GÜLBEYAZ BAYRAM ÖZER

GİTHUB REPO: HTTPS://GİTHUB.COM/GULBEYAZZB/İMAGE_CLASSİFİCATİON_CNN

WTECH YAPAY ZEKA EĞİTİMİ BİTİRME PROJESİ

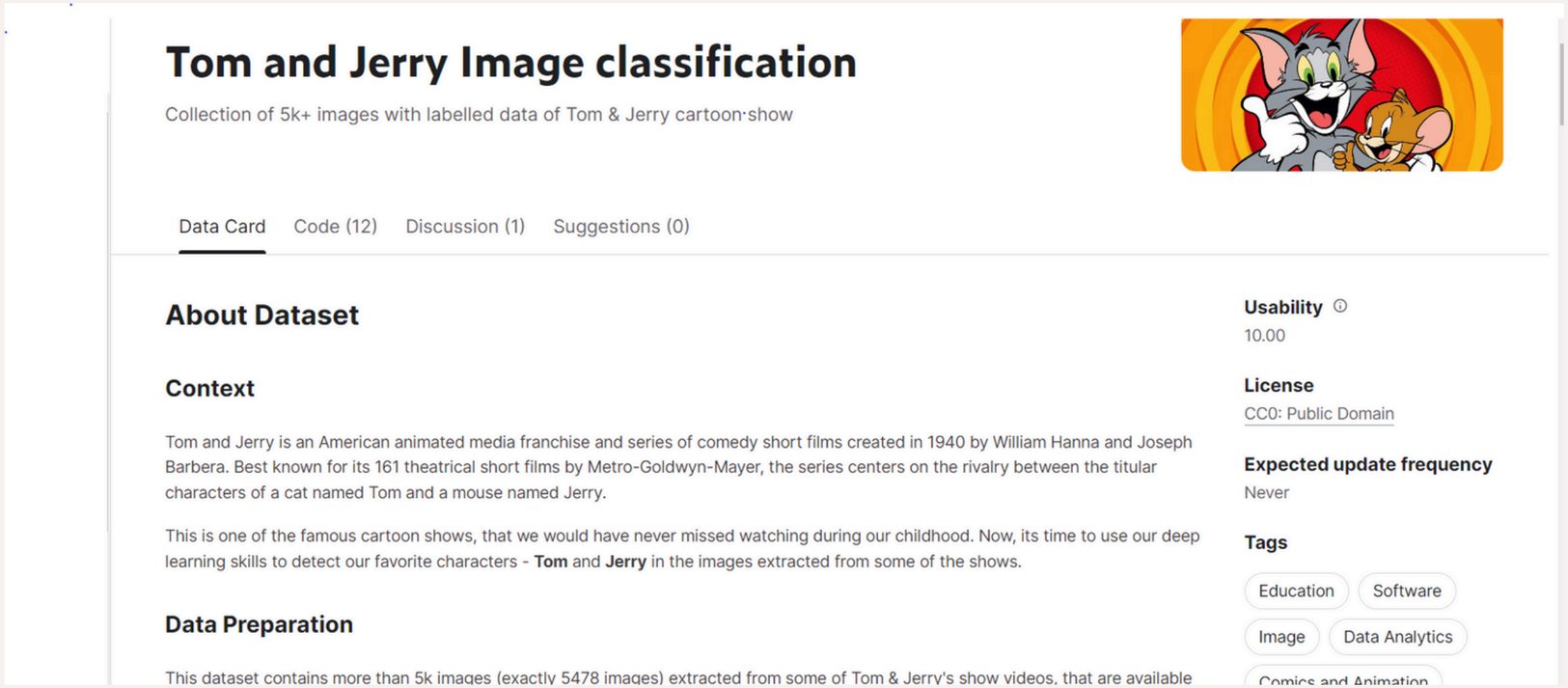
03.05.2024

İÇİNDEKİLER

- Veri Seti Seçimi
- Model Oluşturma
- Compile İşlemi
- Veri Ön İşleme ve Veri Arttırma
- Modelin Eğitilmesi
- FastApi Kodlaması
- API Test

1.VERİ SETİ SEÇİMİ

Veri seti Kaggle’dan indirildi (Resim 1.1).
Kaggle link: https://www.kaggle.com/datasets/balabaskar/tom-and-jerry-image-classification
Veri seti tom ve jerry olmak üzere **iki kategori**de resimler içeriyor; **Tom** kategorisine ait **1,930**, **Jerry** kategorisine ait **1,240** adet resim içeriyor.
Görseller **RGB** formatında.



