1.Model Seçimi

CNN modeli

Convolutional Layer — Özellikleri saptamak için kullanılır

Non-Linearity Layer — Sisteme doğrusal olmayanlığın (non-linearity) tanıtılması

Pooling (Downsampling) Layer — Ağırlık sayısını azaltır ve uygunluğu kontrol eder

Flattening Layer — Klasik Sinir Ağı için verileri hazırlar

Fully-Connected Layer — Sınıflamada kullanılan Standart Sinir Ağı

Transfer Öğrenme Modelleri

- **InceptionV3 modeli**

Giriş katmanı (Input layer): Giriş verilerini kabul eden bir tensör katmanı.

Evrişim katmanları (Convolutional layers): Görüntü özelliklerini çıkarmak için kullanılan çeşitli evrişim katmanlarından oluşur.

Inception modülleri: Birbirine paralel olarak çalışan farklı boyutlarda evrişimlerin birleştirildiği bir yapıdır. Inception modülleri hem hesaplama maliyetini azaltmak hem de daha iyi performans elde etmek için kullanılır.

Aktivasyon fonksiyonu (Activation function) katmanları: Her evrişim katmanının sonunda kullanılan ve modelin öğrenme yeteneğini artıran bir katmandır. Inceptionv3 modelinde genellikle ReLU aktivasyon fonksiyonu kullanılır.

MaxPooling katmanları: Özellik haritalarını küçültmek için kullanılır ve aşırı öğrenmeyi önlemeye yardımcı olur.

Global Average Pooling katmanı: Özellik haritalarını düzleştirir ve son çıktıyı oluşturmak için kullanılır.

Tam bağlı (Fully connected) katmanlar: Son çıktıyı oluşturmak için kullanılır. Inceptionv3 modelinde, tam bağlı katmanların yanı sıra çıkış katmanı olarak bir softmax katmanı kullanılır.

- **Xception Modeli**

Giriş katmanı (Input layer): Giriş verilerini kabul eden bir tensör katmanıdır.

Evrişim katmanları (Convolutional layers): Görüntü özelliklerini çıkarmak için kullanılan çeşitli evrişim katmanlarından oluşur. Bu katmanlar, derin öğrenme modellerinde genellikle özellik haritalarını oluşturmak için kullanılır.

Seperable Convolution katmanları: Normal konvolüsyon işlemi yerine adım adım özellik çıkarma işlemi uygulayan bir tür evrişim katmanıdır. Daha az parametre kullanarak daha az hesaplama gücüyle daha iyi sonuçlar elde etmek için kullanılır.

Aktivasyon fonksiyonu (Activation function) katmanları: Her evrişim katmanının sonunda kullanılan ve modelin öğrenme yeteneğini artıran bir katmandır. Xception modelinde genellikle ReLU aktivasyon fonksiyonu kullanılır.

MaxPooling katmanları: Özellik haritalarını küçültmek için kullanılır ve aşırı öğrenmeyi önlemeye yardımcı olur.

Global Average Pooling katmanı: Özellik haritalarını düzleştirir ve son çıktıyı oluşturmak için kullanılır.

Tam bağlı (Fully connected) katmanlar: Son çıktıyı oluşturmak için kullanılır. Xception modelinde, tam bağlı katmanların yanı sıra çıkış katmanı olarak bir softmax katmanı kullanılır.

2.Veri Seti ve Konu Belirleme

Dermnet veri setini kullanmak üzere seçtim.

Bu veri seti;

Toplam görüntü sayısı 19.500 civarında olup, bunların yaklaşık 15.500'ü eğitim setinde ve geri kalanı test setinde bölünmüştür. 23 kategoriye ayrılmış olan veri seti her bir kategori için test ve eğitim verileri barındırmaktadır. Veri setinde bulunan fotoğraf verileri JPEG formatındadır. Fotoğrafların çözünürlükleri görüntüden görüntüye ve kategoriden kategoriye değişmektedir. Seçmiş olduğum hastalıklar için görüntüler 720x480 veya 720x 480'in aşağısı şeklindedir, ancak genel olarak bunlar çok yüksek çözünürlüklü görüntüler değildir. Bunun yanı sıra bazı görüntülerde insan vücudu kıllı olduğu için kıl bulunmakta ve Lyme hastalığının nedenlerinden biri kene sokması olduğu için fotoğrafların bazılarında semptomlarla beraber kene görüntüsü bulunmaktadır. Normal şartlarda filtreleme yöntemi ile kıllar ekarte edilebilmektedir. Fakat proje için çok vakit olmadığından kaynaklı olduğu gibi bırakılmıştır. Dermnet veri setinin içerisinden birbirine görüntü olarak çok benzeyen fakat birbiri ile alakası olmayan Lupus ve Lyme hastalıklarının bulundukları veri kümelerini seçtim böylece derin öğrenme yolu ile bunların farklını göz dışında tayin edilebilmesini sağlamak amacım oldu.

Görüntülere erişim, çevrimiçi tıp eğitimi sağlamak amacıyla oluşturulmuş en büyük çevrimiçi dermatoloji kaynağı olan kamu portalı Dermnet 'ten (http://www.dermnet.com/) alındı. Bu veri setindeki her bir fotoğraf uzman dermatologlar tarafından onaylanmış olup doğruluğu kanıtlanmış verilerdir ve veri seti olmasının dışında Dermnet sitesine girildiği zaman ayrı ayrı olarak hastalıklara ait fotoğraflar ve bilgi edinmek mümkündür. Ayrıca bu veri seti Kaggle'da da mevcuttur.

Class Label	Abbreviation	Super-Class Name	Np. of Images	No. of Sub-Classes
0	ACROS	Acne and Rosacea	912	21
1	AKBCC	Actinic Keratosis, Basal Cell Carcinoma, and other Malignant Lesions	1437	60
2	ATO	Atopic Dermatitis	807	11
3	BUL	Bullous Diseases	561	12
4	CEL	Cellulitis, Impetigo, and other Bacterial Infections	361	25
5	ECZ	Eczema Photos	1950	47
6	WXA	Exanthems and Drug Eruptions	497	18
7	ALO	Alopecia and other Hair Diseases	195	23
8	HER	Herpes, Genetal Warts and other STIs	554	15
9	PIG	Pigmentation Disorder	711	32
10	LUPUS	Lupus and other Connective Tissue diseases	517	20
11	MEL	Melanoma and Melanocytic Nevi	635	15
12	NAIL	Nail Fungus and other Nail Disease	1541	48
13	POI	Poison Ivy and other Contact Dermatitis	373	12
14	PSO	Psoriasis Lichen Planus and related diseases	2112	39
15	SCA	Scabies Lyme Disease and other Infestations and Bites	611	25
16	SEB	Seborrheic Keratoses and other Benign Tumors	2397	50
17	SYS	Systemic Disease	816	43
18	TIN	Tinea Candidiasis and other Fungal Infections	1871	36
19	URT	Urticaria	265	9
20	VASCT	Vascular Tumors	603	18
21	VASCP	Vasculitis	569	17
22	WARTS	Common Warts, Mollusca Contagiosa and other	1549	26
		Total	**21844**	**622**

Şekil 1: Dermnet veri setinin içeriği

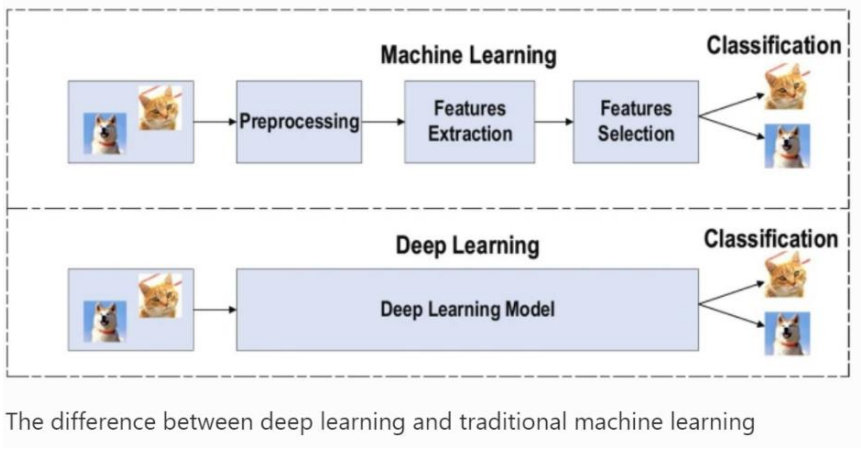
Belirlediğim konumda Dermnet veri setini alarak modeli kendi belirlediğim parametrelere göre düzenledim ve eğittim. Sonrasında başarı değerlerimi belirttim, bulduğum bazı görüntüleri eğittiğim modelim üzerinde test ettim ve en son olarak görselleştirdim.

3.Derin Öğrenme Modeli / Modellerinin Oluşturulması

Model olarak CNN ve Xception ve CNN ile kullanılan InceptionV3 modellerini seçtim. CNN evrişimli sinir ağı temelli iken Xception ve InceptionV3 modelleri de aynı şekilde kullanılmaktadır.

CNN

Bir CNN'nin mimarisi, insan beyninin bağlantı modeline benzer. Beynin milyarlarca nörondan oluşması gibi, CNN'lerin de belirli bir şekilde düzenlenmiş nöronları vardır. Basit bir CNN kurmanın yolu, birkaç Convolutional Katmanı arka arkaya koymak ve her birinden sonra ReLU katmanı eklemektir ve bundan sonra Pooling katmanı(ları) ve Flattening katmanı eklenmelidir. En son olarak ReLu katmanı kadar Fully-Connceted katmanı eklenir. Günümüzde, CNN'ler yüz tanıma, görüntü arama ve düzenleme, artırılmış gerçeklik ve daha fazlası gibi birçok bilgisayarla görme uygulamasında kullanılmaktadır.

