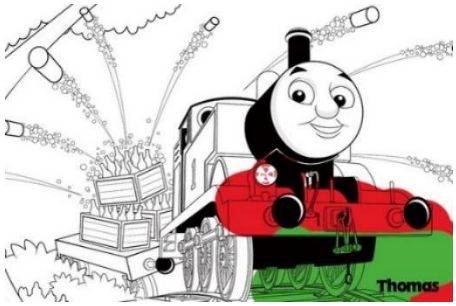
I-SUNS: Zadanie č.3

Konvolučne neurónové siete a prenos vedomostí – JupyterLab Python

Úloha: Spoznajte, analyzujte a pripravte dáta na ďalšie spracovanie

Vykreslené obrázky z každej triedy (na obrázku nie všetky)

Už len na základe vykreslených obrázkov si môžeme všimnúť že rozlíšenia sú rozdielne, ako aj pomery strán. Ak si pozrieme množiny nájdeme tam obrázky, ktoré ale nie moc zapadajú.



train/hot_dog/3222202.jpg

train/pho/1840846.jpg

train/cup_cakes/ 833854.jpg

Dávid Gavenda 98533

Honorable mentions:

/train/caeasr_salad/1303023.jpg

/train/apple_pie/484038.jpg

Bohužiaľ som sa nakoniec nedostal k bonusu, čistenie datasetu.

```
Number of files in apple_pie = 900
Number of files in baby_back_ribs = 900
Number of files in caesar_salad = 900
Number of files in caprese_salad = 900
Number of files in chicken_quesadilla = 900
Number of files in chicken_wings = 900
Number of files in chocolate_cake = 900
Number of files in cup_cakes = 900
Number of files in donuts = 900
Number of files in dumplings = 900
Number of files in french_fries = 900
Number of files in garlic_bread = 900
Number of files in grilled_salmon = 900
Number of files in guacamole = 900
Number of files in hamburger = 900
Number of files in hot_dog = 900
Number of files in ice_cream = 900
Number of files in lasagna = 900
Number of files in macaroni_and_cheese = 900
Number of files in macarons = 900
Number of files in onion_rings = 900
Number of files in oysters = 900
Number of files in pancakes = 900
Number of files in pho = 900
Number of files in pizza = 900
Number of files in red_velvet_cake = 900
Number of files in risotto = 900
Number of files in sashimi = 900
Number of files in spaghetti_bolognese = 900
Number of files in waffles = 900
```

Vypísaním počtu súborov v každom priečinku zistíme, že obrázky sú rozložené rovnomerne.

```
train = ImageDataGenerator(rescale = 1/255, validation_split=0.3)
train_generator = train.flow_from_directory(r'./train/', target_size = (128,128),color_mode="rgb",
                                            batch_size = BS, class_mode = 'categorical',
                                            subset='training', shuffle = True, seed = 42)
valid_generator = train.flow_from_directory(r'./train/', target_size = (128,128),color_mode="rgb",
                                            batch_size = BS, class_mode = 'categorical',
                                            subset='validation', shuffle = True, seed = 42)
test = ImageDataGenerator(rescale = 1/255)
test_generator = test.flow_from_directory('test/', target_size = (128,128),color_mode="rgb",
                                          batch_size = 1, class_mode = 'categorical',shuffle = False)
```

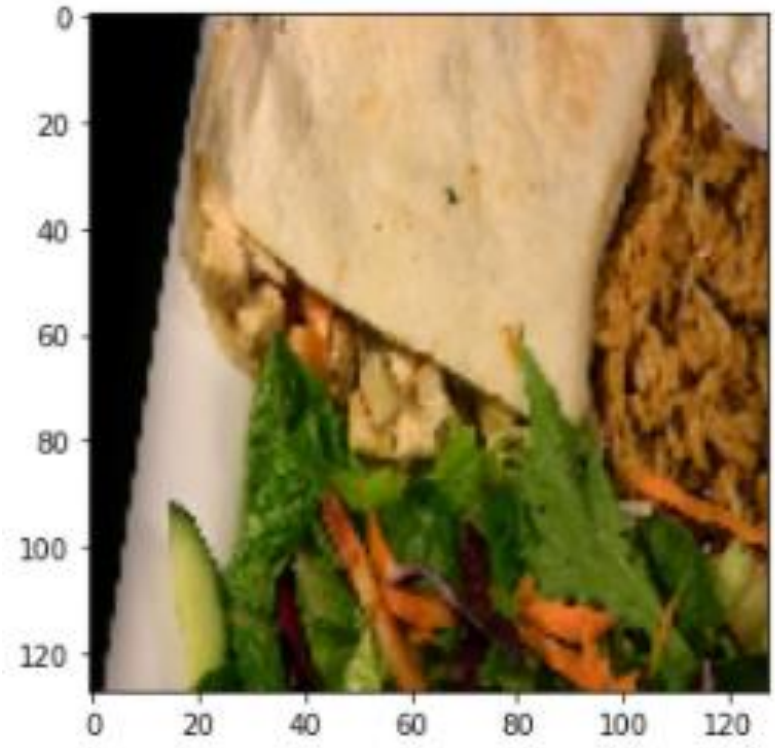
Generátory sú vytvorené pre trénovacie (70%), validačné (30%) aj testovacie množiny. Načítané sú v RGB farebnom móde, veľkosť batchu je v tomto prípade 10. Obrázky sú zmenšené na 128x128 pixelov. Normalizované sú predelením 255, keďže RGB je v rozsahu 0 až 255 (vrátane).

```
Found 18900 images belonging to 30 classes.
Found 8100 images belonging to 30 classes.
Found 3000 images belonging to 30 classes.
```

Trénovacie
Validačné
Testovacie

Dávid Gavenda 98533

Opätovným vykreslením náhodných obrázkov zistíme, že sú správne farby a taktiež rozlíšenie. Dáta sú taktiež už normalizované.



