d-11          [-1, 256, 13, 13]         884,992
             ReLU-12          [-1, 256, 13, 13]               0
        MaxPool2d-13            [-1, 256, 6, 6]               0
          Dropout-14                 [-1, 9216]               0
           Linear-15                 [-1, 4096]      37,752,832
             ReLU-16                 [-1, 4096]               0
          Dropout-17                 [-1, 4096]               0
           Linear-18                 [-1, 4096]      16,781,312
             ReLU-19                 [-1, 4096]               0
           Linear-20                    [-1, 5]          20,485
================================================================
Total params: 58,301,829
Trainable params: 58,301,829
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.59
Forward/backward pass size (MB): 11.08
Params size (MB): 222.40
Estimated Total Size (MB): 234.07
----------------------------------------------------------------
```

Fotoğraf 4. AlexNet Model Özeti

Geliştirilen mimarinin açıklaması:

1. Özellik Çıkarıcı Katmanlar (Feature Extractor Layers)

Özellik çıkarıcı katmanlar, girdi görüntüsündeki düşük seviyeli özellikleri (kenarlar, dokular, renkler) çıkararak başlar ve daha yüksek seviyeli özellikleri (nesne parçaları) öğrenerek devam eder. Bu mimarideki katmanlar sırasıyla şunlardır:

- Katman 1:
    - Conv2d: 3 girdi kanalı (RGB görüntü), 96 çıktı kanalı, 11x11 kernel boyutu, 4 adım (stride), 0 padding
    - ReLU: Aktivasyon fonksiyonu

    - MaxPool2d: 3x3 kernel boyutu, 2 adım, 0 padding

- Katman 2:
    - Conv2d: 96 girdi kanalı, 256 çıktı kanalı, 5x5 kernel boyutu, 1 adım, 2 padding
    - ReLU: Aktivasyon fonksiyonu
    - MaxPool2d: 3x3 kernel boyutu, 2 adım, 0 padding
- Katman 3:
    - Conv2d: 256 girdi kanalı, 384 çıktı kanalı, 3x3 kernel boyutu, 1 adım, 1 padding
    - ReLU: Aktivasyon fonksiyonu
    - Katman 4:
    - Conv2d: 384 girdi kanalı, 384 çıktı kanalı, 3x3 kernel boyutu, 1 adım, 1 padding
    - ReLU: Aktivasyon fonksiyonu
- Katman 5:
    - Conv2d: 384 girdi kanalı, 256 çıktı kanalı, 3x3 kernel boyutu, 1 adım, 1 padding
    - ReLU: Aktivasyon fonksiyonu
    - MaxPool2d: 3x3 kernel boyutu, 2 adım, 0 padding

3.2. VGG.Net:

VGG.Net ve PyTorch için çeşitli github örnekleri [6] incelenmiştir. Medium blogu [5] incelenmiş ve Tablo 2'deki katmanlar temel alınarak model geliştirilmiştir.

VGG16 - Structural Details													
#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total													138,423,208

Tablo 2. VGGNet Yapısal Detaylar

Geliştirilen modelin kodları 5. fotoğrafta, modelin özeti 6'nci fotoğrafta yer almaktadır.

```
class VGG16(nn.Module):
    def __init__(self, num_classes=5):
        super(VGG16, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

