Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по предмету:
«Машинное обучение»
По лабораторной работе №5
«Применение сверточных нейронных сетей (бинарная классификация)»

Выполнил: Сенькович Дмитрий Сергеевич
магистрант кафедры информатики
группы №858642

Проверил: Стержанов Максим Валерьевич
доцент, кандидат технических наук

Минск 2020

Оглавление

1 Постановка задачи

Имеется две выборки по 25000 изображений: тренировочная и тестовая. Выборки наполовину состоят из изображений котов и собак. Требуется создать модель для классификаций животных на изображении.

2 Чтение данных и формирование выборок

Покажем, как будем преобразовывать, считывать данные и дополнять данные:

```
def crop_and_resize_images(dataset_name):

    processed_dataset_directory = PROCESSED_DIRECTORY_PREFIX + dataset_name

    if os.path.isdir(processed_dataset_directory):

        return


    os.makedirs(processed_dataset_directory)


    for image_file_name in os.listdir(dataset_name):

        full_file_name = dataset_name + '/' + image_file_name

        image = cv2.imread(full_file_name)

        resized_image = cv2.resize(image, (64, 64))

        cv2.imwrite(processed_dataset_directory + '/' + image_file_name,
resized_image)




def augment_vertical_flip():

    augmented_dataset_directory = AUGMENTED_VERTICAL_FLIP_DIRECTORY

    if os.path.isdir(augmented_dataset_directory):

        return


    os.makedirs(augmented_dataset_directory)


    processed_dataset_directory = PROCESSED_DIRECTORY_PREFIX +
TRAIN_DATASET_NAME
```

```
    for image_file_name in os.listdir(processed_dataset_directory):
        full_file_name = processed_dataset_directory + '/' + image_file_name
        image = cv2.imread(full_file_name)
        flipped = np.flipud(image)
        cv2.imwrite(augmented_dataset_directory + '/' + image_file_name,
flipped)



def augment_horizontal_flip():
    augmented_dataset_directory = AUGMENTED_HORIZONTAL_FLIP_DIRECTORY
    if os.path.isdir(augmented_dataset_directory):
        return


    os.makedirs(augmented_dataset_directory)


    processed_dataset_directory = PROCESSED_DIRECTORY_PREFIX +
TRAIN_DATASET_NAME


    for image_file_name in os.listdir(processed_dataset_directory):
        full_file_name = processed_dataset_directory + '/' + image_file_name
        image = cv2.imread(full_file_name)
        flipped = np.fliplr(image)
        cv2.imwrite(augmented_dataset_directory + '/' + image_file_name,
flipped)



def augment_blur():
```

```
    augmented_dataset_directory = AUGMENTED_BLUR_FLIP_DIRECTORY

    if os.path.isdir(augmented_dataset_directory):

        return


    os.makedirs(augmented_dataset_directory)


    processed_dataset_directory = PROCESSED_DIRECTORY_PREFIX +
TRAIN_DATASET_NAME


    for image_file_name in os.listdir(processed_dataset_directory):

        full_file_name = processed_dataset_directory + '/' + image_file_name

        image = cv2.imread(full_file_name)

        blurred = cv2.GaussianBlur(image, (11, 11), 0)

        cv2.imwrite(augmented_dataset_directory + '/' + image_file_name,
blurred)




def build_all_train_dataset(use_vertical_flip=False, use_horizontal_flip=False,
use_blur=False):

    original_dataset, original_labels = load_dataset(PROCESSED_DIRECTORY_PREFIX
+ TRAIN_DATASET_NAME)

    all_train_dataset = original_dataset

    all_train_labels = original_labels

    if use_vertical_flip:

        vertical_flip_dataset, vertical_flip_labels =
load_dataset(AUGMENTED_VERTICAL_FLIP_DIRECTORY)

        all_train_dataset = np.concatenate((all_train_dataset,
vertical_flip_dataset))

        all_train_labels = np.concatenate((all_train_labels,
vertical_flip_labels))
```

```
        del vertical_flip_dataset

        del vertical_flip_labels

    if use_horizontal_flip:

        horizontal_flip_dataset, horizontal_flip_labels =
load_dataset(AUGMENTED_HORIZONTAL_FLIP_DIRECTORY)

        all_train_dataset = np.concatenate((all_train_dataset,
horizontal_flip_dataset))

        all_train_labels = np.concatenate((all_train_labels,
horizontal_flip_labels))

        del horizontal_flip_dataset

        del horizontal_flip_labels

    if use_blur:

        blur_dataset, blur_labels = load_dataset(AUGMENTED_BLUR_FLIP_DIRECTORY)

        all_train_dataset = np.concatenate((all_train_dataset, blur_dataset))

        all_train_labels = np.concatenate((all_train_labels, blur_labels))

        del blur_dataset

        del blur_labels

    print(all_train_dataset.shape)

    print(all_train_labels.shape)

    batch_remainder = all_train_dataset.shape[0] % TPU_BATCH_SIZE

    all_train_dataset = all_train_dataset[:-batch_remainder]

    all_train_labels = all_train_labels[:-batch_remainder]

    return all_train_dataset, all_train_labels



def load_dataset(directory):

    dataset = []

    labels = []
```

```
    for image_file_name in os.listdir(directory):
        full_file_name = directory + '/' + image_file_name
        dataset.append(io.imread(full_file_name) / 255)
        if "dog" in image_file_name:
            labels.append(1)
        else:
            labels.append(0)
    print(np.array(dataset).shape)
    return np.reshape(np.array(dataset), (len(dataset), 64, 64,
3)).astype('float32'), np.array(labels)



def build_test_dataset():
    dataset = []
    processed_dataset_directory = 'processed_rgb_' + TEST_DATASET_NAME
    for image_file_name in os.listdir(processed_dataset_directory):
        full_file_name = processed_dataset_directory + '/' + image_file_name
        dataset.append(io.imread(full_file_name) / 255)
    return np.reshape(np.array(dataset), (len(dataset), 64, 64,
3)).astype('float32')
```

