Yapay Sinir Ağları Kullanarak Alzheimer Tespiti

Murat GÜVENÇ – 170290046

Fırat Üniversitesi – Mühendislik Fakültesi

Yazılım Mühendisliği

Biyomedikal Mühendisliği Ders Projesi

ÖZET

Alzheimer hastalığı yaygın olarak görülen bir demans türü olup, ilerleyen ve tedavisi bulunmayan nörodejeneratif bir hastalıktır. Hastalığı teşhis edebilmek için birçok görüntüleme tekniği kullanılmaktadır. Bu tekniklerden biri Manyetik Rezonans Görüntüleme (MRI) tekniğidir. Erken teşhis edilmesi hastalığın ilerlemesini yavaşlatmak ve gerekli önlemleri alma konusunda hasta ve ailesi için büyük önem taşımaktadır. Erken ve doğru teşhis için derin öğrenme yöntemleriyle bu konuda destekleyici çalışmalar gerçekleştirilmiştir. Aynı zamanda hastalığın seyrini takip etmek için de derin öğrenme yöntemleri kullanılmaktadır.

GİRİŞ

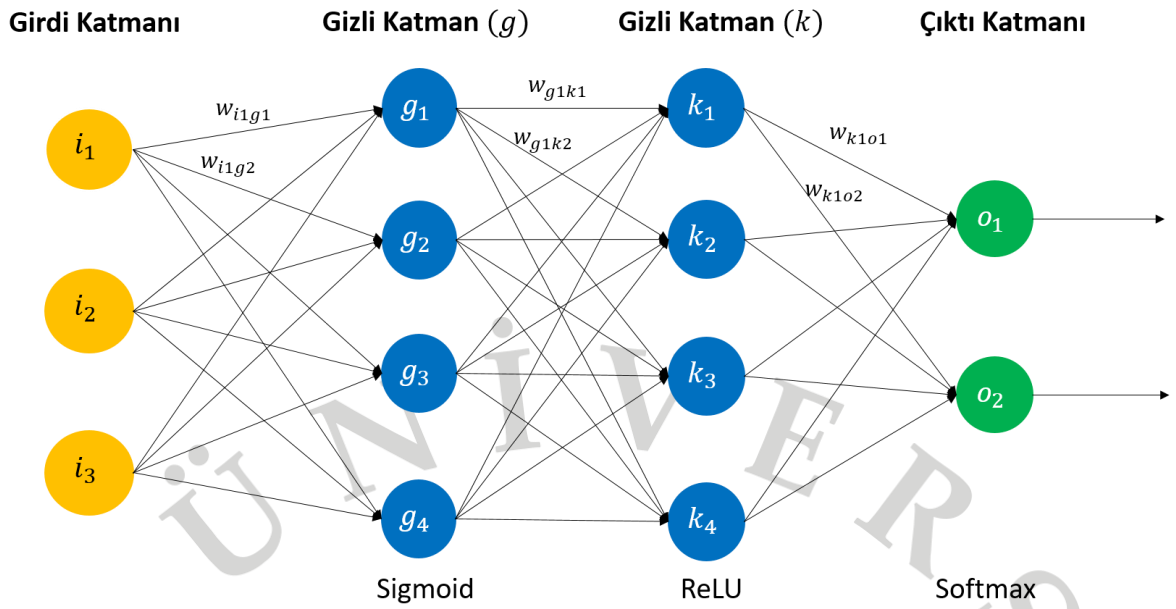
Alzheimer hastalığı, beyin hücrelerinin ölümü sonucu oluşan, nörodejeneratif bir hastalıktır. Bu hastalık, hafıza kaybı, düşüncelerin zorlaşması, dil ve konuşma zorlukları, yön bulma güçlüğü, kişilik değişimleri ve sosyal ilişkilerde zorluklar gibi belirtileri içerebilir. Yaş ilerledikçe daha yaygın hale gelir ve yaşlılık döneminde en sık görülen nörolojik hastalıktır. Alzheimer hastalığının tanısı genellikle zor ve zaman alıcı bir süreçtir. Bu nedenle, hastalığın erken tanısı önemlidir.

