



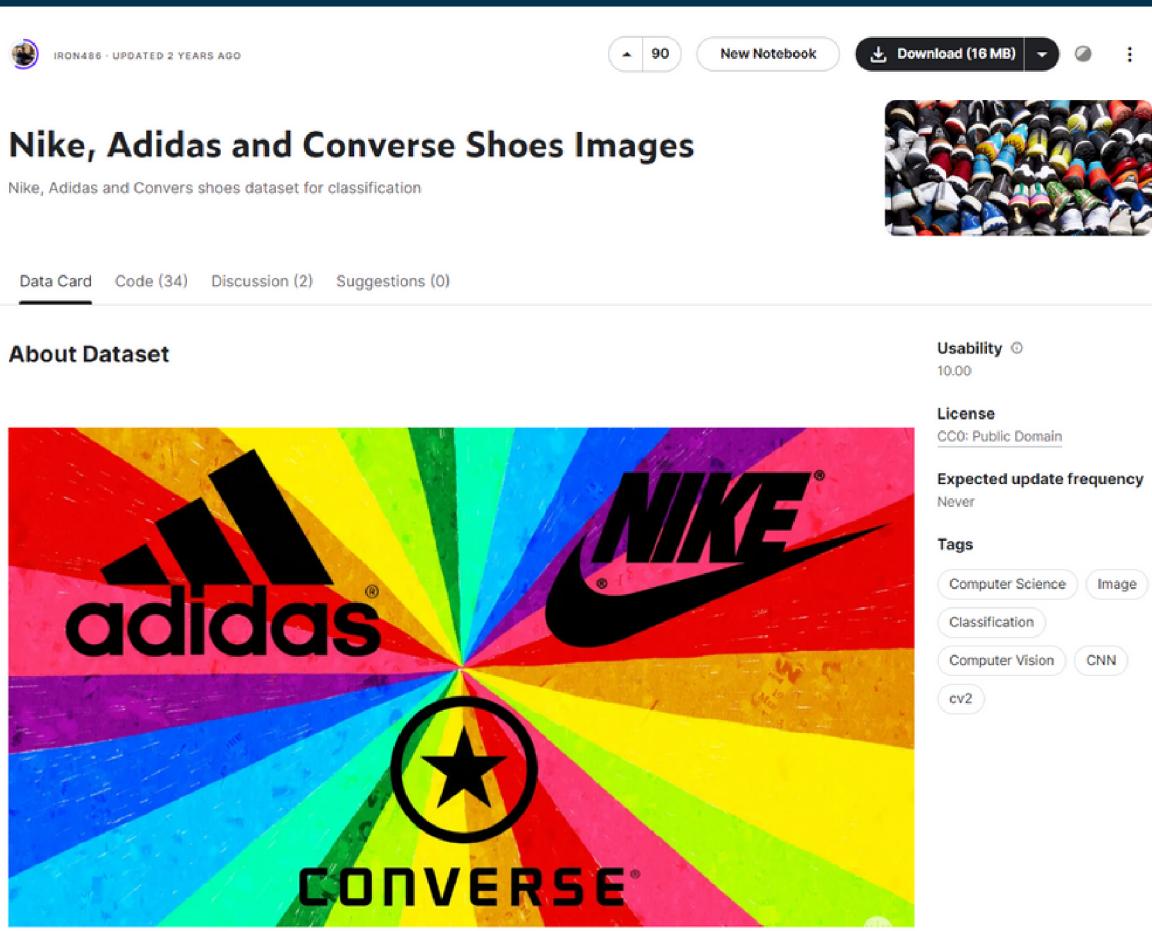
# Sneak-AI

## Nike mi? Adidas mi?

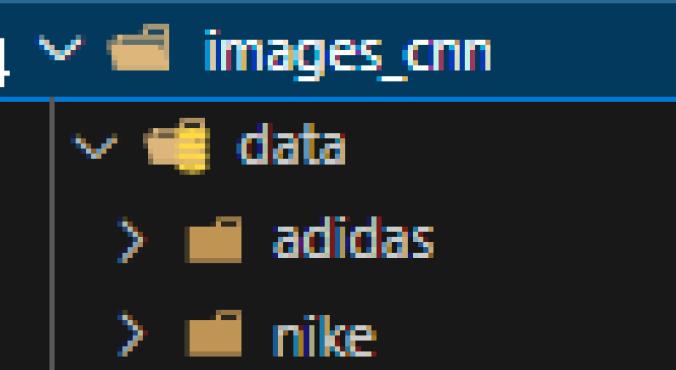
# Veri Seti Seçimi

Kaggle'dan model oluşturabilecek nitelikte görselle sahip olan bir veri seti seçildi.

