МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по Индивидуальному домашнему заданию

по дисциплине «Искусственные нейронные сети»

Тема: «Stanford Dogs Dataset»

Студент гр. 7383 _____ Корякин М.П.

Студент гр. 7383 _____ Васильев А.И.

Преподаватель _____ Жангиров Т.Р

Санкт-Петербург

2020

Постановка задачи и описание датасета

Данный датасет представляет собой 20000 фотографий собак различных пород, которые создатели из Stanford University позаимствовали из более обширного каталога изображений ImageNet. Пород в датасете 120.

Картинки небольшого размера (притом разного, т.е они не нормализованы по ширине и высоте), 3-х канальные RGB, общий вес датасета ~ 800мб. Часть картинок зашумлена, имеет разную цветовую гамму. Где-то видна только часть животного, животные одной породы могут быть с разным цветом шерсти и в то же время разные породы могут быть очень похожи друг на друга. Помимо этого в датасете довольно много классов (120) для такого количества картинок (20000), что наверняка скажется на точности работы сети. Для сравнения, в CIFAR-10 картинок 60000, а классов всего 10 штук.

Соответственно, предлагается решить задачу многоклассовой классификации набора изображений, чтобы определить к какой конкретно породе относится собака на изображении.

Ссылка на датасет: http://vision.stanford.edu/aditya86/ImageNetDogs/

Начальный анализ и преобразование данных

Датасет имеет следующую структуру:

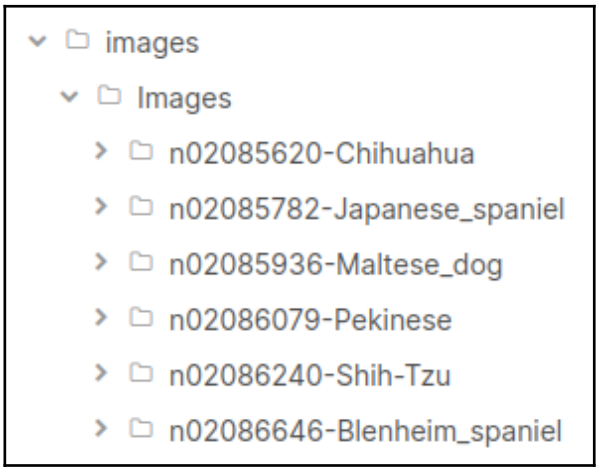

Рисунок 1 - структура исходного датасета

В каждой папке вида <id>-<имя_породы> находятся изображения собак, соответствующие породе в названии родительской директории. Примеры изображений из папки с породой “Родезийский риджбек” (можно обратить внимание, что изображения “из коробки” разного размера):

Рисунок 2 - изображения разного размера из исходного датасета

Метки пород можно получить из названий директорий, то есть при считывании в массив X (массив исх. картинок) очередной папки с N изображениями одной породы, в массив Y (массив меток) добавится N

одинаковых меток.

Проанализируем датасет и определим необходимые для подготовки к обработке сетью действия\преобразования:

1. Необходимо выгрузить датасет с оф сайта и разархивировать его. Напишем для этого функцию:

```
def download(url, dir='./datasets/', name='dogs.tar'):
    if not os.path.isdir(dir):
        os.mkdir(dir)
        os.mkdir(dir + "dogs")
    # Combine the name and the downloads directory to get the local filename
    filename = os.path.join(dir, name)
    # Download the file if it does not exist
    if not os.path.isfile(filename):
        urllib.request.urlretrieve(url, filename)

    my_tar = tarfile.open(filename)
    my_tar.extractall(dir + "dogs")  # specify which folder to extract to
    my_tar.close()
```

Рисунок 3 - функция скачивания и untar-а датасета

2. Необходимо загрузить датасет, а именно нужно выгрузить сами изображения (массив X) и метки (массив Y). Напишем такую функцию:

```
def loadDataset(path='./datasets/dogs/Images/'):
    X = []
    Y = []
    valid_images = [".jpg", ".png", ".jpeg"]
    label = 1
    for dir in os.listdir(path):
        dir_path = path + str(dir) + "/"
        print(dir_path)
        for f in os.listdir(dir_path):
            ext = os.path.splitext(f)[1]
            if ext.lower() not in valid_images:
                continue
            img = np.asarray(Image.open(os.path.join(dir_path, f)))
            X.append(img)
            Y.append(label)
        label = label + 1
    return X, Y
```