Projenin amacı Alzheimer hastalığının erken tanısını kolaylaştırmak ve hastalığın seyrini izlemek için kullanılabilecek bir yöntem sunmaktır. Alzheimer hastalığının tanısını yapmak için MRI görüntüleri kullanır. Proje, veri kümesindeki görüntüleri eğitmek ve sınıflandırmak için derin öğrenme yöntemlerini kullanır. Eğitim ve test verileri, Alzheimer hastalığına sahip kişiler ile sağlıklı kişiler arasından seçilmiştir. Proje, Evrişimsel Sinir Ağı (CNN) kullanarak görüntüleri sınıflandırmayı ve hastalık tanısını yapmayı amaçlar. Proje sonunda, modelin Alzheimer hastalığını doğru bir şekilde tanımada yüksek bir başarı oranı elde ettiği sonucuna varılmıştır.

YÖNTEM

Yapay sinir ağı (Artificial Neural Network - ANN), insan beyninin yapısını ve çalışmasını modelleyen, çok sayıda düzenli veya düzensiz bir şekilde bağlantılı nöronların bulunduğu bir sistemdir. ANN'ler, verileri analiz etmek, öğrenmek ve tahmin etmek için kullanılır.

Derin öğrenme, makine öğrenmesi alt dalıdır ve karışık verilerin analizi ve modelleme için ağırlıklı olarak aşırı büyük miktarda veri kullanır. Bu, makineyi gerçek dünya problemlerinin çözümüne yardım etmek için eğitmek ve daha iyi sonuçlar elde etmek için katmanlar halinde farklı özellikler öğrenmesini sağlar.

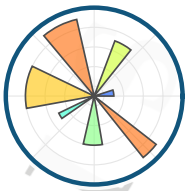


YSA Modeli

Kullanılan Kütüphane ve Modüller:

NumPy, Python programlama dili için büyük, çok boyutlu dizileri ve matrisleri destekleyen, bu diziler üzerinde çalışacak üst düzey matematiksel işlevler ekleyen bir kitaplıktır.

Matplotlib, Python programlama dili ve sayısal matematik uzantısı NumPy için bir çizim kitaplığıdır. Tkinter, wxPython, Qt veya GTK gibi genel amaçlı GUI araç takımlarını kullanarak grafikleri uygulamalara gömmek için nesne yönelimli bir API sağlar.



OS Module

```
>>> import os
```

OS modülü, Python'da dosya işlemleri için vazgeçilmezdir. Bir işletim sisteminde dosya, dizin işlemlerini OS modülü ile düzenlemekteyiz. Bu modül sayesinde farklı işletim sistemlerinde çalışabilecek dosya işlemleri yapabiliriz. Yani windows için yazmış olduğunuz bir program linux dosya sisteminde de çalışacaktır.

Random modülü, rastgele sayılar ve örnekler oluşturmak için çeşitli işlevler sunar. Makine öğrenimi modelleri, verileri rastgele örnekleyerek ve dönüştürerek fazla uydurmayı önleyebilir ve daha sağlam tahminler yapabilir. Random modülü, bu açıdan birçok makine öğrenimi görevinde önemli bir araç haline gelir.

Pathlib, nesneye yönelik dosya sistemi yolları işlemleridir. Dosya yolu objede ve ona özgü metotlarda saklanır. Glob modülü içine dahildir ve özel operatörleri ile hızlı çalışmayı sağlar.

TensorFlow, makine öğrenimi için ücretsiz ve açık kaynaklı bir yazılım kütüphanesidir. Bir dizi görevde kullanılabilir, ancak derin sinir ağlarının eğitimi ve çıkarımına özel olarak odaklanmaktadır. Tensorflow, veri akışına ve türevlenebilir programlamaya dayalı sembolik bir matematik kitaplığıdır.