Bližší pohľad na jeden z obrázkov

Úloha: Natrénujte konvolučnú neurónovú sieť na riešenie tohoto problému

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 32)      2432
_________________________________________________________________
activation (Activation)      (None, 128, 128, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 126, 126, 64)      18496
_________________________________________________________________
activation_1 (Activation)    (None, 126, 126, 64)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 63, 63, 128)       73856
_________________________________________________________________
dense (Dense)                (None, 63, 63, 32)        4128
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 63, 63, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 127008)            0
_________________________________________________________________
dense_1 (Dense)              (None, 1028)              130565252
_________________________________________________________________
activation_2 (Activation)    (None, 1028)              0
_________________________________________________________________
dropout (Dropout)            (None, 1028)              0
_________________________________________________________________
dense_2 (Dense)              (None, 30)                30870
=================================================================
Total params: 130,695,034
Trainable params: 130,695,034
Non-trainable params: 0
```

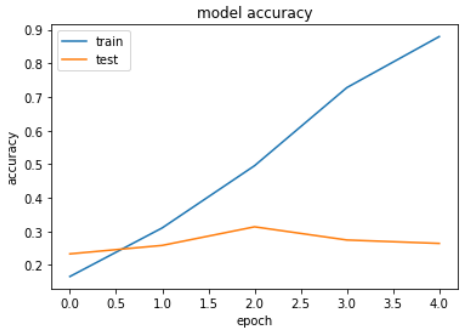
Zhrnutie modelu

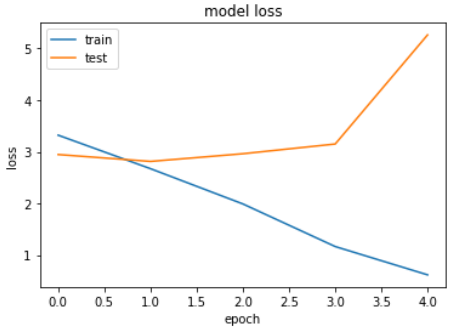
```
Epoch 1/50
1890/1890 [==============================] - 113s 56ms/step - loss: 3.3218 - accuracy: 0.1658 - val_loss: 2.9469 - val_accuracy: 0.2332
Epoch 2/50
1890/1890 [==============================] - 102s 54ms/step - loss: 2.6727 - accuracy: 0.3105 - val_loss: 2.8137 - val_accuracy: 0.2584
Epoch 3/50
1890/1890 [==============================] - 102s 54ms/step - loss: 1.9902 - accuracy: 0.4956 - val_loss: 2.9625 - val_accuracy: 0.3137
Epoch 4/50
1890/1890 [==============================] - 105s 55ms/step - loss: 1.1648 - accuracy: 0.7279 - val_loss: 3.1509 - val_accuracy: 0.2744
Epoch 5/50
1890/1890 [==============================] - 102s 54ms/step - loss: 0.6166 - accuracy: 0.8797 - val_loss: 5.2645 - val_accuracy: 0.2643
```

```python
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience=2, monitor='val_accuracy'),
    tf.keras.callbacks.ModelCheckpoint(filepath='model.{epoch:02d}-{val_accuracy:.2f}.h5',
                                       save_weights_only=True),
    tf.keras.callbacks.TensorBoard(log_dir='./logs'),
]
```

Callbacks sa využívali na predčasné zastavenie trénovania, uloženie **checkpointov** a taktiež logov.

Dávid Gavenda 98533



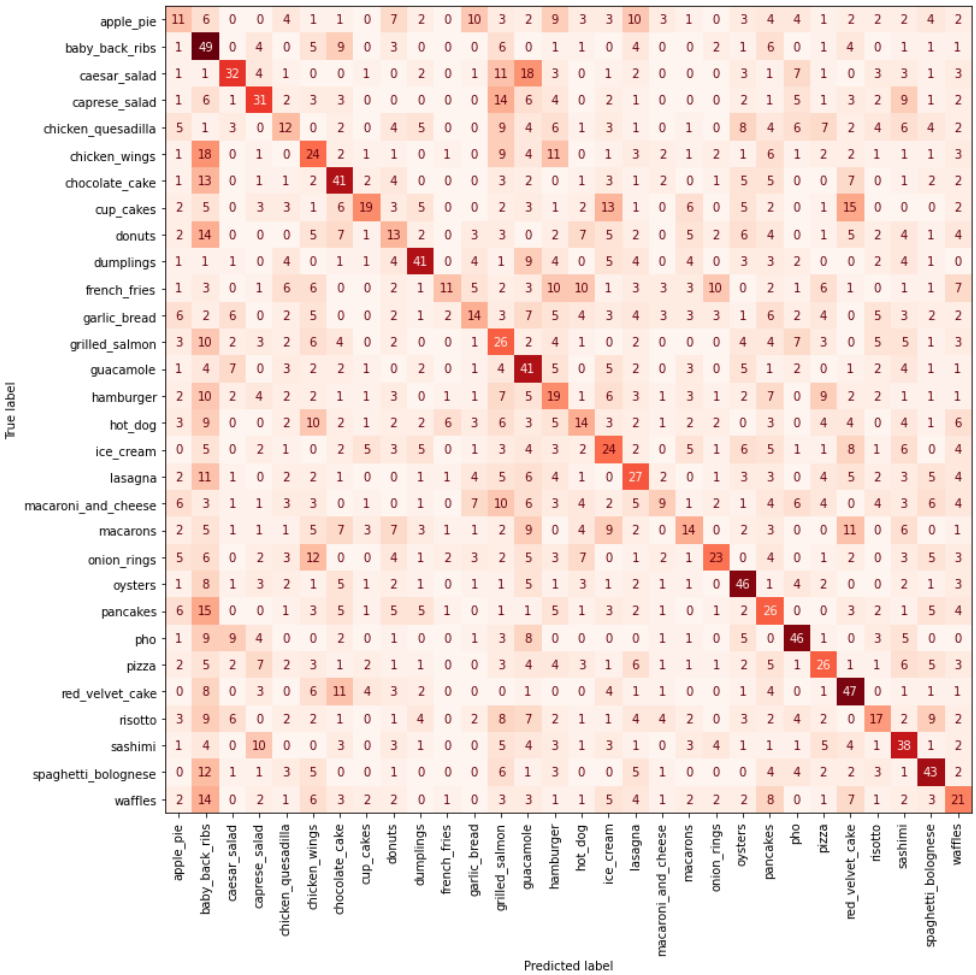


Tento model je jasne pretrénovaný, keďže presnosť na testovacej množine stúpala, avšak presnosť na validačnej začala klesať. Maximálna presnosť na validačnej dosiahla 31.37%, zatiaľ čo presnosť na testovacej bola 49.56%. Hodnota postupne klesala a pomocou *early stoppingu* sa zastavila po tom ako dve epochy presnosť na validačnej množine nestúpla.

Presnosť poslednej bola 26.43%, táto sa uložila a testovala taktiež na testovacej množine

