T.C

TEKİRDAĞ NAMIK KEMAL ÜNİVERSİTESİ

ÇORLU MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ PROJE-II DERSİ BİTİRME ÇALIŞMASI

OTONOM ARAÇLAR İÇİN TRAFİK IŞIKLARININ SINIFLANDIRILMASI

Ad ve Soyad :YAĞMUR KAHYA

No :2170656025

DANIŞMAN

Dr. Öğr. Üyesi Erkan ÖZHAN

ÖZET

Projenin Amacı : Yapay Zeka ve teknoloji dünyasında birçok araştırmacı ve Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi gibi büyük şirketler otonom araçlar ve sürücüsüz arabalar üzerinde çalışıyor. Dolayısıyla bu teknolojide doğruluğa ulaşmak için araçların trafik işaretlerini yorumlayabilmesi ve buna göre kararlar alabilmesi gerekmektedir.

Yolcuların seyahat için tamamen araçlara güvenebilmesi için otonom araçların trafik işaretlerini, kurallarını anlaması ve bunlara uyması gerekir.

Bu projede otonom araçların bu trafik işaretlerini anlaması için trafik işaretleri sınıflandırma işleminin yapılması amaçlanmıştır.

Trafik işaretleri sınıflandırması, bir trafik işaretinin hangi sınıfa ait olduğunu belirleme işlemidir.

Projenin İlerleyişi: Bu Python projesinde, görüntüde bulunan trafik işaretlerini farklı kategorilere ayırabilen derin bir sinir ağı modeli oluşturulur. Bu model ile tüm otonom araçlar için çok önemli bir görev olan trafik işaretlerini okuyup anlayabilmesi amaçlanmıştır.

İÇİNDEKİLER

Sayfa

ÖNSÖZ

Proje çalışmamızın planlanmasında, yürütülmesinde ilgi ve desteğini esirgemeyen, Ekan Özhan ’a, Bilgisayar Mühendisliği Bölüm Öğretim Elemanlarına ve Tekirdağ Namık Kemal Üniversitesi’ne en içten teşekkürlerimizi sunarız.

Ayrıca eğitimimiz süresince bize her konuda tam destek veren ailemize ve bize değerli bilgiler kazandıran tüm hocalarımıza saygı ve sevgilerimizi sunarız.

Tarih **Ad ve Soyadlar**

Yağmur Kahya

1. GİRİŞ

Bu projede, görüntüde bulunan trafik işaretlerini farklı kategorilere ayırabilen derin bir sinir ağı modeli oluşturacağız. Bu model ile tüm otonom araçlar için çok önemli bir görev olan trafik işaretlerini okuyup anlayabiliyoruz.

Kaggle ‘ da bulunan GTSRB - German Traffic Sign Recognition Benchmark dataseti kullanılmıştır.

1. Projenin Tanımı

Yapay Zeka ve teknoloji dünyasında birçok araştırmacı ve Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi gibi büyük şirketler otonom araçlar ve sürücüsüz arabalar üzerinde çalışıyor. Dolayısıyla bu teknolojide doğruluğa ulaşmak için araçların trafik işaretlerini yorumlayabilmesi ve buna göre kararlar alabilmesi gerekmektedir.

Yolcuların seyahat için tamamen araçlara güvenebilmesi için otonom araçların trafik işaretlerini, kurallarını anlaması ve bunlara uyması gerekir.

Bu projede otonom araçların bu trafik işaretlerini anlaması için trafik işaretleri sınıflandırma işleminin yapılması amaçlanmıştır.

2. MATERYAL VE YÖNTEM

2.1. Proje Gereksinimleri

Bu projede Keras, Matplotlib, Scikit-learn, Pandas, PIL ve görüntü sınıflandırması kullanılmıştır.

GTSRB - German Traffic Sign Recognition Benchmark dataseti kullanılmıştır.

