SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

KLASIFIKACIJA ODJEĆE

Projektni zadatak

Leon Vučko

Osijek, 2023. godina.

SADRŽAJ

1. UVOD

Korištenjem strojnog učenja i dubokog učenja je moguće kreirati razne sustave i modele koji automatski prepoznaju objekte sa slike, koji se zatim mogu koristi za klasifikaciju raznih stvari.

Kroz ovaj rad se opisuje postupak na koji se način može kreirati model koji služi za raspoznavanje i klasifikaciju različitih odjevnih predmeta.

Prvi korak u razvoju modela klasifikacije odjeće, ali i općenito pri razvoju modela, je prikupljanje podataka. Za tu svrhu se koriste već postojeće baze podataka ili datasetovi koji sadrže slike različitih komada odjeće koji su već klasificirani. Osim postojećih i gotovih datasetova se mogu koristiti i ručno prikupljeni podaci, za koje je potrebno dosta vremena, kao i drugi izvori poput prikupljanja podatka sa web tražilica poput Google Images, i DuckDuckGo-a.

Nakon što su podaci prikupljeni korištenjem Python-a i Jupyter Notebook-a, kao i pripadajućih biblioteka obrađuju se podaci, te se korištenjem Keras-a i Tensorflowa kreira model za klasifikaciju odjeće.

Kada je dobiven zadovoljavajući model, kreirana je Android aplikacija korištenjem razvojnog okruženja Android Studio. Zatim je ta mobilna aplikacija korištena za testiranje modela u stvarnome svijetu.

Rad je podijeljen na 6 osnovnih poglavlja: uvod, pretraživanje slika, programsko rješenje, analiza rezultata i zaključak. U prvom poglavlju je dan uvod u temu obrađenu u sklopu ovog projektnog zadatka. U drugom poglavlju objašnjeni su osnovni pojmovi vezani uz princip dubokih neuronskih mreža, kao i klasifikacija. U trećem poglavlju je opisan postupak prikupljanja podataka potrebnih za treniranje modela. U četvrtom poglavlju je opisan postupak obrade podataka kao i kreiranje i treniranje modela. U petom poglavlju je dan uvid u kreiranje Android aplikacije koja koristi kameru i prethodno trenirani model za klasifikaciju. U šestom poglavlju su prikazani dobiveni rezultati. Posljednje poglavlje predstavlja zaključak u kojem se komentiraju postignuti rezultati.

1.1. Zadatak projektnog zadatka

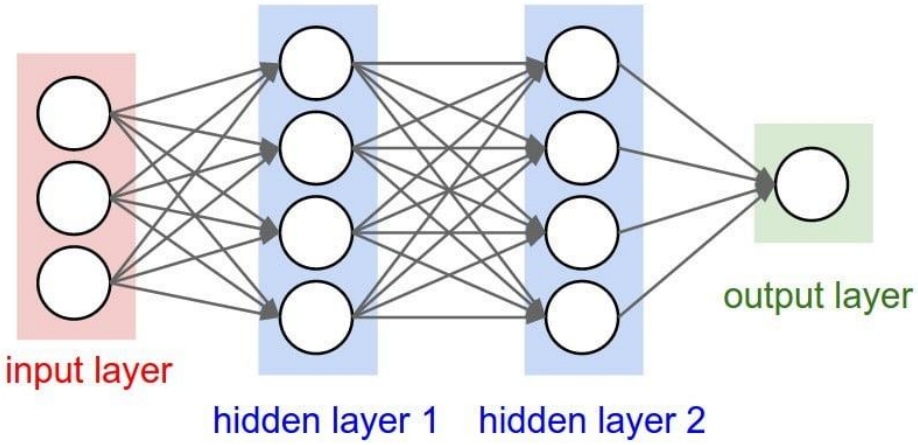
Napraviti i istrenirati model duboke neuronske mreže korištenjem Keras biblioteke, te primijeniti taj model u stvarnom svijetu korištenjem Android aplikacije.

2. STROJNO UČENJE

Strojno učenje je područje umjetne inteligencije koje se bavi razvojem algoritama i modela koji omogućuju računalima da "uče" iz podataka i donose predikcije ili donose odluke bez izričitog programiranja. Umjesto da se programiraju specifične upute za obavljanje zadataka, strojno učenje se temelji na analizi podataka i automatskom izvlačenju uzoraka i znanja iz tih podataka [1].

2.1. Duboke Neuronske Mreže

Duboke neuronske mreže su vrsta neuronskih mreža koje se sastoje dva ili više slojeva neurona [2]. Ove mreže su dizajnirane da simuliraju način na koji ljudski mozak obrađuje informacije, koristeći hijerarhijski pristup za izgradnju sve složenijih reprezentacija podataka.



Sl. 2.1. Duboka neuronska mreža [2].

2.2. Klasifikacija podataka

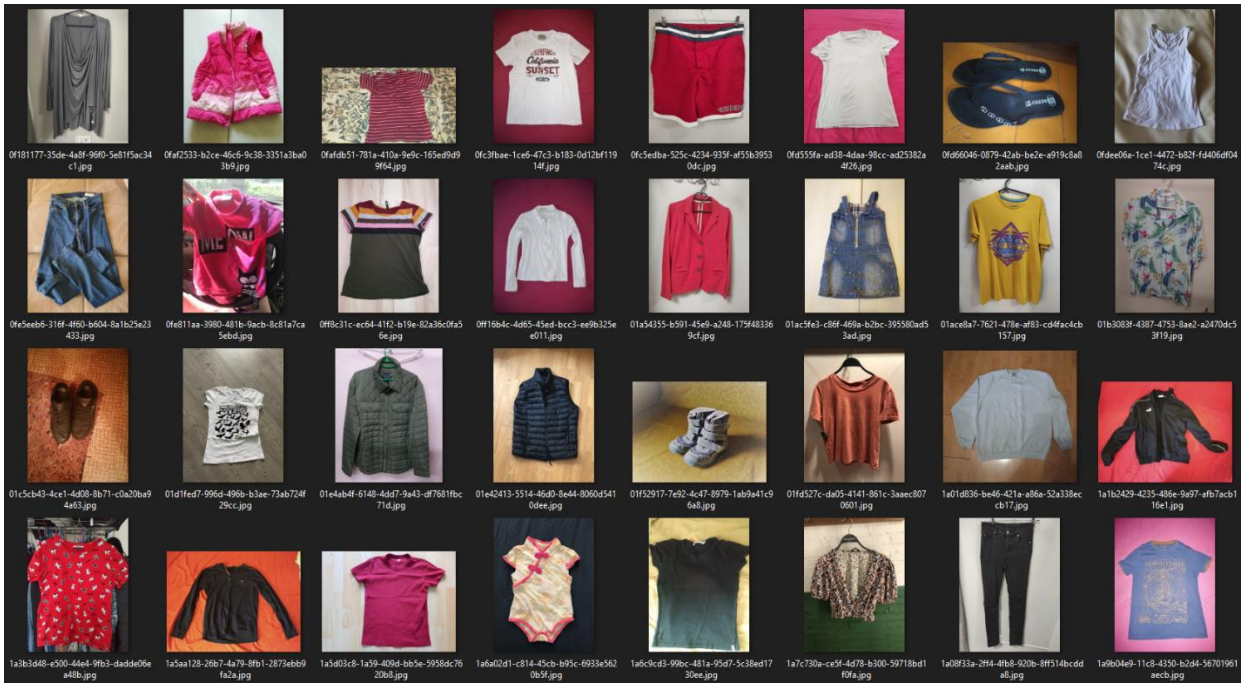
Klasifikacija je proces pridruživanja ulaznih podataka određenim klasama ili kategorijama na temelju njihovih karakteristika ili značajki. Duboke neuronske mreže su posebno dobre u rješavanju problema klasifikacije zbog njihove sposobnosti da nauče složene reprezentacije značajki iz sirovih podataka. Kada se koriste za klasifikaciju slika, duboke neuronske mreže obično imaju ulazni sloj koji prima slike. Nakon toga slijedi više slojeva koji obrađuju te podatke, izvlačeći korisne značajke (*features*) iz njih. Ti slojevi se nazivaju skrivenim slojevima, a broj i veličina slojeva može varirati ovisno o složenosti problema. Svaki skriveni sloj koristi težine (*weights*) kako bi transformirao ulazne podatke i generirao nove reprezentacije. Ove težine se prilagođavaju tijekom faze učenja mreže kako bi se minimizirala pogreška klasifikacije. Na kraju, izlazni sloj daje vjerojatnosti ili odluke o kojoj klasi pripada ulazni podatak.