```
Test Accuracy: 26.833
```

Výsledky sú lepšie ako náhoda, dajú sa ale dosiahnuť aj omnoho lepšie. Najlepšie sieť fungovala na baby_back_ribs a red_velvet_cake. Naopak najhoršie na lasagna a apple_pie/french_fires.



Dávid Gavenda 98533

Úloha: Sledujte vplyv regularizátorov na neurónovú sieť

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 64, 64, 32)	2432
activation_6 (Activation)	(None, 64, 64, 32)	0
conv2d_7 (Conv2D)	(None, 62, 62, 64)	18496
activation_7 (Activation)	(None, 62, 62, 64)	0
conv2d_8 (Conv2D)	(None, 62, 62, 128)	73856
dense_4 (Dense)	(None, 62, 62, 64)	8256
activation_8 (Activation)	(None, 62, 62, 64)	0
flatten_2 (Flatten)	(None, 246016)	0
dense_5 (Dense)	(None, 30)	7380510

Total params: 7,483,550
Trainable params: 7,483,550
Non-trainable params: 0

Sieť je zvolená iná (jednoduchá) , aby to netrvalo celé rok...

Batch_size = 50

Optimizer = Adam (learning_rate=2e-5)

Počet epoch bol zvolený pre všetky rovnako, 15, aby bolo dosiahnuté pretrénovanie.

Nepoužil som dropout, pre zvýraznenie výsledkov.

Použite boli kernel regularizátory.

Komentár:

Všimneme si, že bez použitia regularizátorov dochádza k silnému pretrénovaniu (rozdiel je 61 %). V prípade že použijeme silný regularizátor L1, naša presnosť klesne na náhodu (100/33). Keď však použijeme stále príliš vysoký, ovplyvní to negatívne nie len úspešnosť na trénovanej množine ale rovnako aj testovacej. V prípade že naopak použijeme príliš nízke hodnoty, ich účinok sa stráca a rozdiel