Sneak-AI projesinde kullandığım veri setinin linki



- Doğruluk oranının daha yüksek olmasını istediğim için Converse marka ayakkabı görsellerini veri setinden çıkardım.
- Veri seti train ve test olarak ayrıldı.
- Data/Adidas klasörünün içerisinde 244 görsel yer almaktadır.
- Data/Nike klasörünün içerisinde 244 görsel yer almaktadır.
- Ayrıca modeli test etmek için hem arama motorundan indirilen hem de test klasörünün içerisinde spor ayakkabı görselleri bulunmaktadır.



# Model Seçimi ve Eğitimi

- deep\_learning isimli sanal ortam ve requirements.txt dosyasındaki kütüphanelerin kurulumu gerçekleştirildi.

## 1. Model Oluşturma:

- Sequential: Model katmanlarını sıralı bir şekilde eklemek için kullanılır. Her katmanın çıktısı, bir sonraki katmanın girdisi olur.

## 2. Evrişim Katmanları (Conv2D):

- İlk katman 32 filtre (özellik çıkarıcı) kullanır. Her filtre 3x3 boyutundadır ve aktivasyon fonksiyonu olarak ReLU kullanılır. Giriş resmi 240x240 boyutunda ve RGB olduğu için 3 kanallıdır.
- Daha sonraki katmanlarda filtre sayısı arttırlarak 64 ve 128 olarak ayarlanmıştır, bu, modelin daha karmaşık özelliklerini öğrenmesine yardımcı olacaktır.

```
import tensorflow as tf
import pandas as pd
import sklearn
import PIL
import psutil
import fastapi
import uvicorn
import pydantic

print("Kütüphane kurulumları başarılı!")

[28] ✓ 1.7s
...
Kütüphane kurulumları başarılı!
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
model = Sequential()
model.add(
    Conv2D(# 1st layer
        filters=32, # filtre sayısı yani resimdeki özellik sayısı
        kernel_size=(3, 3), # filtre boyutu
        activation='relu', # aktivasyon fonksiyonu
        input_shape=(240, 240, 3) # giriş şekli 3 kanallı 240x240 piksel resim (RGB) eğer siyah beyaz olsaydı 1 olacaktı
    )
)

model.add(
    MaxPooling2D( # 2nd layer
        pool_size=(2, 2) # max pooling boyutu
    )
)

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu')) # 3rd layer
model.add(MaxPooling2D(pool_size=(2, 2))) # 4th layer
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu')) # 5th layer

model.add(Flatten()) # düzleştirme
model.add(Dense(units=128, activation='relu'))
model.add(Dropout(0.5)) # overfitting önlemek için dropout
model.add(Dense(units=256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=1, activation='sigmoid')) # çıkış katmanı

model.summary()

[29] ✓ 0.3s
```

## 3. Havuzlama Katmanları (MaxPooling2D):

- Havuzlama katmanları, giriş boyutunu küçültmek ve önemli özellikleri korumak için kullanılır. Bu kodda 2x2 havuz boyutu kullanılmıştır.

## 4. Düzleştirme Katmanı (Flatten):

- Evrişim ve havuzlama katmanlarından çıkan çok boyutlu özellik haritalarını tek boyutlu bir vektöre dönüştürür. Bu, tam bağlantılı katmanlar için gerekli bir adımdır.

## 5. Tam Bağlantılı Katmanlar (Dense):

- İlk tam bağlantılı katmanda 128, ikincisinde 256 nöron bulunur. Her ikisi de ReLU aktivasyon fonksiyonunu kullanır.

## 6. Dropout Katmanları:

- Aşırı öğrenmeyi (overfitting) önlemek için kullanılan Dropout katmanları, eğitim sırasında rastgele seçilen nöronları aktif olmaktan çıkarır. İlk katmanda %50, ikincisinde %20 oranında dropout uygulanmıştır.