2.2.Keras

Keras, neredeyse her tür derin öğrenme modelini tanımlamak ve eğitmek için uygun bir yol sağlayan Python için bir derin öğrenme kütüphanesidir. Keras, **Tensorflow** , **Theano** ve **CNTK** üzerinde çalışabilen Python ile yazılmış bir üst düzey sinir ağları API’sıdır.
İçerdiği çok fazla işlevsel fonksiyon sayesinde Keras kolayca bir derin öğrenme modeli oluşturmamızı ve onu eğitmemizi sağlıyor. Bu nedenle derin öğrenmeye yeni başlayanlara önerilen kütüphanelerin başında Keras geliyor.

2.3.Matplotlib

Matplotlib; veri görselleştirmesinde kullandığımız temel python kütüphanesidir. 2 ve 3 boyutlu çizimler yapmamızı sağlar. Matplotlib genelde 2 boyutlu çizimlerde kullanılırken, 3 boyutlu çizimlerde başka kütüphanelerden yararlanılır.

2.4.Scikit-Learn

Scikit-learn, veri bilimi ve machine learning için en yaygın kullanılan Python paketlerinden biridir. Birçok işlemi gerçekleştirmenizi sağlar ve çeşitli algoritmalar sağlar. Scikit-learn ayrıca sınıfları, yöntemleri ve işlevleri ile kullanılan algoritmaların arka planıyla ilgili belgeler sunar.

2.5.Pandas

Pandas, “ilişkisel” ve “etiketli” verilerle çalışmayı kolay ve sezgisel hale getirmek için tasarlanmış hızlı, esnek ve etkileyici veri yapıları sağlayan bir Python paketidir. Python’da pratik, gerçek dünya veri analizi yapmak için temel yapı taşı olmayı hedefler. Ayriyeten, her dilde mevcut olan en güçlü ve esnek açık kaynak veri analizi / manipülasyon aracı olmak gibi daha geniş bir amaca sahiptir.

2.6.PIL

Python Image Library , yani Python Resim Kütüphanesi, Pythonda image işlemlerini kolayca yapabilmek için geliştirilmiş kütüphanedir .

2.7.Tkinter

Tkinter, Python'un fiili standart GUI (Grafik Kullanıcı Arayüzü) paketidir.

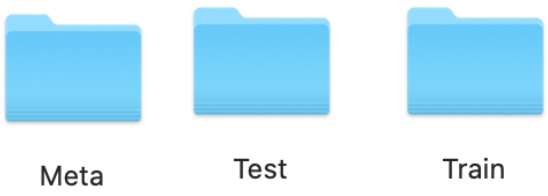
Python, Tkinter ile birlikte kullanıldığında GUI uygulamaları hızlı ve kolay bir şekilde oluşturulur. Tkinter, Tk GUI araç setine güçlü bir nesne yönelimli arayüz sağlar. Python2’de “Tkinter”, Python3’te “tkinter” modülü ile temsil edilir.

Tkinter kullanarak bir GUI uygulaması oluşturmak kolaydır. Tek yapmamız gereken aşağıdaki adımları gerçekleştirmektir;

- Tkinter modülünü uygulamaya import edin.
- GUI uygulama ana penceresini oluşturun.
- GUI uygulamasına yukarıda belirtilen widget’lardan birini veya daha fazlasını ekleyin.
- Kullanıcı tarafından tetiklenen her etkinliğe karşı harekete geçmek için ana etkinlik döngüsünü girin.

3.PROJE KODLARI

3.1.Dataset



Train klasörü her biri farklı bir sınıfı temsil eden 43 klasör içerir. Klasörün aralığı 0 ile 42 arasındadır. İşletim sistemi modülünün yardımıyla, tüm sınıfları yineler ve veri ve etiketler listesine görüntüleri ve ilgili etiketleri ekleriz.

PIL kitaplığı, görüntü içeriğini bir diziye açmak için kullanılır.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
```

```
data = []
labels = []
classes = 43

for i in range(classes):
    path = os.path.join(os.getcwd(),'Train',str(i))
    images = os.listdir(path)

    for a in images:
        try:
            image = Image.open(path + '/'+ a)
            image = image.resize((30,30))
            image = np.array(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")

data = np.array(data)
labels = np.array(labels)

print(data.shape, labels.shape)
X_train, X_test, y_train, y_test= train_test_split(data, labels, test_size=0.2, random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

y_train= to_categorical(y_train,43)
y_test= to_categorical(y_test,43)
```

```
Error loading image
(39209, 30, 30, 3) (39209,)
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

Tüm görüntüleri ve etiketlerini listelere (veriler ve etiketler) kaydettik.