Fotoğraf 5. VGGNet Kodları

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 227, 227]           1,792
              ReLU-2         [-1, 64, 227, 227]               0
            Conv2d-3         [-1, 64, 227, 227]          36,928
              ReLU-4         [-1, 64, 227, 227]               0
         MaxPool2d-5         [-1, 64, 113, 113]               0
            Conv2d-6        [-1, 128, 113, 113]          73,856
              ReLU-7        [-1, 128, 113, 113]               0
            Conv2d-8        [-1, 128, 113, 113]         147,584
              ReLU-9        [-1, 128, 113, 113]               0
        MaxPool2d-10          [-1, 128, 56, 56]               0
           Conv2d-11          [-1, 256, 56, 56]         295,168
             ReLU-12          [-1, 256, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         590,080
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
        MaxPool2d-17          [-1, 256, 28, 28]               0
           Conv2d-18          [-1, 512, 28, 28]       1,180,160
             ReLU-19          [-1, 512, 28, 28]               0
           Conv2d-20          [-1, 512, 28, 28]       2,359,808
             ReLU-21          [-1, 512, 28, 28]               0
           Conv2d-22          [-1, 512, 28, 28]       2,359,808
             ReLU-23          [-1, 512, 28, 28]               0
        MaxPool2d-24          [-1, 512, 14, 14]               0
           Conv2d-25          [-1, 512, 14, 14]       2,359,808
             ReLU-26          [-1, 512, 14, 14]               0
           Conv2d-27          [-1, 512, 14, 14]       2,359,808
             ReLU-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
        MaxPool2d-31            [-1, 512, 7, 7]               0
           Linear-32                 [-1, 4096]     102,764,544
             ReLU-33                 [-1, 4096]               0
          Dropout-34                 [-1, 4096]               0
           Linear-35                 [-1, 4096]      16,781,312
             ReLU-36                 [-1, 4096]               0
