Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
|  |

**ЛабоРАТОРНАЯ РАБОТА №5**

«Применение свёрточных нейронных сетей (бинарная классификация)»

|  |  |  |
| --- | --- | --- |
| Студент |  | К. Д. Зюсько |
| Преподаватель |  | М. В. Стержанов |

Минск 2020

ХОД РАБОТЫ

**Задание.**

Набор данных DogsVsCats, который состоит из изображений различной размерности, содержащих фотографии собак и кошек. Обучающая выборка включает в себя 25 тыс. изображений (12,5 тыс. кошек: cat.0.jpg, …, cat.12499.jpg и 12,5 тыс. собак: dog.0.jpg, …, dog.12499.jpg), а контрольная выборка содержит 12,5 тыс. неразмеченных изображений. Скачать данные, а также проверить качество классификатора на тестовой выборке можно на сайте Kaggle -> https://www.kaggle.com/c/dogs-vs-cats/data

1. Загрузите данные. Разделите исходный набор данных на обучающую, валидационную и контрольную выборки.
2. Реализуйте глубокую нейронную сеть с как минимум тремя сверточными слоями. Какое качество классификации получено?
3. Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?
4. Поэкспериментируйте с готовыми нейронными сетями (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение. Как это повлияло на качество классификатора? Какой максимальный результат удалось получить на сайте Kaggle? Почему?

**Результат выполнения:**

Поскольку в данной лабораторной предлагается сравнить результаты трёх различных конфигураций нейронных сетей, я установил количество эпох для обучение равным 30, чтобы условия, в которых происходит сравнение, были одинаковыми.

1. Сначала я убедился в том, что данные распределены одинаково и классы сбалансированы:

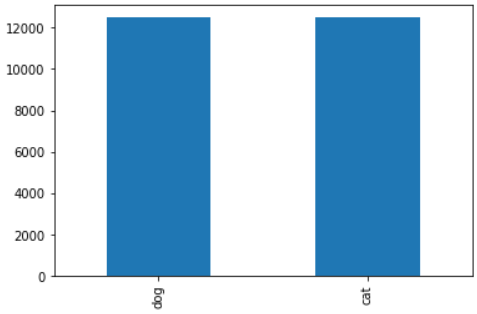


Рисунок 1 – распределение данных

Для того, чтобы уменьшить нашрузку на ОЗУ я использовал ImageDataGenerator из пакета keras.preprocessing.image, и его метод flow\_from\_dataframe. С помощью класса ImageDataGenerator я также нормализировал данные, и в последующих пунктах применял аугментацию данных также с его использованием:

!mkdir ~/.kaggle

!cp /content/.kaggle/kaggle.json ~/.kaggle/kaggle.json

!kaggle competitions download -c dogs-vs-cats -p /content/kaggle/dogscats

!unzip kaggle/dogscats/train.zip -d train

!unzip kaggle/dogscats/test1.zip -d test

import pandas as pd

import os

TRAIN\_PATH = './train/train'

TEST\_PATH = './test/test1'

filenames = os.listdir(TRAIN\_PATH)

categories = []

for filename in filenames:

category = filename.split('.')[0]

if category == 'dog':

categories.append(1)

else:

categories.append(0)

df = pd.DataFrame({

'filename': filenames,

'category': categories

})

df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})

test\_filenames = os.listdir(TEST\_PATH)

test\_df = pd.DataFrame({

'filename': test\_filenames

})

nb\_samples = test\_df.shape[0]

from sklearn.model\_selection import train\_test\_split

train, val = train\_test\_split(df, test\_size=0.20, random\_state=42)

from keras.preprocessing.image import ImageDataGenerator, load\_img

# train

train\_datagen = ImageDataGenerator(rescale=1./255)

total\_train = train.shape[0]

total\_validate = val.shape[0]

batch\_size = 32 # since Andrew Ng told, that better use 2^n

IMAGE\_WIDTH=128

IMAGE\_HEIGHT=128

IMAGE\_SIZE=(IMAGE\_WIDTH, IMAGE\_HEIGHT)

IMAGE\_CHANNELS=3

train\_generator = train\_datagen.flow\_from\_dataframe(

train,

TRAIN\_PATH,

x\_col='filename',

y\_col='category',

target\_size=IMAGE\_SIZE,

class\_mode='categorical',

batch\_size=batch\_size

)

# validation

validation\_datagen = ImageDataGenerator(rescale=1./255)

validation\_generator = validation\_datagen.flow\_from\_dataframe(

val,

TRAIN\_PATH,

x\_col='filename',

y\_col='category',

target\_size=IMAGE\_SIZE,

class\_mode='categorical',

batch\_size=batch\_size

)

test\_gen = ImageDataGenerator(rescale=1./255)

test\_generator = test\_gen.flow\_from\_dataframe(

test\_df,

TEST\_PATH,

x\_col='filename',

y\_col=None,

class\_mode=None,

target\_size=IMAGE\_SIZE,

batch\_size=batch\_size,

shuffle=False

)

2. В качестве архитектуры нейронной сети я использовал ту же самую, как и в третьей лабораторной:

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation, BatchNormalization

IMAGE\_WIDTH=128

IMAGE\_HEIGHT=128

IMAGE\_SIZE=(IMAGE\_WIDTH, IMAGE\_HEIGHT)

IMAGE\_CHANNELS=3

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(IMAGE\_WIDTH, IMAGE\_HEIGHT, IMAGE\_CHANNELS)))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax')) # 2 classes: dog and cat

model.compile(loss='categorical\_crossentropy', optimizer='rmsprop', metrics=['acc'])

model.summary()

model.save\_weights('initial.h5')

model.load\_weights('initial.h5')

history = model.fit\_generator(

train\_generator,

epochs=30,

validation\_data=validation\_generator,

validation\_steps=total\_validate//batch\_size,

steps\_per\_epoch=total\_train//batch\_size,

)

Результаты обучения приведены в таблице ниже.

