Středoškolská technika 2018

Setkání a prezentace prací středoškolských studentů na ČVUT

Detekce gest ruky pomocí strojového učení

Roman Šíp

SPŠ a VOŠ Písek, Karla Čapka 402, 397 11 Písek

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Písku dne 31. 3. 2018 ……

Roman Šíp

Anotace

Tato práce se zabývá zjednodušením procesu tvorby prediktivních modelů, které dokážou rozpoznat jakékoli gesto ruky, které byly naučeny. Soustředí se na všechny tři fáze tvorby prediktivního modelu a to: sběr a zpracování dat, trénink prediktivního modulu pomocí získaných dat a následné nasazení daného modelu do produkce. Pro tvorbu a učení prediktivního modelu jsou využity algoritmy strojového učení.

Klíčová slova

Strojové učení; počítačové vidění; klasifikace; predikce

Annotation

This work is focus on simplifying the process of making a predictive model, can detect any hand gesture it was trained for. It focuses on all three stages of the process of making a predictive model and these are: data collecting and data processing, training of a predictive model on collect date and deployment of the said model to production. For creating and training of the models are used Machine learning algorithms.

Keywords

Machine learning; Computer vision; classificationprediction

Obsah

1 Úvod

Většina lidí, kteří se učí využívat strojové učení pracuje s veřejně dostupnými daty, která jsou předem upravená. Při tvorbě modelu je porozumění dat stejně důležité, jako porozumění algoritmů strojového učení, a proto jsem chtěl vytvořit projekt, který mi dovolí prozkoumat všechny fáze procesu tvorby takového modelu a jejich úskalí, za účelem zdokonalení se v použití algoritmů strojového učení. Problematiku detekce gest ruky jsem si zvolil z důvodu jednoduchosti tvorby dat a testování modelu.

V této práci se tedy zabývám tvorbou počítačových programů pro tvorbu dat (získávaní obrázků z živého videa z kamery), trénink prediktivního modelu (program který dokáže rozpoznat, které gesto ruky se nachází na obrázku na základě předešlého natrénovaní na podobných obrázcích). Využití natrénovaného modelu v aplikaci, která v reálném čase klasifikuje obraz z kamery. Aplikace pak může dále využít informaci o gestu pro např.: ovládání chytré domácnosti, ovládání počítače, atd..

Tyto programy jsem využil pro tvorbu třech konkretních příkladů detekce gest ruky a to: palec nahoru nebo palec dolů, detekce počtu zdvižených prstů na ruce a rozpoznání gest z hry Kámen, nůžky, papír.

2 POUŽITÉ TECHNOLOGIE

Jako programovací jazyk jsem zvolil Python 3 (verze 3.6.5 a 3.5.3) kvůli jeho široké prezenci jak v oblasti strojového učení, tak i v práci s operačním systémem a web kamerami. Nejedná se o jeden z nejrychlejších jazyků, ale tato nevýhoda je překonána použitím knihoven pro strojové učení, které jsou vytvořeny v rychlejších programovacích jazycích C a C++.

2.1 Knihovny

2.1.1 Numpy

Numpy je vědecká a matematická knihovna pro práci s homogenními číselnými poli.Velikou výhodou Numpy je kompatibilita s ostatními Python knihovnami. V tomto projektu používám Numpy pro načtení hodnot jednotlivých pixelů obrazu do Numpy pole.

2.1.2 OpenCV

OpenCV je knihovna zaměřená na počítačové vidění a zpracování obrazu v reálném čase. Použil jsem OpenCV pro úpravu obrazu z kamery a jeho zobrazení na displeji v reálném čase.

2.1.3 Tensorflow

Knihovna Tensorflow je používaná zejména pro strojové učení. Dokáže vytvořit statický komputační graf a optimalizuje jeho náročnost a umožňuje použití grafické karty pro výpočet, což zrychluje proces trénování modelu na datech až čtyřicetkrát.

2.1.4 Picamera

Picamera je knihovna pro práci s Raspberry Pi kamerovým modulem. Umožňuje načtení obrazu z kamery do numpy polí v reálném čase.

2.2 Hardware

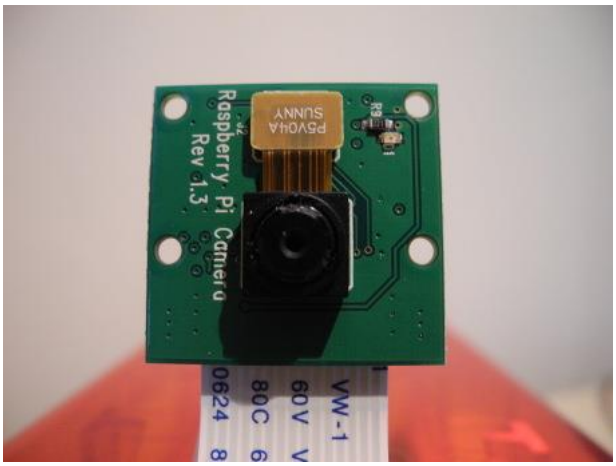
2.2.1 Raspberry Pi 3 B

Raspberry Pi 3 B je malý jednodeskový počítač s deskou plošných spojů o velikosti zhruha platební karty [1]. V tomto projektu je použito jako hlavní počítač pro program na sbírání dat a aplikace natrénovaného modelu.