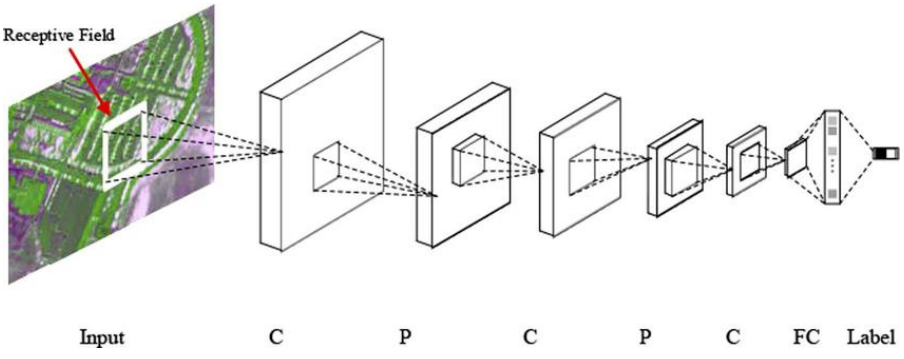


The difference between deep learning and traditional machine learning

Şekil 2: Makine öğrenimi ile deri öğrenmenin arasındaki temel fark.

Convolutional Layer

CNN'nin ana yapı taşıdır ve resmin özelliklerini algılamaktan sorumludur. Bu katman, görüntüdeki düşük ve yüksek seviyeli özellikleri çıkarmak için resme bazı fitreler uygular. Filtre ilk olarak görüntünün sol üst köşesine konumlandırılır. Burada, iki matris arasında (resim ve filtre) indisler birbiri ile çarpılır ve tüm sonuçlar toplanır, daha sonra sonucu çıktı matrisine depolanır. Ardından, bu filtreyi sağa 1 piksel ("basamak" olarak da bilinir) kadar hareket ettirip işlemi tekrarlanır. 1. Satır bittikten sonra 2 satıra geçilir ve işlemler tekrarlanır. Tüm işlemler bittikten sonra çıktı matrisi oluşturulur. Genellikle, birden çok özelliği tespit etmek için birden fazla filtre kullanılır, yani bir CNN ağında birden fazla konvolüsyonel (Convolutional) katman bulunmaktadır. Kullanılan her bir filtre başka bir özellik türünü algılayan feature map'leri oluşturur.



Şekil 3: Convolutional katmanda filtreleme işleminin görselleştirilmiş hali. Solda ilk olarak ana resim/fotoğrafa filtre uygulanmaktadır.

Stride

Stride, filtrenin giriş görüntüsünün etrafında nasıl evrildiğini denetler. Kısacası filtrenin ana görsel boyunca kaç pikselde kayacağını belirler.

Padding

CNN'nin ilk aşamalarında, ilk filtreleri uygularken, diğer Convolutional katmanlar için mümkün olduğunca çok bilgiyi koruması gerektiğinden padding kullanılmaktadır. Padding, resmin boyutunu korumak için haritaya boş alanlarına sıfır değerler katar. Bunun yapılmasındaki amaç katmanlardan geçtikten sonra görsellerin boyutlarının küçülmesinden kaynaklı orijinal boyutlara yaklaştırmaktır.

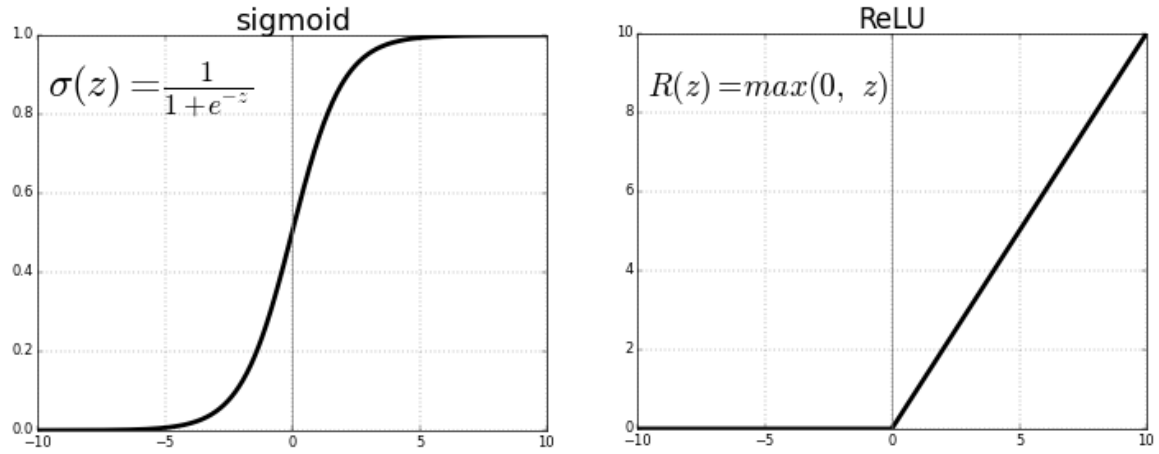
Non-linearity

Tüm Convolutional katmanlarından sonra genellikle Non-Linearity(doğrusal olmayan) katmanı gelir.

Activation Layer

Bu katman aktivasyon katmanı (Activation Layer) olarak adlandırılır çünkü aktivasyon fonksiyonlarından birini kullanılır. Geçmişte, sigmoid ve tahn gibi doğrusal olmayan fonksiyonlar kullanıldı, ancak Sinir Ağı eğitiminin hızı konusunda en iyi sonucu Rectifier (ReLu) fonksiyonu verdiği için artık bu fonksiyon kullanılmaya başlamıştır.

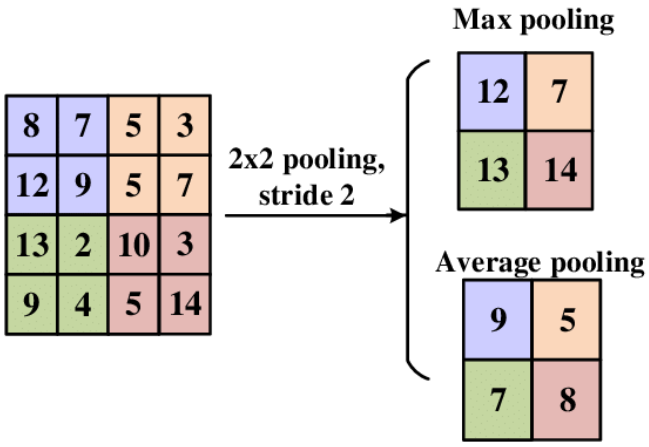
ReLu Fonksiyonu f (x) = max (0, x) fonksiyonunu kullanmaktadır. ReLu'daki ana amaç negatif değerlerden kurtulmaktır.



Şekil 4: Sol tarafta doğrusal olmayan sigmoid fonksiyonu ve matematiksel gösterimi, sağ tarafta doğrusal olan ReLu fonksiyonu ve matematiksel gösterimi

Pooling Layer

Bu katmanın görevi, gösterimin kayma boyutunu ve ağ içindeki parametreleri ve hesaplama sayısını azaltmak içindir. Bu sayede ağdaki uyumsuzluk ve karmaşıklık kontrol edilmiş olur. Ayrıca oldukça kullanılan bir katmandır. Bu katmanı kullanmayan kişiler bunun yerine Convolutional katmanında daha büyük Stride işlemini yapmayı tercih ederler. Evrişimli katman gibi, havuzlama katmanı da giriş görüntüsü boyunca bir çekirdek veya filtre tarar. Ancak evrişim katmanından farklı olarak, havuzlama katmanı girdideki parametre sayısını azaltır ve ayrıca bir miktar bilgi kaybına neden olur. Bu katmada max pooling uyguluyorsanız filtrenin kapsadığı alandaki en büyük değeri, average pooling uyguluyorsanız ise filtredeki değerlerin ortalamasını almaktadır. Bunun sonucunda boyut azalır ve önemli özellikler elimizde kalmış olur.



