Slovenská technická univerzita v Bratislave
Fakulta elektrotechniky a informatiky

Neurónové siete

Zadanie č.3

2021 **Jakub Jančík**

Obsah

Zoznam obrázkov

1 Analýza, príprava a spracovanie

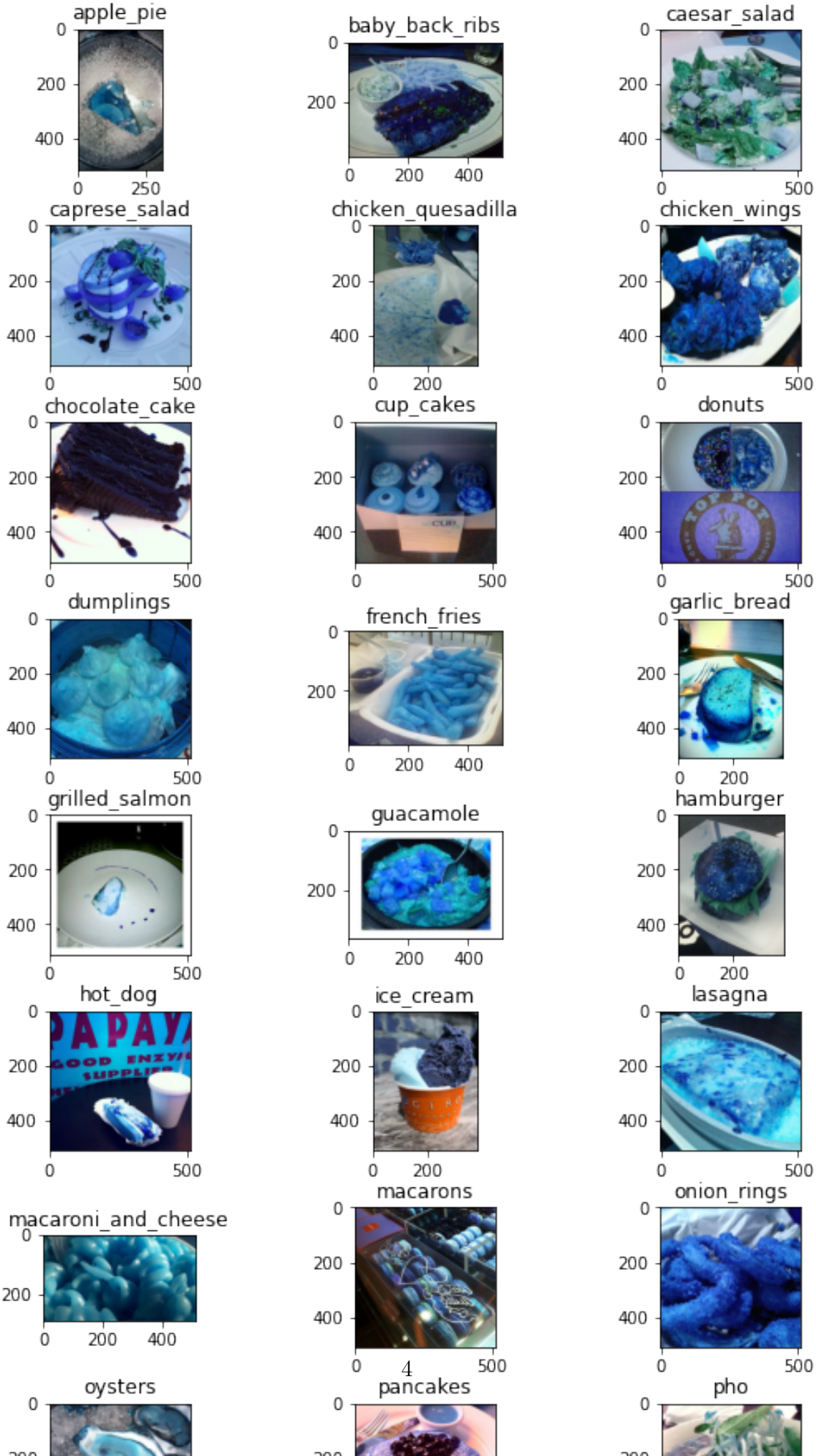
Naším cieľom bolo implementovať program, ktorý bude vedieť klasifikovať druhy jedál.

Dané dáta sme načítali a vytvorili kategórie. Z každej kategórie sme vykreslili jeden obrázok.

```
imgPrint = []
ctgry= []
fig = plt.figure(figsize=(8, 8))
columns = 3
rows = 10
#Lets view some of the pics
def show_imgs():
    for i in range(1, columns*rows + 1):
        image = imgPrint[i-1]
        fig.add_subplot(rows, columns, i)
        plt.imshow(image)
        plt.title(ctgry[i-1])

    # set the spacing between subplots
    plt.subplots_adjust(left=0.1,
                        bottom=1,
                        right=0.9,
                        top=3,
                        wspace=1,
                        hspace=0.4)
    plt.show()

#VYKRESLI 1 jedlo per kategoria
for category in categories:
    trainPath = os.path.join(trainDataDir, category)
    num = categories.index(category)
    for img in os.listdir(trainPath):
        images = cv2.imread(os.path.join(trainPath, img))
        # show_imgs(category, images)
        imgPrint.append(images)
        ctgry.append(category)
        break


show_imgs()
```

Listing 1: .