## 7. Çıkış Katmanı:

- Tek bir nöron içerir ve aktivasyon fonksiyonu olarak sigmoid kullanılır. Bu genellikle ikili sınıflandırma görevleri için kullanılır, çünkü çıktı 0 ile 1 arasında bir değerdir.

## 8. Model Özeti:

- model.summary(): Modeldeki tüm katmanları ve parametre sayılarını gösterir, modelin nasıl bir yapıya sahip olduğunu görsel olarak anlamayı kolaylaştırır.

# CNN Model Derleme

Compile (Derleme) aşamasında TensorFlow ve Keras kütüphaneleri kullanılarak yazılan aşağıdaki kod, modelin nasıl optimize edileceğini, kayıp (loss) fonksiyonunu ve değerlendirme metriklerini tanımlar.

## Model Özeti

Model: "sequential_3"		
Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 238, 238, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 119, 119, 32)	0
conv2d_10 (Conv2D)	(None, 117, 117, 64)	18496
max_pooling2d_7 (MaxPooling2D)	(None, 58, 58, 64)	0
conv2d_11 (Conv2D)	(None, 56, 56, 128)	73856
flatten_3 (Flatten)	(None, 401408)	0
dense_9 (Dense)	(None, 128)	51380352
dropout_6 (Dropout)	(None, 128)	0
dense_10 (Dense)	(None, 256)	33024
dropout_7 (Dropout)	(None, 256)	0
...		
Total params: 51506881 (196.48 MB)		
Trainable params: 51506881 (196.48 MB)		
Non-trainable params: 0 (0.00 Byte)		

```
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
```



0.0s

### 1. Optimize Edici (Optimizer):

- adam: Adam optimize edici, öğrenme hızını ve diğer parametreleri adaptif bir şekilde ayarlayarak modelin eğitim sürecini optimize eder. Çoğu durumda, genel amaçlı bir optimizasyon yöntemi olarak tercih edilir.

### 2. Kayıp Fonksiyonu (Loss Function):

- binary\_crossentropy: İkili sınıflandırma görevlerinde kullanılan bir kayıp fonksiyonudur. Modelin tahminlerinin, gerçek değerlerle olan farkını ölçmek için kullanılır.

### 3. Metrikler (Metrics):

- ["accuracy"]: Modelin performansını değerlendirmek için kullanılan metrik. Doğruluk (accuracy), modelin tahminlerinin ne kadarının doğru olduğunu yüzdesel olarak ifade eder.

# Veri Çoğaltma ve Ön İşleme

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
  
data_path='data'  
  
train_datagen = ImageDataGenerator(  
    rescale=1./255, # resim değerlerini 0-1 arasına çekme  
    shear_range=0.2, # kesme açısı  
    zoom_range=0.2, # yakınlaştırma  
    horizontal_flip=True # yatay çevirme  
)  
  
test_datagen = ImageDataGenerator(  
    rescale=1./255, # resim değerlerini 0-1 arasına çekme  
    validation_split = 0.2 # doğrulama verisi oranı  
)  
  
train_generator = train_datagen.flow_from_directory(  
    data_path, # veri yolu  
    target_size=(240, 240), # resim boyutu  
    batch_size = 32, # her seferinde kaç resim alınacağı  
    subset='training', # eğitim verisi  
    class_mode='binary' # sınıflandırma türü  
)  
  
validation_generator = test_datagen.flow_from_directory(  
    data_path,  
    target_size=(240, 240),  
    batch_size=32,  
    subset='validation', # doğrulama verisi  
    class_mode='binary'  
)  
✓ 0.0s  
  