Modeli beslemek için listeyi numpy dizilerine dönüştürmemiz gerekiyor.

Verinin şekli (39209, 30, 30, 3), yani 30×30 piksel boyutunda 39.209 görüntü olduğu ve son 3'ü verinin renkli görüntüler (RGB değeri) içerdiği anlamına gelir.

Sklearn paketiyle, eğitim ve test verilerini bölmek için train_test_split () yöntemini kullanıyoruz.

Keras.utils paketinden, y_train ve t_test'te bulunan etiketleri one-hot encodinge dönüştürmek için to_categorical yöntemini kullanıyoruz.

3.2.CNN Model

Görüntüleri kendi kategorilerine göre sınıflandırmak için bir CNN modeli oluşturacağız.
Modelin mimarisi;

- 2 Conv2D layer (filter=32, kernel_size=(5,5), activation="relu")
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- 2 Conv2D layer (filter=64, kernel_size=(3,3), activation="relu")
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- Flatten layer to squeeze the layers into 1 dimension
- Dense Fully connected layer (256 nodes, activation="relu")
- Dropout layer (rate=0.5)
- Dense layer (43 nodes, activation="softmax")

Modeli, iyi performans gösteren Adam optimizer ile derliyoruz ve kategorize edilecek birden fazla sınıfımız olduğu için kayıp "categorical_crossentropy" oluyor.

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))

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

3.3.Modelin Eğitilmesi

Model mimarisini oluşturduktan sonra modeli **model.fit()** kullanarak eğitiriz.

```
epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
model.save("my_model.h5")
```

```
Epoch 1/15
981/981 [==============================] - 72s 73ms/step - loss: 3.5436 - accuracy: 0.3062 - val_loss: 0.7043 - val
_accuracy: 0.8373
Epoch 2/15
812/981 [=======================>......] - ETA: 11s - loss: 1.0100 - accuracy: 0.7044
```

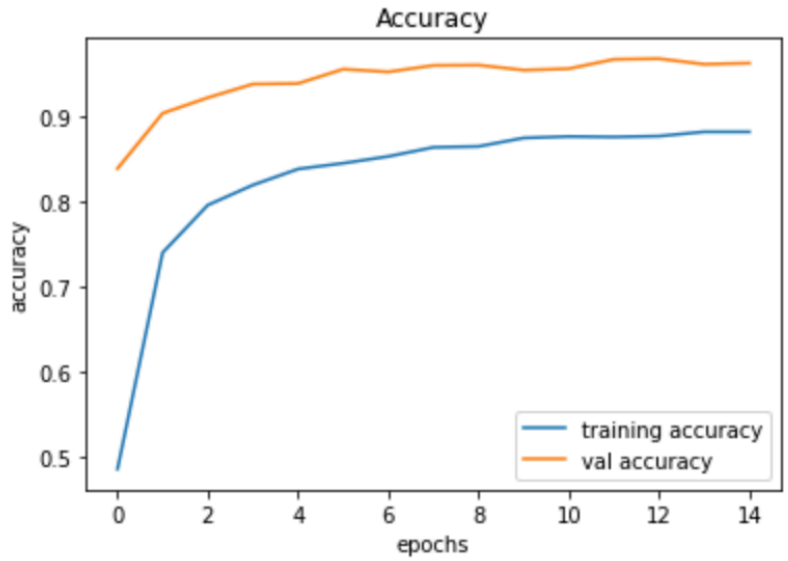
```
Epoch 4/15
981/981 [==============================] - 70s 72ms/step - loss: 0.5390 - accuracy: 0.8406 - val_loss: 0.1778 - val
_accuracy: 0.9500
Epoch 5/15
981/981 [==============================] - 70s 72ms/step - loss: 0.4231 - accuracy: 0.8723 - val_loss: 0.1568 - val
_accuracy: 0.9541
Epoch 6/15
981/981 [==============================] - 66s 67ms/step - loss: 0.4040 - accuracy: 0.8830 - val_loss: 0.1639 - val
_accuracy: 0.9501
Epoch 7/15
981/981 [==============================] - 66s 67ms/step - loss: 0.3627 - accuracy: 0.8925 - val_loss: 0.1116 - val
_accuracy: 0.9688
Epoch 8/15
981/981 [==============================] - 75s 76ms/step - loss: 0.3330 - accuracy: 0.9023 - val_loss: 0.1247 - val
_accuracy: 0.9651
Epoch 9/15
981/981 [==============================] - 70s 72ms/step - loss: 0.3272 - accuracy: 0.9046 - val_loss: 0.0835 - val
_accuracy: 0.9776
Epoch 10/15
981/981 [==============================] - 70s 71ms/step - loss: 0.3016 - accuracy: 0.9127 - val_loss: 0.0951 - val
_accuracy: 0.9739
Epoch 11/15
981/981 [==============================] - 70s 71ms/step - loss: 0.2994 - accuracy: 0.9142 - val_loss: 0.0897 - val
_accuracy: 0.9750
Epoch 12/15
981/981 [==============================] - 65s 66ms/step - loss: 0.2650 - accuracy: 0.9256 - val_loss: 0.0716 - val
_accuracy: 0.9791
Epoch 13/15
981/981 [==============================] - 64s 65ms/step - loss: 0.2699 - accuracy: 0.9263 - val_loss: 0.0881 - val
_accuracy: 0.9762
Epoch 14/15
981/981 [==============================] - 69s 70ms/step - loss: 0.2368 - accuracy: 0.9312 - val_loss: 0.0679 - val
_accuracy: 0.9809
Epoch 15/15
981/981 [==============================] - 72s 73ms/step - loss: 0.2496 - accuracy: 0.9302 - val_loss: 0.0914 - val
_accuracy: 0.9746
```