Obr 1. Raspberry Pi 3 B

2.2.2 Raspberry Pi Camera Module v2

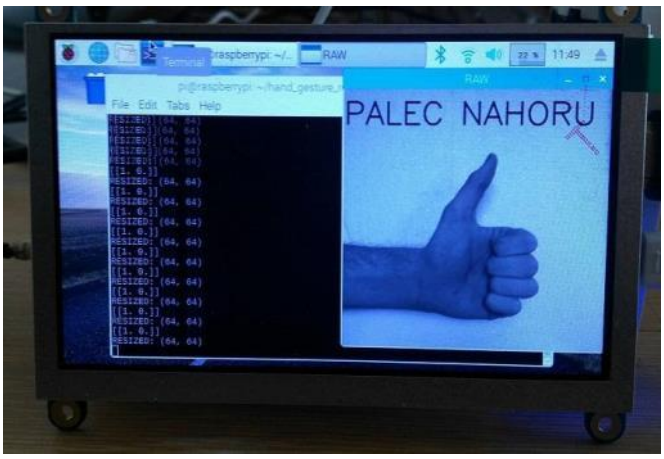
Raspberry Pi kamerový modul je malá kamera připojená pomocí speciálního kamerového konektoru k Raspberry Pi. Umožňuje nahrávání videa v 1080p kvalitě o frekvenci 30 snímků za sekundu [2]. Je použita pro nahrávání snímků za účelem tvorby dat a klasifikace pomocí natrénovaného modelu.



Obr 2. Raspberry Pi Camera Module v2

2.2.3 Adafruit 5" 800x480 HDMI display

Pěti palcový displej s HDMI konektorem pro zobrazení video z kamery.



Obr 3. Adafruit 5" 800x480 display

2.3 Služby

2.3.1 Paperspace VM

Paperspace VM (virtual machine - virtuální počítač) je placená služba poskytující vzdálený přístup k počítači s GPU (grafickou kartou), které je CUDA (rozhraní pro využití grafických karet pro paralelní komputaci [3]) kompatibilní a lze je tedy využít pro zvýšení rychlosti tréninku algoritmů strojového učení.

3 TEORIE STROJOVÉHO UČENÍ

Strojové učení je podoblastí umělé inteligence, zabývající se algoritmy a technikami, které umožňují počítačovému systému 'učit se'. Učením v daném kontextu rozumíme takovou změnu vnitřního stavu systému, která zefektivní schopnost přizpůsobit se změnám okolního prostředí.[4]

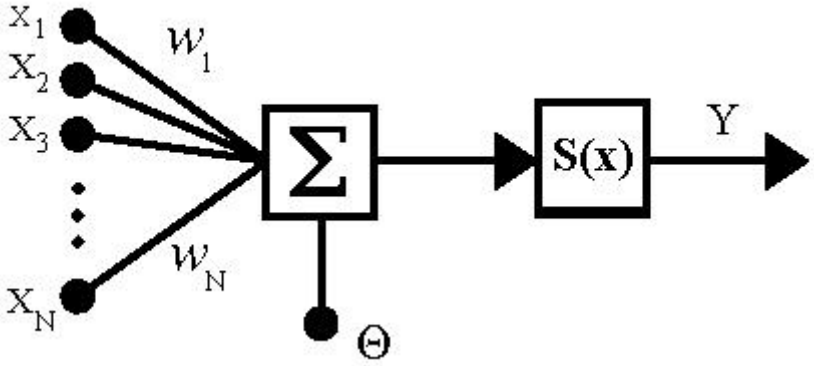
3.1 Učení s učitelem

Učení s učitelem je metoda strojového učení pro učení funkce z trénovacích dat obsahujících správnou třídu.[5]

3.2 Neuronová síť

Neuronová síť je jeden z výpočetních modelů používaných v umělé inteligenci. Jejím vzorem je chování odpovídajících biologických struktur. Umělá neuronová síť je struktura určená pro distribuované paralelní zpracování dat.

Skládá se z umělých (nebo také formálních) neuronů, jejichž předobrazem je biologický neuron. Neurony jsou vzájemně propojeny a navzájem si předávají signály a transformují je pomocí určitých přenosových funkcí. Neuron má libovolný počet vstupů, ale pouze jeden výstup. [6]



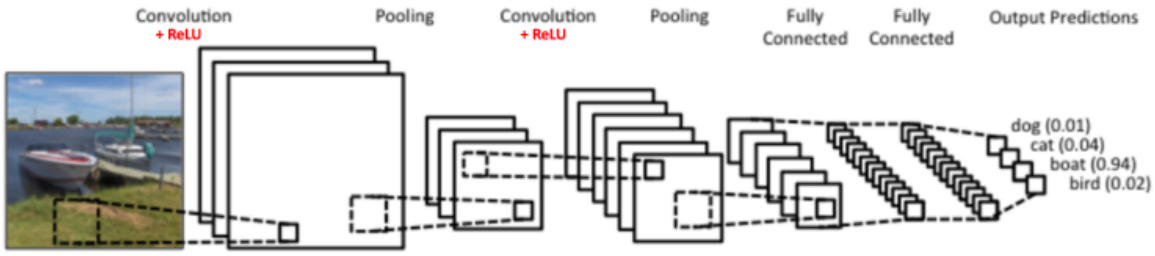
Obr 4. Model umělého neuronu

3.3 Cost funkce

Aby bylo možné změřit, jak moc dobře nebo špatně neuronová síť modeluje data, používá se funkce, která vrací číslo reprezentující míru chyby sítě. Této funkci se říká cost funkce. [7]$L(\theta, X, Y) = -\frac{1}{m}\Sigma(Y * ln(f(X)) + (1-Y) * (1 - ln(1 - f(X)))$