          Dropout-37                 [-1, 4096]               0
           Linear-38                    [-1, 5]          20,485
================================================================
Total params: 134,281,029
Trainable params: 134,281,029
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.59
Forward/backward pass size (MB): 222.21
Params size (MB): 512.24
Estimated Total Size (MB): 735.04
----------------------------------------------------------------
```

Fotoğraf 6. VGGNet Model Özeti

3.3. ResNet18 Modeli

3.3.1. ResNet ResidualBlock

ResidualBlock, ResNet mimarisinin temel yapı taşıdır. Bu blok, girdi tensörünü doğrudan çıktıya ekleyerek "kalan bağlantı" (residual connection) oluşturur. Bu yapı, derin ağlarda gradyan sorunlarını hafifletmeye yardımcı olur. Fotoğraf 7'de Residual Kod bloğu verilmiştir.

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

Fotoğraf 7. ResidualBlock

ResidualBlock Açıklaması

• **İlk Katman (conv1 ve bn1)**: 3x3 kernel boyutunda ve `stride` parametresi ile belirtilen adım büyüklüğünde evrişim katmanı (conv2d). Bunu, çıkışları normalize eden bir batch normalization katmanı (bn1) takip eder. Aktivasyon fonksiyonu olarak ReLU kullanılır.

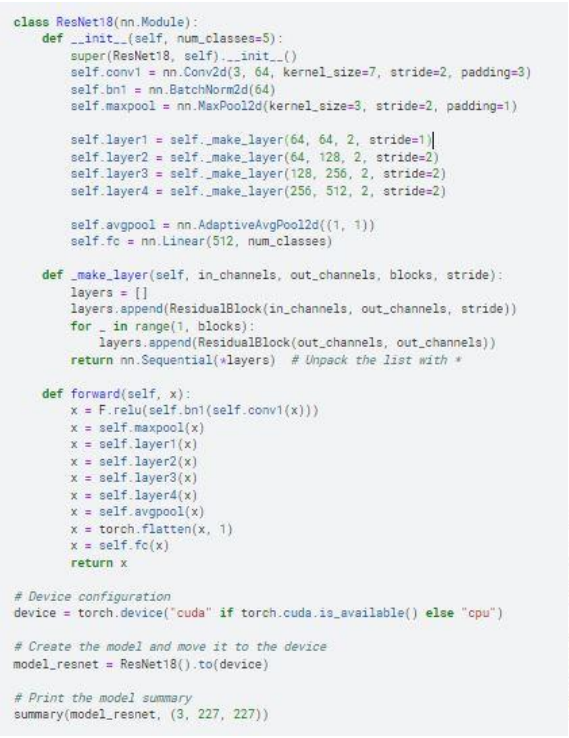
• **İkinci Katman (conv2 ve bn2)**: 3x3 kernel boyutunda ve 1 adım büyüklüğünde evrişim katmanı (conv2d). Bunu, çıkışları normalize eden bir batch normalization katmanı (bn2) takip eder.

• **Shortcut**: Eğer giriş ve çıkış kanalları veya stride farklı ise, giriş verisinin boyutunu değiştirmek için 1x1 kernel boyutunda evrişim ve batch normalization katmanlarından oluşan bir shortcut katmanı oluşturulur. Bu, residual bağlantıdaki boyut uyuşmazlıklarını giderir.

• **Forward Metodu**: İki ana adımı içerir:

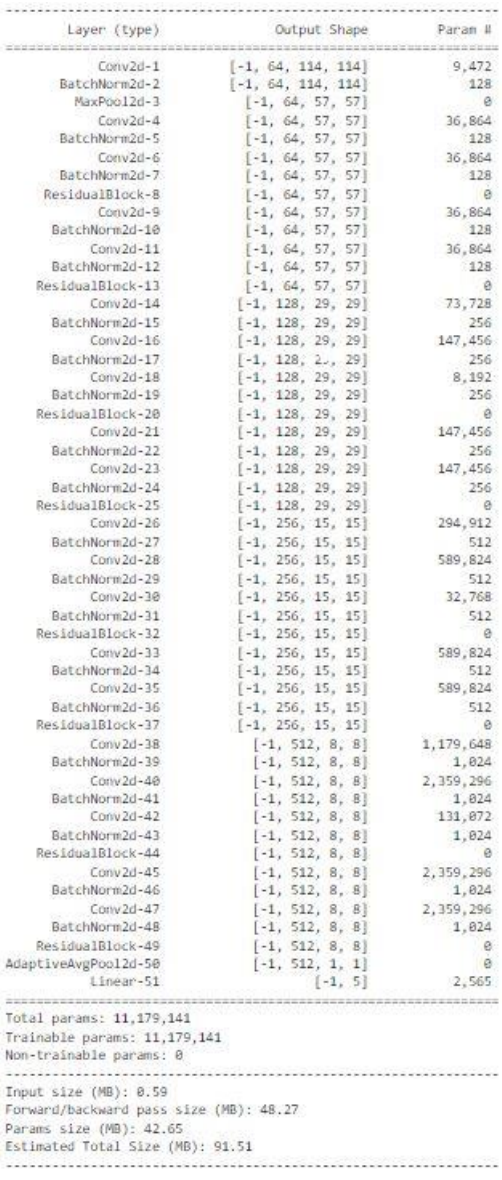
1. Giriş verisi iki evrişim katmanı ve batch normalization katmanından geçer.
2. Elde edilen çıktı, shortcut yolundan gelen giriş verisiyle toplanır ve ReLU aktivasyon fonksiyonundan geçirilir.

Fotoğraf 8'de ResNet18 modelinin kodu verilmiş olup mimarinin açıklaması aşağıda mevcuttur. Fotoğraf 9'da model özeti verilmiştir.