Keras, Python'da yazılmış açık kaynaklı bir sinir ağı kütüphanesidir. Keras TensorFlow, Microsoft Cognitive Toolkit, R, Theano veya PlaidML ile beraber çalışabilir. Derin sinir ağları ile hızlı deney yapabilmek için tasarlanan bu cihaz kullanıcı dostu, modüler ve genişletilebilir olmaya odaklanıyor.

ANALİZ

İlk olarak NumPy, Matplotlib, TensorFlow ve Keras kütüphanelerini içe aktarıyoruz.

```
[ ]  import numpy as np
     import matplotlib.pyplot as plt
     import tensorflow as tf
     import os, random, pathlib

     from keras.utils import image_dataset_from_directory
     from keras import layers
```

Veri setimiz klasörler halinde ayrılmış olarak bulunuyor. Dosya yolunu belirlemek için **dir** ve **data_dir** değişkenlerini tanımlıyoruz. "dir" değişkeni, "drive/MyDrive/Biyomedikal Projesi/Dataset/" dizin yolunu içerir. "data_dir" değişkeni ise, "dir" değişkeninin değerini "pathlib.Path()" sınıfının nesnesi olarak içerir. Bu, dosya yolunun veri dizisi olarak manipüle edilmesini kolaylaştırır.

```
dir = 'drive/MyDrive/Biyomedikal Projesi/Dataset/'
data_dir = pathlib.Path(dir)
```

Veri dizininde bulunan klasör adlarını **classes** değişkenine atıyoruz. **data_dir.glob("*")** ifadesi, veri dizinindeki tüm dosya ve klasör adlarını içeren bir sıralı liste oluşturur. **sorted()** fonksiyonu, bu listeyi alfabetik olarak sıralar. Daha sonra, her bir klasör adını "classes" değişkenine atan **item.name** ifadeleri ile tek bir dizi halinde saklar. Sonuçta "classes" değişkeni, veri dizinindeki klasör adlarını içeren bir NumPy dizisidir. Aynı zamanda hedef kümemizdir ve demanssız, çok hafif demanslı, hafif demanslı ve orta demanslı olmak üzere 4 adet sınıf içerir.

```
classes = np.array([sorted(item.name for item in data_dir.glob("*"))])
classes
```

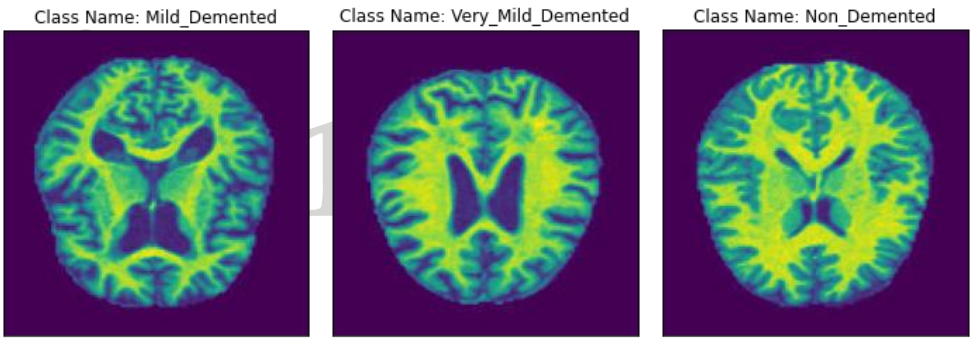
```
array([['Mild_Demented', 'Moderate_Demented', 'Non_Demented',
        'Very_Mild_Demented']], dtype='<U18')
```

Rastgele bir sınıf ve rastgele bir görüntü veren **exImage** fonksiyonu tanımlıyoruz. İlk olarak, **randomClass** değişkeni için **classes** dizisinden rastgele bir sınıf seçiyoruz. Daha sonra, bu sınıfın içindeki rastgele bir görsel için **randomImage** değişkeni tanımlıyoruz. randomImage değişkeni, **data_dir.glob(randomClass+"/*.jpg")** ile belirlenir ve randomClass değişkeninin değerine göre bu sınıfın içindeki tüm **.jpg** uzantılı dosyalardan birini seçer.

