SUNS – Zadanie 5:
Konvolučné siete

Petra Kirschová

1 NAČÍTANIE A SPRACOVANIE DÁT

Na prácu s dataframe a csv som použila knižnice `pandas` a `numpy`, na načítavanie obrázkov `keras`.

Ako prvé som načítala dataframe zo styles.csv s atribútmi obrázkov. Pridala som doňho nový stĺpec *filename*, ktorý obsahuje názvy obrázkov vytvorené podľa *id*.

Na klasifikáciu som si vybrala stĺpec *masterCategory*. V tomto stĺpci sa nachádzali triedy *„Home“*, „Sporting Goods“ a „Free Items“, ktoré sa vyskytovali v datasete veľmi málo, preto som záznamy s týmito triedami odstránila. V datasete po úprave zostalo 44293 záznamov a v rámci masterCategory sa klasifikujú 4 triedy: „Accessories“, „Apparel“, „Footwear“ a „Personal Care“.

Dataframe som následne rozdelila na trénovaciu a testovaciu množinu v pomere 8:2.

Na načítavanie obrázkov som použila ImageDataGenerator. Vytvorila som 2 generátory – jeden pre testovacie a jeden pre trénovacie a validačné dáta. Generátor testovacích dát spracováva celú testovaciu množinu, generátor trénovacích dát rozdeľuje trénovacie dáta ďalej na trénovaciu a validačnú množinu v pomere 8:2. V obidvoch generátoroch prebieha normalizácia obrázkov.

```
# img generator pre trenovacie a validacne data
train_val_gen = ImageDataGenerator(rescale=1.0 / 255.0, validation_split=0.2)
# img generator pre testovacie data
test_gen = ImageDataGenerator(rescale=1.0 / 255.0)
```

Pomocou funkcie `flow_from_dataframe()` som namapovala obrázky k triedam z datasetu pre každú množinu zvlášť.

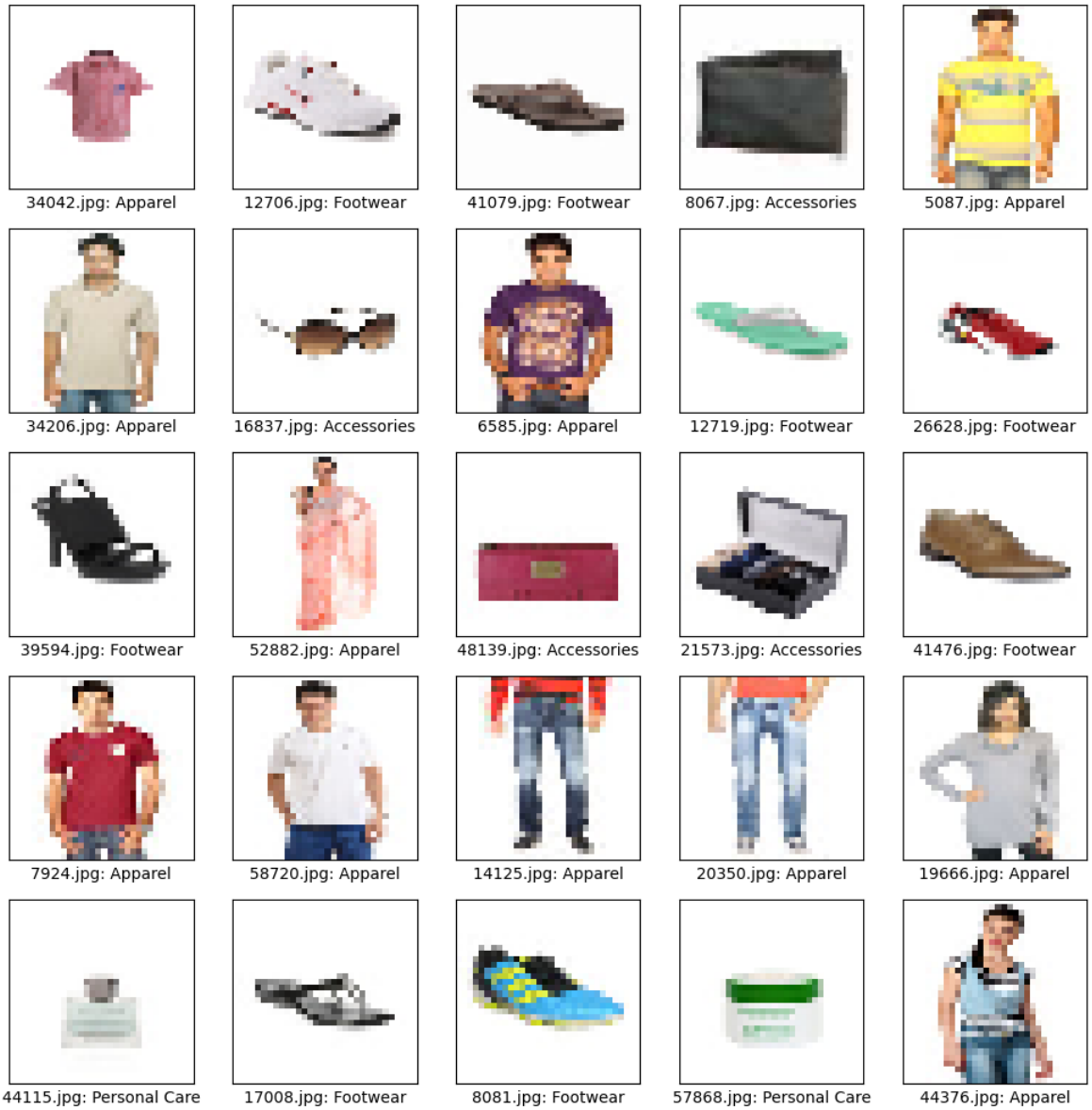
Napr. pre trénovaciu množinu:

- **dataframe** = dataframe načítaný zo styles.csv
- **directory** = priečinok s obrázkami
- **x_col** = stĺpec s názvami obrázkov
- **y_col** = stĺpec s triedami
- **target_size = (32,32)** – veľkosť obrázku
- **batch_size = 400** – počet obrázkov v 1 batchi
- **subset** = trénovacia podmnožina
- **class_mode** = triedy sú reprezentované ako kategórie v one-hot encodingu
- **shuffle=False** – zachováva sa poradie

```
# trenovacie data
train_img_generator = \
    train_val_gen.flow_from_dataframe(
        dataframe=train_labels,
        directory="data/images",
        x_col="filename",
        y_col="masterCategory",
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        subset="training",
        class_mode="categorical",
        shuffle=False
    )
```

Obrázky som spätne vykreslila pomocou `seaborn` a `matplotlib`. Podľa vykreslených obrázkov je vidno, že do image generátora boli načítané správne a majú aj správne priradené triedy.

Prvých 25 obrázkov v trénovacej množine:



2 KLASIFIKÁTOR

Na trénovanie konvolučnej siete som použila knižnice `keras` a `tensorflow`, na kreslenie grafov `seaborn` a `matplotlib`.

2.1 ROZDELENIE DÁT

- **Pomer rozdelenia dát:** Trénovacie : testovacie : validačné = 6 : 2 : 2
- **Vstup neurónovej siete:** Obrázky veľkosti (32, 32, 3) vygenerované pomocou ImageDataGeneratora
- **Výstup neurónovej siete:** Triedy zo stĺpca masterCategory v one-hot encodingu (4 triedy = 4 neuróny)

2.2 NASTAVENIE KLASIFIKÁTORA

Pre vytvorenie neurónovej siete som použila Sequential model:

```
model = Sequential()
```