3. PRIKUPLJANJE PODATAKA

Podaci za testiranje su prikupljeni s interneta. Osim gotovog *dataseta* su se koristili i takozvani *image scraperi* kako bi se povečala količina podataka za treniranje. Kao osnova se koristi Clothing Dataset s Kaggle-a, a broj klasa koji će se koristiti u ovome radu iznosi petnaest.

3.1. Kaggle Clothing Dataset

Clothing Dataset je skup slika koji se koristi za trening i testiranje algoritama za prepoznavanje odjeće, a sadrži slike 20 različitih klasa, poput majica, hlača, suknji, kapa i obuće. Sadrži preko 5000 slika, a dostupan je za slobodnu upotrebu u bilo koje svrhe, čak i komercijalne. Dataset je dostupan na linku: https://www.kaggle.com/datasets/agrigorev/clothing-dataset-full, sadrži kompresirane i nekompresirane slike. Na slici 3.1. je vidljivo nekoliko slika iz skupa.



Sl. 3.1. Primjer slika iz Clothing Dataseta

Zatim je radi lakšeg kasnijeg dodavanja novih slika razdijeliti svaki tip odjeće u poseban direktorij.

```
import pandas as pd
import shutil
import os
labels = pd.read_csv('images.csv')
labels.loc[labels['label']=='Not sure','label'] = 'Not_sure'
labels['image'] = labels['image'] + '.jpg'
labels['label_cat'] = labels['label']
label_df = labels[['image', 'label_cat']]
path = 'images_original'
```

```
source_folder = r"C:\Users\Vucko\Documents\Clothing\images_original\\"
destination_folder1 = r"C:\Users\Vucko\Documents\Clothing\Tshirt\\"
destination_folder2 = r"C:\Users\Vucko\Documents\Clothing\Longsleeve\\"
destination_folder3 = r"C:\Users\Vucko\Documents\Clothing\Shoes\\"
destination_folder4 = r"C:\Users\Vucko\Documents\Clothing\Shirt\\"
destination_folder5 = r"C:\Users\Vucko\Documents\Clothing\Outwear\\"
destination_folder6 = r"C:\Users\Vucko\Documents\Clothing\Not_sure\\"
destination_folder7 = r"C:\Users\Vucko\Documents\Clothing\Hat\\"
destination_folder8 = r"C:\Users\Vucko\Documents\Clothing\Skirt\\"
destination_folder9 = r"C:\Users\Vucko\Documents\Clothing\Polo\\"
destination_folder10 = r"C:\Users\Vucko\Documents\Clothing\Undershirt\\"
destination_folder11 = r"C:\Users\Vucko\Documents\Clothing\Blazer\\"
destination_folder12 = r"C:\Users\Vucko\Documents\Clothing\Hoodie\\"
destination_folder13 = r"C:\Users\Vucko\Documents\Clothing\Body\\"
destination_folder14 = r"C:\Users\Vucko\Documents\Clothing\Other\\"
destination_folder15 = r"C:\Users\Vucko\Documents\Clothing\Top\\"
destination_folder16 = r"C:\Users\Vucko\Documents\Clothing\Blouse\\"
destination_folder17 = r"C:\Users\Vucko\Documents\Clothing\Skip\\"
destination_folder18 = r"C:\Users\Vucko\Documents\Clothing\Pants\\"
os.makedirs(destination_folder1)
os.makedirs(destination_folder2)
os.makedirs(destination_folder3)
os.makedirs(destination_folder4)
os.makedirs(destination_folder5)
os.makedirs(destination_folder6)
os.makedirs(destination_folder7)
os.makedirs(destination_folder8)
os.makedirs(destination_folder9)
os.makedirs(destination_folder10)
os.makedirs(destination_folder11)
os.makedirs(destination_folder12)
os.makedirs(destination_folder13)
os.makedirs(destination_folder14)
os.makedirs(destination_folder15)
os.makedirs(destination_folder16)
os.makedirs(destination_folder17)
os.makedirs(destination_folder18)
for x in range(5403):
    if(label_df.label_cat[x]=='T-Shirt'):
        shutil.move(source_folder+label_df.image[x], destination_folder1)
    if(label_df.label_cat[x]=='Longsleeve'):
        shutil.move(source_folder+label_df.image[x], destination_folder2)
    if(label_df.label_cat[x]=='Shoes'):
        shutil.move(source_folder+label_df.image[x], destination_folder3)
    if(label_df.label_cat[x]=='Shirt'):
        shutil.move(source_folder+label_df.image[x], destination_folder4)
    if(label_df.label_cat[x]=='Outwear'):
        shutil.move(source_folder+label_df.image[x], destination_folder5)
    if(label_df.label_cat[x]=='Not_sure'):
```

```
        shutil.move(source_folder+label_df.image[x], destination_folder6)
    if(label_df.label_cat[x]=='Hat'):
        shutil.move(source_folder+label_df.image[x], destination_folder7)
    if(label_df.label_cat[x]=='Skirt'):
        shutil.move(source_folder+label_df.image[x], destination_folder8)
    if(label_df.label_cat[x]=='Polo'):
        shutil.move(source_folder+label_df.image[x], destination_folder9)
    if(label_df.label_cat[x]=='Undershirt'):
        shutil.move(source_folder+label_df.image[x], destination_folder10)
    if(label_df.label_cat[x]=='Blazer'):
        shutil.move(source_folder+label_df.image[x], destination_folder11)
    if(label_df.label_cat[x]=='Hoodie'):
        shutil.move(source_folder+label_df.image[x], destination_folder12)
    if(label_df.label_cat[x]=='Body'):
        shutil.move(source_folder+label_df.image[x], destination_folder13)
    if(label_df.label_cat[x]=='Other'):
        shutil.move(source_folder+label_df.image[x], destination_folder14)
    if(label_df.label_cat[x]=='Top'):
        shutil.move(source_folder+label_df.image[x], destination_folder15)
    if(label_df.label_cat[x]=='Blouse'):
        shutil.move(source_folder+label_df.image[x], destination_folder16)
    if(label_df.label_cat[x]=='Skip'):
        shutil.move(source_folder+label_df.image[x], destination_folder17)
    if(label_df.label_cat[x]=='Pants'):
        shutil.move(source_folder+label_df.image[x], destination_folder18)
```

