Dokumentácia - Zadanie 5

Meno: Martin Michale
AisID: 92296
Programovací jazyk: Python 3.8
Použité knižnice: Pandas, SciKit learn, cv2/OpenCV, Tensorflow/Keras

Úloha 1: Načítanie obrázkov a príprava na spracovanie. Normalizácia, prevedenie obrázkov do štvorcového formátu, priradenie kategórií k obrázkom, vytvorenie množín pre trénovanie, testovanie a validáciu a overenie správnosti načítania.

Prvý pokus o spracovanie dát prebiehal tak, že som načítal každý obrázok do stĺpca v dataframe a do ďalšieho stĺpca som zapísal príslušné informácie o obrázku, teda pre aké pohlavie je určený. Tento pokus fungoval ale nakoľko nemám nejak extra veľa RAM v počítači tento pokus zlyhal približne pri 20 000-om obrázku.

Druhý pokus, teraz už funkčný fungoval tak že som implementoval ImageDataGenerator z knižnice keras_preprocesing. Táto implementácia má fungovať tak že model konvolučnej siete si bude sám načítavať obrázky podľa vlastnej potreby pred každou epochou, čo znamená že nebudú musieť byť všetky uložené v RAM. Pred samotným použitím generátora je treba vytvoriť Dataframe ktorý bude obsahovať 2 stĺpce, jeden kde sú uvedené názvy jednotlivých obrázkov a v druhom sú k nim priradené pohlavie pre ktoré sú určené. Výsledný dataframe vyzeral nasledovne:

	id	gender
0	57029.jpg	Women
1	16617.jpg	Men
2	24154.jpg	Women
3	13826.jpg	Men
4	4039.jpg	Unisex
5	2880.jpg	Women
6	16404.jpg	Men
7	30661.jpg	Men
8	19912.jpg	Unisex
9	57352.jpg	Women
10	22130.jpg	Men
11	53029.jpg	Women
12	50746.jpg	Men
13	47331.jpg	Women
14	55802.jpg	Women
15	7135.jpg	Women
16	17141.jpg	Women
17	59972.jpg	Women
18	10216.jpg	Women
19	13253.jpg	Men
20	24069.jpg	Men

Pred vložením do Dataframu som musel rozdeliť dáta na trénovanie a testovanie. Tieto dáta som rozdelil v pomere 9 ku 1, teda 90% pre trénovanie a 10% pre testovanie. Následne som tieto dáta spojil do dvoch Dataframov ktoré vyzerali tak ako ten uvedený vyššie. Tieto Dataframy som dal do generátorov. Generátor pre dáta na trénovanie vyzeral nasledovne:

```
trainGenerator = trainDataGenerator.flow_from_dataframe(dataframe=trainDF,
                                                        directory=dataPath,
                                                        x_col='id',
                                                        y_col='gender',
                                                        target_size=(imgHeightWidth, imgHeightWidth),
                                                        batch_size=25,
                                                        subset="training",
                                                        class_mode="categorical")
```

Generátor musí obsahovať niekoľko parametrov. Do parametru dataframe sa vkladá hore uvedený dataframe, parameter directory je celá cesta k zložke k obrázkom. Parametre x_col a y_col sú názvy stĺpcov ktoré majú byť použité ako vstupné a ktoré ako výstupné, target_size udáva na akú veľkosť má byť obrázok prevedený (resiznutý), batch_size udáva na koľko batchov má byť daná množina dát rozdelená, subset uchováva to načo je určený daný generátor. Pre každú množinu som si vytvoril jeden generátor. Na normalizovanie dát som použil nasledovné príkazy:

```
trainDataGenerator = ImageDataGenerator(rescale=1 / 255, validation_split=0.1)
testDataGenerator = ImageDataGenerator(rescale=1 / 255)
```

Normalizovanie prebiehalo tak že som dáta obrázkov (jednotlivé pixely) predelil 255, nakoľko to je maximálna možná hodnota jednej z troch farieb každého pixelu.