Obr. 1: Vykreslenie jedneho obrazku z kazdej kategorie.



Dáta sme načítali ako z trénovacej tak aj z testovacej množiny. Následne sme vytvorili validačné dáta z trénovacej množiny. Úprava veľkosti obrázkov je aplikovaná pomocou resize() na veľkosť 32x32 a normalizovanie pomocou normalize()

```
#A function to read and process the images to an acceptable format
   for our model
def read_and_process_image(dataPath, num):
    """
    Returns array of resized images
    """
    data = []  # images
    for img in os.listdir(dataPath):
        image = cv2.imread(os.path.join(dataPath, img))
        arr = cv2.resize(image, (SIZE, SIZE))
        arr = cv2.normalize(arr, arr, 0, 255, cv2.NORM_MINMAX)
        data.append([arr, num])

    return data

for category in selectedCategories:
    trainPath = os.path.join(trainDataDir, category)
    testPath = os.path.join(testDataDir, category)
    num = selectedCategories.index(category)
    train = train + read_and_process_image(trainPath, num)
    test = test + read_and_process_image(testPath, num)

for t, label in train:
    X.append(t)
    y.append(label)
X = np.array(X).reshape(-1, SIZE, SIZE, 3)
y = np.array(y).reshape(-1, 1)
X.astype(float) / 255.0
y.astype(float)
```

Listing 2: .

Naše dáta si rozdelíme na trénovacie a validačné v pomere 80:20 a následne one-hot kódovanie pomocou numpy súvisiacich nástrojov keras.

```
from sklearn.model_selection import train_test_split
from keras.utils import np_utils

#Now lets split our data into train and test
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=1)
```

```
Y_train = np_utils.to_categorical(y_train, 4)
Y_test = np_utils.to_categorical(y_test, 4)
```

Listing 3: .

Výpis tvarov spracovaných dát.

```
Shape of train images is: (9000, 32, 32, 3)
Shape of labels is: (9000, 1)
```

Listing 4: .

Vybrali sme si 10 tried s ktorými chceme nadalej pracovať.

2 Konvolučná sieť

Budeme používať batch size 64, pričom výhodnejšia je aj vyššia veľkosť dávky 128 alebo 256, všetko závisí od pamäte. Masívne prispieva k určovaniu parametrov učenia a ovplyvňuje presnosť predikcie. Sieť budeme trénovať 10 epoch. Naša sieť sa trénovala veľmi pomaly [2].

```
batch size = 64
epochs = 10
num classes = 4
```

Listing 5: Parametre.

Architektúru sme si zvolili ľubovoľne. Výšku a širku obrázka sme zvolili na 32. Počet farebných kanálov 3 - ak by sme chceli pracovať s bezfarebnými obrázkami tak by sme zvolili počet 1.

V Kerase môžeme vrstvy jednoducho naskladať tak, že jednu po druhej pridáme požadovanú vrstvu. To je presne to, čo tu urobíte: najprv pridáte prvú konvolučnú vrstvu pomocou Conv2D().

Ďalej pridávame vrstvu max-pooling pomocou MaxPooling2D() a tak ďalej. Posledná vrstva je vrstva Dense, ktorá má aktivačnú funkciu softmax s 4 jednotkami, ktorá je potrebná pre tento problém klasifikácie viacerých tried.

```
#Now lets create our model
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

num_classes = np.unique(y_train)
# Determine shape of the data
img_width, img_height, img_num_channels = 32, 32, 3
input_shape = (img_width, img_height, img_num_channels)

# building a linear stack of layers with the sequential model
model = Sequential()

# convolutional layer
model.add(layers.Conv2D(50, kernel_size=(3,3), strides=(1,1), padding
   ='same', activation='relu', input_shape=(32, 32, 3)))

# convolutional layer
model.add(layers.Conv2D(75, kernel_size=(3, 3), strides=(
    1, 1), padding='same', activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Dropout(0.25))

model.add(layers.Conv2D(125, kernel_size=(3,3), strides=(1,1),
   padding='same', activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))
model.add(layers.Dropout(0.25))

# flatten output of conv
model.add(layers.Flatten())

# hidden layer
model.add(layers.Dense(500, activation='relu'))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(250, activation='relu'))
model.add(layers.Dropout(0.3))
# output layer
model.add(layers.Dense(4, activation='softmax'))
```

Listing 6: Model konvolučnej siete.