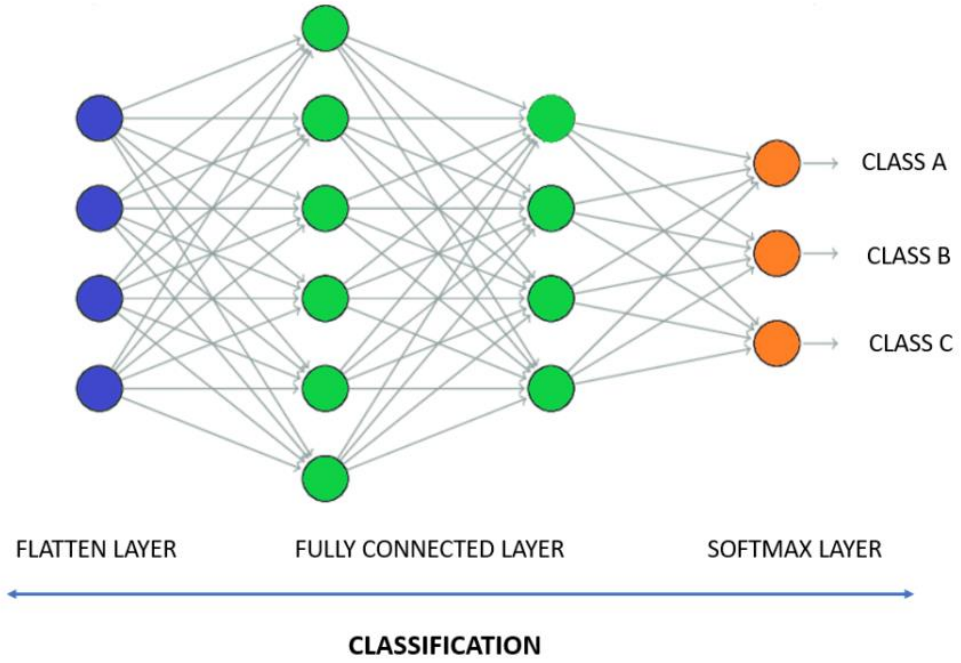
Şekil 5: Maxpooling ve avaragepooling'in çalışma prensibinin görselleştirilmesi

Flattening Layer

Bu katmanın görevi, son ve en önemli katman olan Fully Connected Layer'ın girişindeki verileri hazırlamaktır. Genel olarak, sinir ağları, giriş verilerini tek boyutlu bir diziden alır. Bu sinir ağındaki veriler ise Convolutional ve Pooling katmanından gelen matrixlerin tek boyutlu diziye çevrilmiş halidir.

Fully-Connected Layer

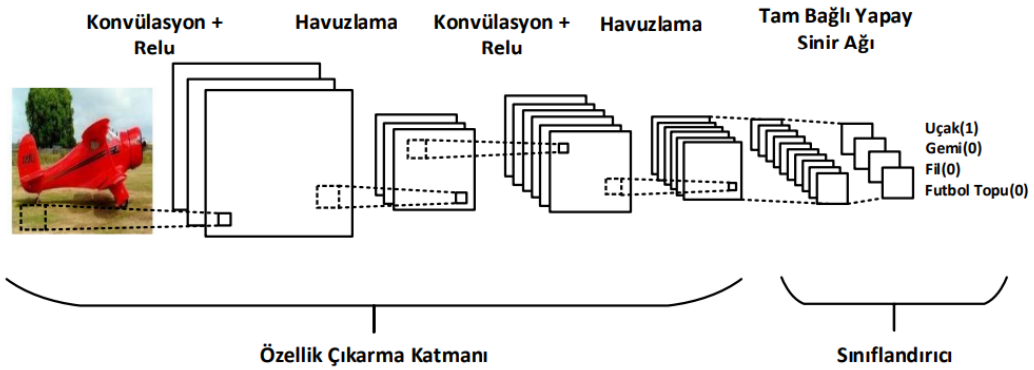
Bu katman son ve en önemli katmanıdır. Verileri Flattening işleminden alır ve sinir ağı yoluyla öğrenme işlemini geçekleştirir. FC katmanı, önceki katmanlarda çıkarılan özelliklere dayalı olarak CNN'de görüntü sınıflandırmasının gerçekleştiği yerdir. Burada tam bağlı, bir katmandaki tüm girişlerin veya düğümlerin bir sonraki katmanın her aktivasyon birimine veya düğümüne bağlı olduğu anlamına gelir.



Şekil 6: Flatten layer, FC ve softmax layerin çalışma prensibinin gösterimi

Derin öğrenme için CNN kullanmanın faydaları

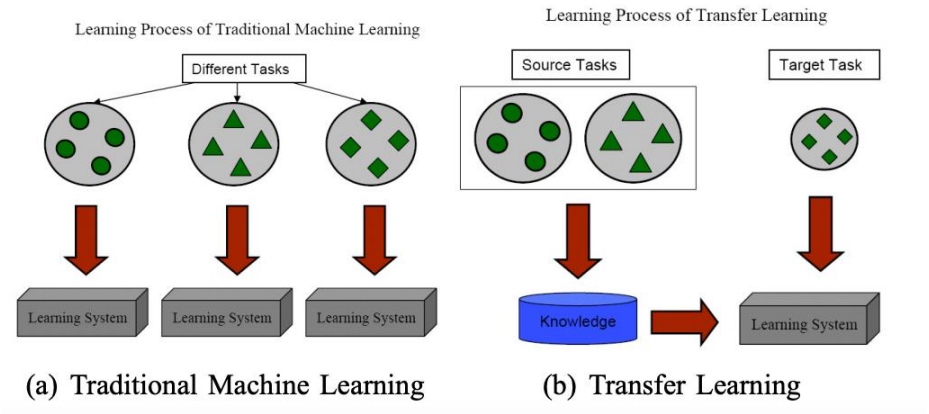
1. Tek katmanlı bir ağa kıyasla, çok katmanlı bir ağ daha doğru sonuçlar vermesi.
2. Görüntü tanıma, görüntü sınıflandırma ve bilgisayarla görme (CV) uygulamaları için CNN'ler özellikle yararlıdır çünkü özellikle çok fazla veri söz konusu olduğunda son derece doğru sonuçlar sağlarlar.
3. CNN'ler yeni tanıma görevleri için yeniden eğitilebilir ve önceden var olan ağlar üzerine inşa edilebilir.



Şekil 7: Basit bir CNN mimarisinin işleme biçimi

Transfer Öğrenme

Transfer öğrenme bir sorunu çözerken elde ettiği bilgileri saklayıp, başka bir sorunla karşılaştığında o bilgileri kullanır ve derin öğrenmede sıklıkla kullanılan bir araştırma yöntemidir. Bu sayede önceki bilgileri kullanılarak daha az eğitim verisi ile daha yüksek başarı gösteren ve daha hızlı öğrenilen modeller elde edilmektedir. Transfer öğrenmede, önce temel veri kümesi ve görevinde temel bir ağ hazırlanır ve daha sonra, öğrenilen özellikleri yeniden tasarlanır veya bunları, bir hedef veri kümesi ve görevi üzerinde eğitilmek üzere ikinci bir hedef ağa aktarılır. Transfer öğrenmede çok büyük veri setleri ile eğitilmiş CNN modellerinin ilk katmanlarındaki ağırlıkların sabit tutularak sadece son katmanının (full connected katmanın) yeniden çok küçük veri setleri ile eğitilmesini sağlar.



Şekil 8: Öğrenme süreçlerinin gösterimi a) geleneksel makine öğrenmesi sürecinin gösterilmesi b) transfer öğrenme ile öğrenme sürecinin gösterilmesi

Aktarma işlemi için önemli detaylar;

- **Neyin aktarılacağı:** Kaynak veriden hedefi hangi aktar bilginin hedef görevin başarıya ulaşacağına cevap aranan bölüm
- **Ne zaman aktarılacağı:** Bu durum verilerinin hangi bölümlerinin aktarılacağı da hangi nesnelerin aktarılmaması gerektiğini belirlemektedir. Hedef verileri ve kaynak verileri birbirinden farklı olduğu zaman tahmin değeri düşmektedir.
- **Nasıl aktarılacağı:** Burada muhafaza/öğrenmenin postası için hangi yöntemlerin yoluna karar verilir.

Avantajları

- **Daha Hızlı Eğitim Süresi:** Hatırlamış olduğu modellerdeki ağırlıklar birçok kullanıcıyı barındırmaktadır. Dolayısıyla bu bilgileri kullanıp ince ayar (ince ayar) yaparak yeni modeli daha hızlı okur.

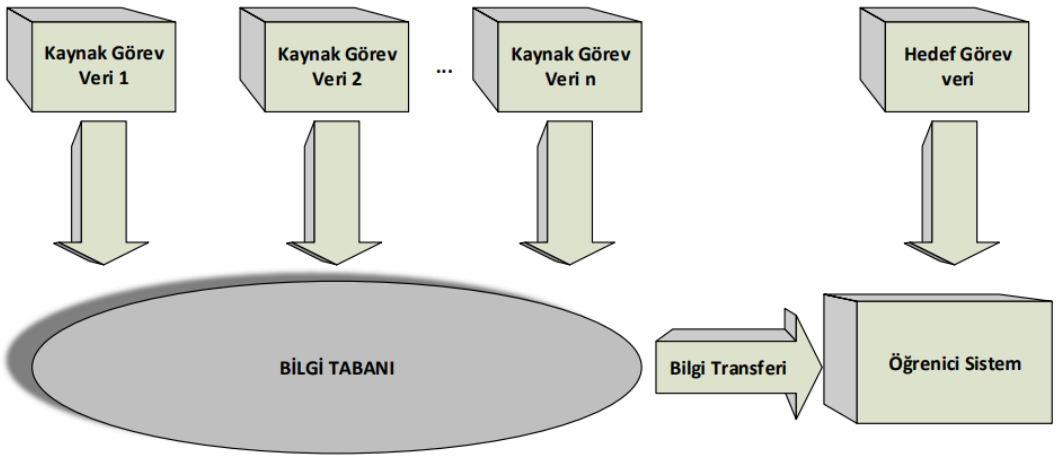
- **Daha az veri:** Standart olarak öğretilmiş modellere ince ayar yapılması daha az veri kullanarak yüksek performanslar elde edilmesini sağlar. IMAGENET gibi veri kümelerini kullanarak öğrenmeye destek olmaktadır.