presnosti sa blíži hodnote ako keby tam ani neboli. Ak si pozrieme aj ostatné dáta zistíme že čím je nižšia hodnota regularizátorov, tým nižšia je aj loss na oboch na testovanej ako aj validačnej množine. Najlepšie výsledky, zároveň s normálnym rozptylom sú pre L1 = 0.0001 a L2 = 0.1.

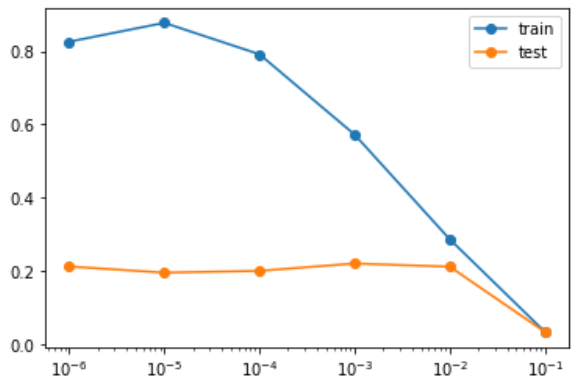
Dávid Gavenda 98533

L1

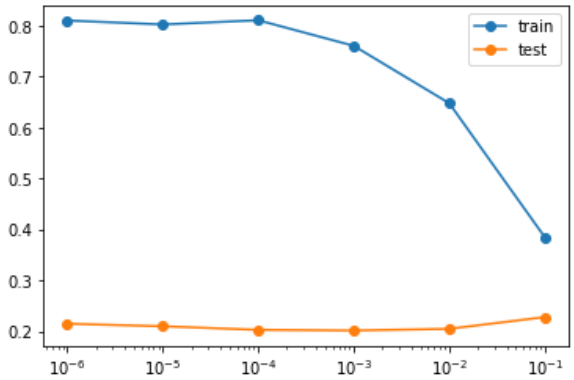
L2	BEZ	0.1	0.01	0.001	0.0001	0.00001	0.000001
BEZ	0.814, 0.206	0.033, 0.033	0.286, 0.211	0.572, 0.220	0.791, 0.200	0.877, 0.195	0.826, 0.212
0.1	0.385, 0.228	0.033, 0.033	0.219, 0.179	0.328, 0.216	0.343, 0.235	0.343, 0.229	0.354, 0.223
0.01	0.648, 0.205	0.033, 0.033	0.270, 0.206	0.501, 0.228	0.610, 0.208	0.612, 0.201	0.667, 0.206
0.001	0.761, 0.202	0.033, 0.033	0.274, 0.209	0.589, 0.207	0.752, 0.207	0.763, 0.211	0.753, 0.216
0.0001	0.811, 0.203	0.033, 0.033	0.250, 0.200	0.561, 0.218	0.753, 0.215	0.801, 0.202	0.813, 0.213
0.00001	0.803, 0.210	0.033, 0.033	0.274, 0.200	0.620, 0.212	0.753, 0.207	0.812, 0.209	0.870, 0.198
0.000001	0.810, 0.210	0.033, 0.033	0.274, 0.214	0.600, 0.209	0.798, 0.206	0.765, 0.221	0.802, 0.207

Údaje: presnosť na trénovanej, presnosť na testovacej

L1



L2



L1

L2	BEZ	0.1	0.01	0.001	0.0001	0.00001	0.000001
BEZ	0.610	0	0.075	0.352	0.591	0.682	0.614
0.1	0.157	0	0.040	0.112	0.108	0.114	0.131
0.01	0.443	0	0.064	0.273	0.402	0.411	0.461
0.001	0.559	0	0.065	0.382	0.545	0.552	0.537
0.0001	0.608	0	0.050	0.343	0.538	0.599	0.600
0.00001	0.593	0	0.074	0.408	0.546	0.603	0.672