Po vytvorení modelu ho skompilujeme pomocou optimalizátora Adam, jedného z najpopulárnejších optimalizačných algoritmov. Okrem toho sme určili loss type - categorical crossentropy, ktorá sa používa na klasifikáciu viacerých tried. Nakoniec špecifikujete metriky ako presnosť - accuracy, ktorú chceme analyzovať počas tréningu modelu.

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam
loss_function = categorical_crossentropy
optimizer = Adam()

# Compile the model
model.compile(loss=loss_function,
              optimizer=optimizer,
              metrics=['accuracy'])
```

Listing 7: Compile convolutional neural network.

Trénovanie modelu pomocou Kerasovej funkcie fit(). Model trénuje 10 epoch. Funkcia fit() vráti objekt histórie.

```
history = model.fit(X_train, Y_train,
            batch_size=batch_size,
            epochs=10,
            verbose=1,
            validation_data=(X_test, Y_test))
```

Listing 8: Trénovanie modelu.

Nakoniec nám vyšli hodnoty po poslednej 10 epoche - loss: 1.0166 - accuracy: 0.5573 - val loss: 1.0888 - val accuracy: 0.5225.

Skúšali sme vyššie epochy ako napríklad 30. Ale to nám trvalo veľmi dlho z časového hľadiska. Pri 30 epoche sme dosahovali výsledky ako napríklad accuracy: 0.8128.

Najprv však zhodnoťme výkon vášho modelu na testovacej súprave, kým dospejete k záveru.

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')

Test loss: 1.0888020992279053 / Test accuracy: 0.5224999785423279
```

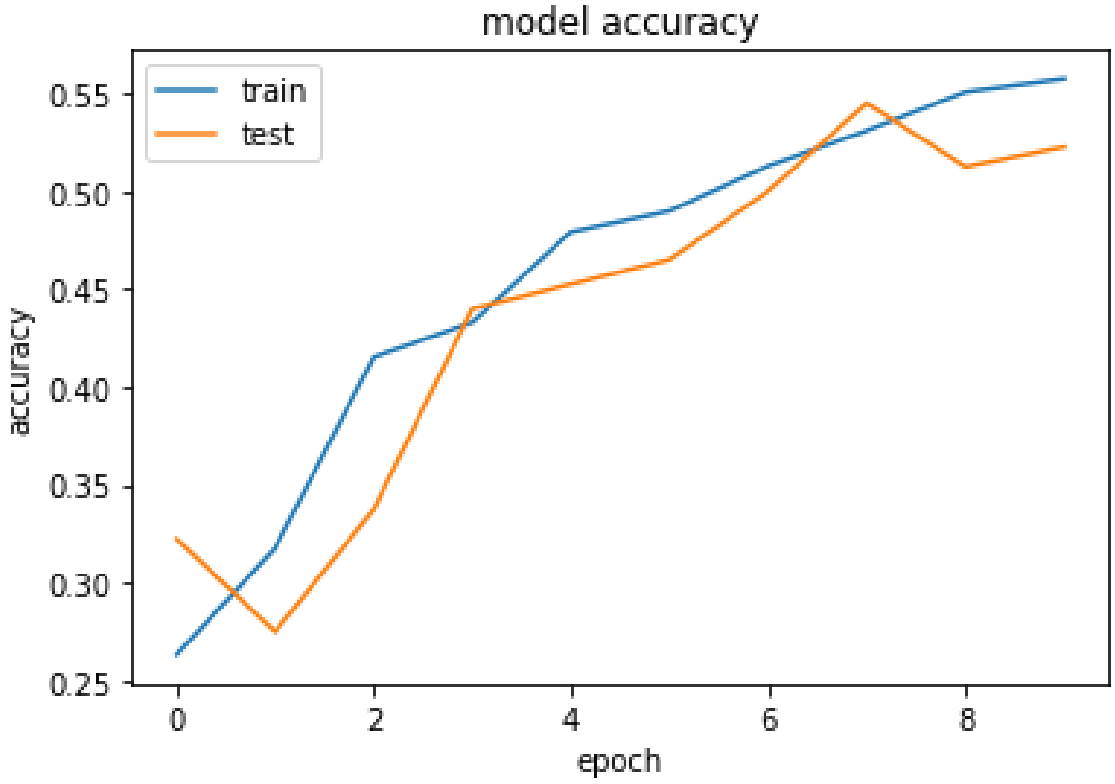
Listing 9: Evaluate model.