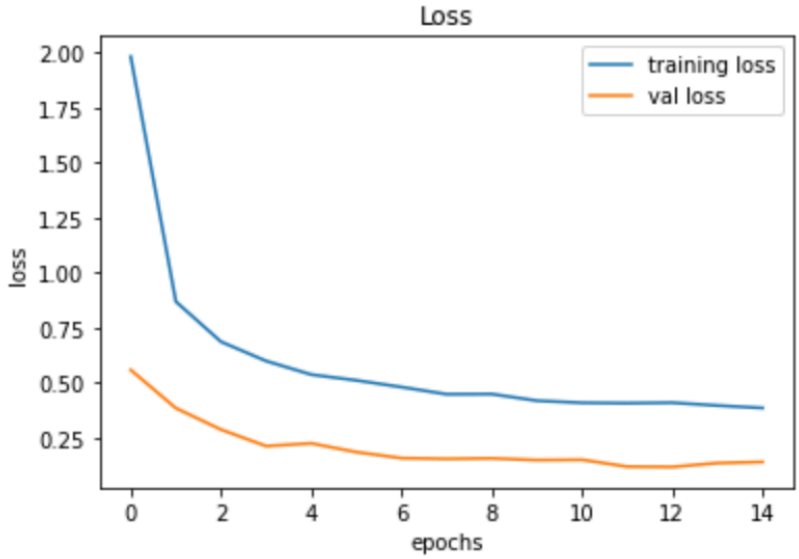
Modelimiz eğitim veri setinde %93 doğruluk elde etti. Matplotlib ile, doğruluk ve kayıp için grafiği çiziyoruz.

In [5]:

```
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

plt.figure(1)
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```





3.4.Modelin Test Edilmesi

Veri kümemiz bir test klasörü içerir ve bir test.csv dosyasında görüntü yolu ve ilgili sınıf etiketleriyle ilgili ayrıntılara sahibiz.

Pandas kullanarak görüntü yolunu ve etiketleri çıkarıyoruz.

Daha sonra modelin tahmini için resimlerimizi 30×30 piksel olarak yeniden boyutlandırmamız ve tüm resim verilerini içeren bir dizi oluşturmamız gerekiyor.

sklearn.metrics'den accuracy_score ‘u import ettik ve modelimizin gerçek etiketleri nasıl tahmin ettiğini gözlemledik. Bu modelde %93 doğruluk elde ettik.

