Makine Öğrenmesine Giriş

Proje Raporu

Ertuğrul Tuğ
201307001
Bilişim Sistemleri Mühendisliği
Kocaeli Üniversitesi

***Özet*—Bu çalışma, Derin Sinir Ağları (CNN) kullanarak otomatik kedi ırkı sınıflandırma sisteminin geliştirilmesi üzerine odaklanmaktadır. VGG16 mimarisini kullanarak, veri kazıma, ön işleme, model eğitimi ve tahmin gibi kapsamlı bir boru hattı oluşturulmuştur. Model, TensorFlow ve Keras kütüphanelerinden faydalanarak eğitilmiş ve değerlendirilmiştir, hedef ise yüksek doğrulukta sınıflandırma sağlamaktır.**

Anahtar Kelimeler—Kedi Irkı Sınıflandırma, Konvolüsyonel Sinir Ağı, VGG16, TensorFlow, Keras, Veri Artırma

Projemin yer aldığı github [sayfası]

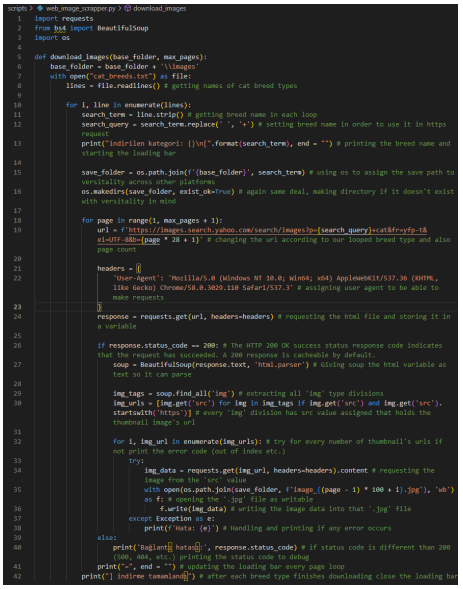
I. Giriş

Derin öğrenme ve bilgisayarla görme alanlarındaki hızlı ilerlemeler, sofistike görüntü sınıflandırma sistemlerinin geliştirilmesine olanak tanımıştır. Bu çalışma, VGG16 mimarisini kullanarak otomatik kedi ırkı sınıflandırma modelinin geliştirilmesine odaklanmaktadır. Model, çeşitli kedi ırklarına ait görüntülerden oluşan bir veri seti üzerinde eğitilmiş olup, ırkları doğru bir şekilde tanımlamayı amaçlamaktadır.

II. Yöntem

A. Veri Toplama ve Ön İşleme

Veri toplama, web_image_scrapper.py kullanılarak gerçekleştirilmiştir. Bu betik, belirli kedi ırklarına ait görüntüleri internetten kazıyarak indirir. İndirilen görüntüler daha sonra duplicate_image_remover.py betiği kullanılarak çoğaltılmış görüntülerin kaldırılması amacıyla işlenmiştir.



```python
import requests
from bs4 import BeautifulSoup
import os

def download_images(base_folder, max_pages):
    base_folder = base_folder + '\\images'
    with open("cat_breeds.txt") as file:
        lines = file.readlines() # getting names of cat breed types

        for i, line in enumerate(lines):
            search_term = line.strip() # getting breed name in each loop
            search_query = search_term.replace(' ', '+') # setting breed name in order to use it in https
            request
            print("indirilen kategori: {}\n".format(search_term), end = "") # printing the breed name and
            starting the loading bar

            save_folder = os.path.join(f'{base_folder}', search_term) # using os to assign the save path to
            uerstiality across other platforms
            os.makedirs(save_folder, exist_ok=True) # again same deal, making directory if it doesn't exist
            with versitality in mind

            for page in range(1, max_pages + 1):
                url = f'https://images.search.yahoo.com/search/images?p={search_query}+cat&fr=yfp-t&
                ei=UTF-8&b={page * 28 + 1}' # changing the url according to our looped breed type and also
                page count

                headers = {
                    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
                    like Gecko) Chrome/58.0.3029.110 Safari/537.3' # assigning user agent to be able to
                    make requests
                }
                response = requests.get(url, headers=headers) # requesting the html file and storing it in
                a variable

                if response.status_code == 200: # The HTTP 200 OK success status response code indicates
                that the request has succeeded. A 200 response is cacheable by default.
                    soup = BeautifulSoup(response.text, 'html.parser') # Giving soup the html variable as
                    text so it can parse

                    img_tags = soup.find_all('img') # extracting all 'img' type divisions
                    img_urls = [img.get('src') for img in img_tags if img.get('src') and img.get('src').
                    startswith('https')] # every 'img' division has src value assigned that holds the
                    thumbnail image's url

                    for i, img_url in enumerate(img_urls): # try for every number of thumbnail's urls if
                    not print the error code (out of index etc.)
                        try:
                            img_data = requests.get(img_url, headers=headers).content # requesting the
                            image from the 'src' value
                            with open(os.path.join(save_folder, f'image_{(page - 1) * 100 + i}.jpg'), 'wb')
                            as f: # opening the '.jpg' file as writable
                                f.write(img_data) # writing the image data into that '.jpg' file
                        except Exception as e:
                            print(f'Hata: {e}') # Handling and printing if any error occurs
                else:
                    print('Bağlantı hatası: ', response.status_code) # if status code is different than 200
                    (500, 404, etc.) printing the status code to debug
                print("-", end = "") # updating the loading bar every page loop
            print("] indirme tamamlandı") # after each breed type finishes downloading close the loading bar
```