- **Daha iyi performans:** Modellere yeni tam bağlantılı katman(lar) eklenerek yapılan basit bir işlemin başarısını iyileştirdiği görülmüştür

Bu tür modellerin üç ana örneği şunlardır:

- Oxford VGG Modeli (VGG16, VGG19 vb.)
- Google Inception Modeli (InceptionV, Xception vb.)
- Microsoft ResNet Modeli (Resnet50)

Olumsuz tutma: Öğrenme her zaman performansını artırmayabilir hatta bazen mevcut performansın düşmesine de sebep olabilir. Bu durum olumsuz tutma olarak adlandırılır. Burada modelin başarısının düşmesinin çeşitli nedenleri olabilir. Kaynak model ile hedef model grupları birbirinden çok farklı olabilir veya aktarılan kaynaklar ve hedef modeller arasındaki ilişki tam olarak ifade edilemiyor olabilir.



Şekil 2. Transfer öğrenme çalışma prensibi [11].

Şekil 9: Transfer öğrenme çalışma prensibi

4. Deneysel Çalışma

CNN modeli

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

- Gerekli kütüphaneler "os", "cv2", "numpy", "matplotlib.pyplot", "tensorflow.keras.models", "tensorflow.keras.layers" ve "tensorflow.keras.preprocessing.image" import ettim. "Sequential" sınıfını, bir sıralı model oluşturmak için kullandım bu sınıf katmanları ardışık olarak birbirine bağlayarak oluşturmaktadır. "Dense" sınıfı, tam bağlantılı bir katman eklemek için kullandım. girdiler üzerinde bir dizi evrişim işlemi gerçekleştirmek için "Conv2D" sınıfını kullandım "MaxPooling2D" sınıfını kullanarak modelin daha az sayıda parametreye sahip olmasını sağlayarak hesaplama yükünü azaltmasını sağladım. "ImageDataGenerator" sınıfını ise, veri artırma işlemi için kullandım. Bu sayede eğitim verilerinin çeşitliliğini artırmış oldum.

```
[ ]  # Veri klasörlerinin yolları
     train_dir = '/content/drive/MyDrive/trans/train/train'
     val_dir = '/content/drive/MyDrive/trans/train/valid'
     test_dir = '/content/drive/MyDrive/trans/test'
```

- Google Colab üzerinden işlemlerimi ve analizlerimi yapacağım için önce verilerimi klasör haline getirip Google Drive'a yükledim sonrasında ise yukarıdaki kodla okuttum. "Train/train" eğitim verilerimi, "train/valid" validasyon verilerimi ve "test" ise test verilerimin klasörlerinin adıdır.

```
# Veri boyutu
img_size = 150

# Veri artırma (data augmentation) yapılandırması
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

- Modelimde kullanılacak veri boyutunu img_size değişkeni ile ayarladım. Bu modelde veri boyutunu 150x150 olarak seçtim. veri artırma işlemi için "ImageDataGenerator" sınıfını kullandım. "rescale" parametresi ile verilerin 0-255 aralığından 0-1 aralığına ölçeklendirilmesini sağladım. Veri arttırma işlemini train_datagen, val_datagen ve test_datagen olarak atayıp bütün verilerimde uyguladım.

```
# Eğitim verileri yükleme
train_generator = train_datagen.flow_from_directory(train_dir, target_size=(img_size, img_size),
                                                    batch_size=32, class_mode='binary')

# Doğrulama verilerini yükleme
val_generator = val_datagen.flow_from_directory(val_dir, target_size=(img_size, img_size),
                                                batch_size=32, class_mode='binary')

# Test verilerini yükleme
test_generator = test_datagen.flow_from_directory(test_dir, target_size=(img_size, img_size),
                                                  batch_size=32, class_mode='binary')
```

- Doğrulama ve test veri setlerini yüklemek için kullanılan "flow_from_directory" fonksiyonunu kullandım. Bu fonksiyon belirtilen dizinlerdeki görüntüleri yükler ve önceden yapılandırılmış veri artırma işlemi yapılandırmalarına göre görüntüleri işler. "batch_size" parametresi ise bir seferde işlenecek görüntü sayısını belirtmektedir. Size'ı 32 olarak belirledim bu yüzden her seferinde 32 görüntü işlenmesini sağladım. Eğitim doğrulama ve test verilerimi yükledim eğitim verimde 811 fotoğraf ve 2 sınıf, doğrulama verilerimde 40 fotoğraf ve 2 sınıf, test verilerimde ise 213 fotoğraf ve 2 sınıf bulunmaktadır. Bu sınıflar Dermnet veri setinden seçmiş olduğum iki hastalığı göstermektedir.

```
# Model oluşturma
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(img_size, img_size, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))
```

- Modeli oluşturdum yukarıda belirttiğim gibi sequential modelini seçtim ve aktivasyon fonksiyonu olarak en çok kullanılan ReLu fonksiyonunu seçtim. Buraya detaylı açıklama gir "Sequential()" fonksiyonu, sıralı bir model oluşturmak için kullandım sonrasında ise "Conv2D" katmanlarını kullanarak görüntülerin özelliklerini çıkarttım. Filtre boyutunu 3x3 olarak ayarladım. Bu modelde toplamda iki tande "Conv2D" katmanı kullandım. "MaxPooling2D" katmanlarında ise Conv2D katmanları tarafından çıkarılan özellik haritalarını özetlemek için kullandım bu modelde iki tane "MaxPooling2D" katmanı kullandım. "Flatten" katmanı, kullanarak çıktıları tek boyutlu bir vektöre dönüştürdüm. "Dense" katmanlarında aktivasyon fonksiyonu olarak "ReLu" ve "sigmoid" fonksiyonlarını kullandım.

```
[ ]  # Modeli derleme
     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
     # Modeli eğitme
     history = model.fit(train_generator, steps_per_epoch=len(train_generator), epochs=30,
                         validation_data=val_generator, validation_steps=len(val_generator))
```

- Modeli derleme aşamasında hangi optimizasyon ve metriklerin kullanılacağını ayarladım. Bu model için "adam" optimizasyon algoritması, kayıp değerleri görmek için "loss" parametresi ve modelin doğruluğunu/başarısını ölçmek için "accuracy" metriği kullandım.
- Modelimi derledikten sonra modelimi eğittim. Modelin eğitimi için model.fit'i kullandım. İlk çalışmamda devir sayısını 10 olarak almıştım fakat istediğim sonuçlar çıkmayınca devir sayısını 30'a yükselterek tekrardan işlem yaptım. Devirden kastım modelin kaç defa eğitileceğidir bu modelimde 30 defa eğittim. History değişkeni olarak modelimi atadım.

```
[ ]  # Modeli test edme
     test_loss, test_acc = model.evaluate(test_generator, steps=len(test_generator))
     print('Test accuracy:', test_acc)

     7/7 [==============================] - 5s 682ms/step - loss: 0.8087 - accuracy: 0.7183
     Test accuracy: 0.7183098793029785
```

- Eğitilmiş modelimin performansını göstermek için modeli test ettim. Çalışmamın en başında test verilerimi Google Drive'dan okutmuştum o yüzden tekrar yazılmasına gerek yoktu. "model.evaluate" metodu ile modelimin test verilerim üzerindeki performansını ölçtüm .Test doğruluk değerim 10 epokta 69 çıkmıştı 30 epok ile yaptığım zaman 71 olarak gözüktü kayıp değeri ise 0.8087 çıktı bu oldukça yüksek bir sayı.

```python
# Eğitim ve doğrulama doğruluğunu ve kaybını gösterme
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

- Eğitim ve doğrulama verilerimin doğruluğunu göstermek için grafik çizdirdim. Bunun için modelimi history olarak kaydetmiştim ve bu model üzerinden değerleri kaydettim. Plt.show() fonksiyonu ile grafiği çizdirdim. Yukarıda yaptığımın bir benzeri şekilde doğrulama verilerinin ve test verilerinin kayıp ve tahmin değerlerini hesaplatarak grafik oluşturttum. Bunların kodlarını yukarıdaki kodla çok benzer oldukları için koymuyorum ama iyileştirme ve karşılaştırma kısmında görüntü olarak mevcutlar

```python
from sklearn.metrics import classification_report, confusion_matrix

# Test veri seti üzerinde modelin tahminlerini alma
y_pred = model.predict(test_generator)

# Sınıflandırma raporunu hesaplama
class_names = list(test_generator.class_indices.keys())
report = classification_report(test_generator.classes, y_pred.argmax(axis=-1), target_names=class_names)

# Karışıklık matrisini hesaplama
cm = confusion_matrix(test_generator.classes, y_pred.argmax(axis=-1))

# Sınıflandırma raporunu ekrana yazdırma
print(report)

# Karışıklık matrisini görselleştirme
import seaborn as sns
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```

- Son olarak modelin test veri seti üzerindeki performansını değerlendirmek için sınıflandırma raporu ve karışıklık (hata) matrisi oluşturdum. Karışıklık matrisi, gerçek sınıfların ve tahmin edilen sınıfların birbirleriyle nasıl karşılaştırıldığını göstermektedir bunun için confusion_matrix fonksiyonunu kullandım.

CNN ve INCEPTIONV3 modeli

```
%matplotlib inline
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

- Öğrenmenin ilk aşamasında kullanılacak kütüphaneleri import ettim fakat daha sonraki aşamalarda kullanacağım başka kütüphaneleri başka kod satırlarında import ettim. CNN modelinde bu kütüphanelerin işlevlerini yazdığım için buraya yazmıyorum.

```
[ ]  BATCH_SIZE = 64
     IMAGE_SIZE = 300
     train_path = "/content/drive/MyDrive/pro/train/"
     test_path = "/content/drive/MyDrive/pro/test"
```