Таблица 1 – результаты обучения

|  |  |  |
| --- | --- | --- |
|  | Loss | Accuracy |
| Тренировочный датасет | 0.0874 | 96.89% |
| Валидационный датасет | 0.5028 | 88.43% |

Как можно видеть выше, ошибка на валидационном датасете больше почти в шесть раз, чем на тренировочном. Точность также ниже на валидационном датасете. Это свидетельствует о том, что мы столкнулись с проблемой high variance.

3. Все дополнительные препроцессинги можно увидеть ниже. Аугментирование данных применялось только к тренировочному датасету:

train\_datagen\_aug = ImageDataGenerator(

rotation\_range=15,

rescale=1./255,

shear\_range=0.1,

zoom\_range=0.2,

horizontal\_flip=True,

width\_shift\_range=0.1,

height\_shift\_range=0.1

)

train\_generator\_aug = train\_datagen\_aug.flow\_from\_dataframe(

train,

TRAIN\_PATH,

x\_col='filename',

y\_col='category',

target\_size=IMAGE\_SIZE,

class\_mode='categorical',

batch\_size=batch\_size

)

model.load\_weights('initial.h5')

history\_aug = model.fit\_generator(

train\_generator\_aug,

epochs=30,

validation\_data=validation\_generator,

validation\_steps=total\_validate//batch\_size,

steps\_per\_epoch=total\_train//batch\_size,

)

Таблица 2 – результаты обучения

|  |  |  |
| --- | --- | --- |
|  | Loss | Accuracy |
| Тренировочный датасет | 0.2572 | 89.22% |
| Валидационный датасет | 0.2261 | 91.75% |

Из таблицы выше видно, что ошибки примерно равны, что свидетельствует о том, что проблема high variance, как и ожидалось, была решена с помощью увеличения датасета (в данном случае аугментированием данных).

4. Здесь я также постарался создать те же условия, как и везде, поэтому полносвязный слой состоит из 512 нейронов, и также имеет дропаут слой с коэффицентом 0.5:

from keras.models import Model

from keras.applications.vgg16 import VGG16

# load model

vgg = VGG16(include\_top=False, input\_shape=(IMAGE\_HEIGHT, IMAGE\_WIDTH, IMAGE\_CHANNELS))

# mark loaded layers as not trainable

for layer in vgg.layers:

layer.trainable = False

# define a new output layer to connect with the last fc layer in vgg

x = Flatten()(vgg.layers[-1].output)

class1 = Dense(512, activation='relu', kernel\_initializer='he\_uniform')(x)

bn = BatchNormalization()(class1)

do = Dropout(0.5)(bn)

output\_layer = Dense(2, activation='softmax', name='predictions')(do)

# combine the original VGG model with the new output layer

vgg = Model(inputs=vgg.input, outputs=output\_layer)

vgg.compile(loss='categorical\_crossentropy', optimizer='rmsprop', metrics=['acc'])

Таблица 3 – результаты обучения

|  |  |  |
| --- | --- | --- |
|  | Loss | Accuracy |
| Тренировочный датасет | 0.2620 | 88.91% |
| Валидационный датасет | 0.2261 | 90.20% |

Как видно выше, результаты, практически, не изменились, по сравнению со вторым пунктом. Есть небольшие различия, однако, это можно списать на погрешность в обучении.

Но в сравнении с предыдущей конфигурации нейронной сети, если посмотреть на историю обучения, то можно заметить, что вариант с транферным обучением «обучается» быстрее, а именно ошибка после первой эпохи значительно меньше, чем в предыдущем случае. Объясняется это тем, что мы использовали предобученную конфигурацию нейронной сети, которая обучалась на других изображения, но конволюционные слои запомнили «фичи» изображений, что и сказалось на скорости уменьшения ошибки.

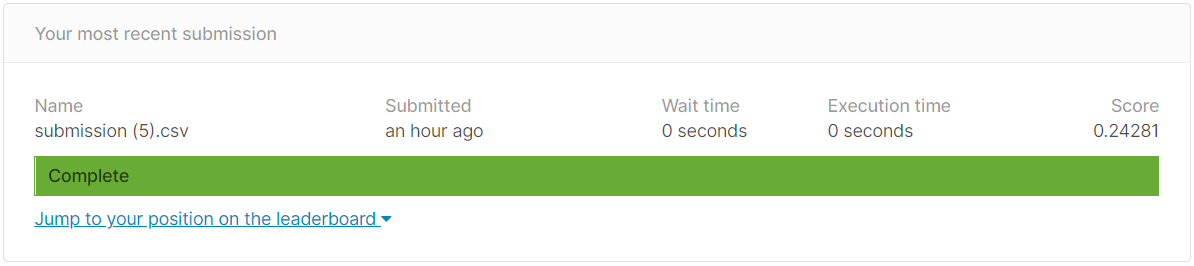


Рисунок 2 – результаты submission на сайте kaggle

На сайт kaggle я загружал результаты, полученные с помощью трансферного обучения с моделью VGG16, и с аугментированными данными. Итоговый score по всем данным (train/val/test) составил 0.262/0.2261/0.24281. Как можно увидеть, ошибки во всех трёх случаях отличаются незначительно. Это значит, что модель была подобрана правильно, и проблем high bias/variance в данном случае нету.