Veri kümesinin bütünlüğü dataset_corruption_checker.py kullanılarak kontrol edilmiştir.

```python
from os import listdir
from os.path import join, isdir
from PIL import Image

base_folder = r'C:\Users\ertug\Desktop\Folder\makine_ogrenmesi_proje\images'
bad_files = []

# Iterate through each breed folder
for breed_folder in listdir(base_folder):
    breed_folder_path = join(base_folder, breed_folder)
    if isdir(breed_folder_path):
        # Iterate through each image file in the breed folder
        for filename in listdir(breed_folder_path):
            if filename.endswith('.jpg'):
                file_path = join(breed_folder_path, filename)
                try:
                    img = Image.open(file_path)  # Open the image file
                    img.verify()  # Verify that it is, in fact, an image
                except (IOError, SyntaxError) as e:
                    print('Bad file:', file_path)
                    bad_files.append(file_path)

print(f"Total bad files: {len(bad_files)}")
if bad_files:
    print("List of bad files:")
    for bad_file in bad_files:
        print(bad_file)
```

B. Model Eğitimi

Model eğitimi, train_vgg16.py betiği kullanılarak gerçekleştirilmiştir. Modelin mimarisi VGG16 tabanlı olup, çeşitli katmanlar ve yoğun bağlantı katmanlarından oluşmaktadır.

scripts > train_vgg16.py > ...

```python

# VGG16 Model Definition
print("Defining the VGG16 model...")
input_layer = Input(shape=(224, 224, 3))

x = Conv2D(64, kernel_size=(3, 3), padding="same", activation="relu")(input_layer)
x = Conv2D(64, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = MaxPooling2D((2, 2), strides=(2, 2))(x)

x = Conv2D(128, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = Conv2D(128, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = MaxPooling2D((2, 2), strides=(2, 2))(x)

x = Conv2D(256, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = Conv2D(256, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = Conv2D(256, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = MaxPooling2D((2, 2), strides=(2, 2))(x)

x = Conv2D(512, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = Conv2D(512, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = Conv2D(512, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = MaxPooling2D((2, 2), strides=(2, 2))(x)

x = Conv2D(512, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = Conv2D(512, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = Conv2D(512, kernel_size=(3, 3), padding="same", activation="relu")(x)
x = MaxPooling2D((2, 2), strides=(2, 2))(x)

x = Flatten()(x)
x = Dense(4096, activation="relu")(x)
x = Dense(4096, activation="relu")(x)
output_layer = Dense(train_generator.num_classes, activation="softmax")(x)

model = Model(inputs=input_layer, outputs=output_layer)

# Model Summary
print("Model summary:")
model.summary()

# Model Compilation
print("Compiling the model...")
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Model Training
print("Starting model training...")
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_steps=val_generator.samples // val_generator.batch_size,
    epochs=25,
    validation_data=val_generator
)

# Save the model
model_path = os.path.join(model_dir, 'qedy_vgg16_model.keras')
print(f"Saving the model as '{model_path}'...")
model.save(model_path)

# Save class indices
class_indices = train_generator.class_indices
with open(class_indices_path, 'w') as f:
    json.dump(class_indices, f)
print(f"Class indices saved to {class_indices_path}")

# Evaluate the model
print("Evaluating the model on validation data...")
loss, accuracy = model.evaluate(val_generator, steps=val_generator.samples // val_generator.batch_size)
print(f'Validation loss: {loss:.4f}')
print(f'Validation accuracy: {accuracy:.4f}')

# Plot and save training & validation accuracy values
accuracy_plot_path = os.path.join(model_dir, 'accuracy_plot.png')
plt.figure()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.savefig(accuracy_plot_path)
print(f"Accuracy plot saved to {accuracy_plot_path}")

# Plot and save training & validation loss values
loss_plot_path = os.path.join(model_dir, 'loss_plot.png')
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.savefig(loss_plot_path)
print(f"Loss plot saved to {loss_plot_path}")
```