Vstupná vrstva = konvolučná 2D vrstva s aktivačnou funkciou „relu“, do ktorej vstupujú obrázky tvaru in_shape = (32, 32, 3) – veľkosť 32x32 a 3 kanály rgb. Obsahuje 32 filtrov, veľkosť kernelu som pri testovaní parametrov skúšala (3,3) a (5,5). Za konvolučnou vstvou nasleduje MaxPooling vrstva.

```
model.add(Conv2D(32, kernel_size, activation="relu",
                 input_shape=in_shape))
model.add(MaxPooling2D())
```

Konvolučná vrstva a MaxPooling sa následne znovu opakujú. Druhá konvolučná vrstva obsahuje 64 filtrov a používa sa v nej regularizer=None alebo „l2“.

```
model.add(Conv2D(64, kernel_size, activation="relu",
                 kernel_regularizer=regularizer))
model.add(MaxPooling2D())
```

Na záver nasleduje vrstva Flatten, ktorá vytvorí z 2D vstupu jednorozmerné pole, Dense vrstva so 64 neurónmi a aktivačnou funkciou „relu“ a výstupná Dense vrstva so 4 neurónmi (out_shape = 4) a aktivačnou funkciou „softmax“.

```
model.add(Flatten())
model.add(Dense(64, activation="relu"))
model.add(Dense(out_shape, activation="softmax"))
```

Pri kompilácii modelu som nastavila optimizer „Adam“ a learning rate som pri testovaní nastavila na 0.01, 0.001 alebo 0.0001. Chyba sa počíta ako categorical crossentropy, vzhľadom na to, že klasifikátor zatrieďuje obrázky do viacerých tried, metrika je accurracy – percentuálne vyjadrenie zhody s pôvodnými triedami vzoriek.

```
# trenovanie klasifikatora
classifier.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                   loss='categorical_crossentropy', metrics=['accuracy'])
```

Trénovanie prebiehalo pri 30 epochách. Pri trénovaní zároveň prebiehala aj validácia na validačných dátach. Parametre steps_per_epoch a validation_steps som vypočítala podielom počtu vzoriek v množinách s veľkosťou batchu (používala som batch size = 400).

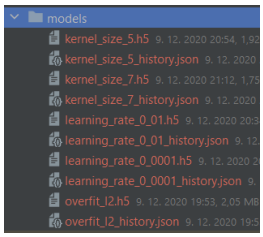
```
steps_per_epoch = math.ceil(train_img_generator.n / BATCH_SIZE)
validation_steps = math.ceil(validation_img_generator.n / BATCH_SIZE)

fit = classifier.fit(train_img_generator,
                     verbose=True, epochs=30,
                     steps_per_epoch=steps_per_epoch,
                     validation_data=validation_img_generator,
                     validation_steps=validation_steps)
```

Vytvorený a natrénovaný model ukladám do súboru vo formáte .h5 a históriu trénovania do .json súboru.

```
# ulozenie modelu do suboru
classifier.save(saved_model_filename+".h5")

# ulozenie history
with open(saved_model_filename+'_history.json', 'w') as file:
    json.dump(fit.history, file)
```



Z týchto súborov následne načítavam model aj históriu trénovania. Načítaný model používam na predikciu tried pre testovacie dáta a z histórie vykresľujem grafy pre priebeh presnosti a kriteriálnej funkcie.

```
# nacitanie modelu zo suboru
loaded_classifier = load_model(saved_model_filename+".h5")

# nacitanie history
with open(saved_model_filename+'_history.json') as file:
    loaded_history = json.load(file)
```