A následne ideme vykresliť náš natrénovaný model, ktorý sme vytvorili pomocou:

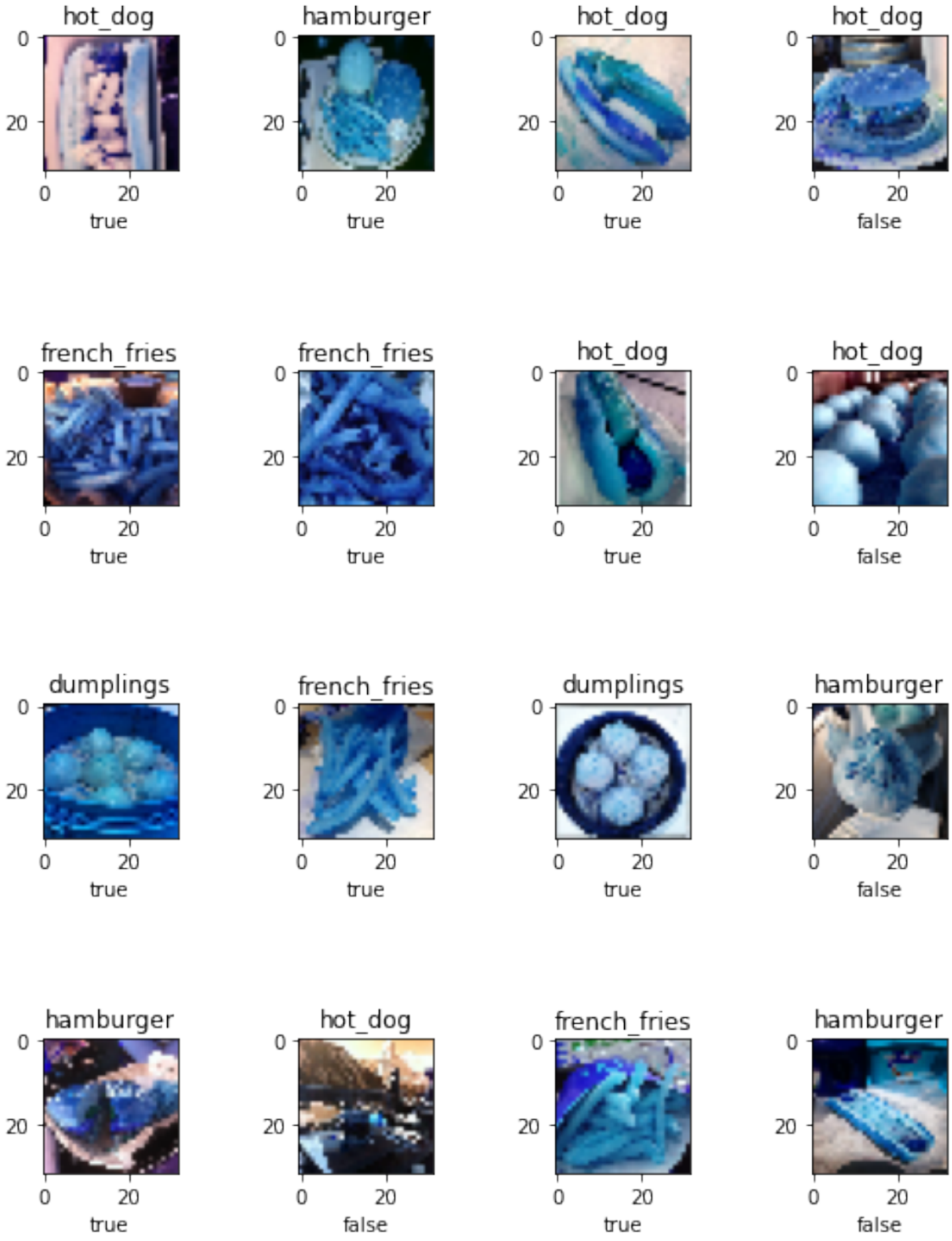
```
ypred = model.predict(X_test)

#lets plot the train and val curve
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.imshow(X_train[0])
plt.show()
plt.imshow(ypred[0])
plt.show()
```

Listing 10: Plot model.

Obr. 2: Plot model.



Obr. 3: Vykreslenie trénovacich dát, kde label značí čo sa predpovedalo.



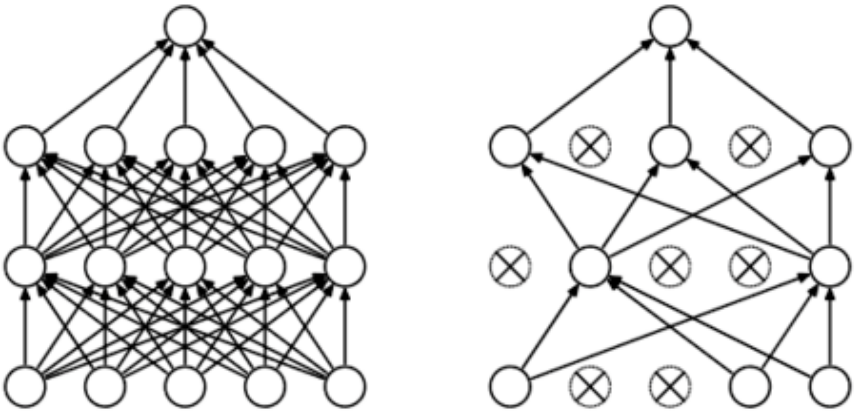
3 Regularizacia

Regularizácia optimalizuje model penalizáciou zložitých modelov, čím sa minimalizuje strata a zložitosť. To núti našu neurónovú sieť, aby bola jednoduchšia. Tu použijeme

regularizátor L2, pretože je najbežnejší a je stabilnejší ako regularizátor L1. Tu pridáme regularizátor do druhej a tretej vrstvy našej siete s rýchlosťou učenia (lr) 0,01 [1].