- CNN modelinde olduğu gibi işlemlerimi Google Colab üzerinden yaptığım için Google Drive'da bulunan test ve eğitim verilerimi okuttum. CNN 'den farklı olarak bu çalışmada validasyon verilerini test verilerinden yaptığım için farklı bir klasör yolu izleyerek yaptım.

```
[ ]  def train_val_generators(TRAINING_DIR, VALIDATION_DIR,IMAGE_SIZE,BATCH_SIZE):
       train_datagen = ImageDataGenerator(rescale=(1./255),
                                          shear_range=0.2,
                                          zoom_range=0.3,
                                          width_shift_range=0.2,
                                          height_shift_range=0.2,
                                          brightness_range=[0.2,1.2],
                                          rotation_range=0.2,
                                          horizontal_flip=True)

       train_generator = train_datagen.flow_from_directory(directory=TRAINING_DIR,
                                                           batch_size=BATCH_SIZE,
                                                           class_mode='categorical',
                                                           target_size=(IMAGE_SIZE, IMAGE_SIZE))

       test_datagen = ImageDataGenerator(rescale=1./255)


       test_generator = test_datagen.flow_from_directory(directory=VALIDATION_DIR,
                                                         batch_size=BATCH_SIZE,
                                                         class_mode='categorical',
                                                         target_size=(IMAGE_SIZE, IMAGE_SIZE))

       return train_generator, test_generator
```

- Modelim için kullanılacak eğitim ve doğrulama veri setleri oluşturdum. ImageDataGenerator sınıfını kullanarak veri artırmayı sağladım. flow_from_directory() yöntemini kullanarak veri setlerini yükledim ve döndürme işlemi sağladım.

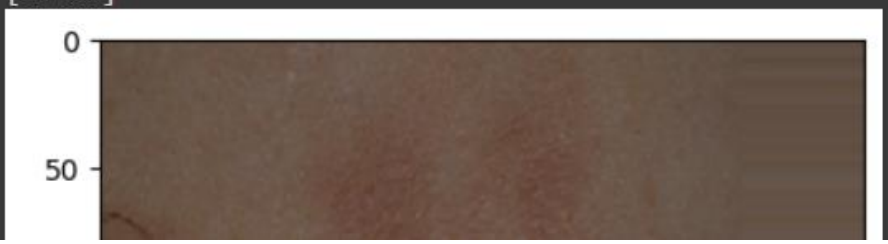
- Sonuç olarak, görüntü sınıflandırma modelinin eğitiminde kullanılacak verileri sağlamış oldum.

```
[ ]  train_generator,test_generator=train_val_generators(train_path, test_path,IMAGE_SIZE,BATCH_SIZE)
```

- Test ve eğitim verisinde kaç tane sınıf ve fotoğraf olduğunu gösterdim. Eğitim verimde 851 fotoğraf ve 2 sınıf, test verimde ise 213 fotoğraf ve iki sınıf bulunmaktadır.

```
[ ]  for _ in range(5):
         img, label = train_generator.next()
         print(img.shape)
         print(label[0])
         plt.imshow(img[0])
         plt.show()
```

```
(64, 300, 300, 3)
[0. 1.]
```



- Verilerimin düzgün okunup okunmadığını anlamak için mini gruplar halinde gösterdim burada beş tanesini görselleştirdim.

```
[ ]  class_names = train_generator.class_indices
     class_names
```

```
{'Lupus and other Connective Tissue diseases': 0,
 'Scabies Lyme Disease and other Infestations and Bites': 1}
```

- Verilerimin sınıflarının isimlerini yazdırdım. Çalışmamda iki tane sınıf bulunmaktadır. Birinci sınıf olan lupus hastalığı 0, ikinci sınıf olan Lyme hastalığı ise 1 olarak sınıflandırılmıştır.

```
[ ]  NUMBER_OF_CLASSES = len(class_names)
     print(f'number of classes : {NUMBER_OF_CLASSES}')
```

```
number of classes : 2
```

```
[ ]  from tensorflow.keras.applications.inception_v3 import InceptionV3
     base_model = InceptionV3(input_shape = (IMAGE_SIZE, IMAGE_SIZE, 3), include_top = False, weights = 'imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 1s 0us/step
```

- İki tane sınıf olduğunu gösterdikten sonra inceptionv3 modelimi çalıştırmak için import ettim. InceptionV3 modeli, özellik çıkarma için kullanılabilecek bir evrişimli sinir ağıdır. Bu model, ImageNet veri kümesiyle eğitilmiştir. weights parametresi, modelin hangi ağırlıkları kullanacağını belirler bu yüzden bu parametre 'imagenet' olarak ayarlandığında, model, ImageNet veri kümesinden eğitilmiş önceden eğitilmiş

ağırlıkları kullanmaktadır. Ben de modelim için imagenet kullandığımdan önceki ağırlıkları kullandı.

```
def output_of_last_layer(pre_trained_model, limit_layer):
  last_desired_layer = pre_trained_model.get_layer(limit_layer)
  print('last layer output shape: ', last_desired_layer.output_shape)
  last_output = last_desired_layer.output
  print('last layer output: ', last_output)

  return last_output
```

```
[ ]  last_output = output_of_last_layer(base_model,'mixed5')
```

- Bu kodda önceden eğitilmiş bir modelin içindeki katmanları gezinerek, her bir katmanın eğitilebilirliğini True olarak ayarladım. Ayrıca, bir önceden eğitilmiş model ve bir katman adı alarak, bu katmandan çıkan çıktıyı döndüren output_of_last_layer adlı bir fonksiyon tanımlar. Bu fonksiyon, bir tam bağlantılı katman oluşturmak için kullanılabilir. (Bu kodu bilmediğim için ChatGPT'ye sordum)

```
[ ]  from tensorflow.keras.optimizers import Adam
     from tensorflow.keras import regularizers

     x = tf.keras.layers.Flatten()(last_output)
     x = tf.keras.layers.Dense(1024, activation='relu', kernel_regularizer=regularizers.l2(0.01))(x)
     x = tf.keras.layers.BatchNormalization()(x)
     x = tf.keras.layers.Dropout(0.3)(x)
     x = tf.keras.layers.Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.01))(x)
     x = tf.keras.layers.BatchNormalization()(x)
     x = tf.keras.layers.Dropout(0.3)(x)
     x = tf.keras.layers.Dense(NUMBER_OF_CLASSES, activation='softmax')(x)

     model = tf.keras.models.Model(base_model.input, x)

     model.compile(
         optimizer = Adam(learning_rate=0.001),
         loss = 'categorical_crossentropy',
         metrics = ['accuracy']
     )
```

- Önceden eğitilmiş bir modelin sonuna, birkaç yoğun (dense) katman ekledim ve çıktı katmanı olarak da veri kümesindeki sınıf sayısına eşit bir softmax aktivasyonlu yoğun katman ekledim. Bu eklediğim yoğun katmanlar, aşırı uyumu (overfitting) azaltmak için düzenlenmesini sağladılar. Sonrasında ise, bu yeni oluşturulan modeli derleme işlemi yaptım. Optimizasyon olarak Adam optimizasyon algoritması kullandım.

```
[ ]  from tensorflow.keras.callbacks import EarlyStopping
     custom_early_stopping = EarlyStopping(
         monitor='val_loss',
         patience=10,
         min_delta=0.001,
         mode='min'
     )
```

```
[ ]  history = model.fit(
         train_generator,
         validation_data = test_generator,
         epochs = 10,
         callbacks=[custom_early_stopping]
         )
```

- Bu kodda, TensorFlow Keras kütüphanesinin EarlyStopping geri çağrısını kullanarak eğitim sırasında modelin otomatik olarak durdurulmasını sağladım. Earlystoping, belirli bir metriğin (burada val_loss, yani doğrulama kaybı) istenen bir değerine ulaşıldığında eğitimi durdurdu. Earlystopping kullanılmasından kaynaklı 20 gibi devir (epok) sayıları hemen hemen aynı sonuçları çıkardığından işlem durdu. Devir sayımı 10 tuttum başta 20 ve 30 devir sayılarını denedim fakat 14. devirde çalışmayı durdurduğu için 10 da karar kıldım. Earlystoppingi kullanmamın sebebi ise devirleri döndürürken CPU değerini aştığından kaynaklı hata almamdı bu yüzden bu şekilde bir yöntem uygulamak zorunda kaldım. Modeli history değişkenine atadım ve çalıştırdım.

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure(figsize=(6,6))

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.figure(figsize=(6,6))


plt.show()
```

- CNN modelinde de yaptığım gibi modelimin doğrulama ve eğitim verilerinin doğruluk ve kayıp değerlerini görselleştirdim.