Found 488 images belonging to 2 classes.  
Found 96 images belonging to 2 classes.
```

## 1. Veri Yolu Tanımlama

- `data_path='data'`: Eğitim ve doğrulama görüntülerinin saklandığı dizini belirtir.

## 2. ImageDataGenerator Kullanımı:

### Eğitim Veri Üreteci (train\_datagen):

- `rescale=1./255`: Görüntü piksel değerlerini 0-1 arasına ölçeklendirir, böylece modelin daha iyi öğrenmesine yardımcı olur.
- `shear_range=0.2`: Görüntüye belirli bir miktar kesme uygular.
- `zoom_range=0.2`: Görüntüleri rastgele bir oranda yakınlaştırır veya uzaklaştırır.
- `horizontal_flip=True`: Görüntüyü yatay eksende rastgele çevirir.

### Test Veri Üreteci (test\_datagen):

- `rescale=1./255`: Eğitim veri üreteci ile aynı ölçeklendirme işlemini uygular, böylece eğitim ve doğrulama verileri aynı şekilde işlenmiş olur.
- `validation_split=0.2`: Toplam veri setinin %20'sini doğrulama verisi olarak ayırır.

## 3. Veri Yükleme:

### Eğitim Veri Üreteci:

- `train_generator = train_datagen.flow_from_directory(...)`: Veri yolu, hedef boyut, toplu işlem boyutu, alt küme türü ('training') ve sınıf modu ('binary') belirlerek eğitim veri seti oluşturulur.

### Doğrulama Veri Üreteci:

- `validation_generator = test_datagen.flow_from_directory(...)`: Eğitim için kullanılan parametrelerle benzer şekilde doğrulama veri seti oluşturulur, ancak 'validation' alt kümesi kullanılır.

## 4. Sonuç Çıktıları:

- Eğitim için kullanılacak toplam 488 görüntü ve doğrulama için 96 görüntü olduğunu belirtir. Her iki durumda da görüntüler iki sınıfa ayrılmıştır.

# Modelin Eğitimi ve Kaydetme Aşaması

## Model Eğitimi

### 1. train\_generator:

- Modelin eğitim için kullanacağı veri üretici. Bu üreteç, belirli bir klasörden otomatik olarak resimleri yükler ve önişleme adımlarını uygular.

### 2. epochs=10:

- Modelin eğitim veri seti üzerinden 10 tam geçiş yapmasını sağlar. Her geçiş, modelin ağırlıklarını güncellerek öğrenme sürecini ilerletir.

### 3. validation\_data=validation\_generator:

- ["accuracy"]: Modelin performansını değerlendirmek için kullanılan metrik. Doğruluk (accuracy), modelin tahminlerinin ne kadarının doğru olduğunu yüzdesel olarak ifade eder.

10 Epochlu eğitim modelinde doğruluk oranları %90'larda olduğu için daha fazla tekrar sayılı eğitim yapılmadı.

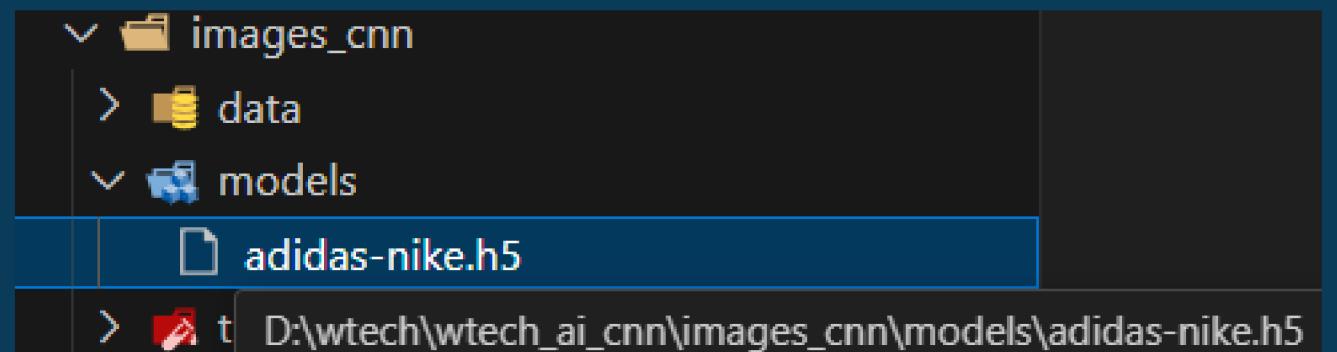
```
history=model.fit(train_generator,epochs=10,validation_data=validation_generator)
✓ 4m 10.6s