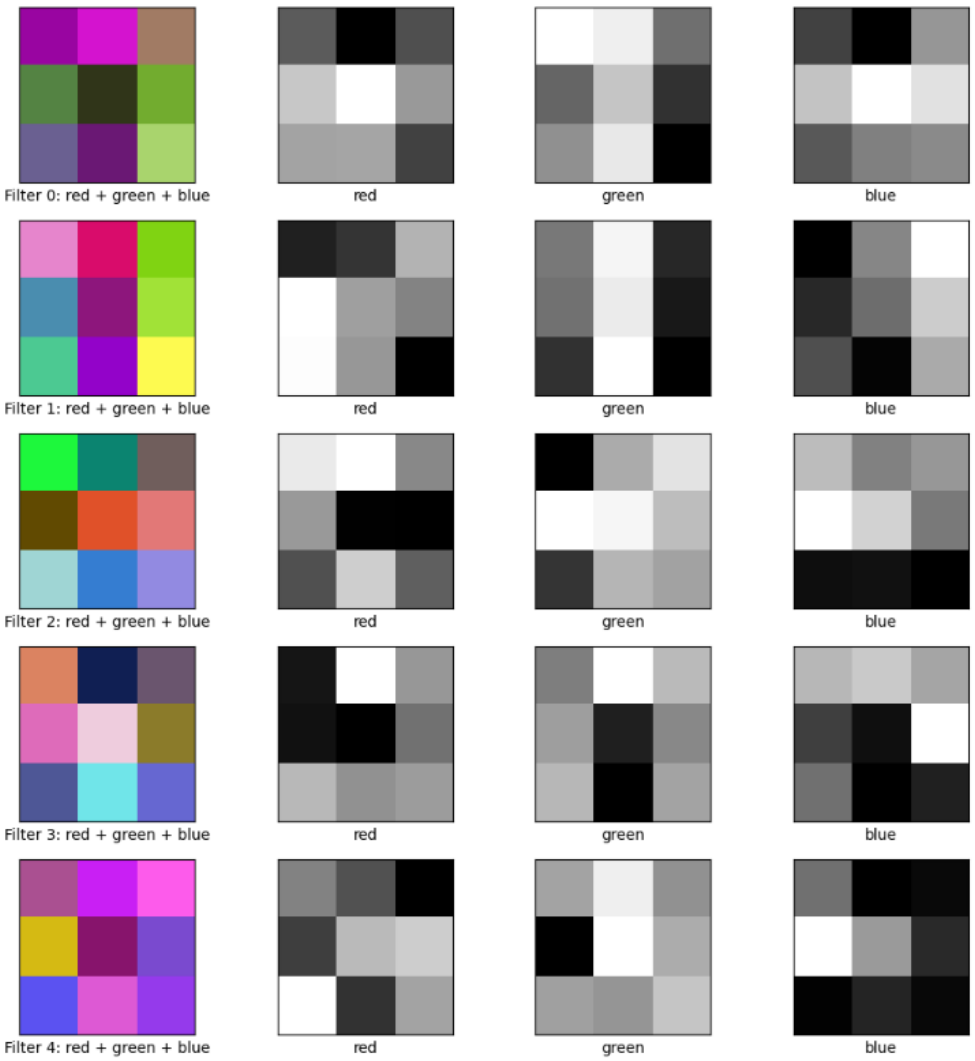
2.3 VIZUALIZÁCIA FILTROV V PRVEJ KONVOLUČNEJ VRSTVE

Prvá vrstva obsahuje 32 filtrov. Pre veľkosť kernelu (3,3) s tromi farebnými (rgb) kanálmi sú filtre reprezentované ako pole veľkosti (3, 3, 3, 32). Filtre z prvej vrstvy som získala takto:

```
# prva konvolucna vrstva
conv_layer = model.layers[0]

# filtre vo vrstve
filters = conv_layer.get_weights()[0]
```

Na obrázku je vykreslených prvých 5 filtrov so všetkými farebnými kanálmi, ako aj pre každý kanál zvlášť. Filtre som normalizovala, vďaka čomu boli farby zobrazené jasnejšie.