```
:
```

```
from sklearn.metrics import accuracy_score
y_test = pd.read_csv('Test.csv')
labels = y_test["ClassId"].values
imgs = y_test["Path"].values
data=[]
for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))
X_test=np.array(data)
pred = model.predict_classes(X_test)
#Accuracy with the test data
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))
model.save("traffic_classifier.h5")
```

```
/Users/yagmurkahya/opt/anaconda3/lib/python3.8/site-packages/tensorflow/python/kera
Warning: `model.predict_classes()` is deprecated and will be removed after 2021-01-
max(model.predict(x), axis=-1)`,   if your model does multi-class classification
t-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model d
. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
0.9361836896278701
```

Son olarak Keras **model.save()** fonksiyonu ile eğittiğimiz modeli kaydediyoruz.

4.Traffic Signs Classifier GUI

Şimdi Tkinter ile trafik işaretleri sınıflandırıcımız için bir grafik kullanıcı arayüzü oluşturacağız. Tkinter, standart python kitaplığındaki bir GUI araç takımıdır.

In [1]:
```
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import numpy

from keras.models import load_model
model = load_model('/Users/yagmurkahya/Desktop/PROJE2/trafiksigns/traffic_classifier.h5')
```

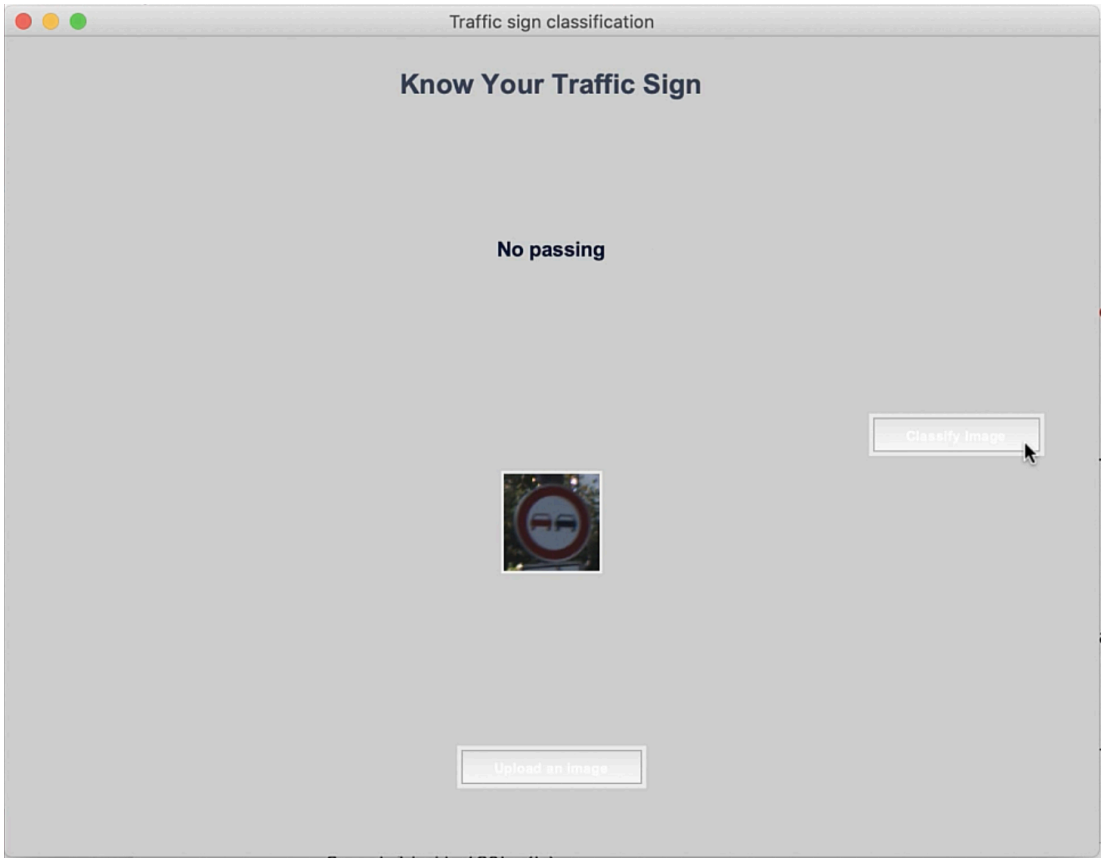
```
classes = { 1:'Speed limit (20km/h)',
            2:'Speed limit (30km/h)',
            3:'Speed limit (50km/h)',
            4:'Speed limit (60km/h)',
            5:'Speed limit (70km/h)',
            6:'Speed limit (80km/h)',
            7:'End of speed limit (80km/h)',
            8:'Speed limit (100km/h)',
            9:'Speed limit (120km/h)',
            10:'No passing',