C. Model Tahmini

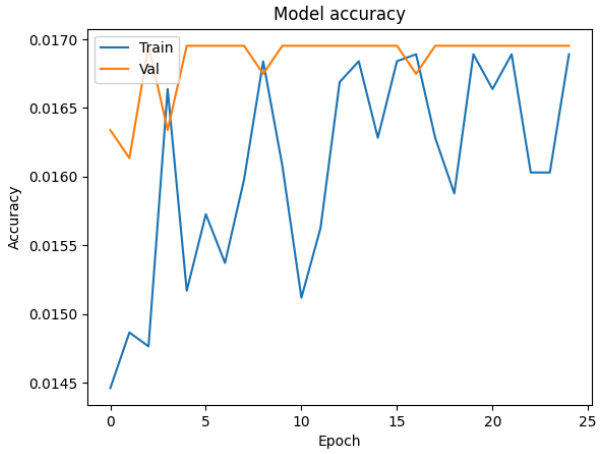
Model tahmini, predict_vgg16.py betiği kullanılarak gerçekleştirilmiştir. Bu betik, eğitilmiş modeli yükler ve yeni görüntüler üzerinde tahmin yapar.

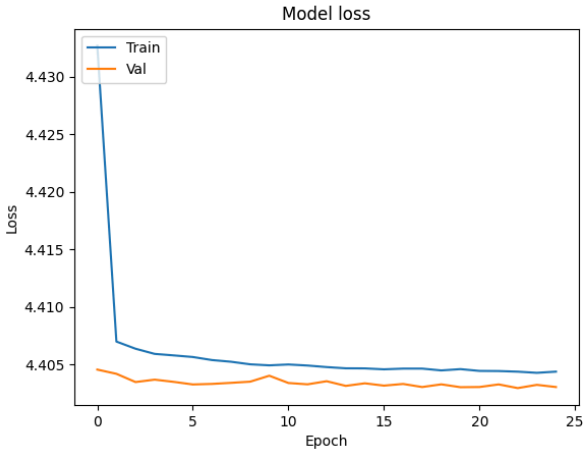
scripts > predict_vgg16.py > ...

```python
import os
import keras
from keras.models import load_model, Model
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import numpy as np
import matplotlib.pyplot as plt
import json

# Define paths
model_dir = r'C:\Users\ertug\Desktop\Folder\makine_ögrenmesi_proje\models'
model_path = os.path.join(model_dir, 'qedy_vgg16_model.keras')
class_indices_path = os.path.join(model_dir, 'class_indices.json')
sample_image_path = r'C:\Users\ertug\Desktop\Folder\makine_ögrenmesi_proje\sample.jpg'

# Load the trained model
print("Loading the model...")
model = load_model(model_path)
print("Model loaded successfully.")

# Load class indices
print("Loading class indices...")
if not os.path.exists(class_indices_path):
    raise FileNotFoundError(f"Class indices file not found: {class_indices_path}")

with open(class_indices_path, 'r') as f:
    class_indices = json.load(f)
class_indices = {v: k for k, v in class_indices.items()}  # Invert the class_indices dictionary

# Load and preprocess the sample image
print("Loading and preprocessing the sample image...")
if not os.path.exists(sample_image_path):
    raise FileNotFoundError(f"Sample image not found: {sample_image_path}")

sample_image = load_img(sample_image_path, target_size=(224, 224))
sample_image_array = img_to_array(sample_image) / 255.0
sample_image_array = np.expand_dims(sample_image_array, axis=0)

# Make predictions
print("Making predictions on the sample image...")
predictions = model.predict(sample_image_array)
predicted_class = np.argmax(predictions, axis=1)

# Display the image and prediction
print("Displaying the sample image with the prediction...")
plt.imshow(sample_image)
plt.title(f'Predicted Class: {class_indices[predicted_class[0]]}')
plt.axis('off')
plt.show()

# Visualizing the feature maps
def plot_feature_maps(model, img_array):
