Министерство науки и высшего образования Российской федерации
Новосибирский Государственный Технический Университет

Исследовательская работа
по нейронным сетям
в рамках проектной деятельности

Факультет: ПМИ

Группа: ПМ-63

Студент: Утюганов Д.С.

Преподаватель: Попов А.А.

Новосибирск, 2019

1. **Основание для разработки, назначение, требования, стадии и этапы разработки описаны в техническом задании.**

2. **Описание**

В рамках данной работы было реализовано 2 скрипта.

make_pickles.py – программа формирования готовых файлов нужного формата для нейронной сети из множества изображений.

convolutional_neural_network.py – программа сверточной нейронной сети, содержащая в себе 3 других сверточных нейронных сети для каждой из подзадач.

3. **Исследования**

Было проведено множество исследований, в рамках которых были найдены лучшие архитектуры для каждой внутренней нейронной сети.

Для каждой архитектуры записывался лог, хранящий в себе информацию о проценте точности предсказаний нейронной сети на обучающей и тестовой выборках.

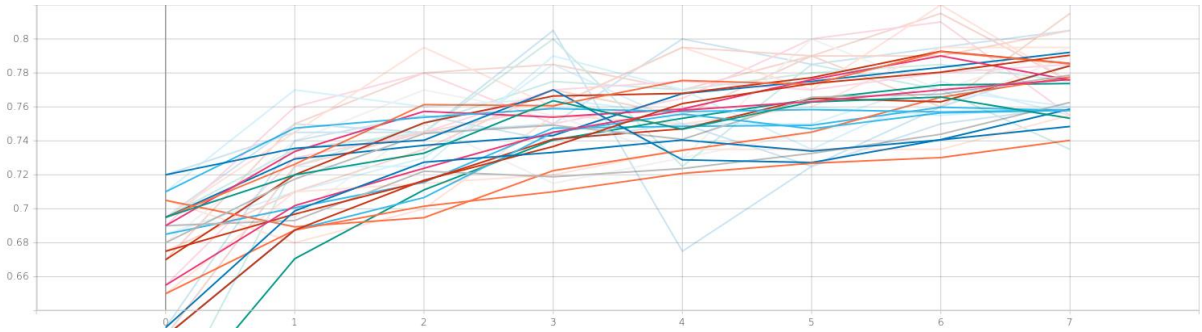
В качестве изменяемых параметров использовались:

- Количество сверточных слоев {1, 2, 3}
- Количество фильтров на каждом сверточном слое {16, 32}
- Размер пакета (batch) {8, 16, 32}

В качестве статичных параметров использовались:

- Количество плотных слоев: 0
- Количество частиц на каждом плотном слое: 0
- Kernel size: 3
- Max Pooling: 2
- Количество эпох: 8

3.1 Исследования по бинарному половому параметру (male, female):



Вертикальная ось – вероятность верного предсказания

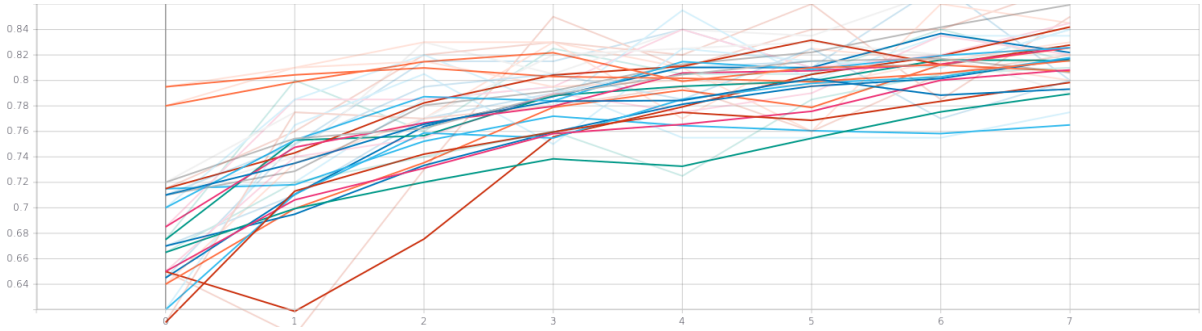
Горизонтальная ось – количество эпох

Лучший результат показала модель с архитектурой:

- 3 сверточных слоя
- 32 фильтра
- 8 пакетов (batch)
- Остальные параметры у всех моделей одинаковы

Вероятность **0.805** на тестовой выборке при затраченных на обучение данной модели 1 минуте 12 секунд.