            11:'No passing veh over 3.5 tons',
            12:'Right-of-way at intersection',
            13:'Priority road',
            14:'Yield',
            15:'Stop',
            16:'No vehicles',
            17:'Veh > 3.5 tons prohibited',
            18:'No entry',
            19:'General caution',
            20:'Dangerous curve left',
            21:'Dangerous curve right',
            22:'Double curve',
            23:'Bumpy road',
            24:'Slippery road',
            25:'Road narrows on the right',
            26:'Road work',
            27:'Traffic signals',
            28:'Pedestrians',
            29:'Children crossing',
            30:'Bicycles crossing',
            31:'Beware of ice/snow',
            32:'Wild animals crossing',
            33:'End speed + passing limits',
            34:'Turn right ahead',
            35:'Turn left ahead',
            36:'Ahead only',
            37:'Go straight or right',
            38:'Go straight or left',
```

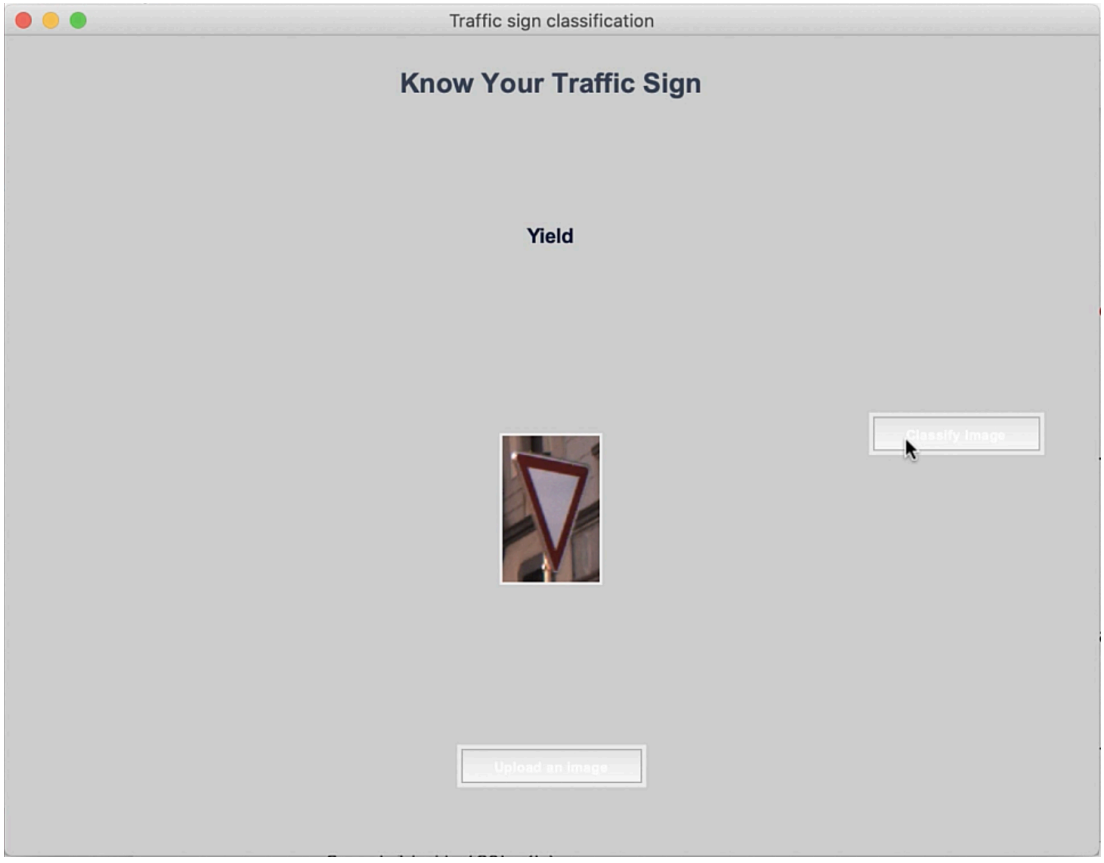
```
            24:'Slippery road',
            25:'Road narrows on the right',
            26:'Road work',
            27:'Traffic signals',
            28:'Pedestrians',
            29:'Children crossing',
            30:'Bicycles crossing',
            31:'Beware of ice/snow',
            32:'Wild animals crossing',
            33:'End speed + passing limits',
            34:'Turn right ahead',
            35:'Turn left ahead',
            36:'Ahead only',
            37:'Go straight or right',
            38:'Go straight or left',
            39:'Keep right',
            40:'Keep left',
            41:'Roundabout mandatory',
            42:'End of no passing',
            43:'End no passing veh > 3.5 tons' }