Sl. 3.2. Programski kod za razvrstavanje slika u zasebne direktorije

Nakon toga su obrisani direktoriji *Not_Sure*, *Skip* i *Other* kako ne bi kasnije došlo do neželjenog učenja klasa koje nisu sortirane.

3.1. Duck Duck Go jmd_imagescraper

Za proširivanje baze za treniranje neuronske mreže se koristi i *jmd_imagescraper* koji služi za prikupljanje slika sa *DuckDuckGo* tražilice. Izvorni kod je dostupan na linku: https://github.com/joedockrill/jmd_imagescraper. Donjim kodom je dan primjer kako ga koristiti:

```
!pip install -q jmd_imagescraper

from pathlib import Path
root = Path().cwd()/"images"
from jmd_imagescraper.core import *
duckduckgo_search(root, "Blouse", "Blouse", max_results=250)
```

Sl. 3.3. Programski kod za korištenje jmd_imagescraper

Na slici 3.4. su vidljive dobivene slike za pretragu *T-shirt* iz kojih je moguće zaključiti kako će biti potrebno malo pregledati i filtrirati podatke.

Sl. 3.4. Primjer slika dobivenih korištenjem *jmd_imagescrapera* za pojam *T-shirt*

3.1. Google Image Scraper

Također, slike je moguće i dobiti sa Googleove tražilice. Za to se koristio *Website Image Scraper* dostupan na https://github.com/JJLimmm/Website-Image-Scraper. Kako bi ispravno radio potrebno je imati instaliran *selenium*, *request* i *pillow*.

```
import os
import concurrent.futures
from GoogleImageScraper import GoogleImageScraper
from patch import webdriver_executable
def worker_thread(search_key):
    image_scraper = GoogleImageScraper(
        webdriver_path,
        image_path,
        search_key,
        number_of_images,
        headless,
        min_resolution,
        max_resolution,
        max_missed)
    image_urls = image_scraper.find_image_urls()
    image_scraper.save_images(image_urls, keep_filenames)
    webdriver_path = os.path.normpath(os.path.join(os.getcwd(), 'webdriver',
webdriver_executable()))
    image_path = os.path.normpath(os.path.join(os.getcwd(), 'photos'))
    search_keys = list(set(["Blazer", "Blazer clothing", "Blouse", "Blouse
clothing", "Blouse colorful", "Body clothes", "Body clothing", "beanie","cap","top
hat", "fedora","Hat" ,"Hoodie", "Hoodie clothing", "Hoodie black", "Hoodie white",
```

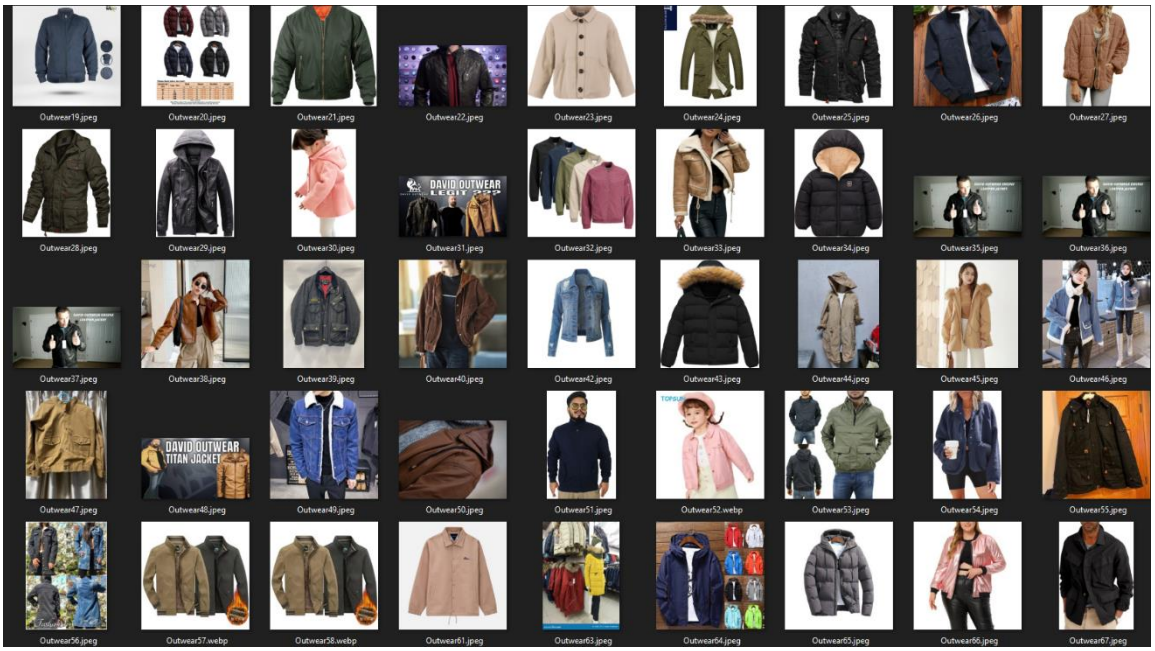
```
"Longsleeve shirt", "Longsleeve shirt gildan", "Outwear", "Outwear jacket",
"Outwear male jacket", "Outwear female jacket","Polo shirt", "Polo shirt
colorful","Shirt","Shoes","Skirt","Top", "Top clothing", "Top black", "Top
colorful","Tshirt","Undershirt", "White undershirt", "black undershirt"]))

    number_of_images = 250                # Desired number of images
    headless = True                      # True = No Chrome GUI
    min_resolution = (100, 100)              # Minimum desired image resolution
    max_resolution = (2000, 2000)        # Maximum desired image resolution
    max_missed = 100                     # Max number of failed images before exit
    number_of_workers = 8               # Number of "workers" used
    keep_filenames = False                # Keep original URL image filenames

    #Run each search_key in a separate thread
    #Automatically waits for all threads to finish
    #Removes duplicate strings from search_keys
    with concurrent.futures.ThreadPoolExecutor(max_workers=number_of_workers) as
executor:
        executor.map(worker_thread, search_keys)
```

Sl. 3.5. Programski kod za korištenje Google Image Scrapera.