    # Ensure the model has been called to set input shape
    _ = model.predict(img_array)

    # Extracts the outputs of the top 8 layers:
    layer_outputs = [layer.output for layer in model.layers[:8]]
    # Creates a model that will return these outputs, given the model input:
    activation_model = Model(inputs=model.input, outputs=layer_outputs)
    # Returns a list of five Numpy arrays: one array per layer activation
    activations = activation_model.predict(img_array)

    layer_names = []
    for layer in model.layers[:8]:
        layer_names.append(layer.name) # Names of the layers, so you can have them as part of your plot

    images_per_row = 16
    for layer_name, layer_activation in zip(layer_names, activations): # Displays the feature maps
        n_features = layer_activation.shape[-1] # Number of features in the feature map
        size = layer_activation.shape[1] # The feature map has shape (1, size, size, n_features)
        n_cols = n_features // images_per_row # Tiles the activation channels in this matrix
        display_grid = np.zeros((size * n_cols, images_per_row * size))
        for col in range(n_cols): # Tiles each filter into a big horizontal grid
            for row in range(images_per_row):
                channel_image = layer_activation[0, :, :, col * images_per_row + row]
                channel_image -= channel_image.mean() # Post-processes the feature to make it visually palatable
                channel_image /= channel_image.std()
                channel_image *= 64
                channel_image += 128
                channel_image = np.clip(channel_image, 0, 255).astype('uint8')
                display_grid[col * size : (col + 1) * size, row * size : (row + 1) * size] = channel_image
        scale = 1. / size
        plt.figure(figsize=(scale * display_grid.shape[1], scale * display_grid.shape[0]))
        plt.title(layer_name)
        plt.grid(False)
        plt.imshow(display_grid, aspect='auto', cmap='viridis')

plot_feature_maps(model, sample_image_array)

```

III. Deneysel Sonuçlar

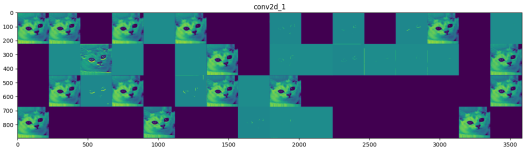
Modelin doğruluğu ve kaybı aşağıdaki grafikte gösterilmiştir.