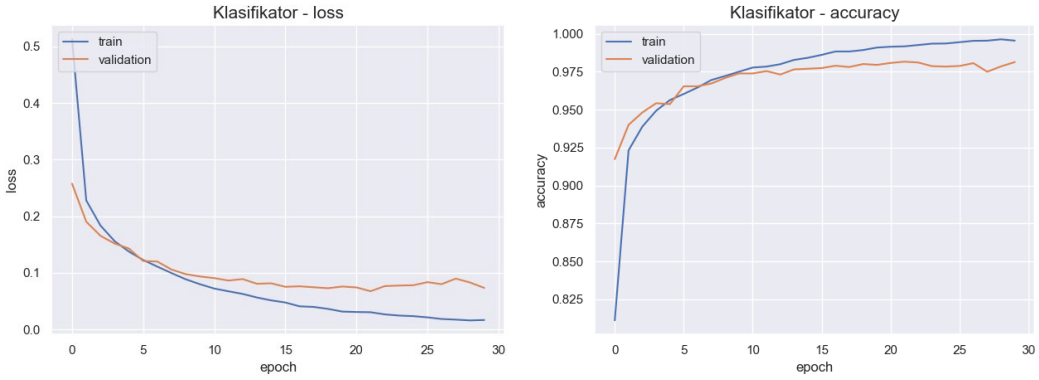


2.4 PRIEBEH TRÉNOVANIA PRE RÔZNE PARAMETRE SIETE

2.4.1 Learning_rate=0.001, kernel_size=(3,3), kernel_regularizer=None

Prvý testovaný model dosiahol veľmi vysokú presnosť trénovania – až 99%, no na grafoch pre vývoj chyby a presnosti je vidno, že je pretrénovaný, preto bolo potrebné do modelu zaviesť regularizáciu.

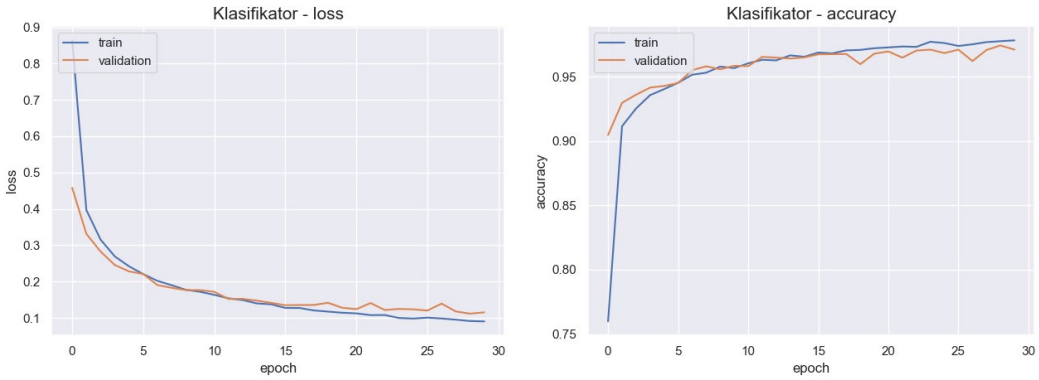
```
- loss: 0.0167 - accuracy: 0.9955 - val_loss: 0.0734 - val_accuracy: 0.9814
```



2.4.2 Learning_rate=0.001, kernel_size=(3,3), kernel_regularizer="l2"

Pri tomto modeli sú všetky pôvodné nastavenia rovnaké, ale je aplikovaná l2 regularizácia na druhú konvolučnú vrstvu. Vďaka regularizácii sa chyba trénovania a validácie líši iba minimálne a výsledná presnosť je 97.7%, čo je stále veľmi vysoká hodnota.