Kao i kod prijašnjeg *scraper*-a vidljivo je na slici 3.6. da ima duplikata i da postoje slike koje bi mogle utjecati na kvalitetu modela, te ih je potrebno ukloniti.



Sl. 3.6. Primjer slika dobivenih korištenjem Google Image Scrapera za pojam Jacket.

Iz dobivenih slika su uklonjene slike koje sadrže ljude jer su znatno utjecale na točnost podataka kao i slike na kojima se nalazi više odjevnih predmeta.

4. KREIRANJE MODELA

Programski kod je pisan u programskom jeziku Python koristeći Jupyter notebook koji omogućava izvođenje programa „blok po blok“.

4.1. Python

Python je jednostavan i popularan programski jezik koji služi mnogim svrhama, od jednostavnog računanja pa do složeniji stvari poput strojnog učenja i obrade slike. Prednosti *python*-a su jednostavna sintaksa i velika zajednica korisnika. Također Python je *open-source* i neovisan je o platformi [3].

Sl. 4.1. Python logo [3.

4.2. Jupyter notebook

Jupyter notebook je interaktivno računalno okruženje koje omogućuje korisnicima da kreiraju dokumente koji sadržavaju izvorni kod kao slike, matematičke formule i grafove [5]. Podržava mnoge programske jezike, uključujući i Python. Moguće ga je koristi lokalno ili preko interneta, jedan od primjera je *Google Colab* koji omogućuje korištenje mnogih *python* biblioteka za strojno učenje i obradu slike bez potrebe za preuzimanjem software-a na računalo.

Sl. 4.2. Jupyter logo [4].

4.3. Korištene biblioteke

Prva biblioteka koja se koristi je *OS* biblioteka. Ona definira funkcije za interakciju s operacijskim sustavom u vezi datoteka i direktorija poput ispisa svih datoteka u folderu, kreiranje novih direktorija i datoteka.

Numpy je biblioteka koja se koristi za složene matematičke operacije, kao i za operacije nad velikim i višedimenzionalnim matricama.

PIL ili *Python Image Library* je biblioteka čija je namjena obrada i manipulacija slikama. Ima funkcionalnosti poput uređivanja, prikazivanja i učitavanja slika.

Keras je među najpopularnijim bibliotekama za duboko učenje, dolje su dane specifične biblioteke koje se koriste, poput sekvencijalnog modela, slojeva (*layera*) prilikom kreiranja modela, *ImageDataGeneratora* za generiranje izmjenitih slika itd.

```
import os
import numpy as np
from PIL import Image
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
```

Sl. 4.3. Uvoz potrebnih biblioteka.

4.4. Učitavanje i obrada podataka

Za početak se učitavaju kategorije koje se nalaze u nazivima poddirektorija dataset direktorija. Zatim se određuje broj klasa kao i inicijaliziraju lista koje predstavljaju ulazne slike i oznake.

```
dataset_folder = "Final_Data_V2"
categories = os.listdir(dataset_folder)
num_classes = len(categories)
X_data = []
y_data = []
```

Sl. 4.4. Početne operacije

Nakon toga se u for petlji prolazi kroz direktoriji svake klase i iz njega se učitavaju slike. Prvo se provjerava da li je učitana slika, i ukoliko je smanjuje se njena veličina na 100 x 100 piksela. Zatim se provjerava da li slika sadrži alpha kanal, poput .png slika, onda se pretvara u RGB sliku. Zatim se slika pretvara u numpy array, i provjerava se da li je slika odgovarajućeg oblika. Konačno se

slika dodaje u images. Ovisno o duljini images i kategoriji zapisuje broj koji predstavlja oznaku klase za svaku sliku dodanu u images. Na kraju se slike i oznake dodaju na kraj liste slika i oznaka.

```
for i, category in enumerate(categories):
    folder_path = os.path.join(dataset_folder, category)
    images = []
    for filename in os.listdir(folder_path):
        img = Image.open(os.path.join(folder_path, filename))
        if img is not None:
            img = img.resize((100, 100))
            if img.mode == 'RGBA':
                img = img.convert('RGB')
            img_array = np.array(img)
            if len(img_array.shape) == 3 and img_array.shape[2] == 3:
                images.append(img_array)
    labels = [i] * len(images)
    X_data.extend(images)
    y_data.extend(labels)
```

Sl. 4.5. Dobivanje ulaznih slika i pripadajućih oznaka

X_data i y_data se pretvaraju u numpy array, zatim se X_data normalizira na raspon od 0 do 1. Oznake klasa y_data se onda kodiraju u binarni oblik, jedna oznaka sadržava 0 za sve indekse klasa kojoj ne pripada, a za indeks klase kojoj pripada je 1. Konačno se test podaci dijele na trening podatke i test podatke u omjeru 80:20, također se postavlja i vrijednost random_state kako bi se moglo ponoviti trening sa istim podacima, poput korekcije parametara modela.

```
X_data = np.array(X_data)
y_data = np.array(y_data)
X_data = X_data.astype('float32') / 255.0
y_data = to_categorical(y_data, num_classes)
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
test_size=0.2, random_state=123)
```

Sl. 4.6. Normalizacija i raspodjela podataka na train i test.

Također je moguće generiranje dodatnih verzija slika za obuku modela korištenjem ImageDataGeneratora. Potrebno je specificirati neke parametre poput raspona kutova i pomaka, horizontalnog okretanja i tome koliko će se podataka koristiti za validaciju na kraju, u ovom slučaju bi to bilo 20%.