Pre pretrénovanie sme tam pridali aj Dropout - regulácia výpadkov ignoruje náhodnú podmnožinu jednotiek vo vrstve, pričom ich váhy nastavuje na nulu počas tejto fázy tréningu. Ideálna miera pre vstupnú a skrytú vrstvu je 0,4 a ideálna miera pre výstupnú vrstvu je 0,3.

Obr. 4: Dropout regularizacia pred a po.



(a) Standard Neural Net (b) After applying dropout.

Weight Constraints - Obmedzenie hmotnosti kontroluje veľkosť sieťových váh a mení ich mierku, ak veľkosť presahuje vopred definovaný limit. Obmedzenie hmotnosti funguje podľa potreby. Nižšie používame obmedzenie unit norm, ktoré núti váhy mať veľkosť 1,0.

```
kernel_constraint=tf.keras.constraints.unit_norm()
```

```
from keras.regularizers import l2

model2 = Sequential()

# convolutional layer
model2.add(layers.Conv2D(50, kernel_size=(3, 3),kernel_constraint=tf.
    keras.constraints.unit_norm()
, kernel_regularizer=l2(0.01), strides=(1, 1),
                         padding='same', activation='relu',
    input_shape=(32, 32, 3)))

# convolutional layer
```

```
model2.add(layers.Conv2D(75, kernel_size=(3, 3), kernel_regularizer=
    l2(0.01), strides=(
     1, 1), padding='same', activation='relu'))
model2.add(layers.MaxPool2D(pool_size=(2, 2)))
model2.add(layers.Dropout(0.4))

model2.add(layers.Conv2D(125, kernel_size=(3, 3), kernel_regularizer=
    l2(0.01), strides=(
     1, 1), padding='same', activation='relu'))
model2.add(layers.MaxPool2D(pool_size=(2, 2)))
model2.add(layers.Dropout(0.4))

# flatten output of conv
model2.add(layers.Flatten())

# hidden layer
model2.add(layers.Dense(500, activation='relu'))
model2.add(layers.Dropout(0.4))
model2.add(layers.Dense(250, activation='relu'))
model2.add(layers.Dropout(0.2))
# output layer
model2.add(layers.Dense(4, activation='softmax'))
```

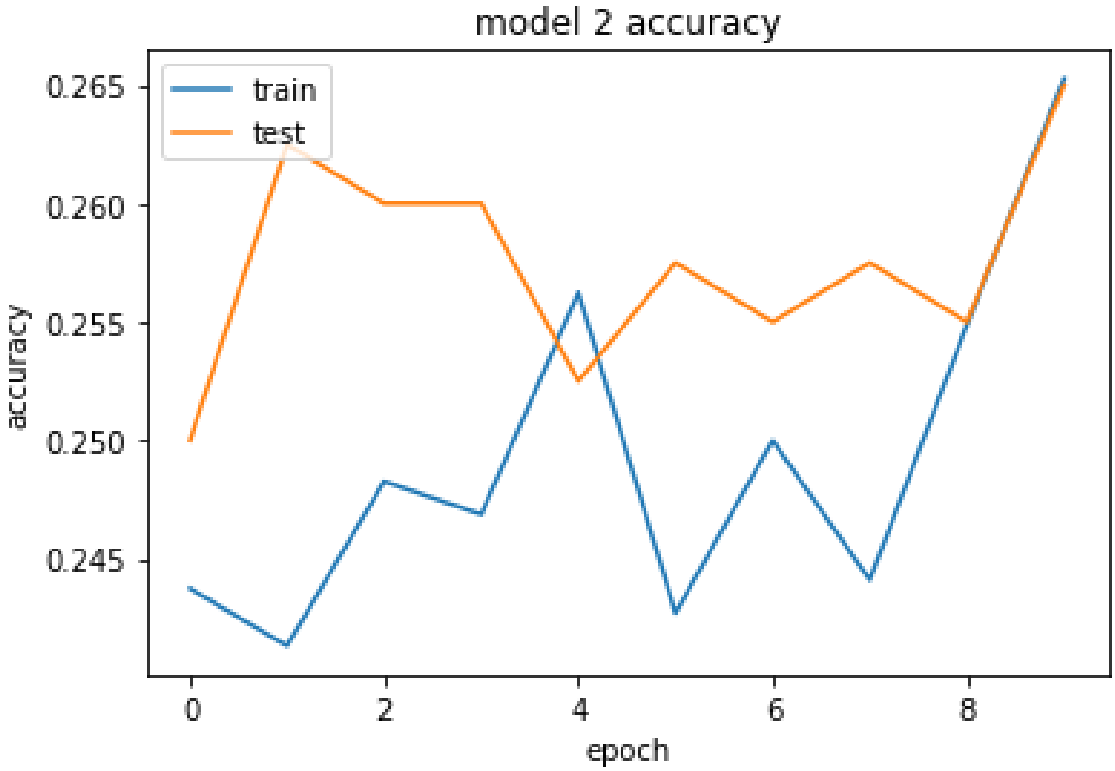
Listing 11: Model pre vyriesenie pretrénovania.