Resim 1.1. Kaggle’da Veri Seti

2.MODEL OLUŞTURMA

İlgili dataset görsel içerdiği için, görüntü sınıflandırmada tercih edilen Convolutional Neural Network yapısı kullanılmıştır.

```
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation="relu", input_shape=(240, 240, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(filters=256, kernel_size=(3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(units=256, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(units=128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(units=64, activation="relu"))
model.add(Dense(1, activation='sigmoid'))
```

Resim 2.1. Model Oluşturma

Sınıflandırma problemleri için **genellikle son katmanda** sigmoid kullanılırken, **gizli katmanlarda** ReLU gibi aktivasyonlar daha yaygın olarak tercih edilir. Modelin eğitiminde yine bu aktivasyon yapısı tercih edildi.

2.MODEL OLUŞTURMA

Derin öğrenmede filtre **başlangıç** sayısı genel olarak **64** tercih edilir. ve her layerda 2 kat arttırılır. Her katmanda **filtre sayısının artması**, modelin daha *karmaşık* özellikleri **öğrenme kapasitesini arttırır**.

```
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation="relu", input_shape=(240, 240, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(filters=256, kernel_size=(3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(units=256, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(units=128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(units=64, activation="relu"))
model.add(Dense(1, activation='sigmoid'))
```

Resim 2.2. Model Oluşturma

Önceki katmanlardan gelen özellik haritalarının **boyutunu azaltmak** ve böylece **hesaplama maliyetini azaltmak** için kullanılan pooling katmanı için burada tanımlanandan **daha büyük bir pooling boyutu**, daha az özetleme ve **daha fazla bilgi kaybı** anlamına gelir ve **küçük** pooling boyutları, **aşırı uyum riskini azaltabilir. Görsel** veriler için, genellikle **2x2** veya **3x3** boyutunda pooling katmanları tercih edilir. **2x2** boyutunda pooling katmanları oldukça **yaygındır** çünkü veriyi yarıya indirir ve hesaplama maliyeti düşüktür. Tüm bu nedenlerden dolayı 2x2 boyutu tercih edildi.

2.MODEL OLUŞTURMA

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 238, 238, 64)      1792

 max_pooling2d (MaxPooling2  (None, 119, 119, 64)      0
 D)

 conv2d_1 (Conv2D)           (None, 117, 117, 128)     73856

 max_pooling2d_1 (MaxPoolin  (None, 58, 58, 128)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 256)       295168

 max_pooling2d_2 (MaxPoolin  (None, 28, 28, 256)       0
 g2D)

 flatten (Flatten)           (None, 200704)            0

 dense (Dense)               (None, 256)               51380480

 dropout (Dropout)           (None, 256)               0

...
Total params: 51792513 (197.57 MB)
Trainable params: 51792513 (197.57 MB)
Non-trainable params: 0 (0.00 Byte)
```

Resim 2.3. Model Özeti

Dense katmanlarındaki **nöron sayısı(units)** modelin her bir katmanda daha fazla **özellik ve karmaşıklığı öğrenmesini sağlar**. Önceki Conv2D ve MaxPooling2D katmanlarından elde edilen özelliklerin bir temsilini alabilmesi için başlangıç nöron sayısı 256 seçildi.

Dense katmanı aynı zamanda **aşırı uyum riskini de artırabilir**. Bu nedenle, **Dropout** katmanlarını kullanarak aşırı uyumu kontrol altında tutmaya çalışıyoruz.
Genellikle **0.2** veya **0.5** olarak tercih edilir. İlgili modelin eğitiminde de yine 0.5 tercih edildi.

Son katmanın sınıf sayısı 2 olduğu için ve **çoğu ikili sınıflandırma** problemi için **standart** bir yaklaşım olduğu için **son** dense katmanı **nöron sayısı 1** olarak belirlendi.

3.COMPILE İŞLEMİ