Ardından, **plt.figure** fonksiyonu ile bir resim oluşturuyoruz ve **plt.imshow** fonksiyonu ile görselleştirme işlemini yapıyoruz. **plt.xticks** ve **plt.yticks** fonksiyonları ile, resmin x ve y eksenindeki etiketlerini sıfırlıyoruz. **plt.title** fonksiyonu ile, resmin başlığını belirliyoruz ve rastgele seçilen sınıfı belirtiyoruz.

Son olarak, exImage fonksiyonunu üç kez çağırıp üç adet farklı görseli ekrana veriyoruz.

```
def exImage():
  randomClass = random.choice(classes[0,:])
  randomImage = random.choice(list(data_dir.glob(randomClass+"/*.jpg")))
  a= plt.figure(figsize=(4, 4))
  image = plt.imread(randomImage)
  plt.xticks([])
  plt.yticks([])
  plt.title("Class Name: "+ str(randomClass))
  plt.imshow(image)
exImage()
exImage()
exImage()
```



Eğitim sırasında kullanılacak verilerin veri seti boyutlarını tanımlıyoruz. **batch_size** değişkeni her bir eğitim adımında kullanılacak veri sayısını tanımlar. **img_height** ve **img_width** değişkenleri, görsellerin boyutlarını (yükseklik ve genişlik) tanımlar.

```
batch_size = 32
img_height = 224
img_width = 224
```

Veri setinden eğitim verilerini ve doğrulama verilerini ayırmamız gerekiyor.

- **image_dataset_from_directory** fonksiyonu, verilen dizin içindeki resim dosyalarından veri seti oluşturur.
- **validation_split** değişkeni, veri setindeki verilerin doğrulama verilerine ayrılması için kullanılır. Değer 0.2 olarak belirlenmiştir, yani veri setindeki verilerin %80'i eğitim verileri olarak kullanılacak ve %20'si doğrulama verileri olarak kullanılacaktır.
- **subset** değişkeni, hangi veri setinin (eğitim veya doğrulama) kullanılacağını belirler.
- **seed** değişkeni, veri setinin rastgele seçim işlemi sırasında kullanılan rastgele sayı üreticisinin değerini tanımlar.
- **image_size** değişkeni, verilerdeki görsellerin boyutlarını tanımlar.

Sonuç olarak, veri setinden eğitim verileri (5120 veri) ve doğrulama verilerini (1280 veri) bu yöntemle ayırıyoruz.

```
train_data = image_dataset_from_directory(data_dir, validation_split=0.2, subset='training', seed=123,
                                          image_size=(img_height, img_width), batch_size=batch_size)
```

```
Found 6400 files belonging to 4 classes.
Using 5120 files for training.
```

```
test_data = image_dataset_from_directory(data_dir, validation_split=0.2, subset='validation', seed=123,
                                         image_size=(img_height, img_width), batch_size=batch_size)
```

```
Found 6400 files belonging to 4 classes.
Using 1280 files for validation.
```

Modelimizi TensorFlow Keras'taki Sequential modeliyle oluşturuyoruz. Modelimizin adımları şunlardır:

- Rescaling resim verilerinin renklerini 0 ile 1 arasındaki değerlere dönüştürür.
- Conv2D katmanları, resim verilerindeki görsel özellikleri öğrenmek için filtreler kullanır. Her Conv2D katmanı, önceki katmanın çıktısındaki veriler üzerinde çalışır.
- MaxPooling2D katmanları, önceki Conv2D katmanından gelen verileri boyutlarını azaltır.
- Dropout katmanı, overfitting (eğitim verileri ile modelin çok iyi ancak test verilerinde kötü performans göstermesi) problemini önlemek için rastgele seçilen nöronların çıktılarını düşürür.
- Flatten katmanı, önceki katmanın verilerini tek bir vektör haline getirir.
- Dense katmanları, veriler üzerinde tam bağlantılı ağ yapısı kullanır. İlk dense katmanı, 128 nöronlu ve relu aktivasyon fonksiyonu kullanır. İkinci dense katmanı ise, 4 nöronlu ve softmax aktivasyon fonksiyonu kullanır. Sonuç olarak, model, her resim için 4 sınıfın olasılık değerlerini verir.

```
model = tf.keras.Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),

    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),

    layers.Dropout(0.5),
    layers.Flatten(),

    layers.Dense(128, activation='relu'),
    layers.Dense(4, activation='softmax')
])
```

Modelin performansını ölçmek için bir derleme (**compile**) işlemi yapmamız gerekiyor. **Optimizer** parametresi **Adam** optimizasyon algoritmasını kullanmak için ayarlanır. **Loss** fonksiyonu **SparseCategoricalCrossentropy** ve metrik parametresi **accuracy** belirlenir. Bu, modelin doğruluğunu ve eğitim sürecindeki kaybın miktarını ölçmek için kullanılacak.

```
model.compile(optimizer='Adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
```

Artık modelimizi eğitebiliriz. **epochs** değişkeni 5 olarak ayarlanmıştır ve **model.fit** fonksiyonu ile eğitim verileri ile eğitim işlemi başlatılmıştır. Eğitim işlemi 5 epoch (iterasyon) sürecektir ve her epoch'ta **train_data** verileri kullanılacaktır. **validation_data** parametresi ile test verileri belirtilmiş ve **batch_size** parametresi ile her eğitim adımında kullanılacak veri miktarı (32) belirtilmiştir. Eğitim sonunda **history** değişkenine eğitim sürecine ait bilgiler kaydedilecektir.

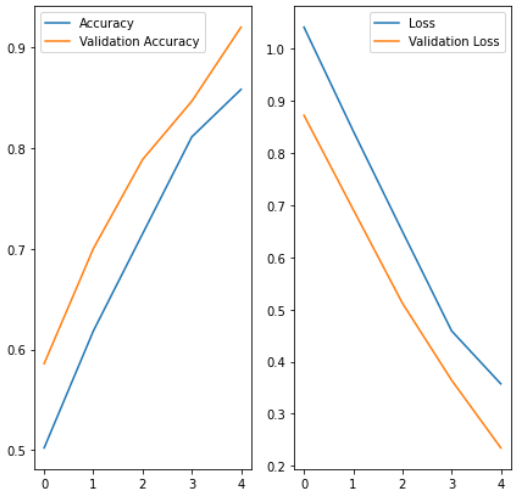
```
epochs = 5
history = model.fit(train_data, epochs=epochs, validation_data=test_data, batch_size=batch_size)
```

```
Epoch 1/5
160/160 [==============================] - 785s 4s/step - loss: 1.0416 - accuracy: 0.5023 - val_loss: 0.8725 - val_accuracy: 0.5859
Epoch 2/5
160/160 [==============================] - 265s 2s/step - loss: 0.8425 - accuracy: 0.6184 - val_loss: 0.6918 - val_accuracy: 0.7000
Epoch 3/5
160/160 [==============================] - 263s 2s/step - loss: 0.6497 - accuracy: 0.7146 - val_loss: 0.5121 - val_accuracy: 0.7883
Epoch 4/5
160/160 [==============================] - 261s 2s/step - loss: 0.4589 - accuracy: 0.8107 - val_loss: 0.3643 - val_accuracy: 0.8461
Epoch 5/5
160/160 [==============================] - 264s 2s/step - loss: 0.3572 - accuracy: 0.8578 - val_loss: 0.2344 - val_accuracy: 0.9195
```

Modelimizin 5 iterasyon sonucundaki doğruluk değeri yaklaşık olarak %92 olduğu görülmektedir.