Рисунок 4 - функция загрузки датасета

3. Существует проблема, что изображения разного размера и нужно как-то

их привести к единому размеру. Первый самый очевидный способ - взять среднее по размерам изображений всего датасета и сделать ресайз по данному среднему. Однако тогда будут нарушены пропорции у большого количества картинок и появятся сильные искажения. Другой вариант - обрезать картинки со всех сторон по размерам минимальной картинки, но тогда очевидно, что потеряется много данных. Более адекватным кажется объединение этих идей. Для каждой картинки будем производить ресайз с сохранением пропорций, пока её размеры не превысят средние, затем сделаем RandomCrop получившейся картинки до средних размеров. Напишем функции для random crop и обработки всего массива картинок:

```
def standartize(raw_X, mean_w, mean_h):
    X = []
    for img in raw_X:
        w, h = img.size
        if mean_h > h:
            new_x = int(w * mean_h/float(h))
            img = img.resize((new_x, mean_h))
        w, h = img.size
        if mean_w > w:
            new_y = int(h * mean_w/float(w))
            img = img.resize((mean_w, new_y))
        img = randomCrop(img, mean_w, mean_h)
        X.append(np.asarray(img))
    return X


def randomCrop(img, cropped_w, cropped_h):
    x, y = img.size
    x1 = np.random.randint(x - cropped_w+1)
    y1 = np.random.randint(y - cropped_h+1)
    return img.crop((x1, y1, x1 + cropped_w, y1 + cropped_h))
```

Рисунок 5 - функции *randomCrop* для обрезания изображения и *standartize* для обработки всего массива с помощью resize и random_crop операций