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    validation_split=0.2
)
```

Sl. 4.7. Proširivanje skupa slika transformacijom postojećih

4.5. Definiranje i treniranje modela

Prvo se inicijalizira prazan sekvencijalni model kojemu se prvo dodaje Conv2D layer koji provodi konvolucijske operacije s 32 filtra veličine 3x3 korištenjem ReLU aktivacijske funkcije. Zatim se dodaje MaxPooling2D sloj koji odabire maksimalnu vrijednost na 2x2 području. Zatim se koristi još jedan Conv2D sloj ali ovaj puta s 64 filtra i s ReLU funkcijom kao aktivacijskom. Zatim se ponovo koristi MaxPooling2D sloj kao i prije. Nakon tih slojeva se dodaje i Flatten sloj koji pretvara izlaz u jedno dimenzionalni vektor. Zatim se dodaje Dense potpuno povezani sloj s 64 neurona i aktivacijskom funkcijom ReLU. Na kraju se dodaje još jedan sloj Dense čiji je broj neurona jednak broju klasa, ovdje se koristi softmax aktivacijska funkcija kako bi generirao vjerojatnost za svaku od klasa. Zatim se model kreira korištenjem adam optimizatora, categorical_crossentropy se koristi za loss jer postoji više klasa i točnosti kao mjerilom.

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

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

Sl. 4.8. Definiranje i treniranje modela

Nadalje se određuje batch size iliti broj slika koji se koristi u jednoj grupi tjekom treniranja, ovdje je kroz nekoliko iteracija započeo s 32, da bi se sa tri postigla optimalna brzina i preciznost. Epoch predstavlja broj puta kroz koji će se model obučavati, tj. proći će kroz cijeli skup trideset puta. Zatim koristimo train_generator i valdiation_generator koji koriste prijašnje postavljen datagen za kreiranje dodatnih slika korištenjem nekih načina izmjene slike.

```
batch_size = 3
epochs = 30
train_generator = datagen.flow(X_train, y_train, batch_size=batch_size,
subset='training')
validation_generator = datagen.flow(X_train, y_train, batch_size=batch_size,
subset='validation')
```

Sl. 4.9. Postavljanje veličine batch-a i broja epoha,

Zatim se trenira model korištenjem naredbe fit, kao ulaz za treniranje se koristi train_generator, a za broj koraka se uzima duljina train_generatora. Kao broj epoha se uzima prijašnja vrijednost od 30, a za validaciju se koristi validation_generator. Na kraju svake epohe se prikazuje loss i preciznost (accuracy).

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

Sl. 4.10. Treniranje modela

Nakon što je model iztreniran možemo vidjeti loss i accuracy na testnom skupu podataka, u ovome slučaju smo prethodno uzeli 20% podataka. Dobila se točnost od 54,61% i loss koji iznosi 1,5509.

```
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

Sl. 4.11. Testiranje modela na testnom skupu.

4.6. Spremanje, učitavanje i testiranje modela na vlastitim primjerima

Model se može spremiti korištenjem .save("name") funkcije, pri čemu se Keras modelima najčešće daje nastavak .h5.

```
model.save("ClothingModel.h5")
```

Sl. 4.12. Spremanje modela

Također je moguće i učitati jednom spremljene modele korištenjem load_model(), ali je pritome potrebno prvo uključiti biblioteku load_model iz keras.models.

```
loaded_model = load_model("ClothingModel.h5")
```

Sl. 4.13. Učitavanje modela

Zatim se može testirati model na vlastitom primjeru, prije toga je potrebno obratiti sliku (preprocess) kako bi mogla se mogla testirati na modelu, poput postavljanja dimenzija na 100x100, kao i pretvaranje u numpy matricu, zatim se vrši normalizacija vrijednosti piksela. Zatim se dodaje dodatna dimenzija korištenjem np.expand_dims koja predstavlja broj uzoraka, a potrebna je jer je

model očekuje (npr. prilikom testiranja više slika), zatim se vrši predviđanje i odabire se klasa kojoj najviše odgovara taj model, na kraju se ispisuje naziv te odabrane klase.

```
img_path = 'test2.jpg'
img = Image.open(img_path)
img = img.resize((100, 100))
img_array = np.array(img)
img_array = img_array / 255.0
img_array = np.expand_dims(img_array, axis=0)
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions[0])
print(categories[predicted_class])
```

Sl. 4.14. Testiranje modela

4.7. Spremanje modela u TFlite formatu

Razlog spremanje slike u ovome formatu je to što je potrebno kako bi se mogao koristiti u android aplikaciji. Primjer spremanja modela u tflite format je dan slikom 4.15.

```
import tensorflow as tf

model = tf.keras.models.load_model('ClothingModel.h5')

converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open('ClothingModel.tflite', 'wb') as f:
    f.write(tflite_model)
```

Sl. 4.15. Spremanje modela u tflite formatu

5. ANDROID APLIKACIJA

Android aplikacija je kreirana za potrebe testiranja ovoga skupa u pravome svijetu.

5.1. Android studio

Android Studio je službeno integrirano razvojno okruženje za Googleov operativni sustav Android, izgrađeno na JetBrainsovom IntelliJ IDEA softveru i dizajnirano posebno za Android razvoj, a dostupan je za preuzimanje na Windows, macOS i Linux operativnim sustavima [5].

5.2. Glavni kod Android aplikacije

Slikom 5.1. su dane korištene biblioteke, ove biblioteke su dodavane tijekom programiranja.

```
package com.example.clothingclassificator
import android.Manifest
import android.content.pm.PackageManager
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.os.Bundle
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import android.hardware.Camera
import android.view.SurfaceHolder
import android.view.SurfaceView
import android.widget.Button
import android.widget.TextView
import java.io.IOException
import androidx.appcompat.app.AppCompatActivity
import org.tensorflow.lite.Interpreter
import java.io.FileInputStream
import java.nio.ByteBuffer
import java.nio.ByteOrder
import java.nio.channels.FileChannel
```

Sl. 5.1. Uvoz potrebnih biblioteka.

Prilikom pokretanja Activity-a potrebno je inicijalizirati varijable te im dodjeliti njihove postavke kako bi se mogle koristiti. Za potrebe ovoga projekta potrebno je ubaciti tfliteInerpreter kako bi se model mogao interpretirati. Potrebno je prilikom kreiranja View-a dohvatiti postavke modela. A to se postiže funkcijom loadModelFromAsset() gdje je unutar assets foldera postavljen istrenirani model. Također se tu postavljaju ulazni parametri za model poput duljine ulaza i oblika.

```
class MainActivity : AppCompatActivity(), SurfaceHolder.Callback,
Camera.PictureCallback {
    private lateinit var camera: Camera
    private lateinit var surfaceView: SurfaceView
    private lateinit var surfaceHolder: SurfaceHolder
    private lateinit var captureButton: Button
    private lateinit var predictionTextView: TextView
    private lateinit var tfliteInterpreter: Interpreter
    private val CAMERA_PERMISSION_REQUEST_CODE = 100
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val tfliteModel = loadModelFromAsset()
        val options = Interpreter.Options()
        tfliteInterpreter = Interpreter(tfliteModel, options)
        surfaceView = findViewById(R.id.cameraSurfaceView)
        captureButton = findViewById(R.id.captureButton)
        predictionTextView = findViewById(R.id.predictionTextView)

        surfaceHolder = surfaceView.holder
        surfaceHolder.addCallback(this)

        captureButton.setOnClickListener {
            camera.takePicture(null, null, this)
        }
    }
    private fun loadModelFromAsset(): ByteBuffer {
        val assetManager = assets
        val modelDescriptor = assetManager.openFd("ClothingModel.tflite")
        val inputStream = FileInputStream(modelDescriptor.fileDescriptor)
        val modelData =
ByteBuffer.allocateDirect(modelDescriptor.declaredLength.toInt())
        inputStream.channel.use { channel ->
            modelData.put(channel.map(FileChannel.MapMode.READ_ONLY,
modelDescriptor.startOffset, modelDescriptor.declaredLength))
        }
        modelData.order(ByteOrder.nativeOrder())
        return modelData
    }
```

Sl. 5.2. Početak MainActivity, inicijalizacija varijabli i učitavanje modela.