top=tk.Tk()
top.geometry('800x600')
top.title('Traffic sign classification')
top.configure(background='#CDCDCD')
label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))
sign_image = Label(top)
```

```
def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((30,30))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    pred = model.predict_classes([image])[0]
    sign = classes[pred+1]
    print(sign)
    label.configure(foreground='#011638', text=sign)
def show_classify_button(file_path):
    classify_b=Button(top,text="Classify Image",command=lambda: classify(file_path),padx=10,pady=5)
    classify_b.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)
def upload_image():
    try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((top.winfo_width()/2.25),(top.winfo_height()/2.25)))
        im=ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass
upload=Button(top,text="Upload an image",command=upload_image,padx=10,pady=5)
upload.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="Know Your Traffic Sign",pady=20, font=('arial',20,'bold'))
heading.configure(background='#CDCDCD',foreground='#364156')
heading.pack()
top.mainloop()
```

İlk olarak Keras kullanarak eğitilmiş "traffic_classifier.h5" modelini yükledik.
Ardından görüntüyü yüklemek için GUI'yi oluşturuyoruz ve classify() işlevini çağıranları sınıflandırmak için bir düğme kullanılıyor.
classify() işlevi, görüntüyü şekil boyutuna (1, 30, 30, 3) dönüştürüyor. Bunun nedeni, trafik işaretini tahmin etmek için modeli oluştururken kullandığımız boyutun aynısını sağlamamız gerektiğidir.
Sonra sınıfı tahmin ederiz, model.predict_classes(image) bize ait olduğu sınıfı temsil eden (0-42) arasında bir sayı verir. Sınıf hakkında bilgi almak için sözlüğü kullanırız.


Traffic sign classification
Know Your Traffic Sign
No passing
Classify Image
Upload an Image


Traffic sign classification
Know Your Traffic Sign
Yield
Classify Image
Upload an Image

5. TARTIŞMA ve SONUÇ

Bu Python projesinde, trafik işaretleri sınıflandırıcısı için bir CNN modeli oluşturduk %93 başarıyla sınıflandırdık ve doğruluk ve kaybımızın zamanla nasıl değiştiğini görselleştirdik. Tkinter GIU ile sınıflandırma işlemi için arayüz oluşturduk, eğittiğimiz CNN modelini ekledik.

6.KAYNAKÇA

HTTPS://WWW.KAGGLE.COM/GTSRB - GERMAN TRAFFIC SIGN RECOGNITION BENCHMARK

HTTPS://WWW.KAGGLE.COM/MARINOVIK/RECOGNIZING-TRAFFIC-SIGNALS-WITH-KERAS-CNN

HTTPS://TOWARDSDATASCIENCE.COM/TRAFFIC-SIGN-RECOGNITION-USING-DEEP-NEURAL-NETWORKS-6ABDB51D8B70

HTTPS://ERDINCUZUN.COM/ILERI-PYTHON/TKINTER-GUI/