3.2 Исследования по бинарному возрастному параметру (adult, child):



Вертикальная ось – вероятность верного предсказания

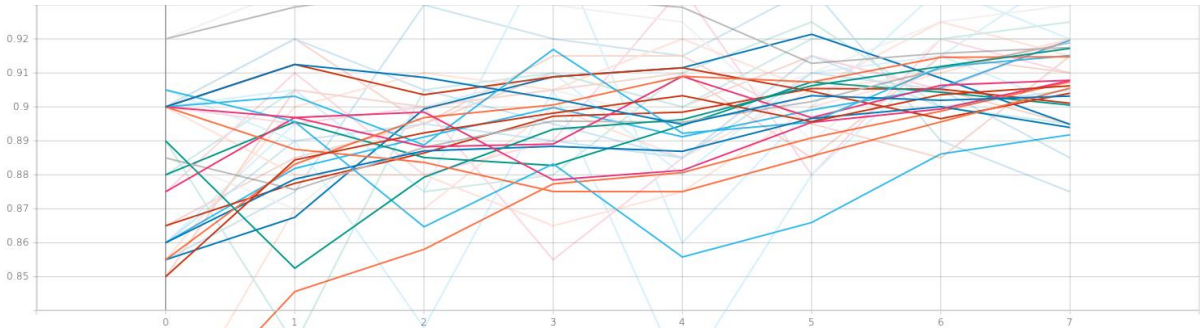
Горизонтальная ось – количество эпох

Лучший результат показала модель с архитектурой:

- 3 сверточных слоя
- 16 фильтров
- 16 пакетов (batch)
- Остальные параметры у всех моделей одинаковы

Вероятность **0.885** на тестовой выборке при затраченных на обучение данной модели 49 секунд.

3.3 Исследования по бинарному параметру цвета волос (blonde, dark):



Вертикальная ось – вероятность верного предсказания

Горизонтальная ось – количество эпох

Лучший результат показала модель с архитектурой:

- 3 сверточных слоя
- 32 фильтра
- 32 пакетов (batch)
- Остальные параметры у всех моделей одинаковы

Вероятность **0.94** на тестовой выборке при затраченных на обучение данной модели 57 секунд.

Среднее время обучения каждой модели от 40 до 70 секунд.

Предполагаю, что это не лучшие результаты, т.к. как видно по графикам: с ростом количества эпох по-прежнему увеличивается точность, а значит модель еще не дошла по состояния переобучаемости, и имеет смысл увеличить число эпох.

Наиболее точные результаты были у моделей с большим числом сверточных слоев, а значит имеет смысл проверить архитектуры с 4 и более свероточными слоями.

Также большое влияние имеет число фильтров на сверточном слое, но т.к. исследования проводились автоматически без внешнего контроля (на ночь), было опасение, что с увеличением фильтров (64, 128), все исследование может упасть и были протестированы наиболее «безопасные» архитектуры.

Было замечено, что точность на тестовой выборке приблизительно равна точности на обучающей выборке. Вероятно, это связано с большим размером обучающей выборки (~1500 изображений на каждый класс), а также с ее разнообразностью (качество фотографий, ракурсы, освещение, эмоции на лице, разнородность людей в целом etc.).

По результатам исследований видно, что точность по параметру цвета волос является наиболее высокой (0.94), меньше же точность у возрастного параметра (0.885), и самая маленькая точность у полового параметра (0.805).

В целом, на мой взгляд, результаты являются крайне хорошими, и это, наверняка, далеко не лучшие результаты, т.к. можно увеличить обучающую выборку, провести больше исследований по архитектуре с лучшей вычислительной мощностью.

Также можно расширять сами классы (например, вместо бинарного возрастного получить множество возрастных интервалов, или разбить цвета волос по оттенкам), а также увеличить число классов в целом (например, ввести класс цвета кожи, наличия растительности на лице, цвета глаз, этнической принадлежности etc.), что существенно увеличит сложность задачи, но сделает ее более прикладной и полезной.