0.000001	0.600	0	0.060	0.391	0.592	0.544	0.595

Kolónky obsahujú rozdiel medzi presnosťou na trénovanej a testovacej množine.

Príklad na loss:

L1 = 0.1	**L2=0.1**	EPOCHA 1 – loss = 360.9218	EPOCHA 15 – loss = 15.1496
L1 = 0.000001	**L2=0.000001**	EPOCHA 1 – loss = 3.1705	EPOCHA 15 – loss = 1.0586

Dávid Gavenda 98533

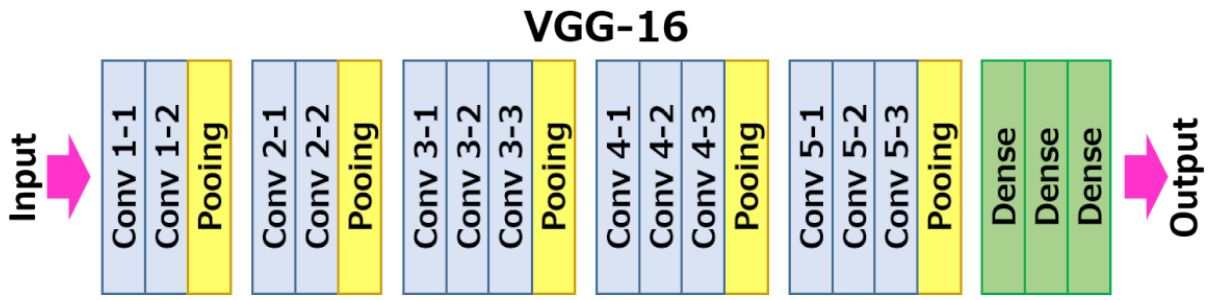
L2 \ L1	BEZ	0.1	0.01	0.001	0.0001	0.00001	0.000001
BEZ	5	3	14	9	6	5	6
0.1	15	3	14	14	14	11	12
0.01	6	2	12	11	7	8	7
0.001	8	2	15	6	8	5	6
0.0001	4	2	15	9	10	5	4
0.00001	5	2	15	9	6	5	5
0.000001	5	3	15	7	5	5	5

Údaje: Epocha, v ktorej bola dosiahnutá najvyššia presnosť na validačnej množine

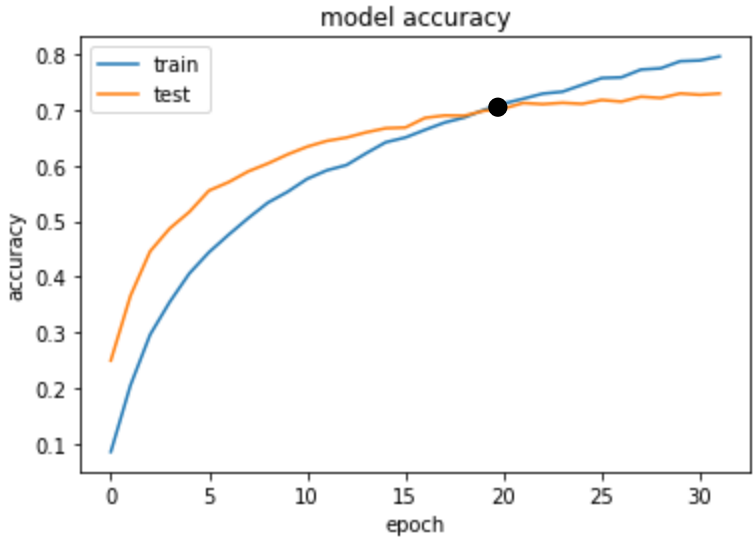
Úloha: Nájdite predtrénovanú sieť (napr. na datasete Imagenet) a prepoužite ju na riešenie problému

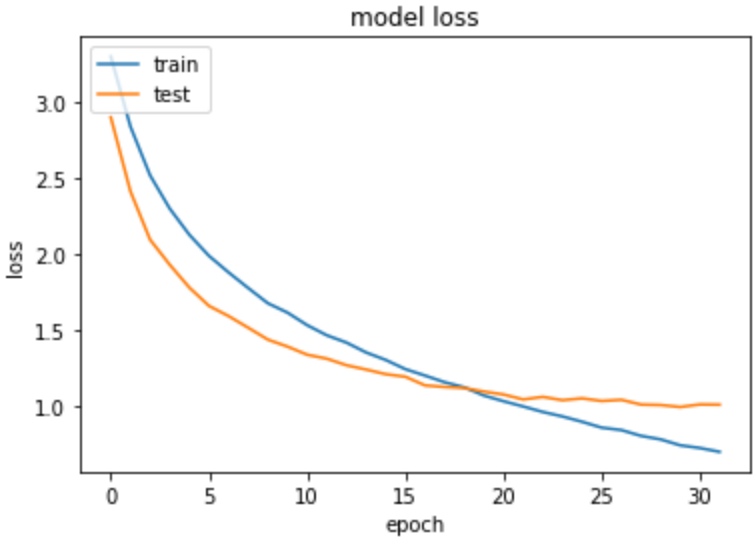
```
baseModel = VGG16(weights="imagenet", include_top=False,input_tensor=Input(shape=(128, 128, 3)))
print("[INFO] summary for base model...")
print(baseModel.summary())
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(30, activation="softmax")(headModel)
model = Model(inputs = baseModel.input, outputs = headModel)
model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate=2e-5), metrics = ['accuracy'])
print("[INFO] summary for  model...")
print(model.summary())
```

Využitý bol predtrénovaný model VGG16 Imagenet, ktorý bol len jemne upravený aby sedel na náš prípad.



Dávid Gavenda 98533



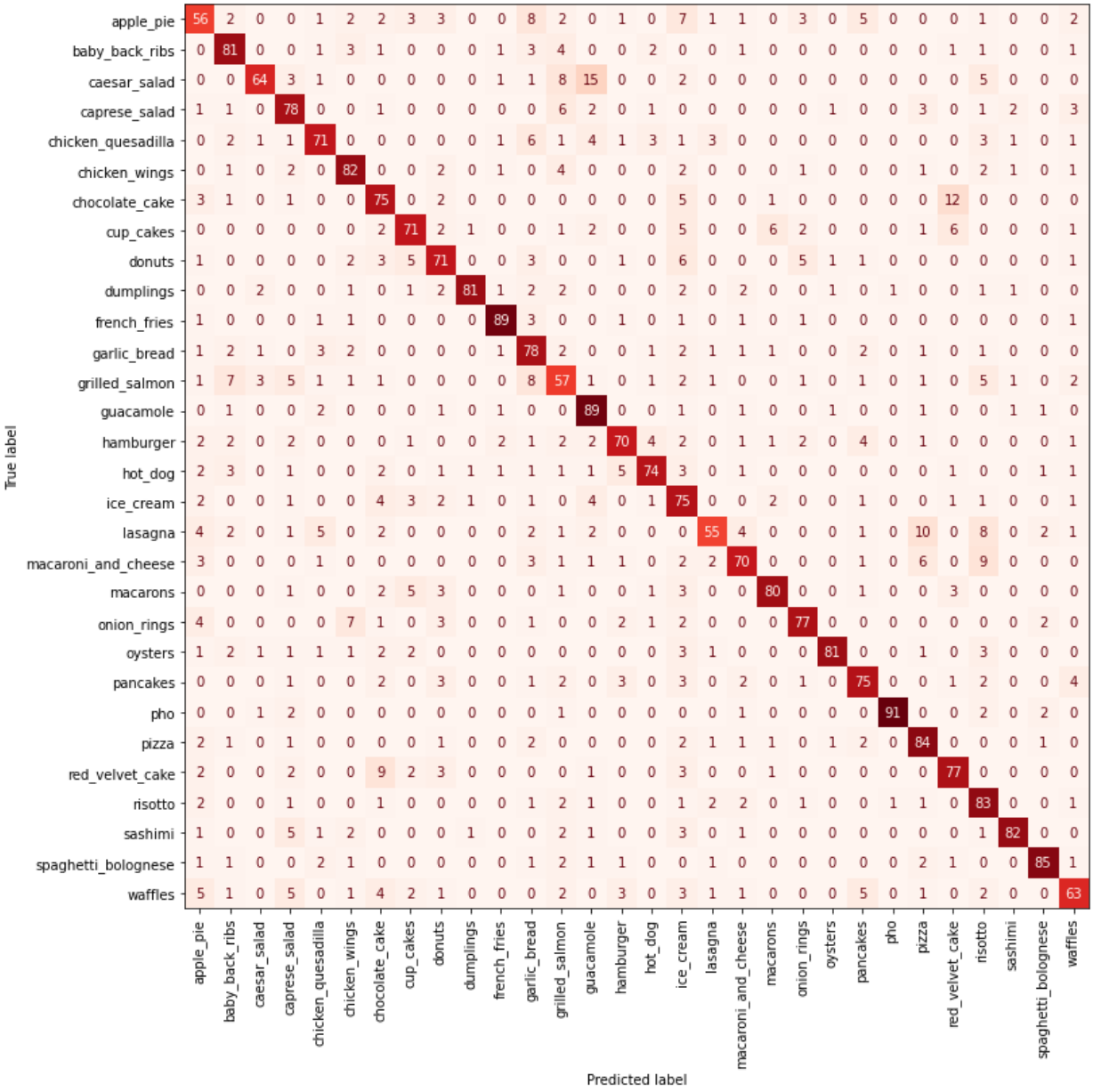