```
model.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy", Precision(), Recall()])
✓ 0.0s
```

Resim 3.1. Compile İşlemi Kod Satırı

Modelin, belirli bir probleme nasıl uyum sağlayacağını, hangi kaybın minimize edileceğini ve ne tür bir performans değerlendirmesi yapılacağını tanımlayan aşamadır.

optimizer="adam" yaygın olarak tercih edilen optimizasyon algoritmasıdır.
"binary_crossentropy", ikili sınıflandırma problemleri için yaygın olarak kullanılan bir kayıp fonksiyonudur.
"accuracy" doğruluğu, **"Precision()"** hassasiyeti ve **"Recall()"** duyarlılığı hesaplayan performans metrikleri kullanıldı.

4.VERİ SETİ ÖN İŞLEME VE VERİ ARTTIRMA

```python
data_dir = r'C:\Users\Hp\Documents\GitHub\image_classification_cnn\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_dir, # 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_dir,
    target_size=(240, 240),
    batch_size=32,
    subset='validation', # doğrulama verisi
    class_mode='binary'
)
```

Resim 4.1. Veri Seti Ön işleme

Eğitim veri setinin **çeşitliliğini artırmak** ve modelin **genelleme yeteneğini artırmak** için **ImageDataGenerator** ile **veri artırma** işlemleri gerçekleştirildi.

Piksel değerlerini normalize ederek modelin **eğitimini daha iyi yapmasını** sağlamak için resim **piksel değerlerini 0 ile 1 arasına** ölçekleriz. Bu işlem için **rescale=1./255** yöntemi kullanıldı.

shear_range=0.2(kırpma), zoom_range=0.2(yakınlaştırma), horizontal_flip=True(çevirme) gibi özelliklerle resim arttırıldı.

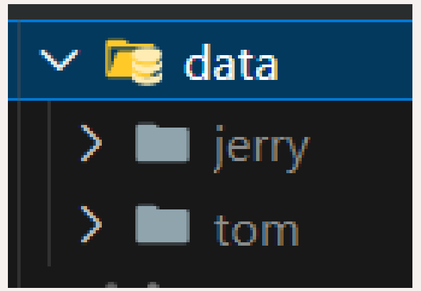
validation_split=0.2 ile Veri setinin **%20**'sini doğrulama verisi olarak ayarlamak için kullanıldı. %20'lik bir doğrulama verisi oranı, genellikle modelin **iyi performans** göstermesini sağlayacak **yeterli sayıda doğrulama örneğ**ini sağlarken, **hesaplama maliyetini** de kontrol altında tutar.

4.VERİ SETİ ÖN İŞLEME VE VERİ ARTTIRMA

```python
data_dir = r'C:\Users\Hp\Documents\GitHub\image_classification_cnn\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_dir, # 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_dir,
    target_size=(240, 240),
    batch_size=32,
    subset='validation', # doğrulama verisi
    class_mode='binary'
)
```

Resim 4.2. Veri Seti Ön işleme

flow_from_directory, veri setini yükler. Veri setinin **her sınıfı bir klasör içinde** bulunmalıdır, ve her **klasörün adı sınıf etiketi** olmalıdır (Resim 4.1).



Resim 4.3. Veri Seti Klasörü

4.VERİ SETİ ÖN İŞLEME VE VERİ ARTTIRMA