Nakon toga se definiraju funkcije za rad s kamerom, prvo je unutar funkcije onResume potrebno zatražiti ovlaštenje (permission) za kameru, a nakon se unutar onRequestPermissionsResult provjerava da li je dano odobrenje i može li se pokrenuti kamera koja se prikazuje na surface-u. Unutar AndroidManifesta je potrebno definirati android:name="android.hardware.camera" kako bi se koristila kamera. U funkciji startCamera se otvara kamera i postavlja se prikaz kamere na surfaceholder koji se zatim pokreće.

```
override fun onResume() {
    super.onResume()
    if (ContextCompat.checkSelfPermission(
            this,
            Manifest.permission.CAMERA
        ) == PackageManager.PERMISSION_GRANTED
    ) {
        startCamera()
    } else {
        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.CAMERA),
            CAMERA_PERMISSION_REQUEST_CODE
        )
    }
}
```

```
override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == CAMERA_PERMISSION_REQUEST_CODE) {
        if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            startCamera()
        }
    }
}
private fun startCamera() {
    try {
        camera = Camera.open()
        camera.setPreviewDisplay(surfaceHolder)
        camera.startPreview()
        camera.setDisplayOrientation(90)
    } catch (e: IOException) {
        e.printStackTrace()
    }
}
override fun surfaceCreated(holder: SurfaceHolder) {}
override fun surfaceChanged(holder: SurfaceHolder, format: Int, width: Int,
height: Int) {
    if (holder.surface == null) {
        return
    }
    try {
        camera.stopPreview()
        camera.setPreviewDisplay(holder)
        camera.startPreview()
    } catch (e: IOException) {
        e.printStackTrace()
    }
}
override fun surfaceDestroyed(holder: SurfaceHolder) {
    camera.stopPreview()
    camera.release()
}
```

Sl. 5.3. Početak MainActivity, inicijalizacija varijabli i učitavanje modela.

Zatim se definira onPictureTaken() metoda koja se poziva kada se klikne na tipku tj. nakon što završi slikanje. Ona radi tako da prvo pretvara sliku dobivenu s kamere iz bajtova u Bitmap, te mjenja njenu veličinu na 100x100. Zatim se stvara slika koja koristi ARGB format, te se vrši pretvorba slike u ByteBuffer kako bi se mogla koristiti za model. Pretvorba se vrši pomoću metode convertBitmapToByteBuffer. Na kraju se pokreće tfliteInterpreter i kao izlaz se dobiva broj iz kojeg se zatim dobiva najvjerojatnija klasa metodom getPredictedClass().

```
    override fun onPictureTaken(data: ByteArray?, camera: Camera?) {
        val bitmap = BitmapFactory.decodeByteArray(data, 0, data?.size ?: 0)
        val resizedBitmap = Bitmap.createScaledBitmap(bitmap, 100, 100,
false)
        val normalizedBitmap = resizedBitmap.copy(Bitmap.Config.ARGB_8888,
true)
```

```
        val modelInputBuffer = convertBitmapToByteBuffer(normalizedBitmap)

        val outputBuffer = Array(1) { FloatArray(NUM_CLASSES) }
        tfliteInterpreter.run(modelInputBuffer, outputBuffer)

        val predictedClass = getPredictedClass(outputBuffer[0])
        predictionTextView.text = predictedClass

        camera?.startPreview()
    }
    private fun convertBitmapToByteBuffer(bitmap: Bitmap): ByteBuffer {
        val inputShape = tfliteInterpreter.getInputTensor(0).shape() // Get
the input tensor shape
        val inputSize = inputShape[1] * inputShape[2] * inputShape[3]
        val modelInputBuffer = ByteBuffer.allocateDirect(inputSize *
4).apply {
            order(ByteOrder.nativeOrder())
            rewind()
        }

        val intValues = IntArray(bitmap.width * bitmap.height)
        bitmap.getPixels(intValues, 0, bitmap.width, 0, 0, bitmap.width,
bitmap.height)

        for (pixelValue in intValues) {
            val r = (pixelValue shr 16) and 0xFF
            val g = (pixelValue shr 8) and 0xFF
            val b = pixelValue and 0xFF

            val normalizedR = r / 255.0f
            val normalizedG = g / 255.0f
            val normalizedB = b / 255.0f

            modelInputBuffer.putFloat(normalizedR)
            modelInputBuffer.putFloat(normalizedG)
            modelInputBuffer.putFloat(normalizedB)
        }

        return modelInputBuffer
    }
    val NUM_CLASSES = 15
    val classLabels = arrayOf("Blazer", "Blouse", "Body", "Hat", "Hoodie",
"Longsleeve", "Outwear", "Pants", "Polo", "Shirt", "Shoes", "Skirt", "Top",
"Tshirt", "Undershirt")

    private fun getPredictedClass(output: FloatArray): String {
        val predictedClassIndex = output.indices.maxByOrNull { output[it] }
?: 0
        return classLabels[predictedClassIndex]
    }
}
```

Sl. 5.4. Početak MainActivity, inicijalizacija varijabli i učitavanje modela.

6. ANALIZA MODELA

Za analizu će analizirati podaci koji su dobiveni na testnom skupu. Osim njih testirati će se i rad aplikacije kroz nekoliko primjera raznih komada odjeće.