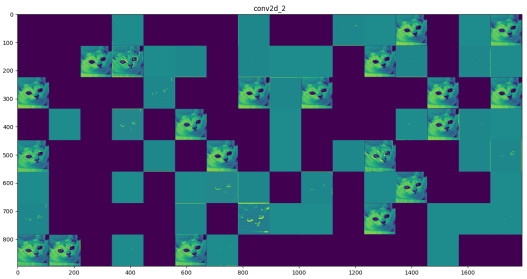


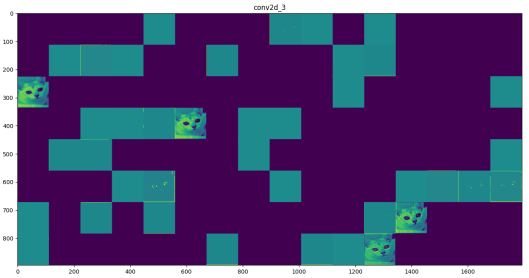


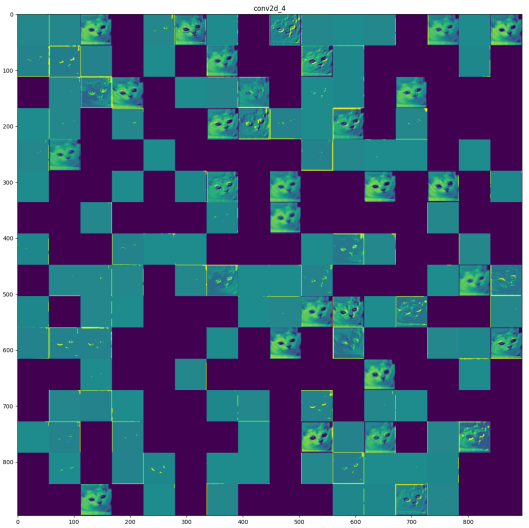
IV. Tartışma ve Gelecek Çalışmalar

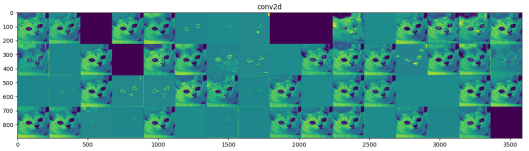
Bu çalışma, derin öğrenme yöntemlerinin kedi ırkı sınıflandırmasında etkin bir şekilde kullanılabileceğini göstermektedir. Gelecekteki çalışmalar, daha büyük veri kümeleri ve daha karmaşık modeller kullanarak sınıflandırma doğruluğunu artırmayı hedeflemektedir.

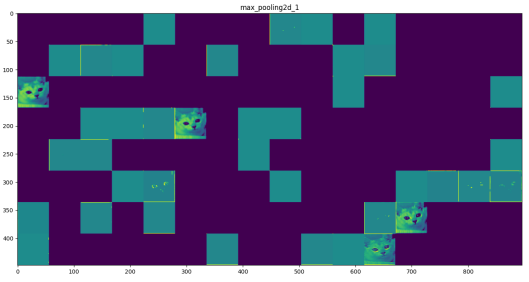


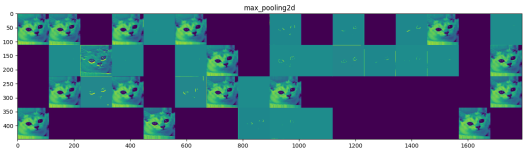












Kaynakça

[1] *Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556.*

[2] *TensorFlow. https://www.tensorflow.org/*

[3] *Keras. https://keras.io/*

[4] *Pillow. https://python-pillow.o*

[5] https://www.thehappycatsite.com/wp-content/uploads/2017/04/ab-4-759x1024.jpg

[6] https://tse1.explicit.bing.net/th?id=OIP.fDO-en7qEf3HAQJCqueUQgHaJ_&pid=Api&P=0&h=220

[7]

[8] https://stackoverflow.com/questions/37158246/how-to-download-images-from-beautifulsoup

[9] https://dev.to/dmitryzub/scrape-yahoo-search-with-python-2bk4

[10] https://stackoverflow.com/questions/55664961/cannot-get-images-from-webpage-with-high-resolution-using-beautifulsoup-and-pyth

[11] https://gist.github.com/genekogan/ebd77196e4bf0705db51f86431099e57

[12] https://github.com/psf/requests/issues/4246

[13] https://stackoverflow.com/questions/23013220/max-retries-exceeded-with-url-in-requests

[14] https://stackoverflow.com/questions/72671331/python-error-read-files-built-in-method-read-of-io-textiowrapper-object-at-0x0

[15] https://stackoverflow.com/questions/11806559/removing-first-x-characters-from-string

[16] https://stackoverflow.com/questions/74751254/removing-all-duplicate-images-with-different-filenames-from-a-directory

[17] https://medium.com/@urvisoni/removing-duplicate-images-through-python-23c5fdc7479e

[18] https://towardsdatascience.com/removing-duplicate-or-similar-images-in-python-93d447c1c3eb

[19] https://docs.python.org/3/library/hashlib.html

[20] https://docs.python.org/3/tutorial/datastructures.html

[21]