```python
class ResNet18(nn.Module):
    def __init__(self, num_classes=5):
        super(ResNet18, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = self._make_layer(64, 64, 2, stride=1)
        self.layer2 = self._make_layer(64, 128, 2, stride=2)
        self.layer3 = self._make_layer(128, 256, 2, stride=2)
        self.layer4 = self._make_layer(256, 512, 2, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)

    def _make_layer(self, in_channels, out_channels, blocks, stride):
        layers = []
        layers.append(ResidualBlock(in_channels, out_channels, stride))
        for _ in range(1, blocks):
            layers.append(ResidualBlock(out_channels, out_channels))
        return nn.Sequential(*layers)  # Unpack the list with *

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.maxpool(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x

# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Create the model and move it to the device
model_resnet = ResNet18().to(device)

# Print the model summary
summary(model_resnet, (3, 227, 227))
```

Fotoğraf 8. ResNet18 modeli



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 114, 114]	9,472
BatchNorm2d-2	[-1, 64, 114, 114]	128
MaxPool2d-3	[-1, 64, 57, 57]	0
Conv2d-4	[-1, 64, 57, 57]	36,864
BatchNorm2d-5	[-1, 64, 57, 57]	128
Conv2d-6	[-1, 64, 57, 57]	36,864
BatchNorm2d-7	[-1, 64, 57, 57]	128
ResidualBlock-8	[-1, 64, 57, 57]	0
Conv2d-9	[-1, 64, 57, 57]	36,864
BatchNorm2d-10	[-1, 64, 57, 57]	128
Conv2d-11	[-1, 64, 57, 57]	36,864
BatchNorm2d-12	[-1, 64, 57, 57]	128
ResidualBlock-13	[-1, 64, 57, 57]	0
Conv2d-14	[-1, 128, 29, 29]	73,728
BatchNorm2d-15	[-1, 128, 29, 29]	256
Conv2d-16	[-1, 128, 29, 29]	147,456
BatchNorm2d-17	[-1, 128, 2., 29]	256
Conv2d-18	[-1, 128, 29, 29]	8,192
BatchNorm2d-19	[-1, 128, 29, 29]	256
ResidualBlock-20	[-1, 128, 29, 29]	0
Conv2d-21	[-1, 128, 29, 29]	147,456
BatchNorm2d-22	[-1, 128, 29, 29]	256
Conv2d-23	[-1, 128, 29, 29]	147,456
BatchNorm2d-24	[-1, 128, 29, 29]	256
ResidualBlock-25	[-1, 128, 29, 29]	0
Conv2d-26	[-1, 256, 15, 15]	294,912
BatchNorm2d-27	[-1, 256, 15, 15]	512
Conv2d-28	[-1, 256, 15, 15]	589,824
BatchNorm2d-29	[-1, 256, 15, 15]	512
Conv2d-30	[-1, 256, 15, 15]	32,768
BatchNorm2d-31	[-1, 256, 15, 15]	512
ResidualBlock-32	[-1, 256, 15, 15]	0
Conv2d-33	[-1, 256, 15, 15]	589,824
BatchNorm2d-34	[-1, 256, 15, 15]	512
Conv2d-35	[-1, 256, 15, 15]	589,824
BatchNorm2d-36	[-1, 256, 15, 15]	512
ResidualBlock-37	[-1, 256, 15, 15]	0
Conv2d-38	[-1, 512, 8, 8]	1,179,648
BatchNorm2d-39	[-1, 512, 8, 8]	1,024
Conv2d-40	[-1, 512, 8, 8]	2,359,296
BatchNorm2d-41	[-1, 512, 8, 8]	1,024
Conv2d-42	[-1, 512, 8, 8]	131,072
BatchNorm2d-43	[-1, 512, 8, 8]	1,024
ResidualBlock-44	[-1, 512, 8, 8]	0
Conv2d-45	[-1, 512, 8, 8]	2,359,296
BatchNorm2d-46	[-1, 512, 8, 8]	1,024
Conv2d-47	[-1, 512, 8, 8]	2,359,296
BatchNorm2d-48	[-1, 512, 8, 8]	1,024
ResidualBlock-49	[-1, 512, 8, 8]	0
AdaptiveAvgPool2d-50	[-1, 512, 1, 1]	0
Linear-51	[-1, 5]	2,565

Total params: 11,179,141
Trainable params: 11,179,141
Non-trainable params: 0

Input size (MB): 0.59
Forward/backward pass size (MB): 48.27
Params size (MB): 42.65
Estimated Total Size (MB): 91.51

Fotoğraf 9. ResNet Model Özeti

• **Başlangıç Katmanı (conv1, bn1 ve maxpool)**: 7x7 kernel boyutunda evrişim katmanı ve batch normalization katmanı. Ardından 3x3 kernel boyutunda ve 2 adım büyüklüğünde bir maksimum havuzlama (maxpool) katmanı bulunur.

• **Layer1-4**: Her biri farklı kanal boyutlarına sahip dört katman. `_make_layer` fonksiyonu ile oluşturulurlar. Bu fonksiyon, verilen kanal ve blok sayısına göre ResidualBlock'lar oluşturur ve sıraya dizer.

• **Ortalama Havuzlama ve Tam Bağlantılı Katman (avgpool ve fc)**: Adaptive average pooling ile boyut 1x1'e düşürülür ve tam bağlantılı (fully connected) katmana geçirilir.

Resnet mimarisi hazırlanırken pytorch'un açık kaynak ResNet [7] kodlarından ve layerlar arasında uyumsuzluk hatalarının giderilmesinde ChatGPT'den yararlanmıştır özellikle _make_layer bölümü github reposu kullanılarak oluşturulmuştur. ResNet mimarisini açıklarken ve Residual Block hakkında bilgi verirken kaynak olarak [5] bloğundan ve [8] makalesinden faydalanılmıştır.

3.4. GoogLeNet/Inception Modeli

İlk birkaç katman, standart konvolüsyonel katmanlar ve maksimum havuzlama katmanları içerir. Takip eden katmanlar, bir dizi Inception modülünden oluşur. Ve sonra adaptive average pooling ve fully connected katmanı gelir. Fotoğraf 10'da modelin kodları verilmiştir. Çok uzun olduğundan dolayı model özetinin sadece son kısmı fotoğraf 11'de verilmiştir.