6.1. Testiranje na testnom skupu

Za testiranje se koristi prethodno definiranih 20% skupa. Za lakše čitanje rezultata je kreirana matrica konfuzije korištenjem koda na slici 6.1. dok je slikom 6.2. dana matrica konfuzije

```
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)

y_pred = model.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = np.argmax(y_test, axis=1)
confusion_mtx = confusion_matrix(y_true_labels, y_pred_labels)

print("Confusion Matrix:")
print(confusion_mtx)
```

Sl. 6.1. Kod za ispis matricu konfuzije

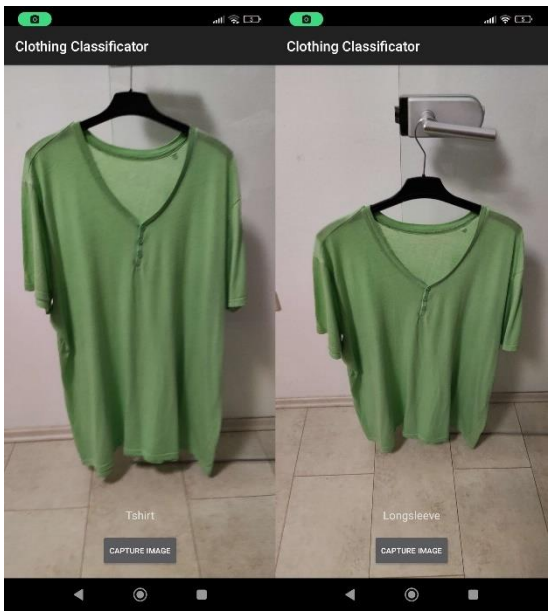
Kroz testiranje na testnom skupu vidljivo je da neke kategorije se često zamjenjuju. Na primjer u ovome testiranju nije ispravno prepoznata niti jedna bluza. Na glavnoj dijagonali su ipak vidljivi najčešći brojevi što je dokaz da je model zadovoljavajući. Isto tako polo se često prepoznaje kao T-shirt iako to nije. Većina klasa se dosta često prepoznaje kao T-shirt. Među „točnijim“ podacima su Body, Hat, Longsleeve, Outwear, Pants, Shirt, Shoes, T-shirt.

Predicted / True	Blazer	Blouse	Body	Hat	Hoodie	Longsleeve	Outwear	Pants	Polo	Shirt	Shoes	Skirt	Top	Tshirt	Undershirt
Blazer	3	0	0	0	0	4	9	0	0	5	0	0	0	0	0
Blouse	0	0	0	0	0	2	2	2	0	8	3	0	0	19	1
Body	0	0	12	0	0	1	0	1	0	0	0	0	0	3	0
Hat	0	0	0	28	0	13	5	2	0	2	11	16	0	15	0
Hoodie	0	0	0	0	2	7	8	1	0	5	1	2	0	6	0
Longsleeve	0	0	1	0	1	101	16	3	0	12	1	0	0	39	0
Outwear	1	0	0	0	2	11	47	1	0	7	2	0	0	9	0
Pants	0	0	0	0	0	1	2	116	0	2	3	0	0	4	0
Polo	0	0	0	0	1	3	0	1	0	0	1	1	1	38	0
Shirt	0	0	0	1	0	18	9	0	0	41	0	0	0	14	0
Shoes	0	0	0	5	0	1	0	1	0	1	68	0	0	4	0
Skirt	0	0	0	2	0	4	3	1	0	7	0	7	0	14	2
Top	0	0	0	0	0	2	2	0	0	0	1	1	2	8	0
Tshirt	0	0	0	0	0	2	0	3	0	2	0	0	0	195	0
Undershirt	0	0	0	1	0	0	0	0	0	1	0	1	1	19	2

Sl. 6.2. Matrica konfuzije

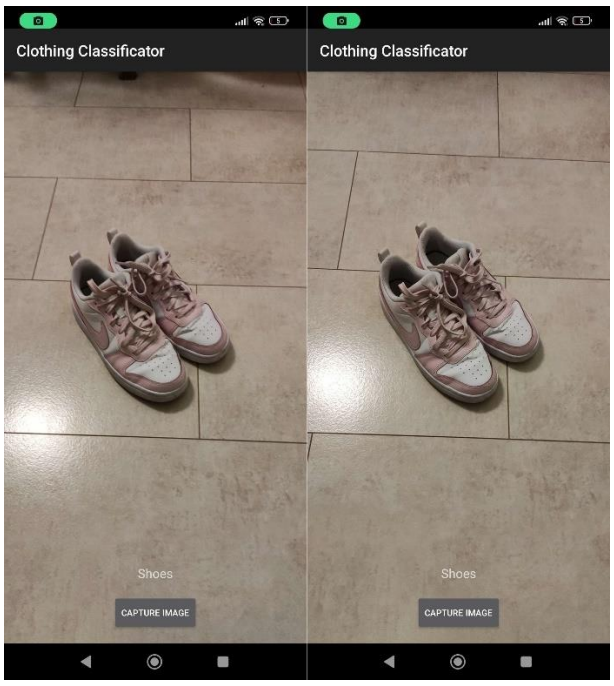
6.2. Testiranje rada aplikacije

Na slici 6.3. vidljiva je klasifikacija koja je dobivena slikanjem iste majice iz dva različita kuta. Vidljivo je da ju u prvome slučaju prepoznaje kao *T-shirt* dok je iz drugoga kuta prepoznaje kao *longsleeve* ili majicu dugih rukava

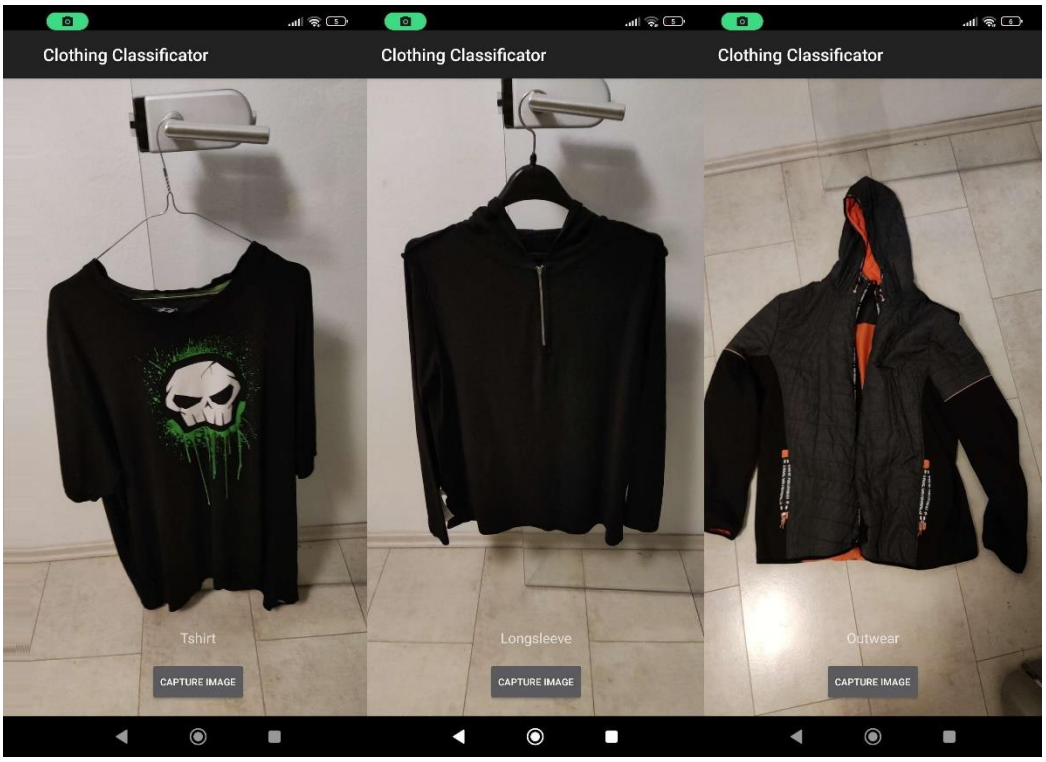


Sl. 6.3. Primjer klasifikacije majice (shirt)

Testiranjem klasifikacije na obući (tenisicama) se gotovo uvijek dobije ispravan rezultat, kao što je to vidljivo na slici 6.4.