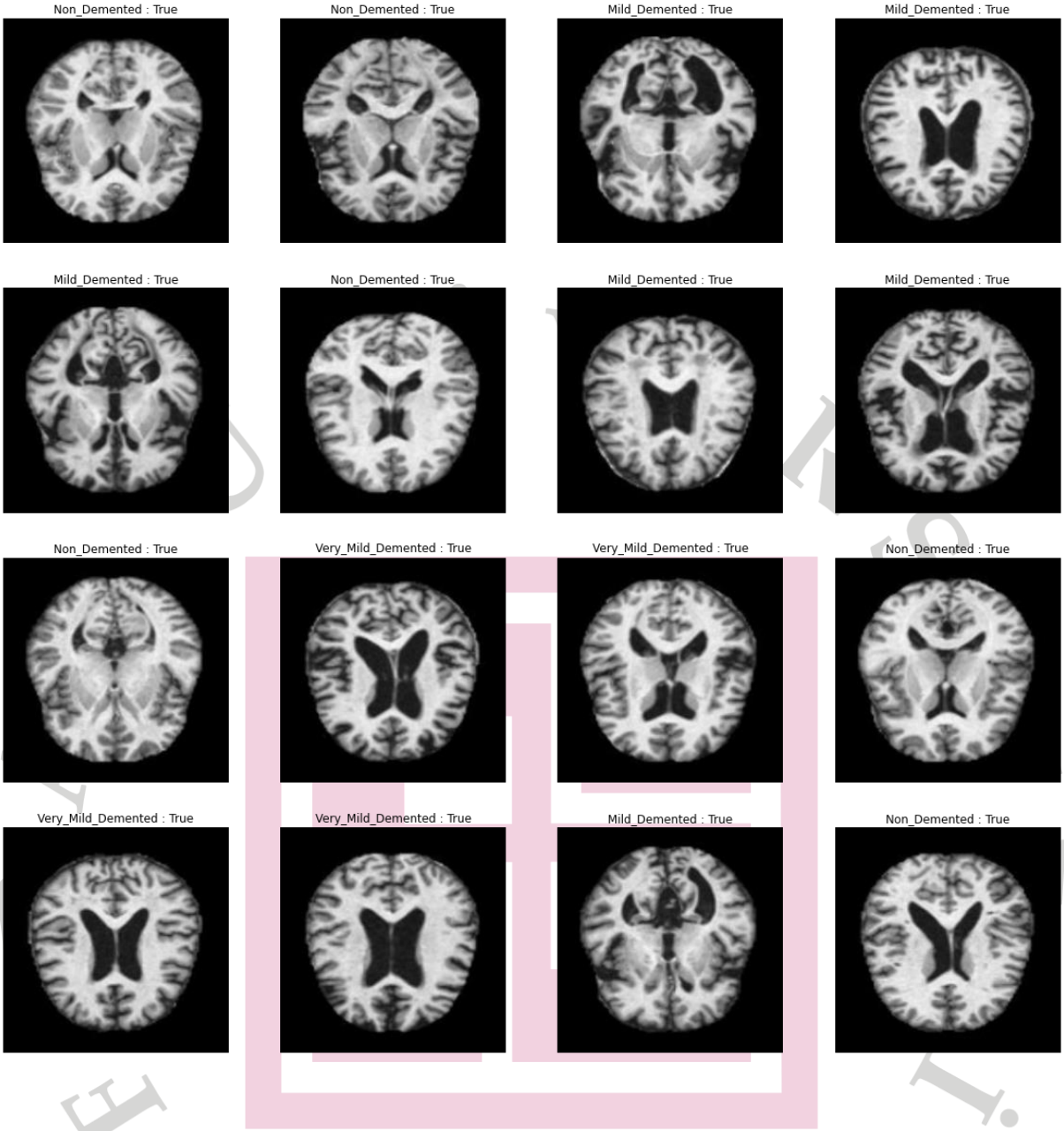
Modelin eğitimi esnasında **accuracy** ve **loss** değerlerinin nasıl değiştiğini göstermek için iki grafik çizdiriyoruz. İlk grafikte epoch sayısına göre accuracy ve **validation accuracy** değerleri görüntülenir. İkinci grafikte de epoch sayısına göre **loss** ve **validation loss** değerleri görüntülenir. Bu grafikler, modelin eğitimi sırasında accuracy ve loss değerlerinin nasıl değiştiğini ve validation accuracy ve validation loss değerlerinin accuracy ve loss değerlerine yakın olup olmadığını gösterir.