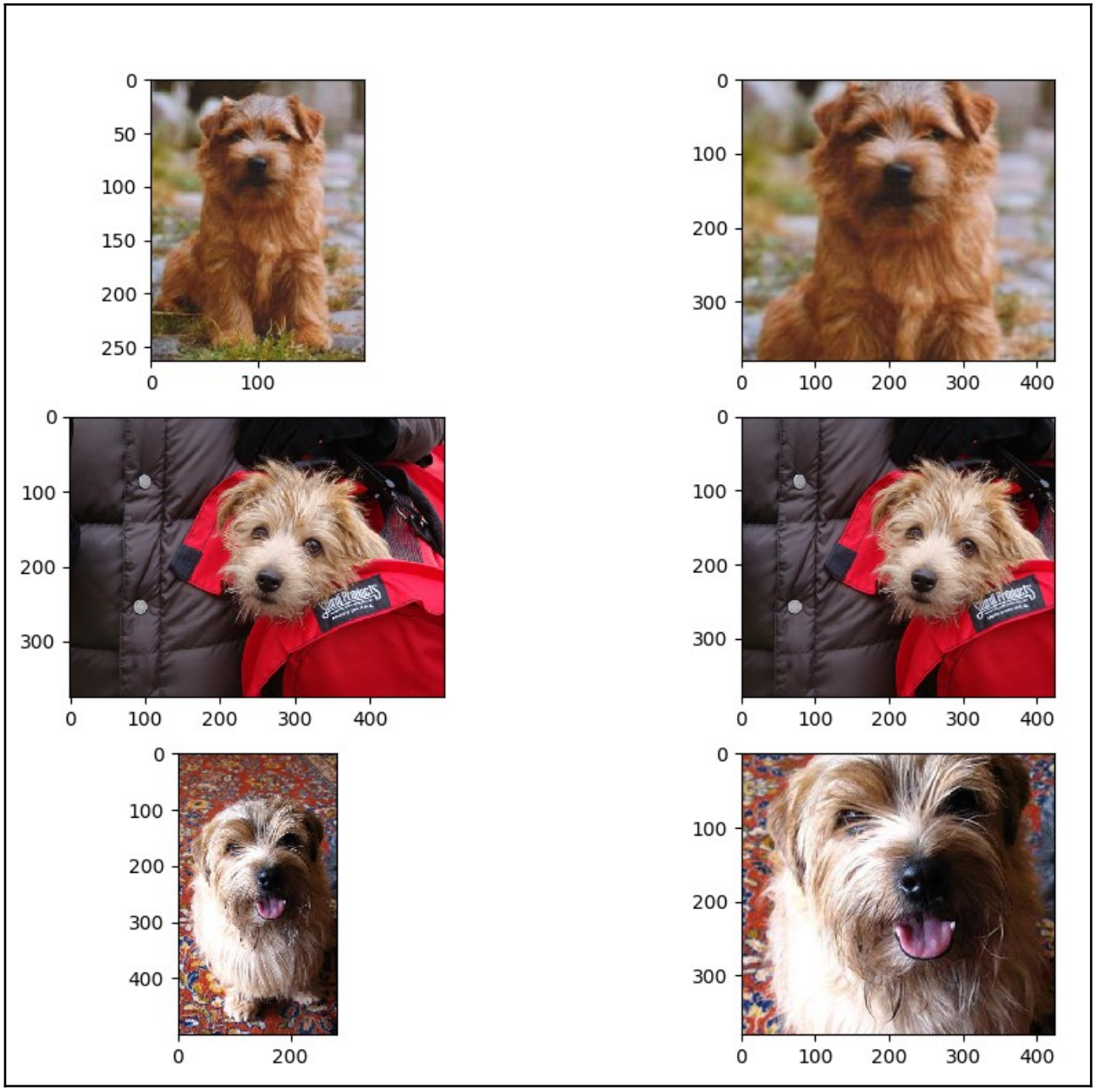

Рисунок 6 - результат работы функции *standartize* на первых 3 изображениях, слева - **до**, справа - **после**

Результат можно считать хорошим, информации теряется не так много, зато получается более менее корректно стандартизировать “сложные” изображения, такие как первое и последнее.

4. Осталось нормализовать картинки по значению пикселей, преобразовать массив меток к виду 0-1 векторов и разделить датасет на тренировочную\тестовую выборки. Код:

```
# normalize pics, categorize labels and split data
X = X / 255.0
Y = to_categorical(Y)

(train_X, test_X, train_Y, test_Y) = train_test_split(X, Y, test_size=0.1, random_state=8888)
```

Рисунок 7 - код нормализации данных

Разработка нейросети-классификатора

Исходная задача - задача многоклассовой классификации, из этого и будем исходить при выборе архитектуры сети и построении модели. С задачей классификации изображений хорошо справляются сверточные нейронные сети.

Наша модель будет последовательной т.е Sequential, содержащей блоки (свертка, пулинг) и одним\несколькими полносвязными слоями на выходе. Это распространенная структура, по подобному принципу построено, например семейство моделей VGG.

Метрика точности для задачи классификации подойдет *Accuracy,* функцию потерь выберем *CategoricalCrossentropy*, она наиболее часто используется в таких задачах. Оптимизатор выберем *Adam,* как наиболее универсальный для подобных задач, learn rate для начальной модели установим немного ниже дефолтного (который указан в документации keras как параметр по умолчанию).

Также, чтобы упростить проектирование модели будем работать не на всём датасете из 120 классов, а возьмем часть, скажем, 8 классов.

Итерация 1

Возьмем в качестве прототипа простую модель CNN с двумя “блоками” (свёртка, пулинг) и одним полносвязным слоем в конце с функцией активации *Softmax* для классификации. Размер фильтров в свертке установим в 3x3. Код:

```
# model initialization
model = Sequential([
    Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    MaxPool2D(pool_size=2),

    Conv2D(filters=16, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    MaxPool2D(pool_size=2),

    Flatten(),
    Dense(units=num_classes, activation='softmax'),
])
```

Рисунок 8 - код инициализации модели

Количество эпох выставим 5, batch_size также 5 и обучим сеть:

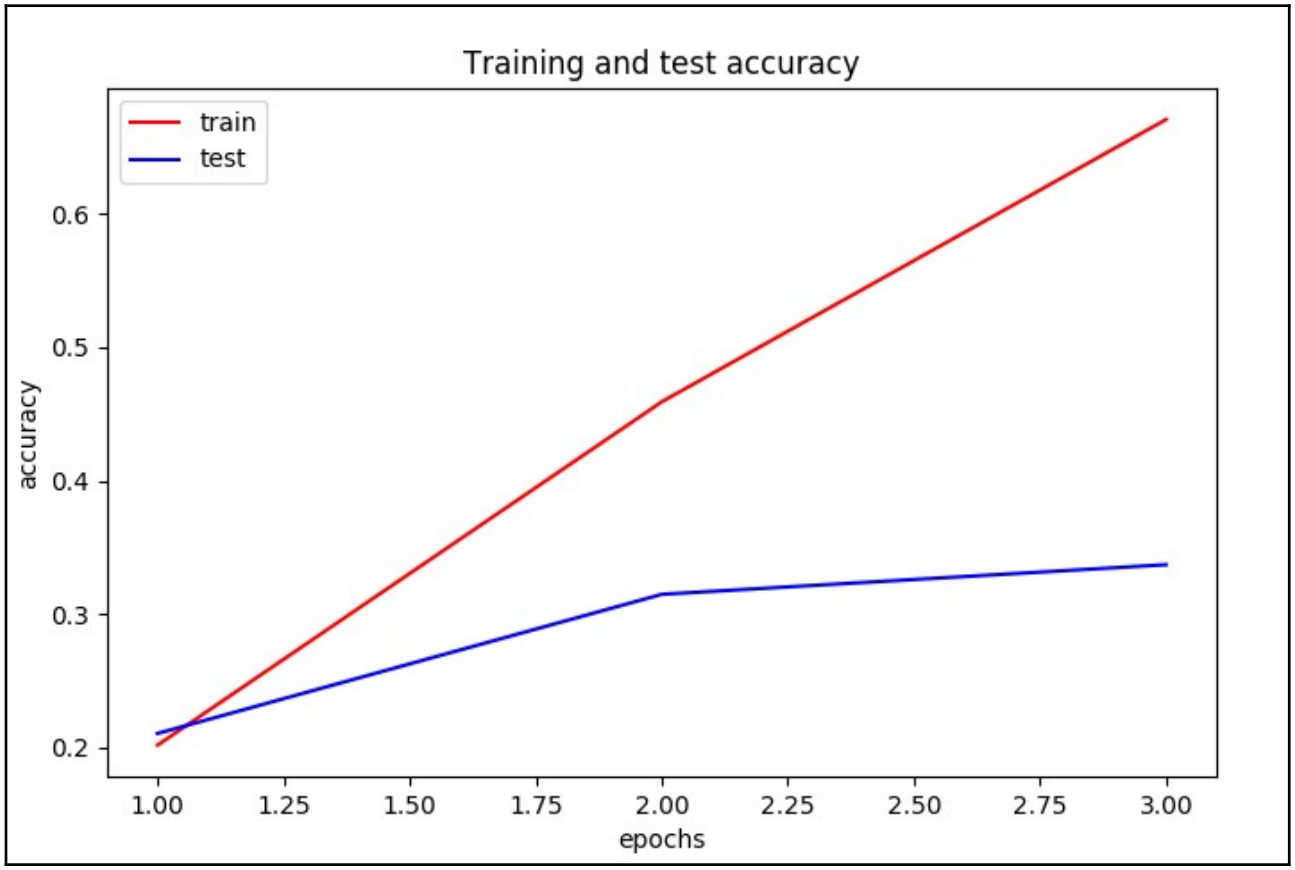

Рисунок 9 - график accuracy на тестовой выборке и тренировочной на 1 итерации

Точность на тестовой выборке - 30%.

Как можно заметить по графику выше - сеть сильно переобучена, необходимо разобраться почему.

Итерация 2

Причина может быть в том, что данных (~1500) слишком мало для 8 классов, поэтому сеть слишком быстро привыкает к одним и тем же картинкам, а когда ей дают что-то непохожее - начинает часто ошибаться. Выход из ситуации - аугментация датасета. Это процедура намеренного искажения изображений путём отзеркаливания по X или Y, поворотов и т.п. В keras есть инструмент для аугментации - ImageDataGenerator, воспользуемся им:

```
aug_generator = ImageDataGenerator(
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.15,
        height_shift_range=0.15,
        horizontal_flip=True,
        vertical_flip=False)
```

Рисунок 10 - ImageDataGenerator

Опции генератора имеют говорящее название, поворот, зум, сдвиги по w;h, флипы по обеим осям.

Обучим сеть на 10 эпохах:

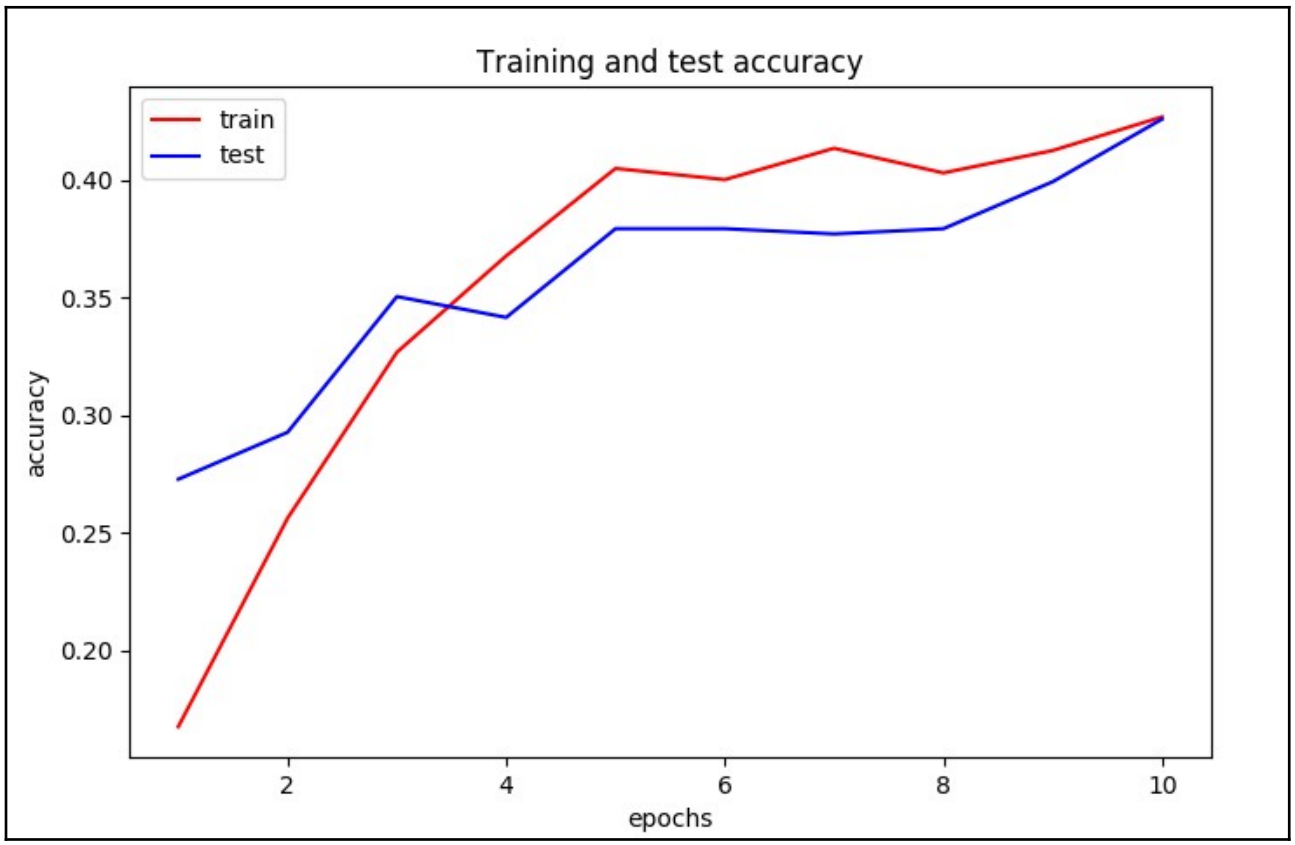
Training and test accuracy
train
test
accuracy
0.40
0.35
0.30
0.25
0.20
2
4
6
8
10
epochs

Рисунок 11 - график accuracy на тестовой выборке и тренировочной на 2 итерации

Данный результат уже существенно лучше предыдущего, точность составила 43%. Теперь, сеть недообучена.

Итерация 3

Чтобы убрать недообучение сети увеличим кол-во эпох до 30, а также, чтобы ускорить и улучшить обучения добавим в сеть слои BatchNormalization:

```
model = Sequential([
    Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    BatchNormalization(),
    MaxPool2D(pool_size=2),

    Conv2D(filters=16, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    BatchNormalization(),
    MaxPool2D(pool_size=2),

    Flatten(),
    Dense(units=num_classes, activation='softmax'),
])
```

Рисунок 12 - код модели на 3 итерации

Обучим сеть:

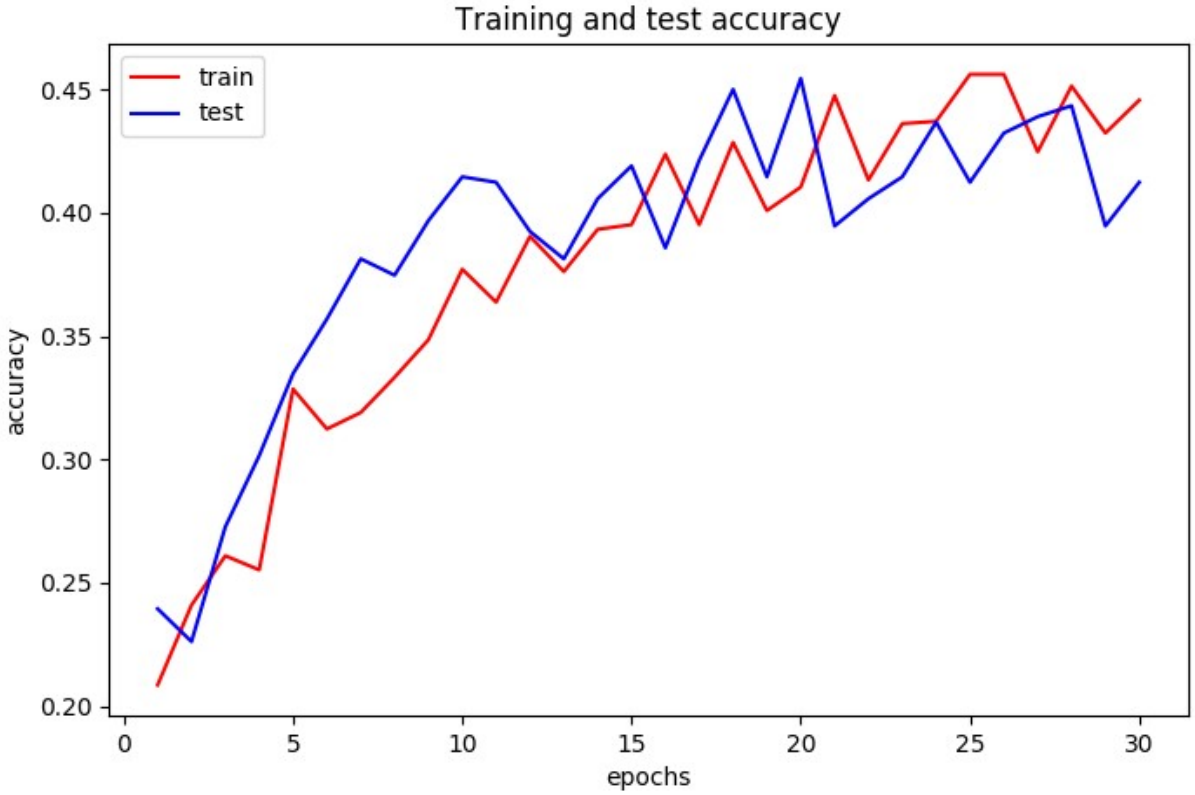

Рисунок 13 - график accuracy на тестовой выборке и тренировочной на 3 итерации

Модель показала аналогичную предыдущей точность в ~ 43 %.

Проанализировав график можно сделать вывод, что данная модель слишком простая для того, чтобы эффективнее различать изображения и 43-45% точности - её предел. Таким образом возникает необходимость в расширении модели, чтобы повысить точность классификации.

Итерация 4

Для увеличения сложности сети добавим в нее ещё одну свертку:

```
model = Sequential([
    Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    BatchNormalization(),
    MaxPool2D(pool_size=2),

    Conv2D(filters=16, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    BatchNormalization(),
    MaxPool2D(pool_size=2),

    Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    BatchNormalization(),
    MaxPool2D(pool_size=2),

    Flatten(),
    Dense(units=num_classes, activation='softmax'),
])
```

Рисунок 14 - код модели на 4 итерации

Количество эпох повысим до 40.

Обучим сеть:

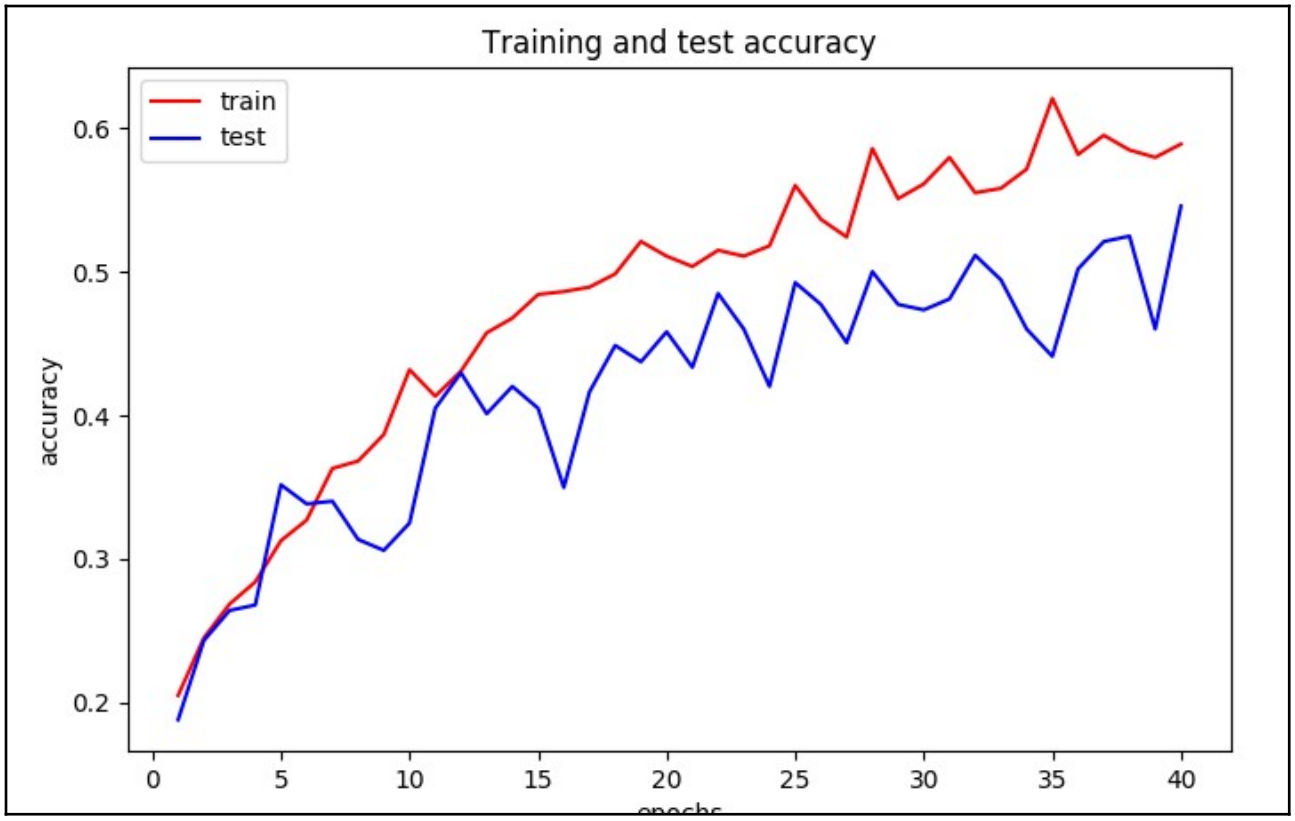

Рисунок 15 - график accuracy на тестовой выборке и тренировочной на 4 итерации

Точность на тестовой выборке составила 53%, что уже довольно неплохо для такого небольшого (по количеству изображений на один класс) датасета, но все еще недостаточно. Несмотря на переобучение, судя по в целом нарастающей точности, у сети ещё есть запас для увеличения точности путем усложнения.

Итерация 5

Расширим сеть еще на одну свертку с 64 фильтрами:

```
model = Sequential([
    Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    BatchNormalization(),
    MaxPool2D(pool_size=2),

    Conv2D(filters=16, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    BatchNormalization(),
    MaxPool2D(pool_size=2),

    Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    BatchNormalization(),
    MaxPool2D(pool_size=2),

    Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), activation='relu'),
    BatchNormalization(),
    MaxPool2D(pool_size=2),

    Flatten(),
    Dense(units=num_classes, activation='softmax'),
])
```

Рисунок 16 - код модели на 5 итерации

Также, настроим ModelCheckpoint для того, чтобы сохранить лучшую модель во время обучения, увеличим количество эпох до 65. Остальные параметры обучения оставим прежними.

Обучим сеть:

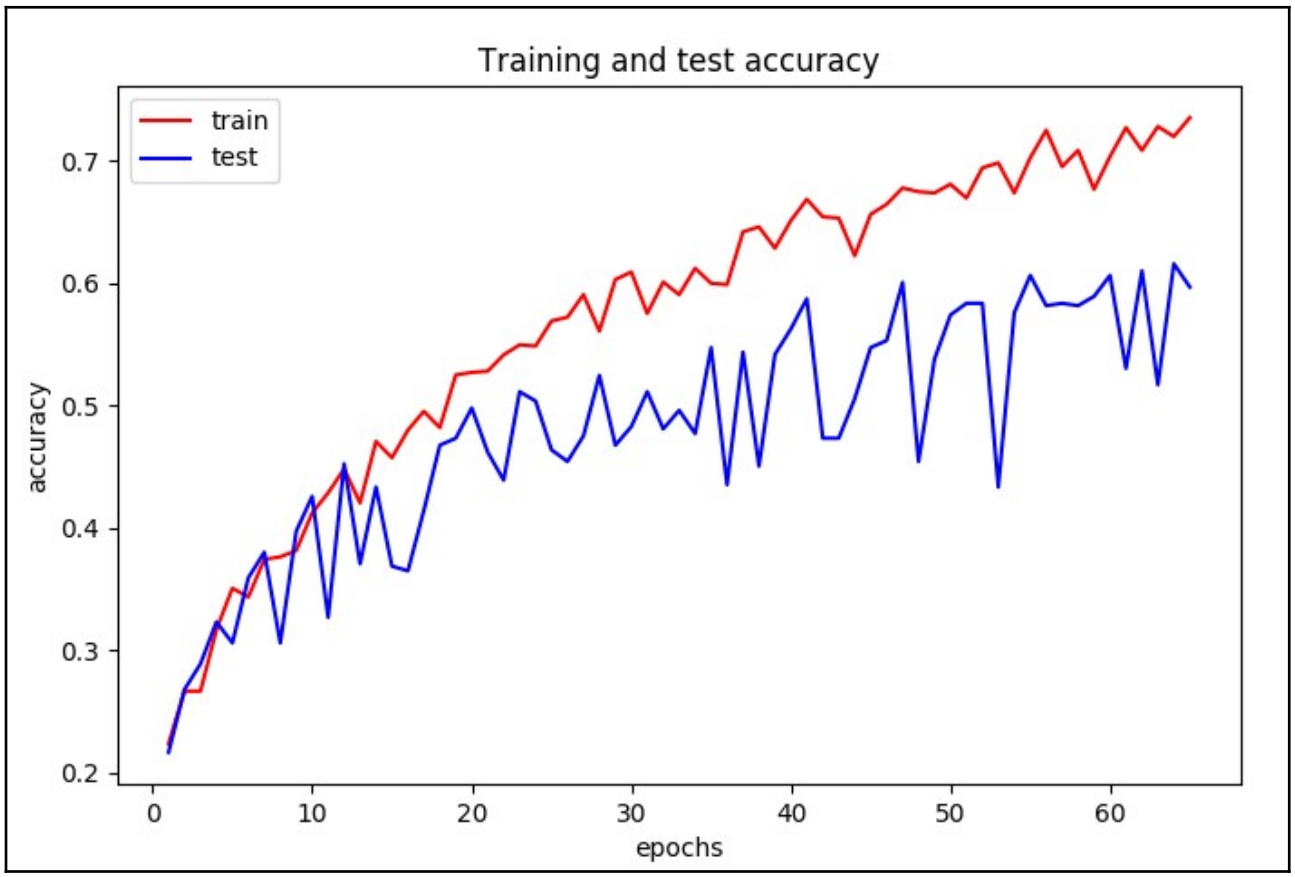

Рисунок 17 - график accuracy на тестовой выборке и тренировочной на 5 итерации

Точность лучшей модели составила 62,6 % на тестовой выборке. Такой результат уже можно считать приемлемым для данного датасета.

Результирующей моделью можно считать эту.

Анализ результирующей модели

Задача классификации для 8 классов была решена с достаточной (для данного датасета) точностью (62,6%) последней моделью (итерация 5). На каждой итерации разработки возникали разные проблемы, среди них:

- Проблема очень быстрого расхождения точности на тестовых и тренировочных данных (30% за 3 эпохи), сеть обучалась сильно переобучилась. Решено было проведением предварительных манипуляций с датасетом (зум, повороты, горизонтальные флипы и т.п)

- Проблема недообучения сети. На итерации 3 выяснилось, что текущая сеть достигла потолка в своей точности классификации изображений. Решено было расширением модели за на несколько уровней свёртки, увеличением количества эпох. Также, для ускорения обучения в структуру сети были встроены BatchNormalization слои.

Помимо проблем, возникших во время проектирования сети были вопросы связанные с подготовкой датасета. В частности, исходный датасет содержал картинки разного размера, что означает невозможность их использования в таком виде нейронной сетью. Решено было стандартизацией картинок под один размер.

Предложения по улучшению модели

Если решать задачу классификации для всего датасета на 120 классов с помощью результирующей модели, то произойдет ожидаемая потеря точности, поскольку признаков, которые нужно запомнить стало больше и сеть начнет чаще ошибаться.

Чтобы этого избежать можно дополнительно расширить модель с помощью новых “уровней свёртки”, добавить, по аналогии с VGG, дополнительные Dense слои в конце модели, таким образом подбирать архитектуру пока точность не восстановится.

Если возникнет необходимость в более высокой точности, то потребуется снизить переобучение, например с помощью L2-регуляризации. Также, могут помочь другие предварительные манипуляции с датасетом.

Выводы

В данной работе была решена задача многоклассовой классификации

изображений. В качестве датасета выступал Stanford Dogs Dataset, выборка картинок из большего датасета ImageNet, изображения разбиты по породам. Задача решена с помощью свёрточной нейронной сети, решены проблемы, связанные с подготовкой датасета (разные размеры картинок, а также малое количество картинок на один класс), проектированием сети (большой разброс при переобучении и недообучение).

Точность предсказаний сети в ходе работы оценивалась метрикой **Accuracy,** лучший результат из представленных выше - 62,6 %, что хороший результат в условиях нехватки изображений на класс.