```
class GoogLeNet(nn.Module):
    def __init__(self, num_classes=5):
        super(GoogLeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3)
        self.maxpool1 = nn.MaxPool2d(3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=1)
        self.conv3 = nn.Conv2d(64, 192, kernel_size=3, padding=1)
        self.maxpool2 = nn.MaxPool2d(3, stride=2, padding=1)

        self.inception3a = Inception(192, 64, 96, 128, 16, 32, 32)
        self.inception3b = Inception(256, 128, 128, 192, 32, 96, 64)
        self.maxpool3 = nn.MaxPool2d(3, stride=2, padding=1)

        self.inception4a = Inception(480, 192, 96, 208, 16, 48, 64)
        self.inception4b = Inception(512, 160, 112, 224, 24, 64, 64)
        self.inception4c = Inception(512, 128, 128, 256, 24, 64, 64)
        self.inception4d = Inception(512, 112, 144, 288, 32, 64, 64)
        self.inception4e = Inception(528, 256, 160, 320, 32, 128, 128)
        self.maxpool4 = nn.MaxPool2d(2, stride=2, padding=1)

        self.inception5a = Inception(832, 256, 160, 320, 32, 128, 128)
        self.inception5b = Inception(832, 384, 192, 384, 48, 128, 128)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.dropout = nn.Dropout(0.4)
        self.fc = nn.Linear(1024, num_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.maxpool1(x)
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = self.maxpool2(x)

        x = self.inception3a(x)
        x = self.inception3b(x)
        x = self.maxpool3(x)

        x = self.inception4a(x)
        x = self.inception4b(x)
        x = self.inception4c(x)
        x = self.inception4d(x)
        x = self.inception4e(x)
        x = self.maxpool4(x)

        x = self.inception5a(x)
        x = self.inception5b(x)

        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.dropout(x)
        x = self.fc(x)
        return x
```

Fotoğraf 10. GoogleNet Modeli