```
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
import numpy as np

test_true=test_generator.classes
test_pred_raw = model.predict(test_generator)
test_pred = np.argmax(test_pred_raw, axis=1)

cm = confusion_matrix(test_true, test_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
fig, ax = plt.subplots(figsize=(15,15))
disp.plot(ax=ax,cmap=plt.cm.Blues)
plt.show()
```

- Son olarak modelimin sınıflandırmayı ne kadar doğru yaptığını anlamak için hata matrisi oluşturdum. Bunun için "sklearn.metrics" içerisinde bulunan "confusion_matrix" ve "confusionMatrixDisplay"'i import ettim. En son olarak hata matriksini plt.show() ile görselleştirdim.

XCEPTION modeli

```
# Gerekli kütüphaneleri içe aktarma
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.applications import Xception
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

- Analizde kullanılacak olan gerekli kütüphaneleri import ettim Xception modeli kullanacağım için "tensorflow. Keras.applications"'tan Xception'u import ettim. Katmanlar olarak dense, flatten ve dropout kullandım. Son olarak optimizer olarak "adam"' ı seçtim.

```
[ ]  # Veri setinin yolunu belirtme
     train_path = '/content/drive/MyDrive/trans/train/train'
     valid_path = '/content/drive/MyDrive/trans/train/valid'
```

- Analizi Google Colab üzerinden yapacağım için validasyon ve eğitim verisini daha önceden Google Drive'a yüklemiştim onları okuttum. "Train/train" eğitim verilerimin, "train/valid" validasyon verilerimin klasörlerinin adıdır.

```
# Veri önişleme ve artırma işlemlerini tanımlama
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

```
[ ]  valid_datagen = ImageDataGenerator(rescale=1./255)
```

- Veri artırma ve önişleme işlemleri için "ImageDataGenerator" sınıfını kullandım. "rescale" parametresi ile verilerin 0-255 aralığından 0-1 aralığına ölçeklendirilmesini sağladım. Veri arttırma işlemini train_datagen, valid_datagen olarak atayıp bütün verilerimde uyguladım.

```
[ ]  # Veri akışını tanımlama
     train_generator = train_datagen.flow_from_directory(
         train_path,
         target_size=(224, 224),
         batch_size=32,
         class_mode='categorical')

     valid_generator = valid_datagen.flow_from_directory(
         valid_path,
         target_size=(224, 224),
         batch_size=32,
         class_mode='categorical')
```

```
Found 811 images belonging to 2 classes.
Found 40 images belonging to 2 classes.
```

- Doğrulama ve test veri setlerini yüklemek için kullanılan "flow_from_directory" fonksiyonunu kullandım. Bu fonksiyon belirtilen dizinlerdeki görüntüleri yükler ve önceden yapılandırılmış veri artırma işlemi yapılandırmalarına göre görüntüleri işler. Bu sayede veri akışını tanımlamış oldum. "batch_size" parametresi ise bir seferde işlenecek görüntü sayısını belirtmektedir. Size'ı 32 olarak belirledim bu yüzden her seferinde 32 görüntü işlenmesini sağladım. Eğitim doğrulama ve test verilerimi yükledim eğitim verimde 811 fotoğraf ve 2 sınıf, doğrulama verilerimde 40 fotoğraf ve 2 sınıf bulunmaktadır. Bu sınıflar Dermnet veri setinden seçmiş olduğum iki hastalığı göstermektedir.

```
# Önceden eğitilmiş Xception modelini yükleyin ve son katmanını çıkarma
base_model = Xception(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = base_model.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(train_generator.num_classes, activation='softmax')(x)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 1s 0us/step
```

- Xception modelimi çalıştırmak için import etmiştim. Bu model, ImageNet veri kümesiyle eğitilmiştir. weights parametresi, modelin hangi ağırlıkları kullanacağını belirler bu yüzden bu parametre 'imagenet' olarak ayarlandığında, model, ImageNet veri kümesinden eğitilmiş önceden eğitilmiş ağırlıkları kullanmaktadır. Ben de modelim için imagenet kullandığımdan önceki ağırlıkları kullandı. Aktivasyon fonksiyonu olarak reLu ve softmax kullandım.

```
# Toplam modeli tanımlama
model = Model(inputs=base_model.input, outputs=predictions)
```

```
[ ] # Önceden eğitilmiş katmanları dondurma
    for layer in base_model.layers:
        layer.trainable = False

    # Modeli derleyin
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

- Model= ile başlayan kodda önceden eğitilmiş bir sinir ağı modeli alıp bu modelin girdi katmanını ve çıktı katmanını belirleyerek yeni bir model oluşturdum. Yeni model, önceden eğitilmiş modelin özelliklerini kullanarak girdi verilerine dayanarak başarının bulunmasını sağladı.
- İkinci kodda ise, önceden eğitilmiş bir sinir ağı modelindeki tüm katmanların eğitilemez hale getirilmesini (donmuş hale getirilmesini) sağladım. Bu sayede, önceden eğitilmiş modelin özelliklerinin korunmasına ve yeni modelin sadece eklenen katmanların eğitilmesine izin verilmesine yardımcı oldu. Bu işlemi layer.trainable = False ifadesi ile sağladım bu ifade katmanların eğitilemez hale getirilmesini sağlamaktadır. Sonrasında ise modelin derlemesini yaptım optimizer olarak "adam"'ı kullandım. Başarıyı bulmak için "accuracy" 'i koydum learning rate ifadesi eskiden lr. Olarak ifadelendirilmekteymiş. Yeni güncelleme ile learning_rate olarak yazılması gerektiğinin hatasını verdi Google Colab buna göre kodu tekrardan değiştirerek çalıştırdım.

```
# Modeli eğitme
history= model.fit(train_generator,
          steps_per_epoch=len(train_generator),
          epochs=10,
          validation_data=valid_generator,
          validation_steps=len(valid_generator))
```

- Modelimi derledikten sonra modelimi eğittim. Modelin eğitimi için model.fit'i kullandım ve history değişkenine atadım. Devir sayısını 10 olacak şekilde belirledim.

```
[ ]  # Gerekli kütüphaneleri içe aktarın
     import tensorflow as tf
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
[ ]  # Test veri setinin yolunu belirtin
     test_path = '/content/drive/MyDrive/trans/test'
```

- Modelimi test etmek için gerekli kütüphaneleri import ettikten sonra Google Driveı'mda daha önceden yüklediğim test verilerini okuttum. "Trans" verilerin olduğu ana klasörü, "Test" ise test verilerinin olduğu klasörlerin adıdır.

```
[ ]  # Test veri seti için veri önişleme işlemlerini tanımlama
     test_datagen = ImageDataGenerator(rescale=1./255)
```

```
     # Test veri seti için veri akışını tanımlama
     test_generator = test_datagen.flow_from_directory(
         test_path,
         target_size=(224, 224),
         batch_size=32,
         class_mode='categorical')
```

```
     Found 213 images belonging to 2 classes.
```

```
[ ]  # Modeli test veri seti üzerinde değerlendirme
     scores = model.evaluate(test_generator)
     print('Test loss:', scores[0])
     print('Test accuracy:', scores[1])
```

```
     7/7 [==============================] - 47s 7s/step - loss: 0.4035 - accuracy: 0.8216
     Test loss: 0.403459757566452
     Test accuracy: 0.8215962648391724
```

- Eğitim ve doğrulama verilerinde yaptığım gibi önişleme işlemlerini tanımladıktan sonra test verilerinin içeriğine baktım ve 213 fotoğraf ve 2 sınıftan oluştuğunu gördüm. Sınıflandırma ve fotoğraf sayısına baktıktan sonra modelimi test ettim. Modelimin test sonuçları yukarıda yapmış olduğum diğer modellere göre daha iyi çıktı.

```python
import matplotlib.pyplot as plt

# Modeli test veri seti üzerinde değerlendirin ve kayıp ve doğruluk skorlarını alma
scores = model.evaluate(test_generator)

# Test kaybı ve doğruluğunu ekrana yazdırma
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

# Test kaybı ve doğruluğunu bir grafikte gösterme
plt.plot(history.history['loss'], label='test loss')
plt.plot(history.history['accuracy'], label='test accuracy')
plt.legend()
plt.show()
```