3.1 Výsledky

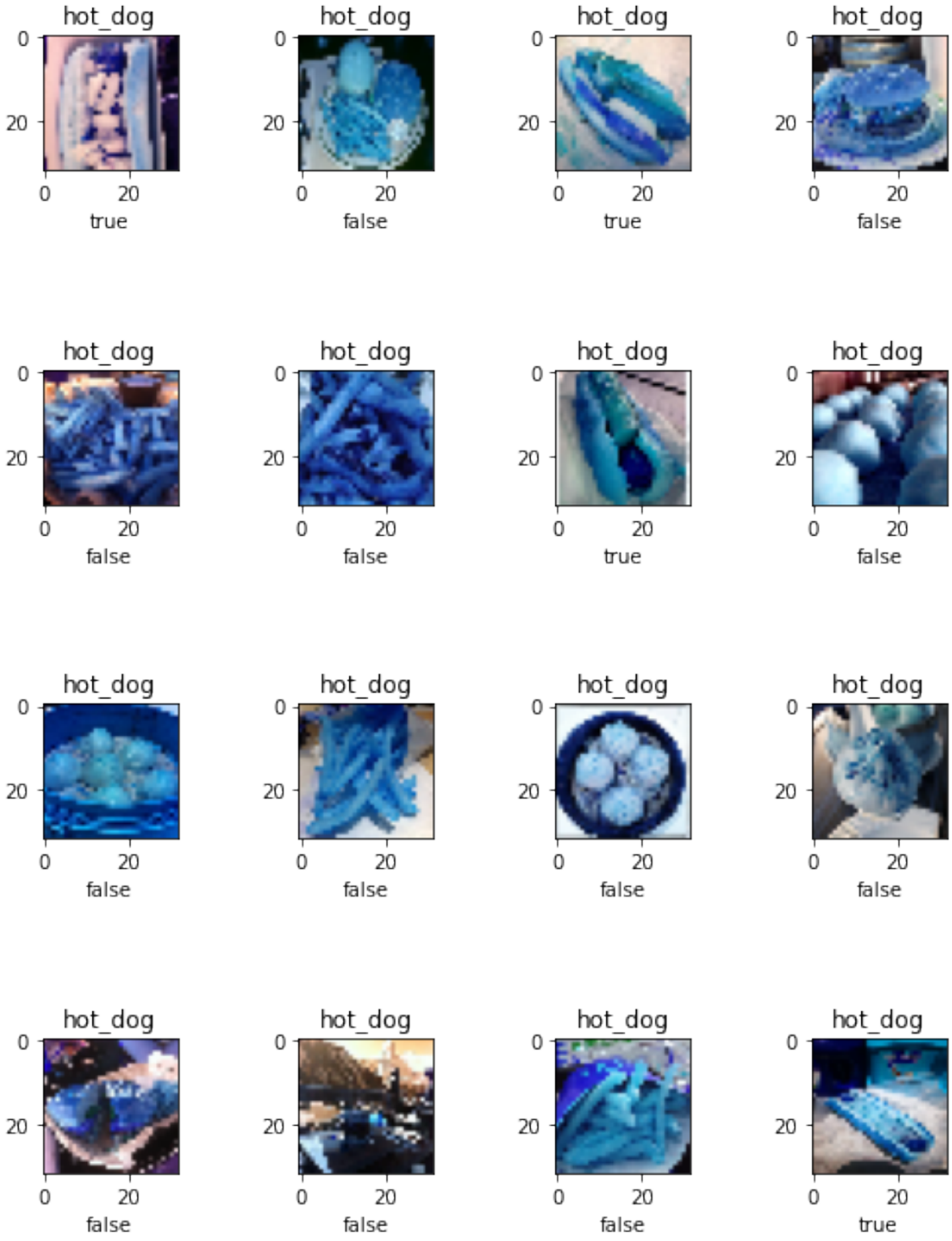
Model ktorý sme chceli vylepšiť tým, že mu nastavíme rôzne parametre aby sme predišli pretrénovaniu sa nám nepodarilo. Pre dobré riešenie by bolo dobre nastaviť idealnejšie parametre. Náš výsledok modelu:

```
loss: 6.9481 - accuracy: 0.2653 - val_loss: 6.9370 - val_accuracy:
    0.2650
Test loss: 6.9369587898254395 / Test accuracy: 0.26499998569488525
```

Obr. 5: Model s regularizatormi 0.01 vo vrstvach pre Conv2d.