```
Epoch 32/50
1890/1890 [==============================] - 84s 45ms/step - loss: 0.7023 - accuracy: 0.7965 - val_loss: 1.0119 - val_accuracy: 0.7298
```

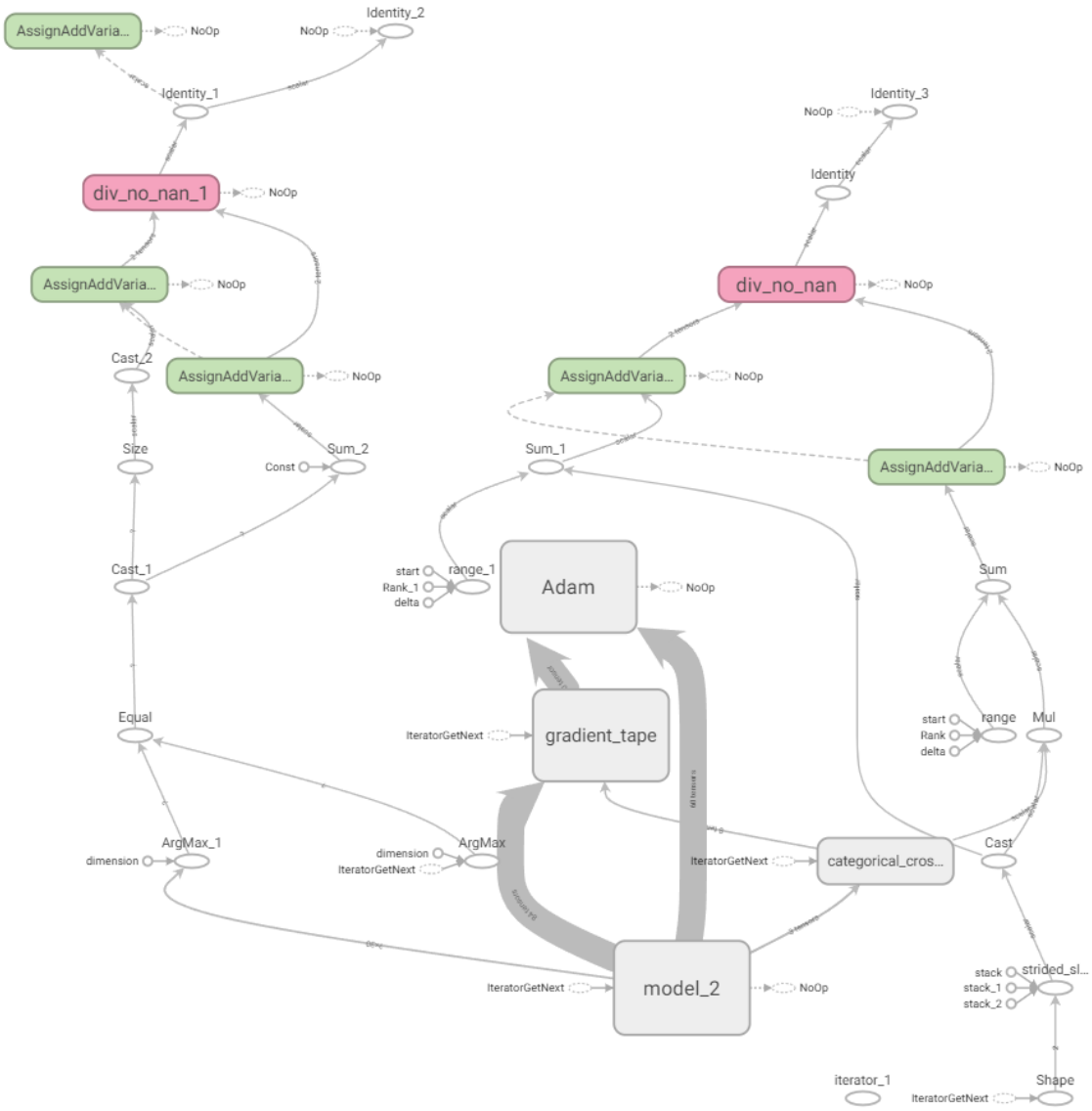
Presnosť na validačnej bola 72.98%

Presnosť na testovacej bola 73.63% (najvyššia sa mi podarila neskôr 75.50%, tá je ukázaná na matici, ako aj v kóde)

Dávid Gavenda 98533



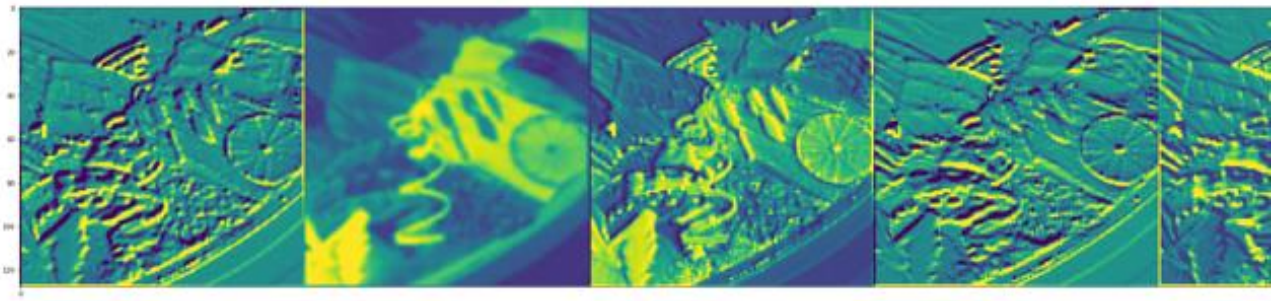
['input_3', 'block1_conv1', 'block1_conv2', 'block1_pool', 'block2_conv1', 'block2_conv2', 'block2_pool', 'block3_conv1', 'block3_conv2', 'block3_conv3', 'block3_pool', 'block4_conv1', 'block4_conv2', 'block4_conv3', 'block4_pool', 'block5_conv1', 'block5_conv2', 'block5_conv3', 'block5_pool', 'average_pooling2d_2', 'flatten', 'dense_4', 'dropout_2', 'dense_5']



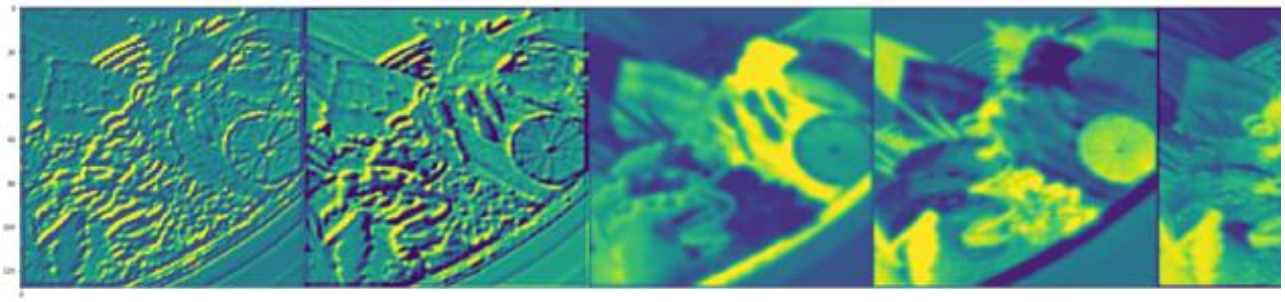
Vizualizovaný model pomocou TensorBoard

Naša predtrénovaná sieť mala omnoho lepšie výsledky ako ktorákoľvek nami vytvorená, skoro trojnásobne. Bola avšak aj zložitejšia ale vďaka predtrénovanosti nebežala o moc dlhšie. Samozrejme oba modely by sa stále dali vylepšiť a po dôkladnom rozšírení nášho modelu by aj ten mohol dosiahnuť výsledky ako predtrénovaný, trvalo by to len omnoho dlhšie. Nie len vytváranie modelu ale aj samotné trénovanie.