4. Листинг кода

Весь код находится в открытом доступе на: https://github.com/Sing3Rous/Human_Classifier_Neural_Network

make_pickles.py

```
import numpy as np
import os
import cv2
from tqdm import tqdm
import random
import pickle

def create_data(dir, categories, imageSize, className, isTest):
    data = []
    for category in categories:
        if (isTest):
            path = os.path.join(dir + "\\" + className + "\\" + "validate",
category)
        else:
            path = os.path.join(dir + "\\" + className, category)

        classNum = categories.index(category)

        for image in tqdm(os.listdir(path)):
            try:
                imageArray = cv2.imread(os.path.join(path, image),
```

```
cv2.IMREAD_COLOR)
                normalizedImageArray = cv2.resize(imageArray, (imageSize,
imageSize))
                data.append([normalizedImageArray, classNum])

            except Exception:
                pass

    return data

def make_pickle(dir, categories, imageSize, className, isTest):
    data = create_data(dir, categories, imageSize, className, isTest)
    random.shuffle(data)
    xData, yData = process_data(data, imageSize)
    if (isTest):
        pickleOut = open(dir + "\\" + className + "\\pickles\\" + "x" +
className + "Validate" + ".pickle", "wb")
    else:
        pickleOut = open(dir + "\\" + className + "\\pickles\\" + "x" +
className + ".pickle", "wb")
    pickle.dump(xData, pickleOut)
    pickleOut.close()
    if (isTest):
        pickleOut = open(dir + "\\" + className + "\\pickles\\" + "y" +
className + "Validate" + ".pickle", "wb")
    else:
        pickleOut = open(dir + "\\" + className + "\\pickles\\" + "y" +
className + ".pickle", "wb")
    pickle.dump(yData, pickleOut)
    pickleOut.close()

def process_data(data, imageSize):
    X = []
    y = []
    for features, label in data:
        X.append(features)
        y.append(label)

    X = np.array(X).reshape(-1, imageSize, imageSize, 3)
    X = X / 255.0
    return X, y

genderCategories = ["male", "female"]
hairColorCategories = ["blonde", "dark"]
ageCategories = ["adult", "child"]

dir = "C:\\Users\\singe\\Documents\\Human Classifier"
imageSize = 200
make_pickle(dir, genderCategories, imageSize, "gender", False)
make_pickle(dir, hairColorCategories, imageSize, "hair_color", False)
make_pickle(dir, ageCategories, imageSize, "age", False)
make_pickle(dir, genderCategories, imageSize, "gender", True)
make_pickle(dir, hairColorCategories, imageSize, "hair_color", True)
make_pickle(dir, ageCategories, imageSize, "age", True)
```