3.4 Konvoluční neuronová síť

Konvoluční neuronová síť je speciální typ neuronové sítě, která se využívá hlavně v oboru počítačového vidění. Oproti klasickým neuronovým sítím exceluje v detekování hran a rohů předmětů podle kterých dokáže velice přesně rozlišit dané předměty. Konvoluční síť typicky disponuje třemi typy vrstev. První je vždy vrstva konvoluční, po ní následuje max-poolingová vrstva a jako poslední obsahuje klasické plně-spojované vrstvy. [8]



Obr 5. Konvoluční neuronová sít

3.5 Augmentace dat

Augmentace dat je technika využívaná ve strojovém učením a hlavně v oboru počítačového vidění pro tvorbu nových dat z existujících dat. Pomocí techniky augmentace dat lze zvětšit množství dat, kterými disponujeme a tím zkvalitnit kvalitu modelu. Technika spočívá v tom, že vezmeme skutečný snímek ruky a zrcadlíme ho kolem vertikální osy. Tím získáme nový obrázek, který stále zobrazuje stejný předmět, avšak jiným způsobem.

3.6 Trénovací a testovací set

Trénovací set (anglicky Training set) obsahuje kolem 80 procent dat a využívá se pro trénink klasifikačního modelu. Testovací set (anglicky Test set) obsahuje kolem 20 procent dat a využívá se pro určení úspěšnosti tréninku modelu. Důvod za použitím test setu je důležitost testování modelu na datech, které předtím nikdy neviděl.

4 POPIS PROGRAMŮ

4.1 Získávání dat z videa

Kvalita každého modelu strojového učení je z velké části závislá na množství a kvalitě dat použitých při trénování. Pro detekci gesta ruky ze snímku o rozlišení 64x64 pixelů je potřeba alespoň 500 obrázků pro každé gesto (třídu).

Za předpokladu videa, které snímá určitý objekt o frekvenci 30 snímků za sekundu můžeme konstatovat, že teoreticky můžeme získat každou minutu 60 x 30 = 1800 snímků. Ovšem hodně snímků, bude natolik podobná, že jejich informační hodnota bude klesat. Proto zavádíme v programu na získávaní dat proměnnou *capture_ratio* (poměr zachycení), která reprezentuje poměr snímků, které budou uloženy pro použití při tréninku klasifikačního modelu a těch, které uloženy nebudou. To znamená, že když bude *capture_ratio* mít hodnotu tří, každý třetí snímek, který kamera zaznamená bude uložen do předem určené složky pro danou třídu (gesto).

Pro efektivní prostředí ke získávání kvalitních dat a testování modelů jsem vytvořil aparaturu skládající se z Raspberry Pi Camery připojené k Raspberry Pi 3 B, která je namířené na bílou polystyrenovou desku.

Obr 6. Aparatura pro testování modelů

4.2 Program pro zaznamenání obrázků z videa

Program lze nastavit tak, aby uložil požadovaný počet snímků do požadované složky v požadovaném rozlišení.

```
# importování knihoven
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import numpy as np
import tensorflow as tf
```