```python
data_dir = r'C:\Users\Hp\Documents\GitHub\image_classification_cnn\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_dir, # 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_dir,
    target_size=(240, 240),
    batch_size=32,
    subset='validation', # doğrulama verisi
    class_mode='binary'
)
```

Resim 4.4. Veri Seti Ön işleme

target_size=(240, 240): Resimlerin boyutu 240x240'a ölçeklenir. **input_shape (240,240)** ile genellikle aynı olmalı.

batch_size=32: Her bir eğitim ve doğrulama adımında kaç resmin işleneceğini belirtir. Batch_size büyük seçilirse maliyet, aşırı uyumluluk, bellek tüketimi ve performans açısından olumsuz sonuçlara sebep olabileceğinden küçük denebilecek 32 değeri seçilmiştir.

4.VERİ SETİ ÖN İŞLEME VE VERİ ARTTIRMA

```python
data_dir = r'C:\Users\Hp\Documents\GitHub\image_classification_cnn\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_dir, # 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_dir,
    target_size=(240, 240),
    batch_size=32,
    subset='validation', # doğrulama verisi
    class_mode='binary'
)
```

Resim 4.5. Veri Seti Ön işleme

subset='training' veya 'validation': Veri setinin hangi alt kümesinin kullanılacağını belirtir. 'training', eğitim verisi için kullanılırken, 'validation' doğrulama verisi için kullanılır.

class_mode='binary': Sınıflandırma türünü belirtir; "binary" olduğunda, modelin ikili sınıflandırma yapması beklenir (örneğin, 0 veya 1).

5.MODELİN EĞİTİMİ

```
model.fit(train_generator,epochs=30,validation_data=validation_generator)
```

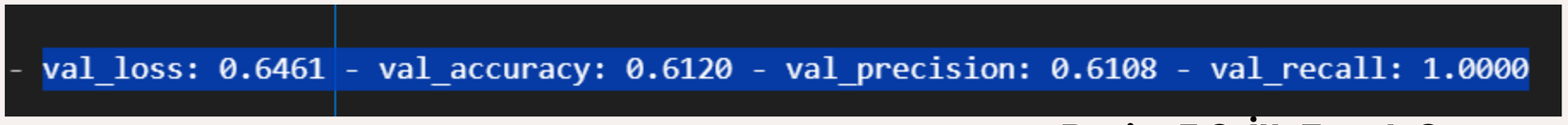
Resim 5.1. Modelin Eğitimi

fit yöntemi, modelin veriye uyum sağlaması için kullanılır ve modelin kayıp fonksiyonunu minimize etmek, belirtilen metrikleri maksimize etmek için ağırlıkları günceller.

epochs 30 tercih edildi çünkü 20 ile yeterli başarı elde edilmedi; 70 ile de çok uyumlu bir başarı elde edildi. 30'nda yeterli olabileceği sonucuna varıldı ve model başarıyla eğitildi.

validation_data, modelin performansını izlemek için kullanılır. Her bir epoch sonunda, model doğrulama veri setindeki performansını ölçer ve eğitim süreci sırasında aşırı uyumu (overfitting) kontrol etmeye yardımcı olur.

5.MODELİN EĞİTİMİ



Resim 5.2. İlk Epoch Sonucu

Resim 5.3. Son Epoch Sonucu

Başlangıçta 0.65 olan loss değeri eğitim sonunda 0.02'ye düşürüldü; accuracy değeri 0.61'den eğitim sonunda 0.99'a yükseltildi.

6. FASTAPI

```python
@app.post("/predict_image")
async def predict_image(file: UploadFile = File(...)):
    load_cnn_model = load_model(r'C:\Users\Hp\Documents\GitHub\image_classification_cnn\fastApi\models\model_cnn.h5')
    with open("uploaded_img.jpg", "wb") as f:
        f.write(await file.read())

    test_image = Image.open("uploaded_img.jpg")      You, 2 seconds ago • Uncommitted changes
    test_image = test_image.resize((240, 240))

    test_image = np.array(test_image)
    test_image = np.expand_dims(test_image, axis=0) / 255.0

    prediction = load_cnn_model.predict(test_image)
    # predicted_class_index = np.argmax(prediction[0][0])

    if prediction[0][0] <= 0.5:
        return 'Jerry'
    else:
        return 'Tom'
```

Resim 6.1. FastApi Kodlama

6. FASTAPI