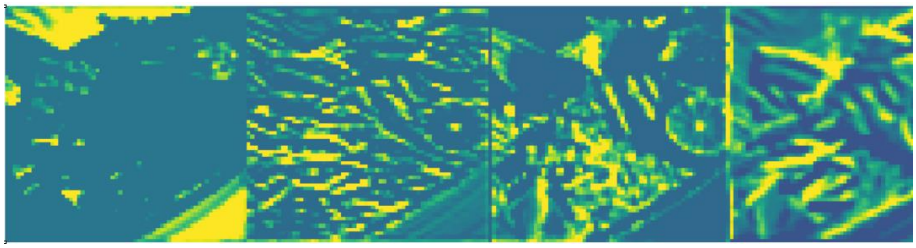
Originálny obrázok sashimi



1. vrstva

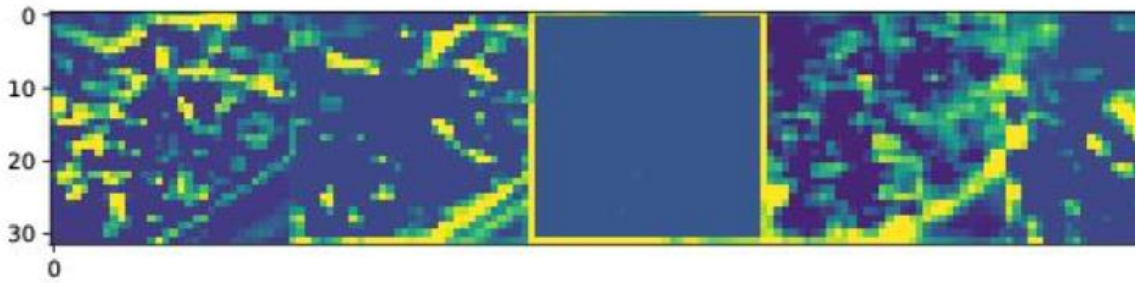


1. vrstva



4. vrstva

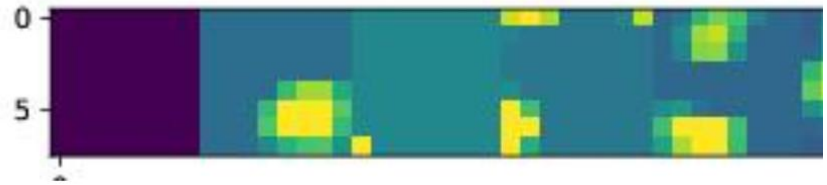
Dávid Gavenda 98533



7. vrstva

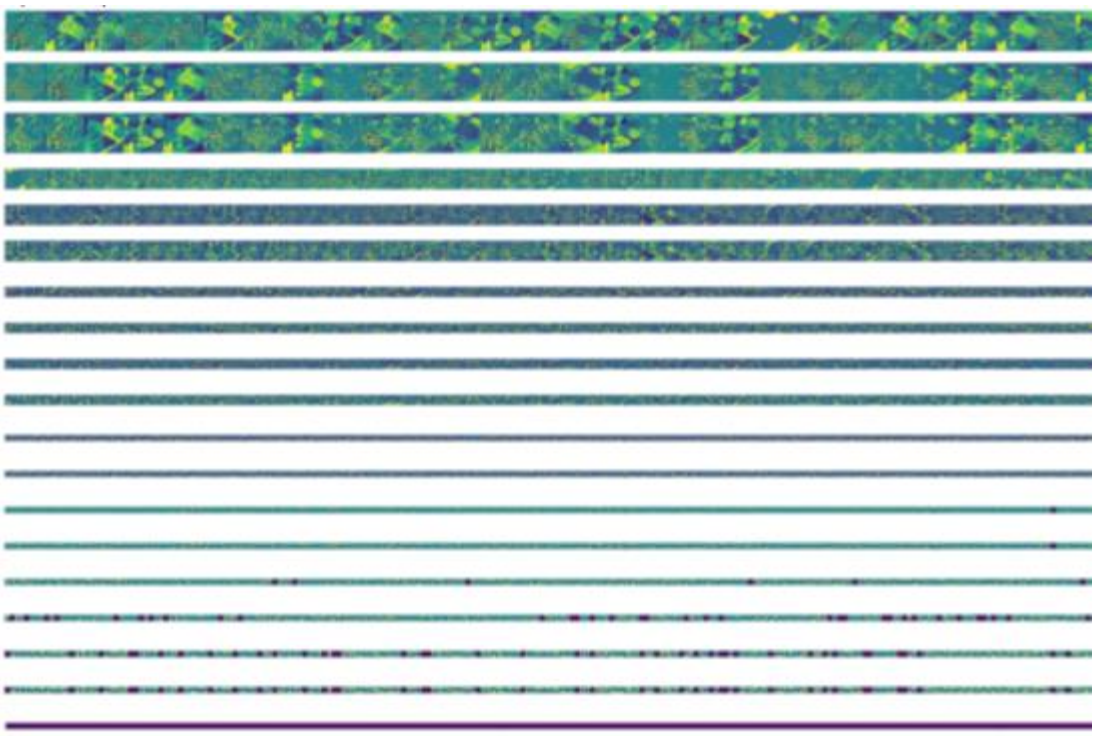
10. vrstva

15. vrstva



16. vrstva

19. vrstva



Na obrázku, môžeme vidieť všetky vrstvy vizualizované. Detailné v plnej veľkosti sa dajú vygenerovať v programe do priečinka visualize, kde majú aj od 8 po 70 mega každý.

Prvý pár vrstiev sa snaží zachytiť čokoľvek z obrázka, čím ďalej tým viac sa filtre koncentrujú viac na určité vlastnosti. Čím hlbšie ideme do siete, tým viac sa špecializuje na viac abstraktné črty.

Dávid Gavenda 98533

Predpovede na pár testovacích obrázkov pomocou VGG16 Imagenet. Môžeme si všimnúť že aspoň s guacamole sa trafil a navyše si je aj istý.

Dávid Gavenda 98533

BONUS

Augmentácie

Originálny obrázok - train/ice_cream/1164.jpg

Augmentované obrázky

Dávid Gavenda 98533

```
train = ImageDataGenerator(rescale = 1/255,    #  zmenšenie obrázkov
                validation_split=0.3,          #  rozdelenie na validačnú a trénovaciu
                rotation_range=30,             #  otáčanie až o 30 stupňov
                width_shift_range=0.2,         #  roztiahne obrázok po šírke
                height_shift_range=0.2,        #  roztiahne obrázok po výške
                shear_range=0.2,               #  Shear Intensity (Shear angle in counter-
                                               clockwisedirection in degrees)
                zoom_range=0.2,                #  priblíži obrázok
                horizontal_flip = True,        #  náhodne otočí obrázok horizontálne
                fill_mode='nearest'            #  mód na vyplnenia prázdneho miesta po otočení a tak
               )
```

Samozrejme sa dali použiť aj iné, ale myslím že toto postačuje.

```
Epoch 15/15
378/378 [==============================] - 77s 203ms/step - loss: 2.4331 - accuracy: 0.3117 - val_loss: 2.4593 - val_accuracy: 0.3011
Train: 0.323, Test: 0.306
```

Náhodné vykreslenie 20 obrázkov z augmentovaného batchu

Dávid Gavenda 98533