convolutional_neural_network.py

```
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.models import model_from_json
import pickle
import time

def add_convolutional_layers(numOfLayers, model, filters, kernelSize, poolSize):
    for i in range(numOfLayers):
        model.add(Conv2D(filters, (kernelSize, kernelSize)))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(poolSize, poolSize)))

def add_dense_layers(numOfLayers, model, units):
    for i in range(numOfLayers):
        model.add(Dense(units))
        model.add(Activation('relu'))
```

```
def add_input_layer(X, model, filters, kernelSize, poolSize):
    model.add(Conv2D(filters, (kernelSize, kernelSize),
input_shape=X.shape[1:]))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(poolSize, poolSize)))

def add_output_layer(model):
    model.add(Dense(1))
    model.add(Activation('sigmoid'))

def load_data(datadir, className, isTest):
    pickles = []
    if (isTest):
        pickleIn = open(datadir + "\\" + className + "\\" + "pickles\\" + "x" +
className + "Validate" + ".pickle", "rb")
    else:
        pickleIn = open(datadir + "\\" + className + "\\" + "pickles\\" + "x" +
className + ".pickle", "rb")
    pickles.append(pickle.load(pickleIn))
    if (isTest):
        pickleIn = open(datadir + "\\" + className + "\\" + "pickles\\" + "y" +
className + "Validate" + ".pickle", "rb")
    else:
        pickleIn = open(datadir + "\\" + className + "\\" + "pickles\\" + "y" +
className + ".pickle", "rb")
    pickles.append(pickle.load(pickleIn))
    return pickles

def save_model(model, datadir, className, CNNArchitecture, CNNparameters):
    modelJson = model.to_json()
    with open(datadir + "\\" + className + "\\models\\" + className + "_conv[" +
str(CNNArchitecture["numOfConvLayers"])
              + "]_" + "filters[" + str(CNNArchitecture["filters"]) + "]_" +
"batches[" + str(CNNparameters["batchSize"])
              + "]" + ".json", 'w') as jsonFile:
        jsonFile.write(modelJson)
    model.save_weights(datadir + "\\" + className + "\\models\\" + className +
"_conv["
                       + str(CNNArchitecture["numOfConvLayers"]) + "]_" +
"filters[" + str(CNNArchitecture["filters"])
                       + "]_" + "batches[" + str(CNNparameters["batchSize"]) +
"]" + ".h5")

def load_model(datadir, modelName):
    jsonFile = open(datadir + "\\models\\" + modelName + ".json", 'r')
    modelJson = jsonFile.read()
    jsonFile.close()
    model = model_from_json(modelJson)
    model.load_weights(datadir + "\\models\\" + modelName + ".h5")
    return model

def build_model(CNNArchitecture, trainData):
    model = Sequential()
    add_input_layer(trainData[0], model,
                    CNNArchitecture["filters"],
                    CNNArchitecture["kernelSize"],
                    CNNArchitecture["poolSize"])
    add_convolutional_layers(CNNArchitecture["numOfConvLayers"], model,
                             CNNArchitecture["filters"],
                             CNNArchitecture["kernelSize"],
                             CNNArchitecture["poolSize"])
    model.add(Flatten())
    add_dense_layers(CNNArchitecture["numOfDenseLayers"], model,
CNNArchitecture["units"])
    add_output_layer(model)

    return model

def train_model(model, CNNparameters, trainData, validateData, board):
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    model.fit(trainData[0], trainData[1],
              batch_size=CNNparameters["batchSize"],
```

```
              epochs=CNNparameters["numOfEpochs"],
              validation_split=0.3,
              callbacks=[board],
              validation_data=validateData)

#cnn architectures:
#########################################################################################
genderCNNArchitecture = {
    'numOfConvLayers': 2,
    'filters': 32,
    'kernelSize': 3,
    'numOfDenseLayers': 0,
    'units': 0,
    'poolSize': 2
}

ageCNNArchitecture = {
    'numOfConvLayers': 2,
    'filters': 32,
    'kernelSize': 3,
    'numOfDenseLayers': 0,
    'units': 0,
    'poolSize': 2
}

hairColorCNNArchitecture = {
    'numOfConvLayers': 2,
    'filters': 32,
    'kernelSize': 3,
    'numOfDenseLayers': 0,
    'units': 0,
    'poolSize': 2
}
#########################################################################################
#########################################

#cnn parameters
#########################################################################################
#########################################
genderCNNparameters = {
    'batchSize': 16,
    'numOfEpochs': 5
}

ageCNNparameters = {
    'batchSize': 16,
    'numOfEpochs': 5
}

hairColorCNNparameters = {
    'batchSize': 16,
    'numOfEpochs': 5
}

genderClassName = "gender"
ageClassName = "age"
hairColorClassName = "hair_color"

dir = "C:\\Users\\singe\\Documents\\Human Classifier"
genderTrain = load_data(dir, genderClassName, False)
ageTrain = load_data(dir, ageClassName, False)
hairColorTrain = load_data(dir, hairColorClassName, False)
genderTrainValidate = load_data(dir, genderClassName, True)
ageTrainValidate = load_data(dir, ageClassName, True)
hairColorTrainValidate = load_data(dir, hairColorClassName, True)
#########################################################################################
#########################################

#process logs
#########################################################################################
#########################################
genderLogName = "gender-{}-conv-{}-filters-{}-batches-
{}".format(genderCNNArchitecture["numOfConvLayers"],

genderCNNArchitecture["filters"],