Здесь, мы сначала приводим все изображения к единому формату - 64x64. Затем, мы создаем 3 дополнительных датасета для увеличения объема информации для обучения. Изменять изображения будем тремя способами:

- Отображать изображение по вертикали
- Отображать изображение по горизонтали
- Добавлять эффект размытия

Также, для ускорения вычислений будет приводить все компоненты цветов к диапазоны 0,1. Для использования TPU к Google Colab дополнительно требуется привести данные к виду float32.

3 Архитектура и результаты для собственной сети

Будем использовать почти такую же сеть, как и в прошлой лабораторной работе:

```
    resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu='grpc://'
+ os.environ['COLAB_TPU_ADDR'])

    tf.config.experimental_connect_to_cluster(resolver)

    tf.tpu.experimental.initialize_tpu_system(resolver)

    strategy = tf.distribute.experimental.TPUStrategy(resolver)

    with strategy.scope():

        model = keras.Sequential()

        model.add(keras.layers.Input(shape=(64, 64, 3)))

        add_conv_layer(model, 48, 0.2)

        add_conv_layer(model, 64, 0.25)

        add_conv_layer(model, 128, 0.3)

        add_conv_layer(model, 160, 0.4)

        add_conv_layer(model, 192, 0.5)

        add_conv_layer(model, 192, 0.5)

        add_conv_layer(model, 192, 0.5)

        add_conv_layer(model, 192, 0.5)

        model.add(keras.layers.Flatten())

        model.add(keras.layers.Dense(512, kernel_initializer='he_normal',
activation="relu"))

        model.add(keras.layers.Dropout(0.5))

        model.add(keras.layers.Dense(512, kernel_initializer='he_normal',
activation="relu"))

        model.add(keras.layers.Dropout(0.5))

        model.add(keras.layers.Dense(1, activation='sigmoid'))
```

```
        t = TicToc('learning')

        t.tic()

        model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

        model.fit(train_dataset, train_labels, batch_size=TPU_BATCH_SIZE,
epochs=EPOCHS_NUMBER)

        t.toc()

        print("Elapsed: %f" % (t.elapsed / 60))

        del train_dataset

        del train_labels

        loss, acc = model.evaluate(validation_dataset, validation_labels)

        del validation_dataset

        del validation_labels

        print("Accuracy: %s" % acc)

        test_dataset = build_test_dataset()

        print(test_dataset.shape)

        predicted = model.predict(test_dataset)

        submission = pd.DataFrame({'id': np.array(range(1, len(test_dataset) +
1)), 'label': predicted.ravel()})

        submission.to_csv('submission.csv', index=False)
```

Выходной файл будет содержать вероятности принадлежности изображения животного к собакам. Так как оригинальное соревнование уже закрыто, то результат будем проверять в этом:

https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition

Тестовые и тренировочные данные здесь такие же, как и в оригинальном соревновании, но система оценки - значения логарифмической функции потерь. Покажем пример оценки с помощью kaggle:

```
~/.local/bin/kaggle competitions submit -c dogs-vs-cats-redux-kernels-edition
-f submission_vgg16.csv -m "Second"
```

После свой score можно посмотреть здесь:

https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/leaderboard#score

Для собственной архитектуры сети результаты получились следующие:

Данные	Accuracy (validation, %)	Score
Без augmentation	78.83	7.07994
blur	95.45	9.25319
horizontal flip	94.31	10.71458
vertical flip	90.75	7.95446