Overenie dát:

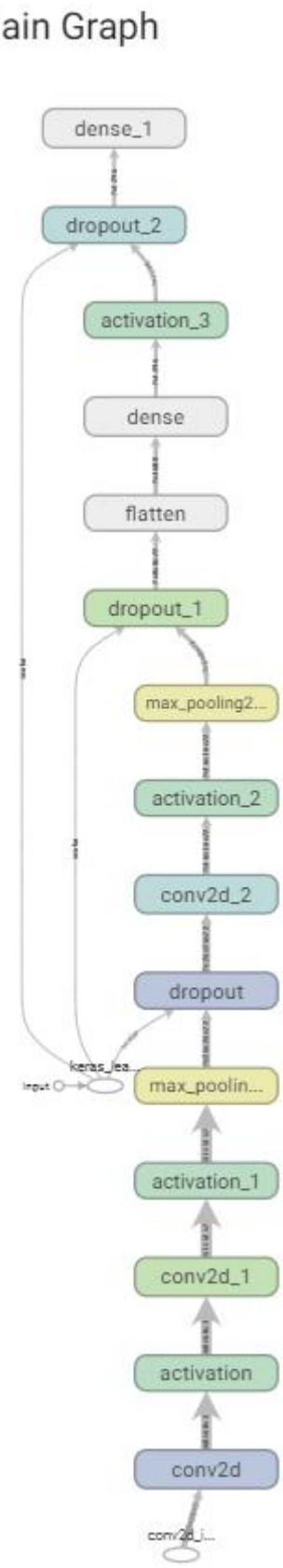
```
Found 35997 validated image filenames belonging to 5 classes.
Found 3999 validated image filenames belonging to 5 classes.
Found 4444 validated image filenames belonging to 5 classes.
```

Pri načítaní dát mi do konzole vypísalo tieto údaje. Tieto údaje som spočítal a zhodujú sa z množstvom obrázkov ktoré som dostal z výnimkou tých ktoré sú v csv súbore, ale neexistujú ako súbory.

Úloha 2: Trénovanie konvolučnej siete na klasifikáciu oblečenia. Sieť musí obsahovať konvolučnú vrstvu, nelinearitu a plne prepojené vrstvy.

Po načítaní, normalizovaní a prevedení veľkosti bolo možné pristúpiť k samotnému trénovaniu siete, moja sieť mala nasledovnú architektúru:

```
# trenovanie
tensorboard = TensorBoard(log_dir="TensorBoardLogs/{}".format(NAME))
model = Sequential()
model.add(Conv2D(36, (10, 10),
                 padding='same',
                 input_shape=(60, 60, 3)))
model.add(Activation('relu'))

model.add(Conv2D(72, (10, 10)))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(72, (10, 10)))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1296, activation='relu'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(5, activation='softmax'))

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



V mojom prípade som vytvoril sieť ktorá obsahovala 3 konvolučné 2D vrstvy, prvá z nich obsahovala 36 filtrov o veľkosti 10x10 pixelov a ako vstup prijímala farebné obrázky z veľkosťou 60x60 pixelov. Ostatné dve konvolučné vrstvy obsahovali 72 filtrov každá, tiež o veľkosti 10x10 pixelov. Medzi týmito vrstvami som dal aktivačné vrstvy relu. Po druhej a tretej vrstve som dal aj maxpooling2D, čo je maximálna operácia združovania priestorových údajov 2D. Dal som tam aj dropout vrstvu s parametrom 0.25. Po týchto vrstvách som dáta "sploštil" pomocou flatten vrstvy a pridal som ešte dve vrstvy s neurónmi. Prvá z týchto vrstiev mala 1296 neurónov a aktivačnú funkciu relu. Posledná, výstupná vrstva, mala 5 neurónov čo je počet možných výstupných údajov, aktivačnú funkciu som zmenil a dal som softmax. Medzi tieto dve vrstvy som pridal ešte aktivačnú vrstvu relu a dorpout vrstvu z parametrom 0.5.

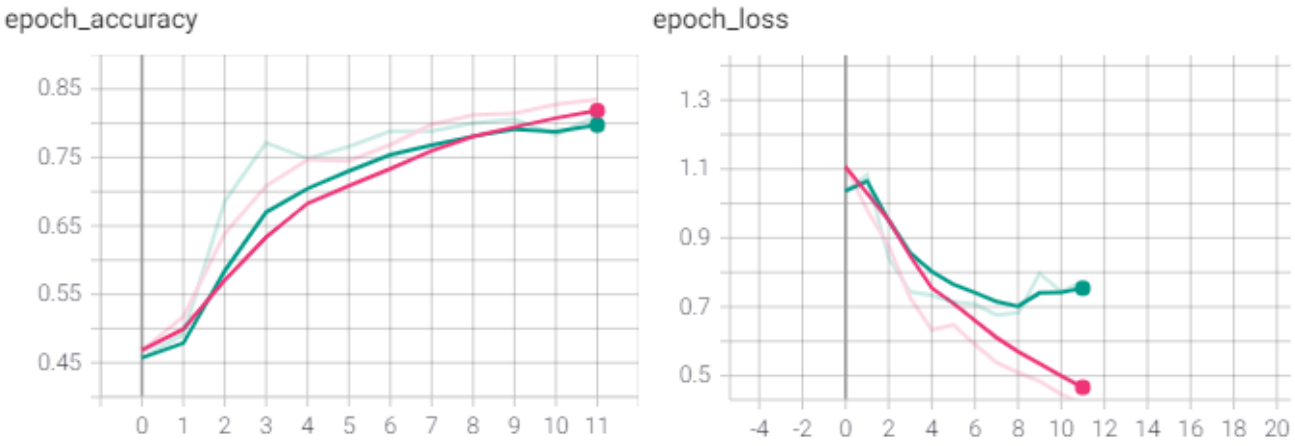
Konečná úspešnosť tohto modelu bola 87,6 % na testovacej množine. Trénovanie trvalo približne 8 hodín na mojom zariadení pri nastavení 12 epoch. Toto som vyhodnotil pomocou nasledujúceho kódu:

```
# evaluate the model
scores = model.evaluate(testGenerator, verbose=0, callbacks=[tensorboard])
print("%s: %.2f%%" % (model.metrics_names[1], scores[1] * 100))
```

Trénovanie prebiehalo nejak nasledovne (obrázok je ilustračný, spustený len na 2000 vzorkách nakoľko som si nezaznamenal ten keď som si ho spúšťal pre všetky dáta):

```
64/64 [==============================] - 85s 1s/step - loss: 1.1066 - accuracy: 0.4683 - val_loss: 1.0362 - val_accuracy: 0.4571
Epoch 2/12
64/64 [==============================] - 82s 1s/step - loss: 0.9824 - accuracy: 0.5179 - val_loss: 1.0826 - val_accuracy: 0.4914
Epoch 3/12
64/64 [==============================] - 84s 1s/step - loss: 0.8758 - accuracy: 0.6389 - val_loss: 0.8407 - val_accuracy: 0.6857
Epoch 4/12
64/64 [==============================] - 85s 1s/step - loss: 0.7263 - accuracy: 0.7085 - val_loss: 0.7434 - val_accuracy: 0.7714
Epoch 5/12
64/64 [==============================] - 108s 2s/step - loss: 0.6337 - accuracy: 0.7467 - val_loss: 0.7340 - val_accuracy: 0.7486
Epoch 6/12
64/64 [==============================] - 111s 2s/step - loss: 0.6473 - accuracy: 0.7448 - val_loss: 0.7139 - val_accuracy: 0.7657
Epoch 7/12
64/64 [==============================] - 109s 2s/step - loss: 0.5908 - accuracy: 0.7687 - val_loss: 0.7078 - val_accuracy: 0.7886
Epoch 8/12
64/64 [==============================] - 108s 2s/step - loss: 0.5384 - accuracy: 0.7981 - val_loss: 0.6762 - val_accuracy: 0.7886
Epoch 9/12
64/64 [==============================] - 108s 2s/step - loss: 0.5086 - accuracy: 0.8119 - val_loss: 0.6829 - val_accuracy: 0.8000
Epoch 10/12
64/64 [==============================] - 108s 2s/step - loss: 0.4841 - accuracy: 0.8144 - val_loss: 0.7968 - val_accuracy: 0.8057
Epoch 11/12
64/64 [==============================] - 107s 2s/step - loss: 0.4465 - accuracy: 0.8276 - val_loss: 0.7450 - val_accuracy: 0.7829
Epoch 12/12
64/64 [==============================] - 108s 2s/step - loss: 0.4164 - accuracy: 0.8345 - val_loss: 0.7716 - val_accuracy: 0.8114
accuracy: 84.50%
```

Priebeh trénovania pre 12 epoch prebiehal takto:



Čo sa týka rýchlosti dobrej hodnoty pre parameter rýchlosti učenia tak som skúsil dve hodnoty nakoľko som to chcel mať presné a teda pre všetky dáta a to je dosť časovo náročne v mojom prípade. Na zmenu rýchlosti učenia som použil nasledujúci kód:

```
optimizer = Adam(learning_rate=0.001)
```

Ten som potom dával ako parameter do compile modelu. Tu som skúsil dve hodnoty: 0.01 a 0.001. V mojom prípade bolo lepšie použiť 0.001 ktoré malo úspešnosť 87.5% na testovacích dátach, oproti tomu 0.01 malo 45.8% na testovacích dátach čo je v mojom prípade dosť veľký rozdiel.

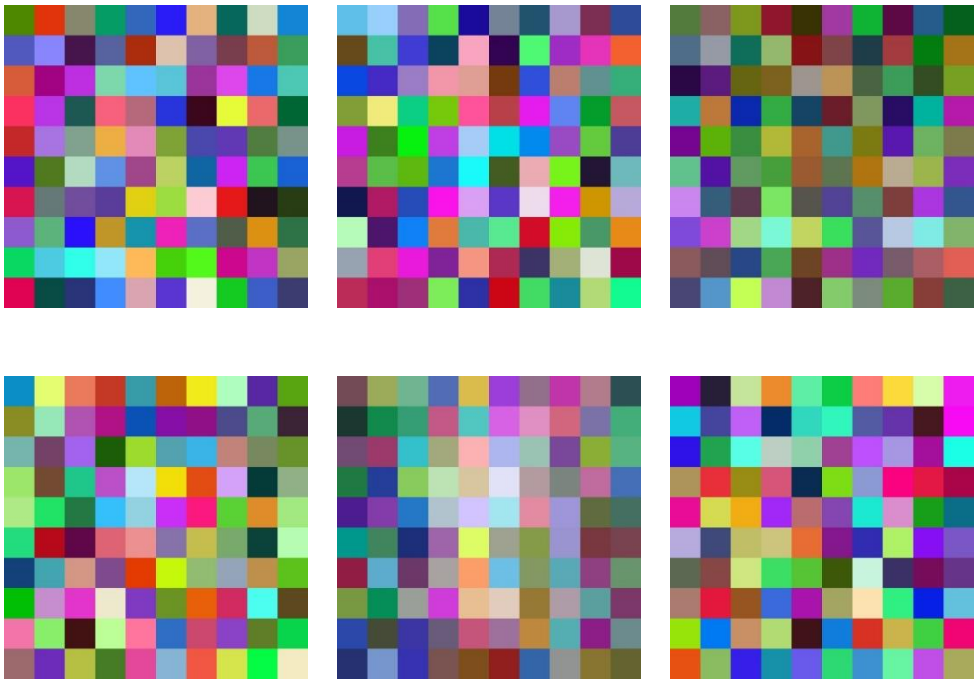
Bonus:

Vizualizácia filtrov z prvej vrstvy:

Vizualizáciu som robil pomocou nasledujúceho kódu:

```
# vykreslenie filtrov
filters, biases = model.layers[0].get_weights()
for pc in range(36):
    filterImage = []
    for row in range(10):
        filterImage.append([])
        for col in range(10):
            filterImage[row].append([])
            filterImage[row][col] = [filters[row][col][0][pc], filters[row][col][1][pc], filters[row][col][2][pc]]
    filterImage = numpy.array(filterImage)
    filterImage = filterImage - filterImage.min()
    filterImage = filterImage / filterImage.max()
    filterImage = filterImage * 255
    filterImage = cv2.resize(filterImage, (500, 500), interpolation=cv2.INTER_NEAREST)
    cv2.imwrite('filters/filter{}.jpg'.format(pc), filterImage)
```

Najprv som si zistil ako pristúpiť k filtrom. To sa robilo pomocou prvého riadku ktorý vrátil do premennej filters filtre z prvej vrstvy. Potom som pomocou for cyklu prešiel a vykreslil všetkých 36 filtrov následne som ich previedol naspäť do normálneho formátu, teda krát 255, zväčšil na veľkosť 500x500 pixelov z veľkosti 10x10 pixelov a uložil po jednom do zložky filters. Tu je vzkreslených prvých 6 filtrov (ostatné sú v priečinku filters ktorý som pribalil k súboru, ak sa zmestil do aisu teda):



Analýza siete pomocou nástroja TensorBoard:

Najprv bolo treba inicializovať tensorbord, to som spravil nasledovne:

```
NAME = "Gender-classification-{}".format(int(time.time()))
```

```
tensorboard = TensorBoard(log_dir="TensorBoardLogs/{}".format(NAME))
```

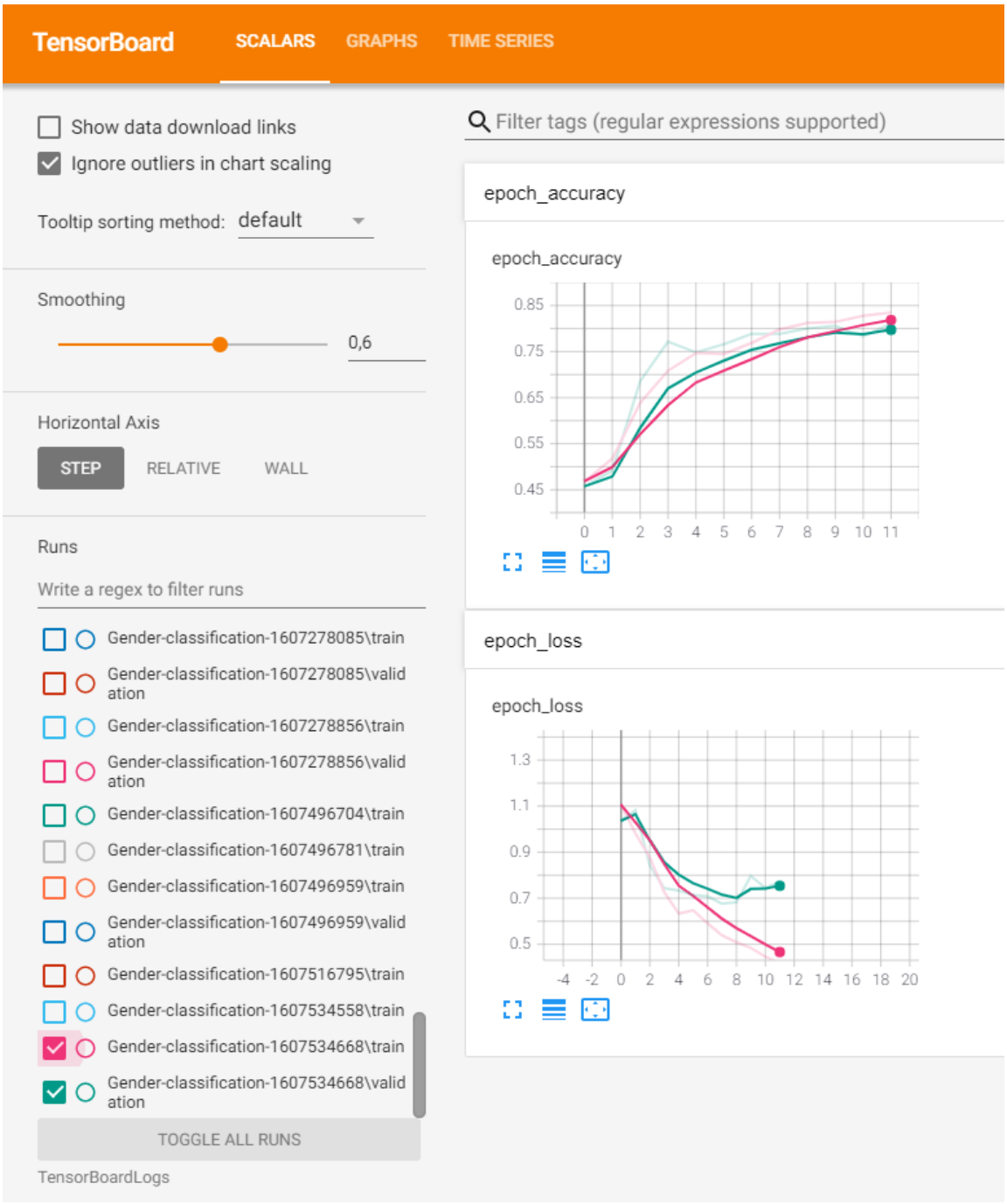
Túto premennú som potom vložil do fit callbacku:

```
history = model.fit(trainGenerator,
                    steps_per_epoch=1620 // 25,
                    epochs=12,
                    validation_data=valGenerator,
                    callbacks=[tensorboard, checkpointCallBack],
                    validation_steps=180 // 25)
```

Po spustení programu sa v zložke TensorBoardLogs vygeneruje zložka v ktorej sú uložené súbory potrebné na spustenie TensorBoard v prehliadači. Na spustenie som si musel v cmd otvoriť zložku ktorá obsahuje zložku TensorBoardLogs a nasledovne spustiť:

```
D:\STU FEI\Year4\SUNS\zadanie5test3>tensorboard --logdir=TensorBoardLogs
TensorFlow installation not found - running with reduced feature set.
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.4.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

Potom som mohol na stránke http://localhost:6006/ v prehliadači otvoriť tensorboard. Ten vyzeral nasledovne:



Od tadeto som si zistil grafy ktoré som už uviedol vyššie pre každé trénovanie ktoré prebehlo od momentu kedy bol tensorboard nastavený.

Ukladanie checkpoitov a testovanie modelu na vlastných obrázkoch:

Uloženie modelu prebiehalo nasledovne:

```
# checkpoints
checkpointPath = "checkPoints/{}".format(NAME)
checkpointCallBack = ModelCheckpoint(filepath=checkpointPath,
                                     save_weights_only=True,
                                     monitor="val_accuracy",
                                     mode="max")
```

Pomocou vyššie uvedeného kódu som si checkpoint callback ktorý uloží váhy modeli pri najlepšej val_accuracy ktorá v tom trénovaní do ktorého je tento callback vložený nastane. Tento súbor sa uloží do zložky ktorá je v parametre fileptah.

Načítanie dát prebiehalo tak že som si v inom python súbore vytvoril ten istý model s tými istými nastaveniami vrstiev a pomocou nasledujúceho kódu som do modelu načítal váhy uložené z predchádzajúceho, uloženého modelu:

```
path = 'checkPoints/Gender-classification-1607496959'
model.load_weights(path)
```

Tento model je už naučený a ďalej som ho použil na odtestovanie na vlastných fotkách, teda na fotkách z internetu že naozaj naučený už je.

Testovanie na vlastných fotkách som spravil nasledovne: načítal som si už vopred naučený model, načítal som si obrázok ktorý sa bude testovať, znormalizoval som hodnoty, zmenšil ho, pridal dimenziu aby ho model vedel zobrať ako vstup a na koniec som nechal model nech predpovedá o aký obrázok sa jedná. Toto všetko riešil nasledujúci kód:

```
img = cv2.imread('data/testImages/borat.jpg')
img = cv2.resize(img, (60, 60))
img = img / 255
img = numpy.expand_dims(img, axis=0)
predict = model.predict(x=img)
predict = pandas.DataFrame(predict, columns=["Boys", "Girls", "Man", "Unisex", "Woman"])
```

Test obrázku a jeho vyhodnotenie 1:

Boys	Girls	Man	Unisex	Woman
0.00001	0.00000	0.93702	0.00005	0.06292

Tento obrázok vyhodnotilo ako pánske čo je z obrázku aj vidieť že asi bude pravda.

Test obrázku a jeho vyhodnotenie 2:

Boys	Girls	Man	Unisex	Woman
0.01110	0.03753	0.39248	0.02802	0.53087

Tento obrázok vyhodnotilo na 53.09% že sa jedná o dámsky kus oblečenia. Čo ma ale zarazilo je vysoké percento pri man.

Test obrázku a jeho vyhodnotenie 3:

Boys	Girls	Man	Unisex	Woman
0.00952	0.00093	0.92931	0.01354	0.04669

Pre posledný obrázok som chcel vybrať niečo trochu zákernejšie, preto som skúsil obrázok kde je muž ktorý má oblečený kilt (v konečnom dôsledku sukňu). Tu sieť nepoľavila a zaradila to tiež správne do mužského (z druhým najväčším skóre ale ženským, teda je vidieť že niečo ženské tam asi vidí).

Poznámka:

Keby bolo treba, prikladám link na GoogleDoc server kde sú všetky filtre z prvej vrsty, súbory potrebné na TensorBoard a Checkpointy z ukladania modelu.

Link:
https://drive.google.com/drive/folders/1X1nIVmbcRcGE81vctUtEefjbk4o4_3q9?usp=sharing