```

```
genderCNNparameters["batchSize"],
                                                    int(time.time()))
genderBoard = TensorBoard(log_dir='C:\\Users\\singe\\Documents\\Human
Classifier\\gender\\logs\\{}'.format(genderLogName))

ageLogName = "age-{}-conv-{}-filters-{}-batches-
{}".format(ageCNNArchitecture["numOfConvLayers"],

ageCNNArchitecture["filters"],

ageCNNparameters["batchSize"],
                                                    int(time.time()))
ageBoard = TensorBoard(log_dir='C:\\Users\\singe\\Documents\\Human
Classifier\\age\\logs\\{}'.format(ageLogName))

hairColorLogName = "hair_color-{}-conv-{}-filters-{}-batches-
{}".format(hairColorCNNArchitecture["numOfConvLayers"],

hairColorCNNArchitecture["filters"],

hairColorCNNparameters["batchSize"],
                                                    int(time.time()))
hairColorBoard = TensorBoard(log_dir='C:\\Users\\singe\\Documents\\Human
Classifier\\hair_color\\logs\\{}'.format(hairColorLogName))
########################################################################
#####################################

#build & train models
########################################################################
########################################
genderModel = build_model(genderCNNArchitecture, genderTrain)
train_model(genderModel, genderCNNparameters, genderTrain, genderTrainValidate,
genderBoard)
save_model(genderModel, dir, genderClassName, genderCNNArchitecture,
genderCNNparameters)

ageModel = build_model(ageCNNArchitecture, ageTrain)
train_model(ageModel, ageCNNparameters, ageTrain, ageTrainValidate, ageBoard)
save_model(ageModel, dir, ageClassName, ageCNNArchitecture, ageCNNparameters)

hairColorModel = build_model(hairColorCNNArchitecture, hairColorTrain)
train_model(hairColorModel, hairColorCNNparameters, hairColorTrain,
hairColorTrainValidate, hairColorBoard)
save_model(hairColorModel, dir, hairColorClassName, hairColorCNNArchitecture,
hairColorCNNparameters)
########################################################################
#####################################

#research architecture & parameters of cnn
########################################################################
########################################
# researchNumOfConvLayers = [1, 2, 3]
# researchFilters = [16, 32]
# researchBatchSize = [8, 16, 32]
#
# researchCNNArchitecture = {
#     'numOfConvLayers': 2,
#     'filters': 16,
#     'kernelSize': 3,
#     'numOfDenseLayers': 0,
#     'units': 0,
#     'poolSize': 2
# }
#
# researchCNNparameters = {
#     'batchSize': 16,
#     'numOfEpochs': 8
# }
########################################################################
#####################################

#research code
########################################################################
########################################
# for numOfConvLayers in researchNumOfConvLayers:
```

```
#     for filters in researchFilters:
#         for batchSize in researchBatchSize:
#             researchCNNArchitecture.pop("numOfConvLayers")
#             researchCNNArchitecture.pop("filters")
#             researchCNNparameters.pop("batchSize")
#             researchCNNArchitecture.setdefault("numOfConvLayers",
numOfConvLayers)
#             researchCNNArchitecture.setdefault("filters", filters)
#             researchCNNparameters.setdefault("batchSize", batchSize)
#
#             genderLogName = "gender-{}-conv-{}-filters-{}-batches-
{}".format(researchCNNArchitecture["numOfConvLayers"],
#
researchCNNArchitecture["filters"],
#
researchCNNparameters["batchSize"],
#
int(time.time()))
#             genderBoard = TensorBoard(
#                 log_dir='C:\\Users\\singe\\Documents\\Human
Classifier\\gender\\logs\\{}'.format(genderLogName))
#
#             ageLogName = "age-{}-conv-{}-filters-{}-batches-
{}".format(researchCNNArchitecture["numOfConvLayers"],
#
researchCNNArchitecture["filters"],
#
researchCNNparameters["batchSize"],
#
int(time.time()))
#             ageBoard = TensorBoard(
#                 log_dir='C:\\Users\\singe\\Documents\\Human
Classifier\\age\\logs\\{}'.format(ageLogName))
#
#             hairColorLogName = "hair_color-{}-conv-{}-filters-{}-batches-
{}".format(
#                 researchCNNArchitecture["numOfConvLayers"],
#                 researchCNNArchitecture["filters"],
#                 researchCNNparameters["batchSize"],
#                 int(time.time()))
#             hairColorBoard = TensorBoard(
#                 log_dir='C:\\Users\\singe\\Documents\\Human
Classifier\\hair_color\\logs\\{}'.format(hairColorLogName))
#
#             genderModel = build_model(researchCNNArchitecture, genderTrain)
#             train_model(genderModel, researchCNNparameters, genderTrain,
genderTrainValidate, genderBoard)
#             save_model(genderModel, dir, genderClassName,
researchCNNArchitecture, researchCNNparameters)
#
#             ageModel = build_model(researchCNNArchitecture, ageTrain)
#             train_model(ageModel, researchCNNparameters, ageTrain,
ageTrainValidate, ageBoard)
#             save_model(ageModel, dir, ageClassName, researchCNNArchitecture,
researchCNNparameters)
#
#             hairColorModel = build_model(researchCNNArchitecture,
hairColorTrain)
#             train_model(hairColorModel, researchCNNparameters, hairColorTrain,
hairColorTrainValidate, hairColorBoard)
#             save_model(hairColorModel, dir, hairColorClassName,
researchCNNArchitecture, researchCNNparameters)
##########################################################################
########################################
```