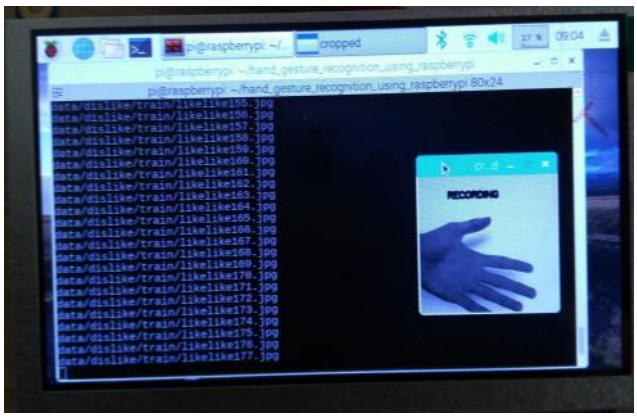
```
import cv2
import math


# pomocné funkce pro přípravu snímků

# zmenší rozlišení snímku, ale zachová poměr stran
def resize_img(img, img_size):
      y = math.floor((img_size/img.shape[0]) * img.shape[1])
      resized = cv2.resize(img, (y, img_size))
      return resized

# ořízne obrázek horizontálně
def crop_img_hor(img, final_width):
      left_border = math.floor((img.shape[1] - final_width) / 2)
      right_border = final_width + left_border
      img_cropped = img[:, left_border:right_border]
      return img_cropped

# ořízne obrázek horizontálně
def crop_img_ver(img, final_width):
      top_border = math.floor((img.shape[0] - final_width) / 2)
      bottom_border = final_width + top_border
      img_cropped = img[top_border: bottom_border, :]
      return img_cropped



# proměnné určené pro konfiguraci nahrávacího procesu

# rozlišení ve kterém bude kamera nahrávat
res = (320, 240)

# velikost stran obrázku, který bude uložen
img_size = 64

# velikost stran po oříznutí (ovlivňuje zorné pole)
cropped_size = 190

# poměr zaznamenanných snímků = každý pátý snímek bude zaznamenán
```

```
capture_ratio = 5

# frekvence snímků za sekund při která kamera pracuje
fps = 30

# slozka kam se obrázky budou ukládat
PATH = 'data/dislike/train/like'

# jméno každého obrázku + číslo
img_name = 'like'

# první obrazek bude mít za jménem toto číslo
start_index = 150

# počet snímků pro zaznamenání
img_count = 200


# inicializace kamery
camera = PiCamera()
camera.resolution = res
camera.framerate = fps
rawCapture = PiRGBArray(camera, size=res)

# zastavení programu pro 0.1 sekundy, aby se kamera mohlo rozehřát
time.sleep(0.1)

# první loop
# slouží pouze k ukázání výstupu z kamery na displeji
for frame in camera.capture_continuous(rawCapture, format='bgr',
use_video_port=True):

      # aktuální snímek jako numpy pole
      img = frame.array

      # převede snímek z RGB do grayscale
      img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
      # print(img_gray.shape)

      # ořízne obrázek tak aby jeho rozlišení byle cropped_size x
```

```
_cropped_size
      img_cropped = crop_img_hor(img_gray, cropped_size)
      img_cropped = crop_img_ver(img_cropped, cropped_size)

      # nastavení písma pro zobrazení textu na obrázku na displeji
      font = cv2.FONT_HERSHEY_SIMPLEX

      # přidá text 'not recording' na obrázek na displej
      cv2.putText(img_cropped, "NOT RECORDING",(40, 30),font, 0.4,
(0,155.0,), 2, cv2.LINE_AA)

      # ukáže snímek na displeji
      cv2.imshow("cropped", img_cropped)


      # zaznamenává zmáčknutí klávesy
      key = cv2.waitKey(1) & 0xFF


    # smaže poslední snímek z paměti
      rawCapture.truncate(0)

      # pokud dojde ke stisknutí klávesy 'q' loop skončí a program
pokraču
      if key == ord('q'):
      break



# první loop
# slouží k nahrávání obrázků do předem určené složky
i = 0
for frame in camera.capture_continuous(rawCapture, format='bgr',
use_video_port=True):

      # aktuální snímek jako numpy pole
      img = frame.array

      # převede snímek z RGB do grayscale
      img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
      # ořízne obrázek tak aby jeho rozlišení byle cropped_size x
_cropped_size
      img_cropped = crop_img_hor(img_gray, cropped_size)
      img_cropped = crop_img_ver(img_cropped, cropped_size)

      # přidá text 'recording' na obrázek na displej
      cv2.putText(img_cropped, "RECORDING",(40, 30),font, 0.4,
(0,255,0), 2, cv2.LINE_AA)

    # ukáže snímek na displeji
      cv2.imshow("cropped", img_cropped)

      # zaznamenává stisknutí klávesy
      key = cv2.waitKey(1) & 0xFF

      # smaže poslední snímek z paměti
      rawCapture.truncate(0)

      # pokud je zbytek dělení pořadí tohoto snímku poměrem
zachycení, tak dojde k uložení snímku
      scaled_index = 0
      if i % capture_ratio == 0:
       # změní číslo snímku, aby bylo o jedna větší než to minulé
      scaled_index = int(i - (i / capture_ratio) * (capture_ratio-
1) + start_index)
      # vypíše cestu k obrázku
      print(PATH + img_name + str(scaled_index) + ".jpg")

      # zmenší rozlišení obrázku na předem určené rozlišení
      img_resized = resize_img(img_cropped, img_size)

      # uloží obrázek to před určené složky pod před určeným názvem
      cv2.imwrite(PATH + img_name + str(scaled_index) +  ".jpg",
img_resized)


      i += 1

      # pokud dojke ke stisknutí klávesy 'q' nebo se uložení před
```

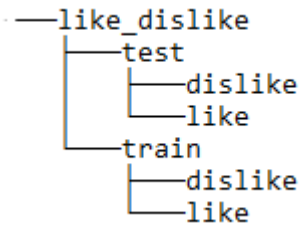
```
určené množství snímků tak se loop ukončí
      if key == ord('q') or scaled_index == (img_count +
start_index):
      break
```



Obr 7. Zaznamenání obrázků

4.3 Složkový systém ukládání dat

Ke každému obrázku gesta ruky musíme mít informaci o které gesto se jedná, aby mohl model při trénování využít učení s učitelem. V oblasti strojového učení existují dva hlavní způsoby, jak uložit informaci o třídě daného obrázku. Prvním způsobem je vygenerovat textový soubor obsahující seznam názvů všech obrázků a u každého připsaný index třídy. Druhý způsob a zároveň ten, který je použit v tomto projektu je rozdělit obrázky podle tříd do složek.