```
        MaxPool2d-60          [-1, 528, 15, 15]               0
           Conv2d-61          [-1, 128, 15, 15]          67,712
        Inception-62          [-1, 832, 15, 15]               0
        MaxPool2d-63            [-1, 832, 8, 8]               0
           Conv2d-64            [-1, 256, 8, 8]         213,248
           Conv2d-65            [-1, 160, 8, 8]         133,280
           Conv2d-66            [-1, 320, 8, 8]         461,120
           Conv2d-67             [-1, 32, 8, 8]          26,656
           Conv2d-68            [-1, 128, 8, 8]         102,528
        MaxPool2d-69            [-1, 832, 8, 8]               0
           Conv2d-70            [-1, 128, 8, 8]         106,624
        Inception-71            [-1, 832, 8, 8]               0
           Conv2d-72            [-1, 384, 8, 8]         319,872
           Conv2d-73            [-1, 192, 8, 8]         159,936
           Conv2d-74            [-1, 384, 8, 8]         663,936
           Conv2d-75             [-1, 48, 8, 8]          39,984
           Conv2d-76            [-1, 128, 8, 8]         153,728
        MaxPool2d-77            [-1, 832, 8, 8]               0
           Conv2d-78            [-1, 128, 8, 8]         106,624
        Inception-79           [-1, 1024, 8, 8]               0
AdaptiveAvgPool2d-80           [-1, 1024, 1, 1]               0
          Dropout-81                 [-1, 1024]               0
           Linear-82                    [-1, 5]           5,125
================================================================
Total params: 5,978,677
Trainable params: 5,978,677
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.59
Forward/backward pass size (MB): 49.28
Params size (MB): 22.81
Estimated Total Size (MB): 72.68
----------------------------------------------------------------
```

Fotoğraf 11. GoogleNet Model Özeti

1- Konvolüsyon Katmanları (Conv Layers):

- Conv1: 3 kanal giriş görüntüsü (örneğin, RGB), 64 filtre ile işlenir. Bu katman büyük bir çekirdek boyutuna (7x7) ve stride 2'ye sahiptir.
- Conv2: 1x1 konvolüsyon katmanı ile 64 filtre uygulanır.
- Conv3: 64 kanal girişten 192 kanal çıkış üreten 3x3 konvolüsyon katmanı, padding=1 ile kullanılır.

2- Max Pooling Katmanları:

- MaxPool1, MaxPool2, MaxPool3, MaxPool4: Farklı aşamalarda özellik haritasını küçültmek ve yerel özellikleri yoğunlaştırmak için 3x3 boyutunda ve stride 2 olan max pooling katmanları kullanılır.

2- Inception modülleri: Inception modülleri, çeşitli boyutlardaki filtrelerle paralel konvolüsyon katmanları kullanarak oluşturulur. Her modül, farklı boyutlardaki filtreleri (1x1, 3x3, 5x5) ve maksimum havuzlama (max pooling) katmanları bir araya getirilerek dört ana bileşenden oluşturulur. Fotoğraf 12'de Inception bloğu verilmiştir.

```python
class Inception(nn.Module):
    def __init__(self, in_channels, out1x1, out3x3_reduce, out3x3, out5x5_reduce, out5x5, out_pool):
        super(Inception, self).__init__()
        self.branch1x1 = nn.Conv2d(in_channels, out1x1, kernel_size=1)

        self.branch3x3_1 = nn.Conv2d(in_channels, out3x3_reduce, kernel_size=1)
        self.branch3x3_2 = nn.Conv2d(out3x3_reduce, out3x3, kernel_size=3, padding=1)

        self.branch5x5_1 = nn.Conv2d(in_channels, out5x5_reduce, kernel_size=1)
        self.branch5x5_2 = nn.Conv2d(out5x5_reduce, out5x5, kernel_size=5, padding=2)

        self.branch_pool = nn.Conv2d(in_channels, out_pool, kernel_size=1)
        self.pool = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)

    def forward(self, x):
        branch1x1 = F.relu(self.branch1x1(x))

        branch3x3 = F.relu(self.branch3x3_1(x))
        branch3x3 = F.relu(self.branch3x3_2(branch3x3))

        branch5x5 = F.relu(self.branch5x5_1(x))
        branch5x5 = F.relu(self.branch5x5_2(branch5x5))

        branch_pool = self.pool(x)
        branch_pool = F.relu(self.branch_pool(branch_pool))