```python
@app.post("/predict_image")
async def predict_image(file: UploadFile = File(...)):
    load_cnn_model = load_model(r"C:\Users\Hp\Documents\GitHub\image_classification_cnn\fastApi\models\model_cnn.h5")
    with open("uploaded_img.jpg", "wb") as f:
        f.write(await file.read())

    test_image = Image.open("uploaded_img.jpg")      You, 2 seconds ago • Uncommitted changes
    test_image = test_image.resize((240, 240))

    test_image = np.array(test_image)
    test_image = np.expand_dims(test_image, axis=0) / 255.0

    prediction = load_cnn_model.predict(test_image)
    # predicted_class_index = np.argmax(prediction[0][0])

    if prediction[0][0] <= 0.5:
        return 'Jerry'
    else:
        return 'Tom'
```

Resim 6.2. FastApi Kodlama

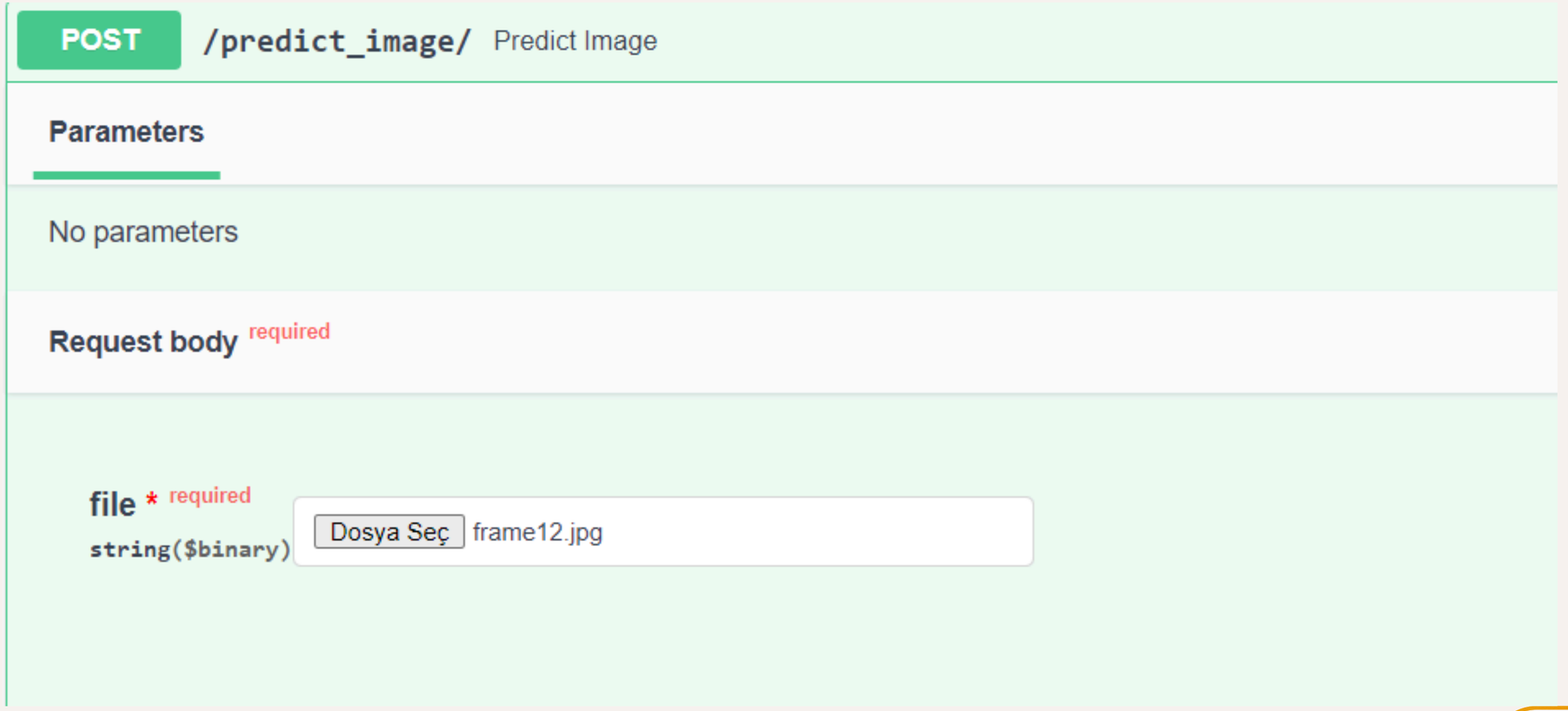
Endpoint, bir görüntü dosyasını (**UploadFile türünde**) alır ve bu görüntüyü tahmin etmek için önceden eğitilmiş bir CNN modelini kullanır.

File(...) varsayılan değer, bu dosyanın **zorunlu** olduğunu belirtir.