4 Готовые сети

В лабораторной работе предлагается использовать и сравнить результаты с какой-нибудь готовой сетью. Я выбрал сеть VGG16, это сеть предназначена для классификации изображений, ее архитектура представлена ниже:

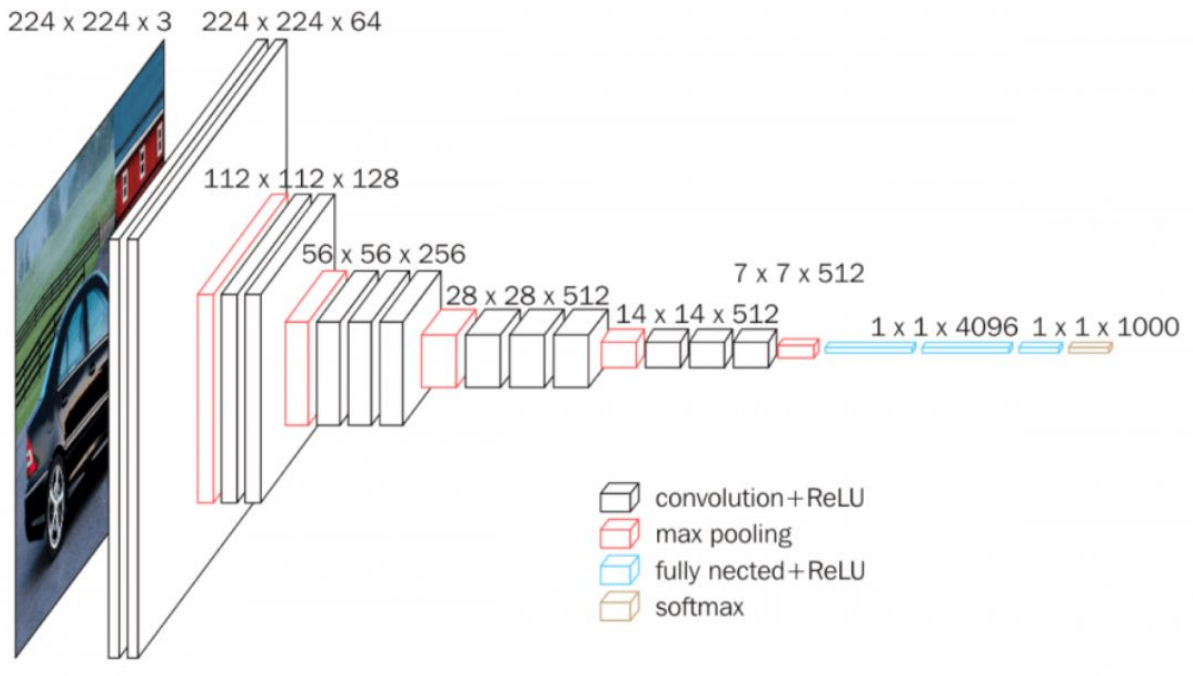

Рисунок 4.1. Архитектура сети VGG16

Мы будем использовать ее без ее последнего softmax слоя с добавлением двух полносвязных слоев по 512 нейронов. Основные варианты обучения: “заморозить” веса VGG16 и обучить свои слои, второй “дообучить” всю сеть. В первом случае не удалось получить accuracy больше 82%, поэтому этот вариант далее не рассматривался. Покажем, как выглядит модель с использованием VGG16 с помощью transfer learning:

```
input_shape=(64, 64, 3)

  vgg = vgg16.VGG16(include_top=False, weights='imagenet',

                    input_shape=input_shape)
```

```
        output = vgg.layers[-1].output

        output = Flatten()(output)

        vgg_model = Model(vgg.input, output)

        model = Sequential()

        model.add(vgg_model)

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

        model.add(Dropout(0.5))

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

        model.add(Dropout(0.5))

        model.add(Dense(1, activation='sigmoid'))

        ...

        model.compile(optimizer=keras.optimizers.RMSprop(lr=1e-5),
loss='binary_crossentropy', metrics=['accuracy'])
```

Покажем результаты обучения данной сети:

Данные	Accuracy (validation, %)	Score
Без augmentation	93.62	9.25319
blur	91.51	13.42928
horizontal flip	95.04	14.67481
vertical flip	91.38	13.09858

Таким образом, лучший accuracy получилось достичь именно с помощью собственной архитектуры сети с blur augmentation (95.45%) а лучший score вообще без augmentation (7.07994) также с собственной архитектурой. Однако видим, что сеть, обученная на основе VGG16 дает значительно лучшие результаты без augmentation. Также стоит заметить, что эта сеть с легкостью обучается до 100% accuracy на тренировочной выборке в течение

500 эпох, поэтому при дальнейшей борьбе с переобучением с ней получился бы результат лучше.