Epoch 1/10
16/16 [=====] - 25s 2s/step - loss: 0.2426 - accuracy: 0.8893 - val_loss: 0.1355 - val_accuracy: 0.9375
Epoch 2/10
16/16 [=====] - 25s 2s/step - loss: 0.2482 - accuracy: 0.9016 - val_loss: 0.1179 - val_accuracy: 0.9479
Epoch 3/10
16/16 [=====] - 24s 1s/step - loss: 0.2606 - accuracy: 0.9016 - val_loss: 0.1283 - val_accuracy: 0.9375
Epoch 4/10
16/16 [=====] - 24s 1s/step - loss: 0.2468 - accuracy: 0.8934 - val_loss: 0.1341 - val_accuracy: 0.9479
Epoch 5/10
16/16 [=====] - 24s 2s/step - loss: 0.2149 - accuracy: 0.9078 - val_loss: 0.1574 - val_accuracy: 0.9375
Epoch 6/10
16/16 [=====] - 24s 1s/step - loss: 0.3056 - accuracy: 0.8750 - val_loss: 0.1422 - val_accuracy: 0.9583
Epoch 7/10
16/16 [=====] - 25s 2s/step - loss: 0.2158 - accuracy: 0.9201 - val_loss: 0.1073 - val_accuracy: 0.9688
Epoch 8/10
16/16 [=====] - 26s 2s/step - loss: 0.2152 - accuracy: 0.8975 - val_loss: 0.1442 - val_accuracy: 0.9167
Epoch 9/10
16/16 [=====] - 25s 2s/step - loss: 0.3630 - accuracy: 0.8832 - val_loss: 0.1078 - val_accuracy: 0.9583
Epoch 10/10
16/16 [=====] - 26s 2s/step - loss: 0.3028 - accuracy: 0.8730 - val_loss: 0.1476 - val_accuracy: 0.9271
```

## Model Kaydetme

```
model.save('models/adidas-nike.h5')
✓ 0.9s
d:\wtech\wtech_ai_cnn\deep_learning\Lib\site-packages\keras\src\engine\training.py:
    saving_api.save_model()
```

- Bu aşamada, eğitilmiş model HDF5 dosya formatında kaydedilir. Dosya adı 'models/adidas-nike.h5' şeklindedir. Bu, modelin daha sonra yeniden yüklenip kullanılabilmesi veya başka bir sistemde kullanılabilmesi için bu aşama oldukça önemlidir.



# Test Aşamaları

## 01 MODELİ VE TEST GÖRSELİ YÜKLEME

```
from tensorflow.keras.models import load_model  
model = load_model('models/adidas-nike.h5')  
  
import numpy as np  
from tensorflow.keras.preprocessing import image  
  
test_image = image.load_img('furyosa.jpg', target_size=(240, 240)) # test resmi yükle  
test_image
```



## 02 GÖRSELİ DİZİYE ÇEVİRME

```
test_image = image.img_to_array(test_image) # resmi diziye çevir  
print(test_image.shape)  
print(test_image)  
✓ 0.0s  
(240, 240, 3)  
[[[242. 242. 242.]  
 [242. 242. 242.]  
 [242. 242. 242.]  
 ...  
 [242. 242. 242.]  
 [242. 242. 242.]  
 [242. 242. 242.]]  
  
[[242. 242. 242.]  
 [242. 242. 242.]  
 [242. 242. 242.]  
 ...  
 [242. 242. 242.]  
 [242. 242. 242.]  
 [242. 242. 242.]]  
  
[[242. 242. 242.]  
 [242. 242. 242.]  
 [242. 242. 242.]  
 ...  
 [242. 242. 242.]  
 [242. 242. 242.]  
 [242. 242. 242.]]  
...  
...  
[242. 242. 242.]  
[242. 242. 242.]  
[242. 242. 242.]]]
```

# Test Aşamaları

03

BOYUT EKLEME VE 0-1 ARASINA ÇEKME

```
> ^
    test_image = np.expand_dims(test_image, axis=0) / 255.0 # boyut ekleyip 0-1 arasına çek
    print(test_image.shape)
[94]   ✓ 0.0s
...   (1, 240, 240, 3)
```

04

TAHMİN YAPMA