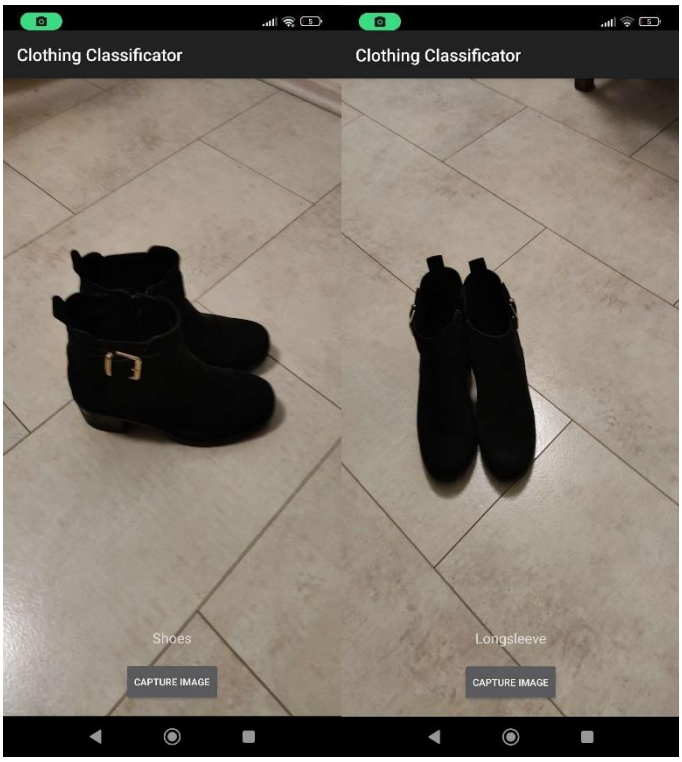


Sl. 6.4. Primjer klasifikacije tenisica (shoes)

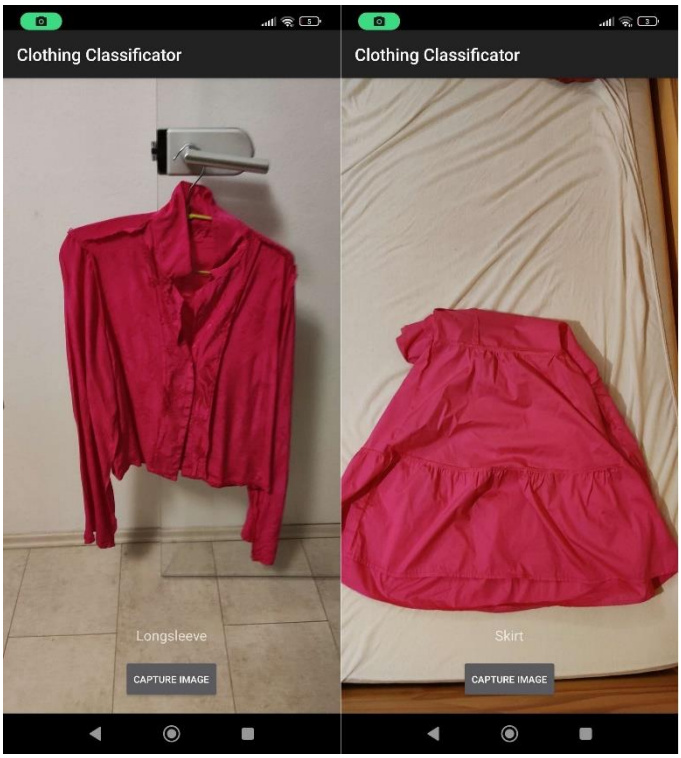


Sl. 6.5. Još nekoliko primjera klasifikacije

Na slici 6.5. je vidljivo kako je ispravno klasificiran *T-shirt* i *Outwear*, dok je umjesto *Hoodie*-a prepoznat *longsleeve*. Na slici 6.6. je vidljivo kako se pomiče kut, tako se mjenja i klasifikacija.



Sl. 6.6. Klasifikacija čizmica (*Shoes*) iz različitog kuta



Sl. 6.7. Još nekoliko primjera klasifikacije

I ono što je vidljivo iz gornje slike, ali i iz matrice konfuzije je to da se bluze gotovo nikada ne prepoznaju kao bluze, već kao druge stvari. Suknja je ipak prepoznata točno.

ZAKLJUČAK

Cilj ovoga projekta je bio kreirati model koji za danu ulaznu sliku određuje njenu klasu. Za potrebe testiranja u stvarnome svijetu kreirana je Android aplikacija koja koristi prethodno istrenirani model kako bi odgonetnula klasu odjeće. Iz dobivenih rezultata vidljivo je da model nije savršen i često se događa da krivo prepozna neku klasu odjeće. Može se vidjeti da su slike koje su skinute sa interneta u neku ruku više nego nerealne. U stvarnome svijetu rijetko se događa da je odjeća prikazana kao na web trgovinama, pod manjim osvjetljenjem i sa različitim pozadinama. Iako model nije savršen, rezultati su pokazali da je moguće prepoznati određene klase odjeće s određenom točnošću. Daljnje poboljšanje ovog modela moglo bi se postići prikupljanjem većeg broja raznolikih slika odjeće koje su snimljene u stvarnom okruženju, koristeći kameru mobilnog uređaja. To bi omogućilo modelu da nauči prepoznavati odjevne predmete u različitim uvjetima osvjetljenja, s različitim pozadinama i drugim varijacijama koje se mogu pojaviti u stvarnom svijetu.

LITERATURA

1. Machine learning https://www.ibm.com/topics/machine-learning pristup ostvaren 20. 6. 2023.
2. Deep neural networks, https://www.bmc.com/blogs/deep-neural-network/ pristup ostvaren 21.6.2023.
3. About Python, https://www.python.org/about/ , pristup ostvaren 21.6.2023.
4. Project Jupyter, https://jupyter.org/ , pristup ostvaren 21.2.2023.
5. Android Studio https://developer.android.com/studio pristup ostvaren 22. 2. 2023.
6. Running ML models in android using tflite https://medium.com/analytics-vidhya/running-ml-models-in-android-using-tensorflow-lite-e549209287f0 pristup ostvaren 22. 2. 2023.