- Test kaybı ve doğruluğunu görselleştirmek için grafik çizdirdim. Bunun için modelimi history olarak kaydetmiştim ve bu model üzerinden değerleri kaydettim. Plt.show() fonksiyonu ile grafiği çizdirdim.

```python
from sklearn.metrics import classification_report, confusion_matrix

# Test veri seti üzerinde modelin tahminlerini alma
y_pred = model.predict(test_generator)

# Sınıflandırma raporunu hesaplama
class_names = list(test_generator.class_indices.keys())
report = classification_report(test_generator.classes, y_pred.argmax(axis=-1), target_names=class_name

# Karışıklık matrisini hesaplama
cm = confusion_matrix(test_generator.classes, y_pred.argmax(axis=-1))

# Sınıflandırma raporunu ekrana yazdırma
print(report)

# Karışıklık matrisini görselleştirme
import seaborn as sns
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```

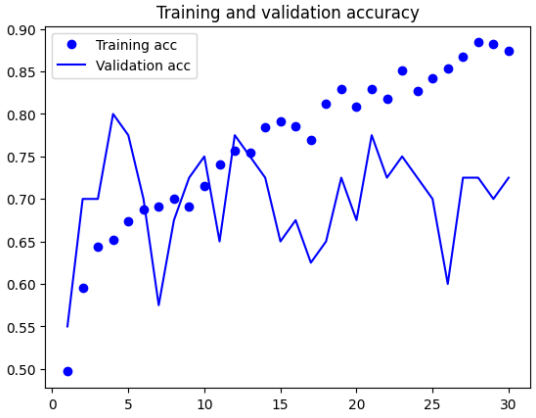
- Son olarak modelin test veri seti üzerindeki performansını değerlendirmek için sınıflandırma raporu ve karışıklık (hata) matrisi oluşturdum. Karışıklık matrisi, gerçek sınıfların ve tahmin edilen sınıfların birbirleriyle nasıl karşılaştırıldığını göstermektedir bunun için confusion_matrix fonksiyonunu kullandım.

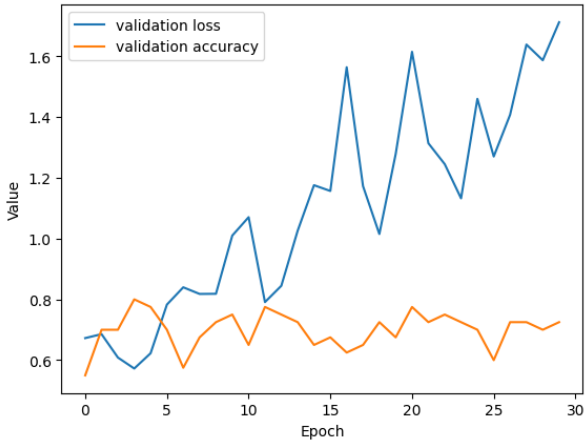
5. İyileştirme ve Karşılaştırma

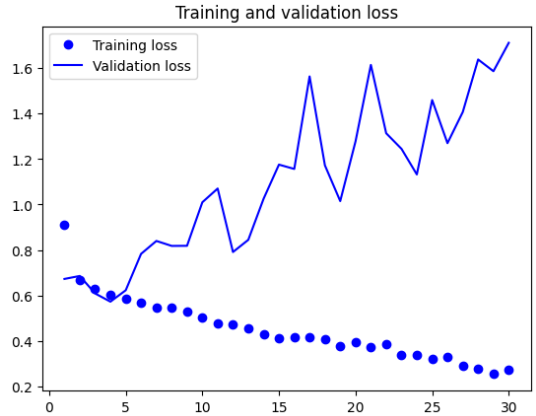
CNN modeli ile yapılan çalışma için;

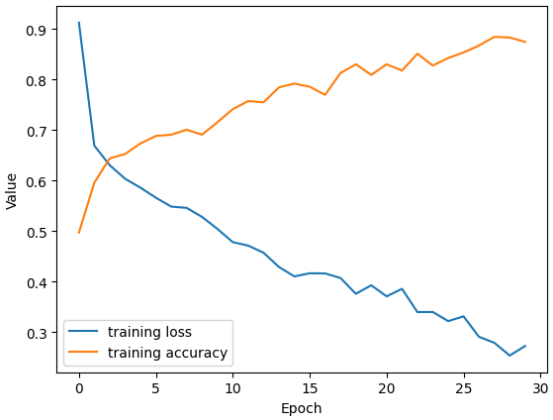
Epoch 1/30	loss: 0.9122	accuracy: 0.4969	val_loss: 0.6725	val_accuracy: 0.5500
Epoch 2/30	loss: 0.6688	accuracy: 0.5956	val_loss: 0.6856	val_accuracy: 0.7000
Epoch 3/30	loss: 0.6302	accuracy: 0.6436	val_loss: 0.6086	val_accuracy: 0.7000
Epoch 4/30	loss: 0.6033	accuracy: 0.6523	val_loss: 0.5724	val_accuracy: 0.8000
Epoch 5/30	loss: 0.5856	accuracy: 0.6732	val_loss: 0.6227	val_accuracy: 0.7750
Epoch 6/30	loss: 0.5658	accuracy: 0.6880	val_loss: 0.7828	val_accuracy: 0.7000
Epoch 7/30	loss: 0.5482	accuracy: 0.6905	val_loss: 0.8396	val_accuracy: 0.5750
Epoch 8/30	loss: 0.5457	accuracy: 0.7004	val_loss: 0.8178	val_accuracy: 0.6750
Epoch 9/30	loss: 0.5279	accuracy: 0.6905	val_loss: 0.8183	val_accuracy: 0.7250
Epoch 10/30	loss: 0.5041	accuracy: 0.7152	val_loss: 1.0098	val_accuracy: 0.7500
Epoch 11/30	loss: 0.4779	accuracy: 0.7411	val_loss: 1.0703	val_accuracy: 0.6500
Epoch 12/30	loss: 0.4710	accuracy: 0.7571	val_loss: 0.7909	val_accuracy: 0.7750
Epoch 13/30	loss: 0.4569	accuracy: 0.7546	val_loss: 0.8446	val_accuracy: 0.7500
Epoch 14/30	loss: 0.4288	accuracy: 0.7842	val_loss: 1.0257	val_accuracy: 0.7250
Epoch 15/30	loss: 0.4101	accuracy: 0.7916	val_loss: 1.1755	val_accuracy: 0.6500
Epoch 16/30	loss: 0.4162	accuracy: 0.7855	val_loss: 1.1565	val_accuracy: 0.6750
Epoch 17/30	loss: 0.4159	accuracy: 0.7694	val_loss: 1.5635	val_accuracy: 0.6250
Epoch 18/30	loss: 0.4070	accuracy: 0.8126	val_loss: 1.1724	val_accuracy: 0.6500
Epoch 19/30	loss: 0.3757	accuracy: 0.8298	val_loss: 1.0149	val_accuracy: 0.7250
Epoch 20/30	loss: 0.3925	accuracy: 0.8089	val_loss: 1.2783	val_accuracy: 0.6750
Epoch 21/30	loss: 0.3706	accuracy: 0.8298	val_loss: 1.6143	val_accuracy: 0.7750
Epoch 22/30	loss: 0.3854	accuracy: 0.8175	val_loss: 1.3135	val_accuracy: 0.7250

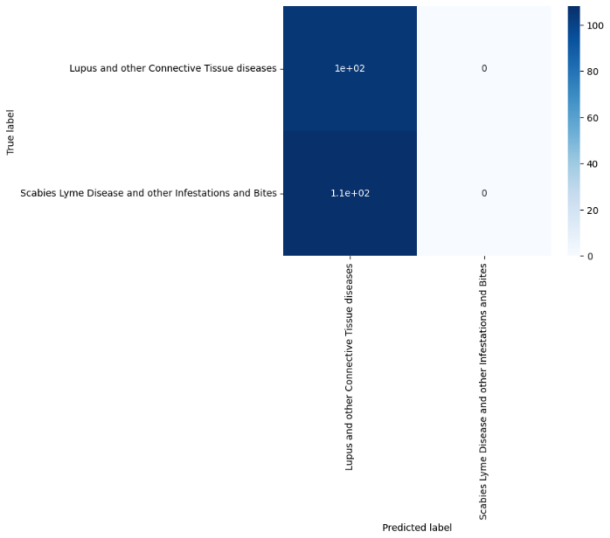
Epoch 23/30	loss: 0.3395	accuracy: 0.8508	val_loss: 1.2448	val_accuracy: 0.7500
Epoch 24/30	loss: 0.3397	accuracy: 0.8274	val_loss: 1.1323	val_accuracy: 0.7250
Epoch 25/30	loss: 0.3216	accuracy: 0.8422	val_loss: 1.4596	val_accuracy: 0.7000
Epoch 26/30	loss: 0.3310	accuracy: 0.8533	val_loss: 1.2699	val_accuracy: 0.6000
Epoch 27/30	loss: 0.2906	accuracy: 0.8668	val_loss: 1.4072	val_accuracy: 0.7250
Epoch 28/30	loss: 0.2787	accuracy: 0.8841	val_loss: 1.6382	val_accuracy: 0.7250
Epoch 29/30	loss: 0.2534	accuracy: 0.8829	val_loss: 1.5865	val_accuracy: 0.7000
Epoch 30/30	loss: 0.2723	accuracy: 0.8742	val_loss: 1.7115	val_accuracy: 0.7250
	loss: 0.8087	accuracy: 0.7183		Test accuracy: 0.7183098793029785









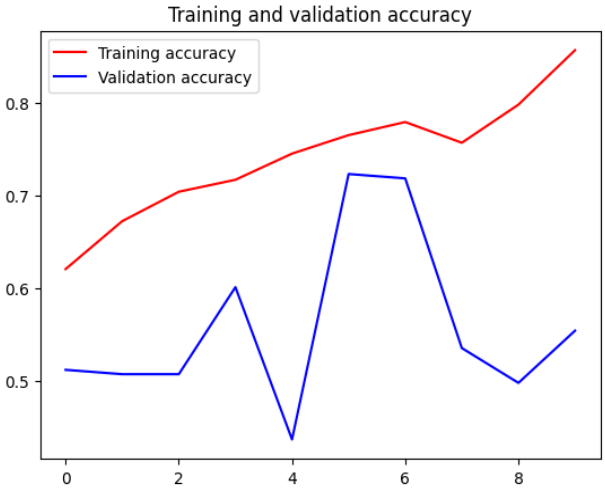


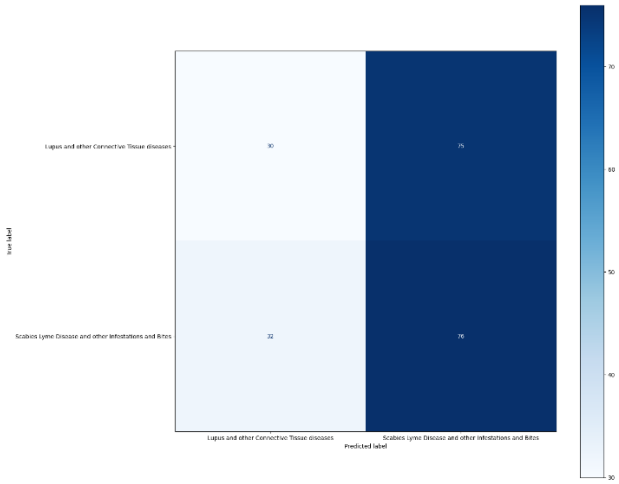
Bu model için 30 devir yapılmıştır. Her bir epokta loss (kayıp değeri) sıfıra yaklaştı ve azaldı. Kayıp değeri kötü bir tahmin cezasıdır kaybın sıfır olması model tahmininin mükemmel olduğunu göstermekte, sıfıra yaklaşması ise tahminin mükemmele yakın olduğunu göstermektedir.

Doğruluk değerim her bir epokta giderek arttı. Son devirde accuracy: 0.8742 (%87) val_accuracy: 0.7250 (%72) olarak bulundu. Test loss değerim 0.8087 ve test accuracy: 0.7183098793029785 olarak bulundu. Sonuç olarak çok iyi bir performans gösterdiğini söylemek güçtür. Çünkü olması gereken kayıp değerlerinin çok yukarısında bir kayıp değeri gözükmekte ve doğruluk değerleri iyileştirmeye rağmen istenilen seviyede değildir. Yukarıdaki grafiklerde de görüldüğü üzere validasyon verilerinde genellikle bir sorun olduğu düşünmekteyim. Çünkü grafiklerde tutarsız çizgiler görünmektedir. Hata matriksinde ise anlamlı sonuçlar çıkmamıştır.