Obr 8. Struktura ukládaní dat

4.4 Trénovaní modelu na datech

```
# importovaní knihoven

import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import math
import time
from datetime import timedelta
from scipy import misc
import imageio
import cv2

# funkce pro načtení dat ze složek

def load_data(PATH):

      # načítá obrázky do numpy pole
      def load_images(filenames):
      images = np.array([imageio.imread(filename) for filename in
filenames])
      return images


      # počáteční složka
      default_dir = os.getcwd()

      # vejde to určené složky s daty
      os.chdir(PATH)


      # celá cesta složky z daty
      main_dir = os.getcwd()
      print("main_dir" + main_dir)

      # nazve slozek "test" a "train"
      sets = os.listdir()
```

```
os.chdir(sets[0])

# nacte tridy gest (napr palec nahoru a palec dolu)
classes = os.listdir()


# slozka s daty určenými k trénovaní modelu
train_dir = os.getcwd()


# inicializace polí pro trenovací data
train_X = []
train_Y = []


for i in range(len(classes)):
os.chdir(classes[i])
filenames = os.listdir()


train_X.append(load_images(filenames))
train_Y.append(np.full((train_X[-1].shape[0], 1), i))
os.chdir(train_dir)

os.chdir(main_dir)
os.chdir(sets[1])
test_dir = os.getcwd()
print(train_X[5].shape)

# inicializace polí pro testovací data
test_X = []
test_Y = []

for i in range(len(classes)):
os.chdir(classes[i])
filenames = os.listdir()
print(os.getcwd())
test_X.append(load_images(filenames))
test_Y.append(np.full((test_X[-1].shape[0], 1), i))
os.chdir(test_dir)
```

```
        os.chdir(default_dir)

        # spojení všech tříd trenujících dat do jednoho pole
        train_X = np.concatenate((train_X[0], train_X[1], train_X[2],
train_X[3], train_X[4], train_X[5]), axis=0)
        train_Y = np.concatenate((train_Y), axis=0)

        # spojení všech tříd testovacích dat do jednoho pole
        test_X = np.concatenate((test_X[0], test_X[1], test_X[2],
test_X[3], test_X[4], test_X[5]), axis=0)
        test_Y = np.concatenate((test_Y), axis=0)

        os.chdir(default_dir)

        return train_X, train_Y, test_X, test_Y


# cesta ke složce s daty
PATH = 'fingers/'


# nahrání dat do trénovacích a testovacích polí
test_X, test_Y_class, train_X, train_Y_class = load_data(PATH)

# rozlišení obrázků
img_size = 64

img_size_flat = img_size * img_size

img_shape = (img_size, img_size)


# pocet kanálů (obrázky jsou černobílé, takže 1 kanál)
num_channels = 1

# počet tříd obrázků (pro příklad počtu prstů na ruce je počet tříd
6 => nula, jedna, dva, tri, ctyri, pet)
num_classes = 6
```

```
# vykreslí 9 určených obrázků
def plot_images(images, true_class, pred_class=None):
      fig, axes = plt.subplots(3, 3)
      fig.subplots_adjust(hspace=0.3, wspace=0.3)

      for i, ax in enumerate(axes.flat):

      ax.imshow(images[i].reshape(img_size, img_size), cmap='gray')

      if pred_class is None:
            xlabel = f"True: {true_class[i]}"
      else:
            xlabel = f"True: {true_class[i]}, Pred: {pred_class[i]}"

      ax.set_xlabel(xlabel)

      ax.set_xticks([])
      ax.set_yticks([])

      plt.show()

index = 150

images = train_X[index:index+10]

true_class = train_Y_class[index:index+10]

plot_images(images=images, true_class=true_class)

# nastaví hodnotu pixelů mezi 0 a 1 místo 0 a 255
def preprocess_images(X):
      X = (X / 255 * 0.99) + 0.01
      return np.float32(X).reshape(-1, img_size, img_size,
num_channels)

# převede hodnotu třídy z přirozeného čísla na pole, kde index
předešlé hodnoty je jedna a zbytek 0
def one_hot(nums):
      output = np.eye(num_classes)[nums]
```

```
    return np.squeeze(output)

# upravená trénovací a testovací data
train_X = preprocess_images(train_X)
test_X = preprocess_images(test_X)

train_Y = one_hot(train_Y_class)
test_Y = one_hot(test_Y_class)


print(f"Number of train examples: {train_X.shape[0]}")
print(f"Number of test examples: {test_X.shape[0]}")
print(f"Train images shape: {train_X.shape}")
print(f"Test images shape: {test_X.shape}")
print(f"Train Labels shape: {train_Y.shape}")
print(f"Test Labels shape: {test_Y.shape}")

# stavba statického grafu modelu

# hyperparametry prvni konvolucni vrstvy
filter_size1 = 3
num_filters1 = 8

# hyperparametry druhe konvolucni vrstvy
filter_size2 = 3
num_filters2 = 16

# hyperparametry treti konvolucni vrstvy
filter_size3 = 3
num_filters3 = 32

# hyperparametry plne spojovane vrstvy
fc_layer = 512

# nasteveni poměru učení
lr=0.0005


# funkce pro definování konvolucni vrstvy
def create_conv_layer(input, num_input_channels, filter_size,
```

```
num_filters, use_pooling=True):

      filter_shape = [filter_size, filter_size, num_input_channels,
num_filters]

      print(filter_shape)

      weights =
tf.Variable(tf.truncated_normal(shape=filter_shape))

      biases = tf.Variable(tf.constant(0.05, shape=[num_filters]))

      layer = tf.nn.conv2d(input=input,
                           filter=weights,
                           strides=[1, 1, 1, 1],
                           padding='SAME')

      layer += biases

      if use_pooling:

      layer = tf.nn.max_pool(value=layer,
                                 ksize=[1, 2, 2, 1],
                                 strides=[1, 2, 2, 1],
                                 padding='SAME')

      layer = tf.nn.relu(layer)

      return layer, weights



# funkce pro definovaní plne-spojovane vrstvy
def create_fully_connected(input,
                                 num_inputs,
                                 num_outputs,
                                 use_relu=True):

      weights = tf.Variable(tf.truncated_normal(shape=[num_inputs,
num_outputs]))