with open("uploaded_img.jpg", "wb") as f: ...: Gelen görüntü dosyası **("uploaded_img.jpg" olarak adlandırılır**) diskte **geçici** olarak saklanır. **With** ifadesi, Python'da bir dosya gibi belirli bir kaynağı kullanırken, bu kaynağın kapatılmasını otomatik olarak sağlayan bir yapıdır. Dosya işlemlerinde tercih edilir.

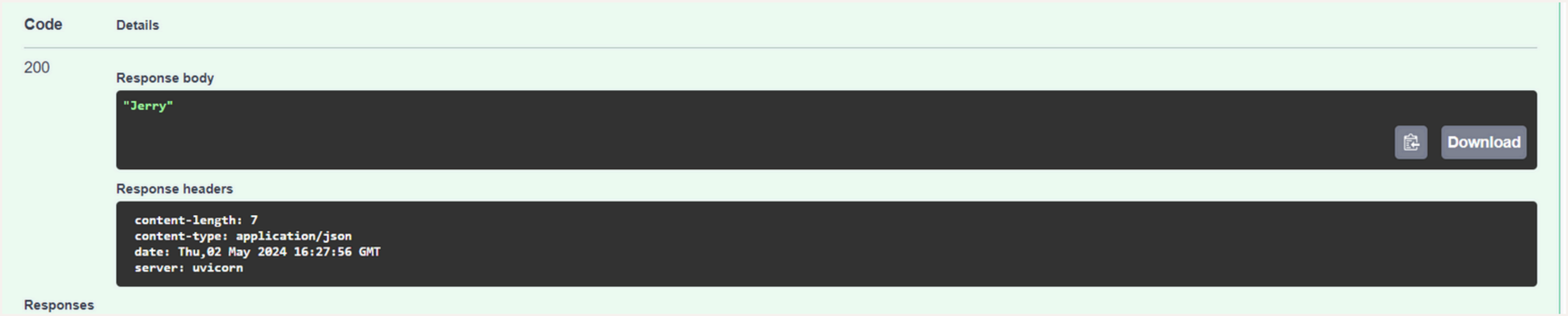
test_image = np.array(test_image): Görüntü numpy dizisine dönüştürülür.
test_image = np.expand_dims(test_image, axis=0) / 255.0: Görüntü, modelin beklentisi olan şekle uyacak şekilde genişletilir ve normalleştirilir.

7. API TEST AŞAMASI



Resim 7.1. İlgili endpointte Resim Dosyası Seçme

7. API TEST AŞAMASI



Resim 7.2 FastApi Jerry'e ait Resim Sonucu

Resim 7.3. FastApi Tom'a ait Resim Sonucu

BONUS (YOLO)

```
from ultralytics import YOLO

model=YOLO("best.pt")
```

✓ 1m 31.3s

```
results=model.predict(source="green.jpg",save=True,verbose=False)
```

✓ 1m 34.8s

```
Results saved to runs\detect\predict14
```

TEŞEKKÜR EDERİM

GÜLBEYAZ BAYRAM ÖZER