```
result = model.predict(test_image) # tahmin yap
print(result)

if result[0][0] > 0.5:
    print('Adidas')
else:
    print('Nike')
[94]   ✓ 0.3s
1/1 [=====] - 0s 186ms/step
[[0.4388455]]
Nike
```

# FastAPI AŞAMASI

The screenshot shows the FastAPI documentation interface for the /cnn endpoint. It includes sections for GET and POST methods, parameters, request body, responses, curl command, request URL, server response, and detailed code examples.

**GET /cnn**

**POST /guess\_images/** Guess Images

Parameters

No parameters

Request body required

file required  
string(\$binary) Dosya Seç furyosa.jpg

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/guess_images/' \
  -H 'Accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@furyosa.jpg;type=image/jpeg'
```

Request URL

http://127.0.0.1:8000/guess\_images/

Server response

Code Details

200 Response body

```
{"final": "Nike"}
```

Download Response headers

```
content-length: 16
content-type: application/json
date: Thu, 02 May 2024 16:05:39 GMT
server: uvicorn
```

Responses

Code Description

200 Successful Response

Media type application/json

Controls Accept header

Example Value Schema

Links No links

```
main.py > ...
1  from fastapi import FastAPI, Request, File, UploadFile
2  from fastapi.responses import HTMLResponse
3  from fastapi.responses import JSONResponse
4  from tensorflow.keras.preprocessing import image
5  from tensorflow.keras.models import load_model
6  import numpy as np
7  from PIL import Image
8
9  app = FastAPI() #uygulamayı başlatır
10
11 model = load_model(r'D:\wtech\wtech_ai_cnn\images_cnn\models\adidas-nike.h5')
12
13 @app.get("/", response_class=HTMLResponse)
14 async def home():
15     return HTMLResponse(open("D:\wtech\wtech_ai_cnn\index.html", "r").read())
16
17 @app.get("/home")
18 def home():
19     return {"message": "CNN Modeli ile Ayakkabı Markası Tahmini Projesi"}
20
21 @app.get("/about") # /about endpointi için GET metodu
22 def about():
23     return {"message": "CNN Modeli ve FastAPI geliştirmesi, Wtech ve IDSA ortaklığında yürütülen PYTHON ve Yapay Zeka eğitimi içerisinde yapılmıştır."}
24
25 @app.get("/cnn", response_class=HTMLResponse)
26 async def cnn(request: Request):
27     return HTMLResponse(open("D:\wtech\wtech_ai_cnn\cnn.html", "r").read())
28
29 @app.post("/guess_images/")
30 async def guess_images(file: UploadFile = File(...)):
31     with open("temp_image.jpg", "wb") as f:
32         f.write(await file.read())
33
34     # Resmi yükle ve işle
35     test_image = Image.open("temp_image.jpg")
36     test_image = test_image.resize((240, 240))
37     test_image = image.img_to_array(test_image)
38     test_image = np.expand_dims(test_image, axis=0) / 255.0
39
40     # Modelle tahmin yap
41     result = model.predict(test_image)
42
43     # Tahmin sonucuna göre sınıf etiketini belirle
44     if result[0][0] > 0.5:
45         guess = "Adidas"
46     else:
47         guess = "Nike"
48
49     # Sonucu JSON olarak döndür
50     return JSONResponse(content={"final": guess})
51
```

# SNEAK-AI WEB SAYFASI

## SNEAK-AI

It was developed using artificial intelligence-supported CNN model

• SNEAKERS RECOGNITION • ABOUT

### Sneak-AI: AI Sneaker Recognition for Nike and Adidas

This project was created as a graduation project within the scope of the "Artificial Intelligence with Python" training of the "100 Women and 25 Men Leaders in Technology During the Centennial of the Republic" project carried out jointly by Wtech Platform and Istanbul Data Science Academy.

This project implements a shoe recognition system using convolutional neural networks (CNNs). The system is trained on a dataset of sneaker images, allowing it to identify and classify sneakers by brand.

Which one is it Nike or Adidas?

Click the button to find out!

Sneakers Recognition

# SNEAK-AI WEB SAYFASI

## SNEAK-AI

It was developed using artificial intelligence-supported CNN model

• SNEAKERS RECOGNITION • ABOUT

Sneakers Image Upload

Choose File



Predict Result

Predict

This is an Adidas brand sneakers.

# Dinlediğiniz için teşekkür ederim

Nursen Özcan



[Github](#)



[LinkedIn](#)