Obr. 6: Vykreslenie trénovacích dát po regularizácií.



4 PCA

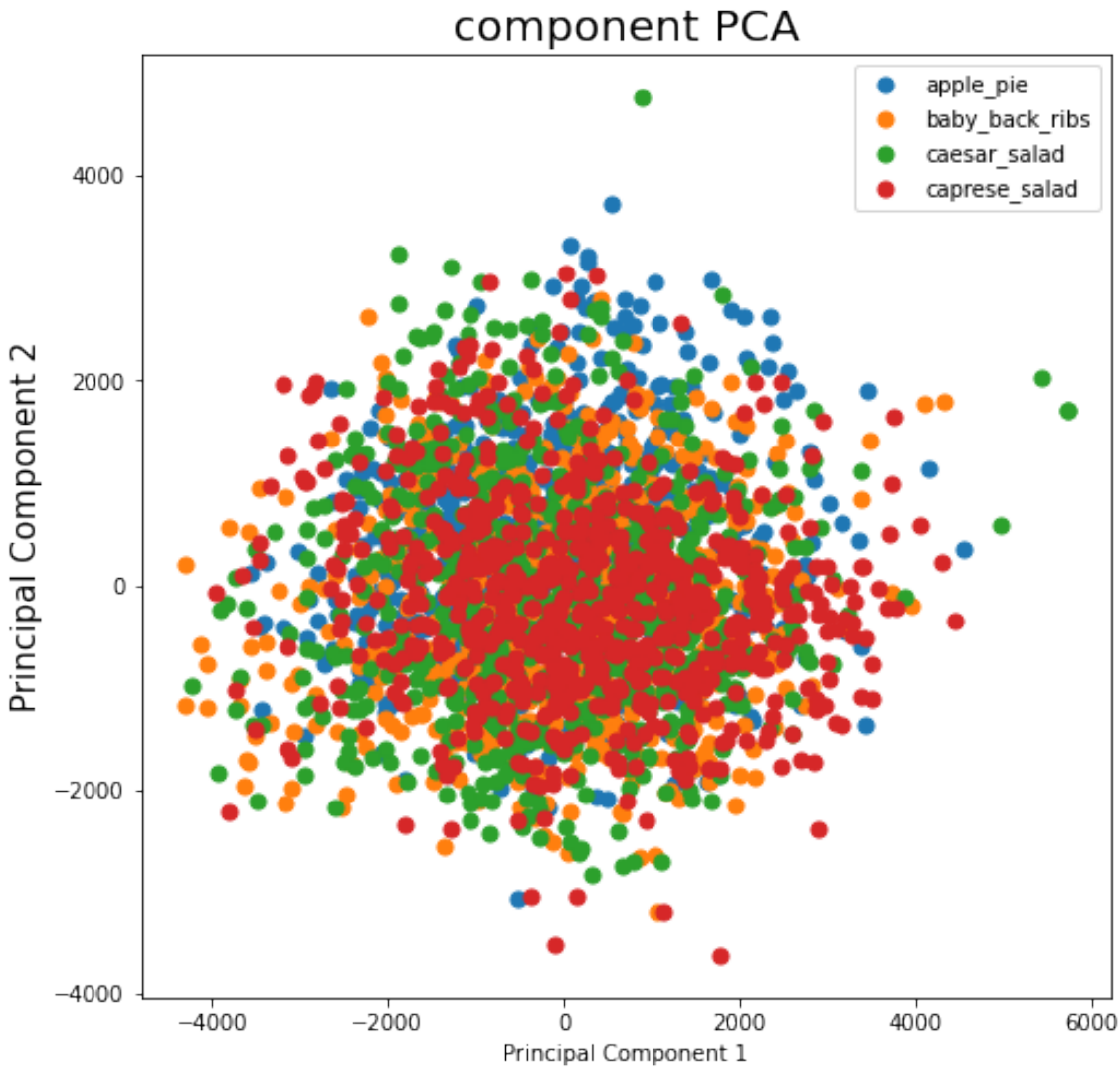
Táto časť zobrazuje iba dvojrozmerné údaje. Pouzili sme metódy na zmenšenie dimenzie pomocou reshape. Targets pre nás predstavujú naše výsledne triedy 0,1,2,3 ktoré

sme si na začiatku zadefinovali.