        outputs = [branch1x1, branch3x3, branch5x5, branch_pool]
        return torch.cat(outputs, 1)
```

Fotoğraf 12. Inception Bloğu

Inception Bloğu Açıklaması

- 1x1 Konvolüsyon: Doğrudan 1x1 filtre kullanarak yapılan konvolüsyon.

- 3x3 Konvolüsyon: Önce 1x1 konvolüsyon ile kanal sayısı azaltılır, ardından 3x3 konvolüsyon uygulanır.
- 5x5 Konvolüsyon: Önce 1x1 konvolüsyon ile kanal sayısı azaltılır, ardından 5x5 konvolüsyon uygulanır.

Max Pooling: 3x3 max pooling uygulanır ve ardından 1x1 konvolüsyon ile kanal sayısı azaltılır. 1x1 konvolüsyonlar, hesaplama yükünü azaltmak ve modelin parametre sayısını düşürmek için kullanılır.

3- Adaptive Average Pooling ve Fully Connected Katman:

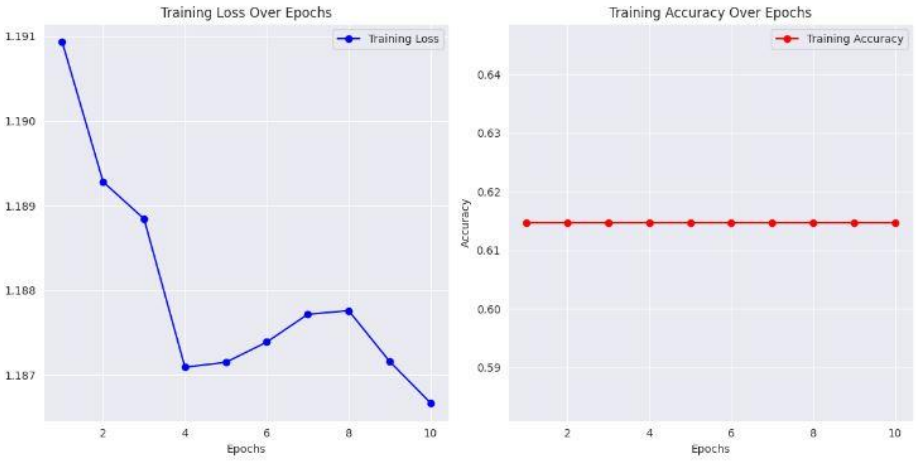
- AdaptiveAvgPool2d: Çıktı boyutunu 1x1'e sabitleyen ortalama havuzlama katmanı.
- Dropout: Aşırı öğrenmeyi (overfitting) önlemek için dropout katmanı.
- Fully Connected Layer: 1024 giriş özellikten (1x1 boyutunda), sınıf sayısı kadar (bu örnekte 5) çıkış üreten tam bağlantılı katman.

Bu bölümün hazırlanma sürecinde oldukça zorlandım. [9] [10] [11] kaynaklarından faydalandım. Benzer kod bloklarını burada görebilirsiniz. Layer değerlerinde daha çok ChatGPT'den yararlandım, hataların giderilmesi konusunda yardımcı oldu.

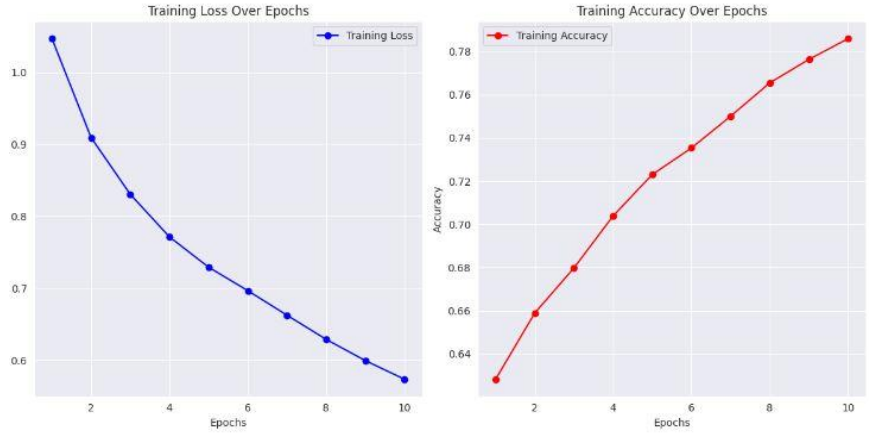
4. Modellerin Grafik Çıktıları

Fotoğraf 1, 2, 3 ve 4'te modellerin Epoch-Loss grafiği ve Epoch-TrainingAccuracy grafikleri verilmiştir.

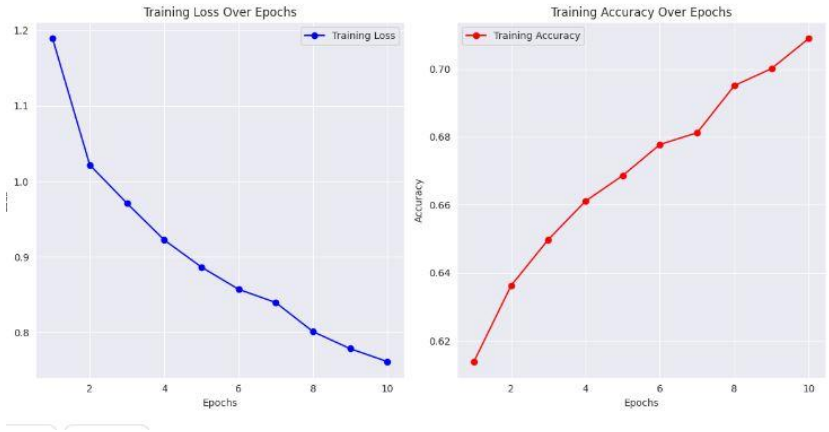
Fotoğraf 13. Alex.Net Eğitim Grafikleri



Fotoğraf 14. VGG.Net Eğitim Grafikleri



Fotoğraf 15. VGG.Net Eğitim Grafikleri



Fotoğraf 16. Google.Net Eğitim Grafikleri