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(7,7))
plt.subplot(121)
plt.plot(epochs_range, accuracy, label='Accuracy')
plt.plot(epochs_range, val_accuracy, label='Validation Accuracy')
plt.legend()

plt.subplot(122)
plt.plot(epochs_range, loss, label='Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend()
plt.show()
```



Test veri setindeki 16 görüntü için modelin tahminlerini ve gerçek sınıfları görselleştirelim. Her görüntü için, görselin numpy dizisi halindeki hali alınır, tek boyutlu bir tensör olarak genişletilir ve model tarafından tahmin edilir. Tahmin edilen sınıfın indeksi alınır ve test veri setindeki sınıf isimleri ile eşleştirilir. Eşleşme varsa, **True** olarak işaretlenir ve görüntün yanında görüntün sınıf tahmininin doğru olduğu görüntülenir. Görüntüler, 4x4 bir düzen içinde görüntülenir ve her birinin altında tahminin doğru olup olmadığı görüntülenir.

```
plt.figure(figsize=(20,20))
class_name = test_data.class_names
result = False
for images, labels in test_data.take(1):
  for i in range(16):
    ax = plt.subplot(4, 4, i+1)
    img = images[i].numpy().astype('uint8')
    img = tf.expand_dims(img, axis=0)
    predictions = model.predict(img)
    prediction_class = np.argmax(predictions)
    if class_name[prediction_class] == class_name[labels[i]]:
      result = True
    plt.imshow(images[i].numpy().astype('uint8'))
    plt.title(class_name[prediction_class] + " : " + str(result))
    plt.axis('off')
```



SONUÇ

Bu projede Alzheimer hastalığını tespit etmek ve sınıflandırmak için bir yapay sinir ağı modeli kullanılmıştır. Model, evrişimli sinir ağı (CNN) tabanlı bir sınıflandırıcıdır ve görüntü verileri üzerinde eğitilmiştir. Eğitilen model, MRI görüntüleri arasındaki farkları tespit etmeyi amaçlamıştır. Veri seti Kaggle'dan alınmış olup 6400 adet görüntü içermektedir.

Modelin başarımı, accuracy ve loss gibi değerler kullanılarak değerlendirilmiştir. Sonuçlar grafikler ile görselleştirilmiş ve test verilerine dayalı tahminler yapılmıştır. Modelin doğruluk oranı oldukça yüksek olup çeşitli yöntemlerle bu oran artırılabilir.

KAYNAKÇA

→ https://numpy.org
→ https://matplotlib.org
→ https://tr.wikipedia.org/wiki/Keras
→ https://keras.io/api/optimizers/adam
→ https://tr.wikipedia.org/wiki/TensorFlow
→ https://docs.python.org/3/library/pathlib.html
→ https://caglarbostanci.com.tr/python-os-modulu-ve-metodlari
→ https://docs.python.org/3/library/random.html#module-random
→ https://www.kaggle.com/datasets/sachinkumar413/alzheimer-mri-dataset
→ https://www.kaggle.com/code/karimkhaled/alzheimer-prediction-92-acc/notebook
→ https://www.derinogrenme.com/2018/06/28/geri-yayilim-algoritmasina-matematiksel-yaklasim