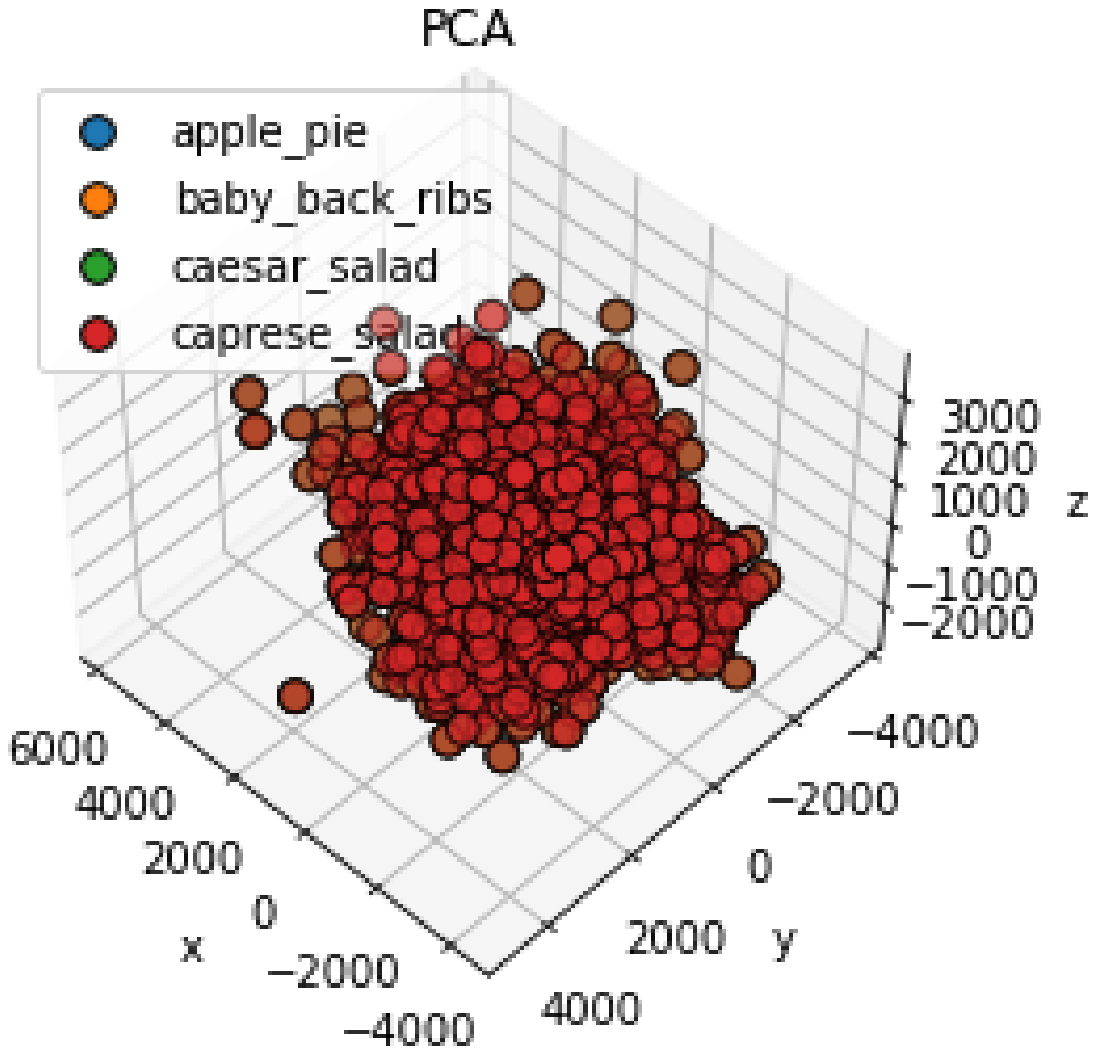
```
X_train = X_train.reshape(len(X_train), np.prod(X_train.shape[1:]))
X_test = X_test.reshape(len(X_test), np.prod(X_test.shape[1:]))

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X_train)
principalDf = pd.DataFrame(data=principalComponents, columns=[
                           'principal component 1', 'principal
   component 2'])
y = pd.DataFrame(data=y_train)
finalDf = pd.concat([principalDf, y], axis=1)

plt.figure(figsize=(8, 8))
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2', fontsize=15)
plt.title('component PCA', fontsize=20)
targets = [0, 1, 2, 3]
for target in targets:
    indicesToKeep = finalDf[0] == target
    plt.scatter(finalDf.loc[indicesToKeep, 'principal component 1'],
                finalDf.loc[indicesToKeep, 'principal component 2'],
   s=50)
plt.legend(categories)
plt.show()
```

Listing 12: Vizualizacia 2D modelu pomocou PCA.

Obr. 7: Vizualizacia 2D modelu PCA.



Obr. 8: Vizualizacia 3D modelu PCA.



Literatúra

[1] Jobs Admin). *Regularization*. https://www.analyticsvidhya.com/blog/2020/09/overfitting-in-cnn-show-to-treat-overfitting-in-convolutional-neural-netwo

[2] Aditya Sharma). *Convolutional Neural Networks in Python with Keras*. 2017,. https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python.