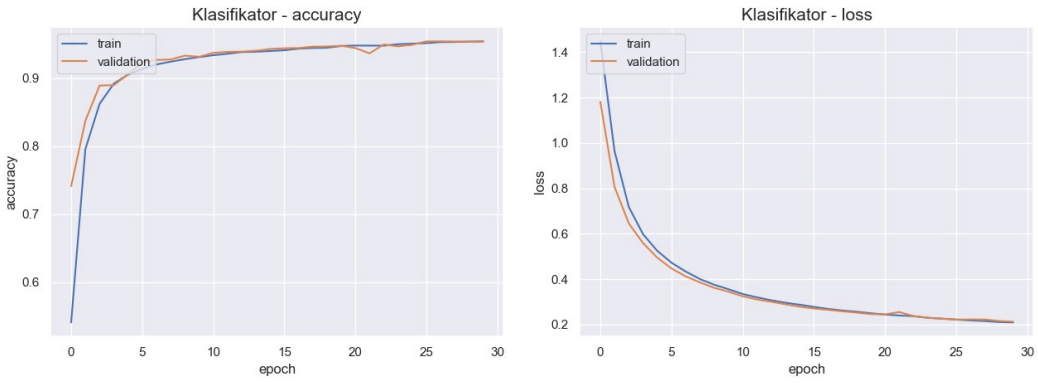
```
loss: 0.0956 - accuracy: 0.9772 - val_loss: 0.1154 - val_accuracy: 0.9716
```



2.4.3 Learning_rate=0.0001, kernel_size=(3,3), kernel_regularizer="l2"

Pre model s menším learning rate boli grafy chyby a presnosti trochu menej strmé, no celková presnosť trénovania klesla na 95.4%.

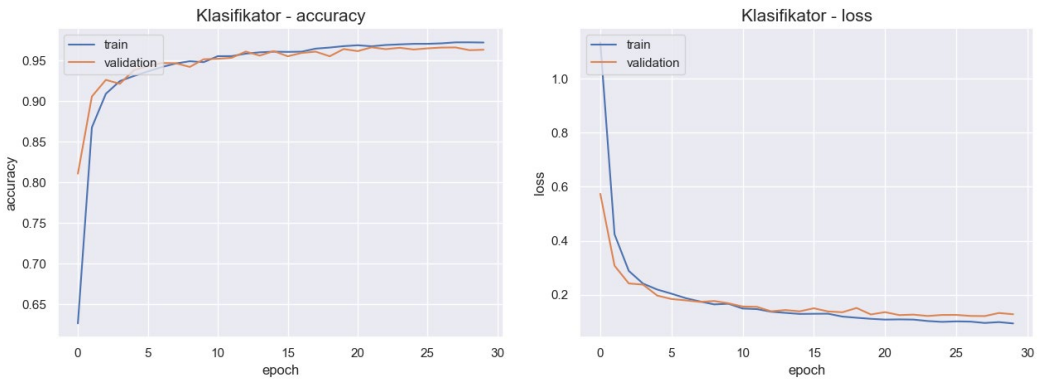
```
loss: 0.2089 - accuracy: 0.9540 - val_loss: 0.2123 - val_accuracy: 0.9534
```



2.4.4 Learning_rate=0.01, kernel_size=(3,3), kernel_regularizer="l2"

Vyšší learning rate takisto nezabezpečil vyššiu presnosť – v tomto prípade je presnosť 97.2%. Grafy sú strmšie, ako pri nižšom learning rate.

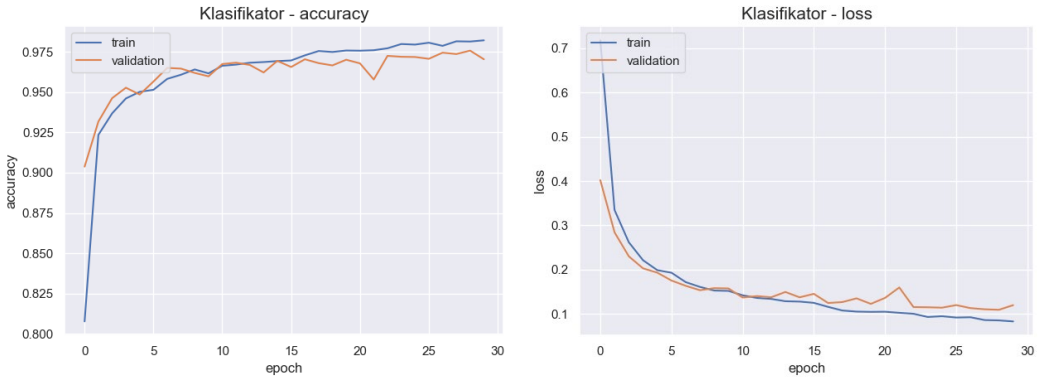
```
loss: 0.0943 - accuracy: 0.9723 - val_loss: 0.1282 - val_accuracy: 0.9634
```