5. Doğruluk, Kesinlik, Geri Çağırma, F-ölçüsüne göre tablo

Fotoğraf 17'de mimarilerin değerlendirme ölçütlerine göre hazırlanmış tablo bulunmaktadır.

	Alex.Net	VGG.Net	Res.Net	Google.Net
Doğruluk(Accuracy)	0.682	0.615	0.639	0.710
Kesinlik (Precision)	0.643	0.378	0.725	0.673
Geri Çağırma (Recall)	0.682	0.615	0.639	0.710
F1-Skor	0.612	0.468	0.638	0.668

Fotoğraf 17. Değerlendirme

Neler Geliştirilebilir?

- Epoch sayısı artırılarak Res.net ve Google.Net için daha yüksek değerler elde edinilebilir.
- Batch parametresi 32'ye düşürülüp daha hızlı train yapılabilirdi.
- VGG.Net için mimarisi kernel-size gibi layerlar değiştirilebilir. Daha güzel sonuçlar elde etmiştim daha önceden lakin layerlarda değişiklikler yaparken yanlışlıkla kaybettim.

6. Parametre değerlerini sununuz.

Alex.Net, VGG.Net, Res.Net, Google.Net için

Learning Rate: 0.01

Batch size: 64

Epoch Sayısı: 10

Optimizasyon Algoritması: Adam

[1] notebookb2cd1cc520 (kaggle.com)

[2] F1 > 0.9 | Crop Diseases Classification | PyTorch (kaggle.com)

[3] Crop Disease Pytorch Lightning (No ImageFolder) (kaggle.com)

[4] https://github.com/Lornatang/AlexNet-PyTorch/blob/main/model.py

[5]https://frightera.medium.com/alexnet-vggnet-inception-ve-resnet-nedir-bddc7482918b

[6] https://github.com/Lornatang/VGG-PyTorch/blob/main/model.py

[7] https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

[8] https://arxiv.org/abs/1512.03385

[9] https://ai.plainenglish.io/googlenet-inceptionv1-with-tensorflow-9e7f3a161e87

[10] https://github.com/walsvid/GoogLeNet-TensorFlow/blob/master/lib/googlenet/inception_v1.py

[11] https://www.youtube.com/watch?v=XxBrvCb6pqc

[12] https://openai.com/index/gpt-4/