```

```
        biases = tf.Variable(tf.constant(0.05, shape=[num_outputs]))

        layer = tf.matmul(input, weights) + biases

        if use_relu:
        layer = tf.nn.relu(layer)

        return layer


# funkce pro upravu vystupu konvolucni vrstvy, tak aby se dal
zpracovat plně spojovanou vrstvou
def flatten_layer(layer):

        layer_shape = layer.get_shape()

        num_features = layer_shape[1:4].num_elements()

        layer_flat = tf.reshape(layer, [-1, num_features])

        return layer_flat, num_features

# definovani vstu neuronove site
x = tf.placeholder(tf.float32, shape=[None, img_size, img_size,
num_channels], name='x')
y = tf.placeholder(tf.float32, shape=[None, num_classes], name='y')


# definovani prvni konvolucni vrstvy
layer_conv1, weights_conv1 = create_conv_layer(input=x,

        num_input_channels=num_channels,

        filter_size=filter_size1,

        num_filters=num_filters1,
                                                use_pooling=True)

# definovani druhé konvolucni vrstvy
```

```
layer_conv2, weights_conv2 = create_conv_layer(input=layer_conv1,
      num_input_channels=num_filters1,
      filter_size=filter_size2,
      num_filters=num_filters2,
                                          use_pooling=True)

# definovani třetí konvoluční vrstvy
layer_conv3, weights_conv3 = create_conv_layer(input=layer_conv2,
      num_input_channels=num_filters2,
      filter_size=filter_size3,
      num_filters=num_filters3,
                                          use_pooling=True)


# uprava vystupu treti konvolucni vrstvy aby mohl byt pouzit jako
vstup do plně-spojovane vrstvy
layer_flat, num_features = flatten_layer(layer_conv3)
print(f"Num features: {num_features}")

# vytvoreni první plně spojované vrstvy
fc_layer1 = create_fully_connected(input=layer_flat,
                                   num_inputs=num_features,
                                   num_outputs=fc_layer,
                                   use_relu=True)

# vytvoreni dropout vrstvy
dropout = tf.layers.dropout(inputs=fc_layer1, rate=0.2,
training=True)


# vytvoreni vystupní spojované vrstvy
fc_layer2 = create_fully_connected(input=dropout,
                                   num_inputs=fc_layer,
                                   num_outputs=num_classes,
```

```
                                    use_relu=False)

# nasazení poslední vrstvy do cost funkce
y_pred = tf.nn.softmax(fc_layer2)
y_pred_class = tf.argmax(y_pred, axis=1)

cross_entropy =
tf.nn.softmax_cross_entropy_with_logits_v2(logits=fc_layer2,
labels=y)
cost = tf.reduce_mean(cross_entropy)

# definovaci optimizeru
optimizer = tf.train.AdamOptimizer(learning_rate=lr).minimize(cost)

# ziskani presnoti naseho modelu
correct_predictions = tf.equal(y_pred_class,
train_Y_class.reshape(-1))
accuracy = tf.reduce_mean(tf.cast(correct_predictions, tf.float32))

# ulozeni struktury statickeho grafu
saver = tf.train.Saver()
save_dir = 'checkpoints/fingerstwo/'

# konecna definice a inicializace statickeho grafu
session = tf.Session()
session.run(tf.global_variables_initializer())

# rozdeleni dat do segmentu pro zrychleni treninku
batch_size = 256

def split_into_batches(X, Y):
      full_sized = math.floor(X.shape[0] / batch_size)
      print(f"Full sized: {full_sized}")
      batches_X = []
      batches_Y = []

      for i in range(full_sized):
      batches_X.append(X[i*batch_size:(i+1) * batch_size])
      batches_Y.append(Y[i*batch_size:(i+1) * batch_size])
```

```
        last_elems = full_sized * batch_size - X.shape[0]
        if last_elems != 0:
        print(f"kasel")
        batches_X.append(X[last_elems:])
        batches_Y.append(Y[last_elems:])

        return batches_X, batches_Y


# vrátí přesnost modelu
def print_accuracy(X, Y):
        feed_dict_test = {x: X,
                          y: Y}


        pred_class = session.run(y_pred_class,
feed_dict=feed_dict_test)


        true_class = np.argmax(Y, axis=1)

        pred_class = np.array(pred_class)
        correct = (pred_class == true_class)

        correct_sum = correct.sum()

        acc = float(correct_sum) / len(correct)

        return acc


# funkce pro optimizaci modelu
total_epochs = 0
def optimize(num_epochs):

        global total_epochs
        start_time = time.time()

        for i in range(total_epochs, total_epochs + num_epochs):
```

```
        for x_batch, y_batch in zip(X_batches, Y_batches):

            feed_dict_train = {x: x_batch,
                               y: y_batch}

            session.run(optimizer, feed_dict=feed_dict_train)


        print(f"Epoch {i}: Train_acc: {str(100)[:5]}% Test accuracy:
{str(print_accuracy(test_X, test_Y) * 100)[:5]}%")

        total_epochs += num_epochs

        end_time = time.time()

        time_dif = end_time - start_time

        print(f"Time usage:
{timedelta(seconds=int(round(time_dif)))}")
        print(f"Test accuracy: {str(print_accuracy(test_X, test_Y) *
100)[:5]}%")


# rozdeleni dat do trenovacich segmentů
X_batches, Y_batches = split_into_batches(train_X, train_Y)

# optimalizovani modelu
optimize(num_epochs=10)


# test kvality modelu pomocí vykreslovani obrazků s test setu s
predikcemi modelu

index = 750


images = test_X[index:index+10]

true_class = test_Y_class[index:index+10]
```

```
feed_dict_test = {x: test_X, y: test_Y}

preds = session.run(y_pred_class,
feed_dict=feed_dict_test)[index:index+10]

# preds = session.run(y_pred_class, feed_dict=feed_dict_test)
# print(preds.shape)


plot_images(images=images, true_class=true_class, pred_class=preds)


# ulozeni hodnot parametru do predem urcene slozky
saver.save(sess=session, save_path=save_dir)
```