CNN-INCEPTINOV3 modeli ile yapılan çalışma için;

Epoch 1/10	loss: 41.9492	accuracy: 0.6204	val_loss: 59.6944	val_accuracy: 0.5117
Epoch 2/10	loss: 45.9395	accuracy: 0.6722	val_loss: 69.2547	val_accuracy: 0.5070
Epoch 3/10	loss: 31.7192	accuracy: 0.7039	val_loss: 37.1832	val_accuracy: 0.5070
Epoch 4/10	loss: 20.4247	accuracy: 0.7168	val_loss: 17.0593	val_accuracy: 0.6009
Epoch 5/10	loss: 13.3769	accuracy: 0.7450	val_loss: 41.2064	val_accuracy: 0.4366
Epoch 6/10	loss: 9.5415	accuracy: 0.7650	val_loss: 8.2916	val_accuracy: 0.7230
Epoch 7/10	loss: 7.3574	accuracy: 0.7791	val_loss: 8.6217	val_accuracy: 0.7183
Epoch 8/10	loss: 6.0360	accuracy: 0.7568	val_loss: 806.3933	val_accuracy: 0.5352
Epoch 9/10	loss: 5.1951	accuracy: 0.7979	val_loss: 136.1263	val_accuracy: 0.4977
Epoch 10/10	loss: 4.3925	accuracy: 0.8566	val_loss: 20.6092	val_accuracy: 0.5540

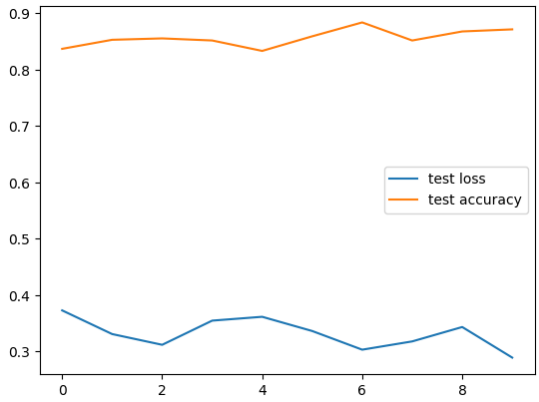


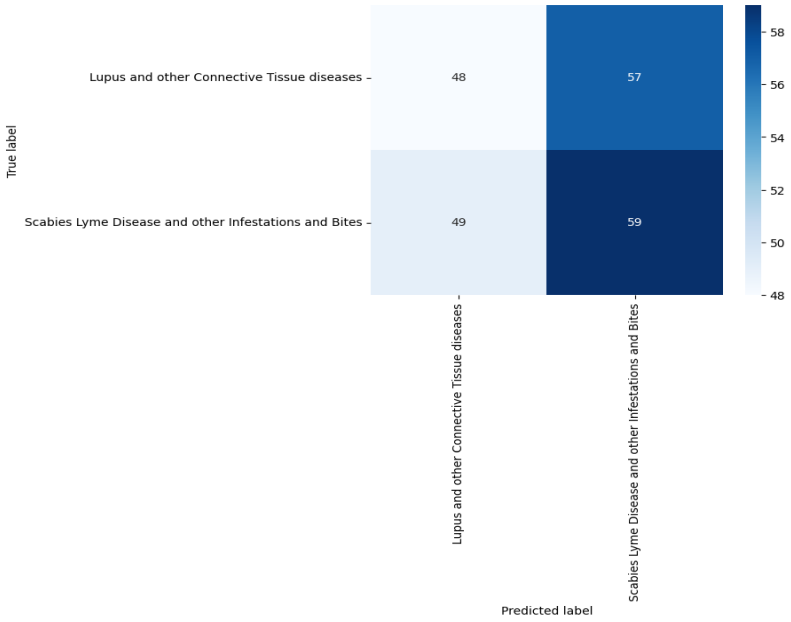


Bu model için 10 devir yapılmıştır. Her bir epokta loss (kayıp değeri) azalmış olmasına karşın bir değerini bile görmediği yüksek kayıplar olduğu görülmüştür. Doğruluk değerim her bir epokta giderek artması gerekirken düzensiz bir şekilde artıp azalmıştır. Son devirde accuracy: accuracy: 0.8566 (%85) ve val_accuracy: 0.5540 (%55) olarak bulunmuştur. Bu modelin her ne kadar transfer öğrenme yolu ile yapılsa da bu model iyi çalışmamaktadır. Bu sonuca kayıp değerlerinin aşırı yüksek olması ve doğruluk değerlerinin düşük çıkması sayılabilir. Aynı şekilde grafiklere bakıldığı zaman validasyon verilerinde sıkıntı olduğu görülmektedir şunu parantez açmam gerekirse validasyon verileri de Dermnet veri seti üzerinden alınmıştır başka bir kaynak kullanılmamıştır. Bunun nedeninin projenin başında da bahsettiğim üzere insan derisinin kıllı olması ve kene gibi başka hayvanların da veri setindeki fotoğraflarda bulunması sayılabilir. Bir araştırma veya eğitim için bu model yapılan kadarıyla başarısız olduğunu düşünmekteyim.

Xception ile transfer learning modeli

Epoch 1/10	loss: 0.3731	accuracy: 0.8372	val_loss: 0.4635	val_accuracy: 0.7250
Epoch 2/10	loss: 0.3311	accuracy: 0.8533	val_loss: 0.4707	val_accuracy: 0.7750
Epoch 3/10	loss: 0.3121	accuracy: 0.8557	val_loss: 0.4318	val_accuracy: 0.8500
Epoch 4/10	loss: 0.3550	accuracy: 0.8520	val_loss: 0.3658	val_accuracy: 0.7750
Epoch 5/10	loss: 0.3618	accuracy: 0.8335	val_loss: 0.4914	val_accuracy: 0.7250
Epoch 6/10	loss: 0.3365	accuracy: 0.8594	val_loss: 0.5185	val_accuracy: 0.7250
Epoch 7/10	loss: 0.3034	accuracy: 0.8841	val_loss: 0.6805	val_accuracy: 0.7000
Epoch 8/10	loss: 0.3181	accuracy: 0.8520	val_loss: 0.3388	val_accuracy: 0.8500
Epoch 9/10	loss: 0.3436	accuracy: 0.8681	val_loss: 0.4410	val_accuracy: 0.7750
Epoch 10/10	loss: 0.2894	accuracy: 0.8718	val_loss: 0.4103	val_accuracy: 0.8250
	loss: 0.4035	accuracy: 0.8216	Test loss: 0.403459757566452	Test accuracy: 0.8215962648391724





Bu model için 10 devir yapılmıştır. Her bir epokta loss (kayıp değeri) sıfıra yaklaştı ve azaldı.

Doğruluk değerim her bir epokta giderek arttı. Son devirde accuracy: 0.8718 ve val_accuracy: 0.8250 olarak bulundu. Test loss değerim 0.403459757566452 (%40) ve Test accuracy: 0.8215962648391724 (%82) olarak bulundu. Sonuç olarak bu modelin de çok iyi bir performans gösterdiğini söylemek güçtür. Çünkü olması gereken kayıp değerlerinin çok yukarısında bir kayıp değeri gözükmekte ve doğruluk değerleri iyileştirmeye rağmen istenilen seviyede değildir. İstenilen sonuçlar elde edilememesine karşın bu üç model arasında en başarı olan model Xception modeli olarak gözükmektedir.

SONUÇ YORUM:

Kişisel olarak bir araştırma yapacak olsaydım ben bu üç modeli de seçmek istemezdim. Sorunun kaynağı benim sağlam bir yazılım bilgisine sahip olmamam da olabilir, yukarıda bahsettiğim gibi verilerin üstündeki kıl ve kene fotoğrafları da olabilir. Bunun dışında üçü arasında seçmek zorunda olsaydım hem loss değerleri düşük olduğu hem de başarı değerleri yüksek olduğu için Xception transfer öğrenme modelini seçerdim.

KAYNAKÇA

https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8

https://medium.com/@tuncerergin/convolutional-neural-network-convnet-yada-cnn-nedir-nasil-calisir-97a0f5d34cad

https://stanford.edu/~shervine/l/tr/teaching/cs-230/cheatsheet-convolutional-neural-networks

https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network#:~:text=A%20CNN%20is%20a%20kind,the%20network%20architecture%20of%20choice.

https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

https://medium.com/novaresearchlab/%C3%B6%C4%9Frenme-aktar%C4%B1m%C4%B1-transfer-learning-c0b8126965c4

https://devhunteryz.wordpress.com/2018/07/28/transfer-ogrenimi-transfer-learning/

https://dergipark.org.tr/tr/download/article-file/833620