2.4.5 Learning_rate=0.001, kernel_size=(5,5), kernel_regularizer="l2"

Zvýšením veľkosti kernelu sa aj pri použití l2 regularizácie javí mierne pretrénovanie. Úspešnosť trénovania je vysoká – 98.2%.

```
loss: 0.0830 - accuracy: 0.9821 - val_loss: 0.1198 - val_accuracy: 0.9704
```



Po vyskúšaní rôznych nastavení som usúdila, že pre túto neurónovú sieť sú optimálne parametre learning_rate=**0.001**, veľkosť kernelu (**3,3)** a použitá **l2 regularizácia**.

2.4.6 Výsledky pre testovaciu množinu

Na predikciu tried na trénovacej množine som použila model s najlepšie nastavenými parametrami: learning_rate=0.001, kernel_size=(3,3), kernel_regularizer="l2" (**2.4.2**).

Výsledky pre testovacie dáta som vizualizovala pomocou classification report a confusion matice. Presnosť zatriedenia nových vzoriek z testovacích dát bola cca 97%. Na confusion matici je tiež vidno, že zatrieďovanie do tried bolo úspešné, pretože najväčšie hodnoty sú na hlavnej diagonále matice – väčšina vzoriek bola zatriedená správne.

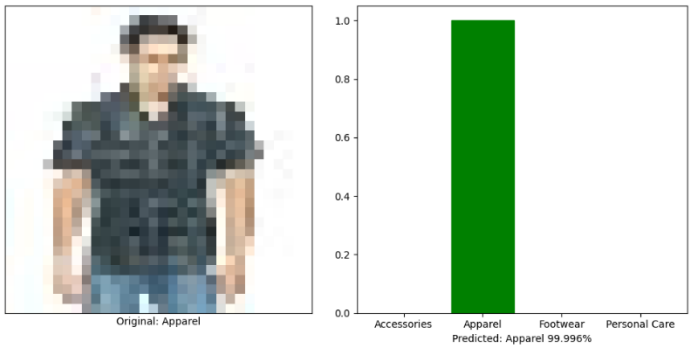
Classification report

	precision	recall	f1-score	support
Accessories	0.9267	0.9655	0.9457	2173
Apparel	0.9853	0.9748	0.9800	4323
Footwear	0.9984	0.9808	0.9895	1873
Personal Care	0.9414	0.9184	0.9298	490
accuracy			0.9707	8859
macro avg	0.9629	0.9599	0.9612	8859
weighted avg	0.9712	0.9707	0.9708	8859

Confusion matrix

Predikciu tried pre niektoré testovacie vzorky som zobrazila aj pomocou bar grafov, kde sú zobrazené percentuálne pravdepodobnosti zatriedenia danej vzorky do jednotlivých kategórií.

Napríklad pánsku košeľu klasifikátor správne zatriedil do triedy Apparel so 100% pravdepodobnosťou.



Tieto šaty naopak boli zatriedené nesprávne do kategórie Accessories. Môže to byť kvôli tomu, že na tomto obrázku sa nenachádza človek a v predošlom áno, z čoho mohol klasifikátor usúdiť, že obrázok nepatrí medzi Apparel.