4.5 Rozpoznání gest ruky v reálném čase

Program nahraje před natrénovaný model a klasifikuje třídy snímků z kamery v reálném čase.

```
# importování knihoven
import tensorflow as tf
import numpy as np
import cv2
import math
import os
import time
from scipy import misc
from picamera.array import PiRGBArray
from picamera import PiCamera



saver = tf.train.Saver()

save_dir = 'checkpoints/onetwo256dropout/'

session = tf.Session()

session.run(tf.global_variables_initializer())

saver.restore(sess=session, save_path=save_dir)


def resize_img(img, img_size):
    y = math.floor((img_size/img.shape[0])*img.shape[1])
    resized = cv2.resize(img, (y, img_size))
    return resized


def crop_img_hor(img, final_width):
    left_border = math.floor((img.shape[1] - final_width) / 2)
    right_border = final_width + left_border
    img_cropped = img[:, left_border:right_border]
    return img_cropped

def crop_img_ver(img, final_width):
```

```
        top_border = math.floor((img.shape[0] - final_width) / 2)
        bottom_border = final_width + top_border
        img_cropped = img[top_border: bottom_border, :]
        return img_cropped


    def preprocess_imgs(X):
        X = (X / 255 * 0.99) + 0.01
        return X


    # proměnné určené pro konfiguracu nahrávacího procesu

    # velikost stran obrázku, který bude použit jako vstup do
neuronove site
    image_size = 64

    # poměr klasifikovaných snímků = každý pátý snímek bude použit
jako vstup do neuronove site
    recognition_ratio = 4
    cropped_size = 380

    # rozlišení ve kterém bude kamera nahrávat
    res = (640,480)

    # frekvence snimku
    fps = 30


    # inicializace kamery0
    camera = PiCamera()
    camera.resolution = res
    camera.framerate = fps
    rawCapture = PiRGBArray(camera, size=res)

    # zastavení programu pro 0.1 sekundy, aby se kamera mohlo
rozehřát
    time.sleep(0.1)
```

```
  pred_cls = [0]
  i = 0
  # slouží k nahrávání obrázků do předem určené složky
  for frame in camera.capture_continuous(rawCapture, format='bgr',
use_video_port=True):


      # aktuální snímek jako numpy pole
      img = frame.array

      # převede snímek z RGB do grayscale
      img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


      # ořízne obrázek tak aby jeho rozlišení byle cropped_size x
_cropped_size
      img_cropped = crop_img_hor(img_gray, cropped_size)
      img_cropped = crop_img_ver(img_cropped, cropped_size)
      img_resized = resize_img(img_cropped, img_size)
      img_shapened = img_resized.reshape(1, img_size, img_size, 1)

      img_scaled = preprocess_imgs(img_shapened)


      pred = 0
      if i % recognition_ratio == 0:

            # upravý snímek, tak aby byl ve stejném formátu jako
data pro které byl náš model naučený
            img_resized = resize_img(img_cropped, img_size)
            img_shapened = img_resized.reshape(1, img_size,
img_size, 1)
            img_scaled = preprocess_imgs(img_shapened)

            # klasifikace aktuálního snímku
            pred_cls = session.run(y_pred_class, feed_dict={x:
img_scaled})
            pred = session.run(y_pred, feed_dict={x: img_scaled})
```

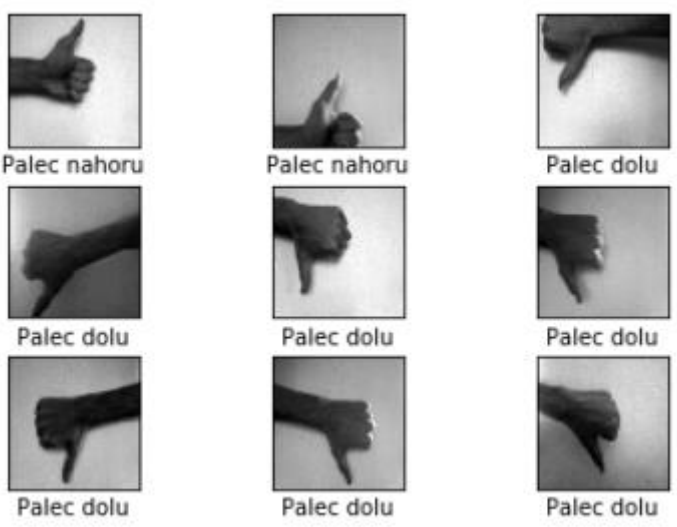
```
        # text, který se objeví na displeji na základě 'pred_cls'
        msg = ""
        if pred_cls[0] == 0:
                msg = "PALEC NAHORU"
        else:
                msg = "PALEC DOLU"

        # nastavení písma pro zobrazení textu na obrázku na displeji
        font = cv2.FONT_HERSHEY_SIMPLEX

        # zprava
        cv2.putText(img_cropped, msg, (0,50), font, 1.5, (0, 255, 0),
2, cv2.LINE_AA)



        # zobrazení aktuálního snímku
        cv2.imshow('RAW', img_cropped)
        key = cv2.waitKey(1) & 0xFF
        rawCapture.truncate(0)
        i += 1

        # po zmáčknutí klávesy q se celý program ukončí
        if key == ord('q'):
                print("FJDASLFJASLFJDASLF")
                break
```

5 TRÉNOVANÉ MODELY

Pomocí předešle popsaných programů jsem natrénoval tři klasifikační modely gest ruky. První dokáže rozeznat palec dolu a palec nahoru. Druhý počet zdvižených prstů na ruce. A třetí kámen, nůžky, papír.

5.1 Palec nahoru a palec dolu

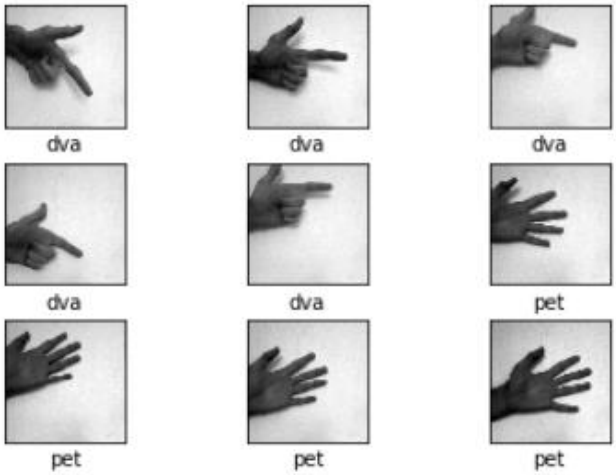
Detekce palce nahoru a palce dolu je jedna z nejjednodušších úloh v oblasti rozeznávání gest ruky. Jedná se o úlohu binární klasifikace. Po tréninku na 6 000 tréninkových obrázcích model dosahoval přesnosti 99.80 % na 1 000 testovacích obrázcích.



Obr 9. Klasifikace palec nahoru a palec dolu

5.2 Počet prstů ukázaných na ruce

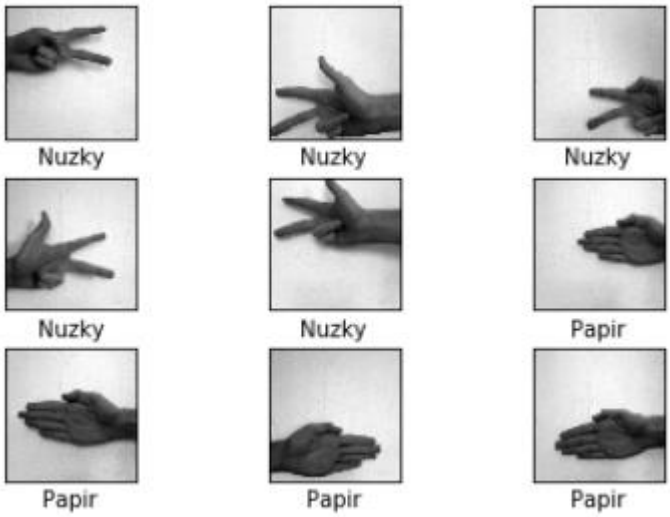
Detekce počtu zdvižených prstů na ruce je mnohem těžší, než detekce palce dolu a nahoru jelikož podobnost mezi třídami je vysoká. Po trénovaní na 5 400 trénovacích obrázcích model klasifikoval správně 99.33 % z 600 testovacích obrázcích.



Obr 10. Klasifikace počet prstů na ruce

5.3 Kámen, nůžky papír

Model trénovaný na 5 700 obrázcích a testovaný na 300 testovacích obrázcích dosahoval přesnosti 99.66 %.



Obr 11. Klasifikace kámen, nůžky, papír

6 ZÁVĚR

Za pomocí programů pro získávání dat z videa, tréninku modelu a predikce v reálném čase lze vytvořit model pro klasifikaci gest ruky ve velice nízkém čase a s vysokou přesností. Jedním z nedostatků je, že všechna data mají relativně podobné osvětlení a čistě bílé pozadí, což znamená, že například při horším osvětlení by modely nedosahovaly stejně vysoké přesnosti. Stejné pravidlo platí i pro pozadí jiné barvy. Dalším problémem je, že všechna data byla vytvořena s pomocí mé ruky, což znamená, že přesnost na snímcích vytvořených pomocí cizí ruky je neznámá a pravděpodobně horší.

Další krok za zlepšením přesnosti modelů i jiných prostředí by bylo nasbírat více dat, která jsou byla vytvořena za jiných světelných podmínek, s jiným pozadím a obsahující ruce více lidí.

7 POUŽITÁ LITERATURA

[1] Raspberry Pi

Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=21471

[2] Raspberry Pi Kamera

Dostupné z: https://www.geeetech.com/wiki/index.php/Raspberry_Pi_Camera_Module

[3] Cuda

Dostupné z: https://cs.wikipedia.org/wiki/CUDA

[4] Strojové učení

Dostupné z: https://cs.wikipedia.org/wiki/Strojov%C3%A9_u%C4%8Den%C3%AD

[5] Učení s učitelem

Dostupné z: https://cs.wikipedia.org/wiki/U%C4%8Den%C3%AD_s_u%C4%8Ditelem

[6] Umělá neuronová síť

Dostupné z: https://cs.wikipedia.org/wiki/Um%C4%9Bl%C3%A1_neuronov%C3%A1_s%C3%AD%C5%A5

[7] Cost funkce

Dostupné z: https://is.muni.cz/th/410034/fi_b/thesis.pdf

[8] Konvoluční neuronová síť

Dostupné z: https://en.wikipedia.org/wiki/Convolutional_neural_network#Design

https://is.muni.cz/th/410034/fi_b/thesis.pdf

https://en.wikipedia.org/wiki/Convolutional_neural_network#Design

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

https://cs.wikipedia.org/wiki/Um%C4%9Bl%C3%A1_neuronov%C3%A1_s%C3%AD%C5%A5

8 SEZNAM OBRÁZKŮ

9 PŘÍLOHY: ZDROJOVÝ KÓD

Zdrojový kód je uložen na úložišti GitHub zde:

https://github.com/SNEKBOI/hand_gesture_recognition_using_raspberrypi

Struktura zdrojového kódu:

1. Script pro získávání dat - image_recorder.py
2. Script pro trénovaní palec nahoru dolu - Like_dislike.ipynb
3. Script pro trénovaní kámen, nůžky, papír - rock_paper_scissors.ipynb
4. Script pro trénovaní počet prstů na ruce - fingers.ipynb
5. Script pro živé rozeznání palec nahoru dolu - like_recognition.py
6. Script pro živé rozeznání kámen, nůžky, papír - rpc_recognition.py
7. Script pro živé rozeznání počet prstů na ruce - finger_recognition
8. Složka checkpoints - předem trénované modely
9. Pomocný script pro náhodné přesouvání obrázků - move_files.py