Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

Курсовой проект

«Распознавание морских животных на изображениях»
по дисциплине
«Глубокое обучение»

Выполнили
студенты гр. 5130904/00101 Кривенко М.Д.
Прудников Д. А.

Руководитель
ст. преподаватель Малеев О.Г.

Санкт-Петербург
2024 г.

Оглавление

Описание задачи

Задача заключается в создании, обучении и тестировании нейронной сети, которая должна быть способна распознавать морских животных на изображениях.

В работе использован датасет морских животных https://www.kaggle.com/datasets/vencerlanz09/sea-animals-image-dataste.
Примеры изображений из сета:

Ход работы

Сверточная нейронная сеть

Первым шагом является загрузка датасета и разделение датасета на train, val и test датасеты:

```
train_path = "/content/train"
val_path = "/content/val"
test_path = "/content/test"
```

Соотношение между сетами примерно равно 0.8 : 0.1 : 0.1 на train, val и test сеты соответственно:

```
split_size = 0.8
val_size = 0.1
train_len = int(len(images) * split_size)
val_len = train_len + int(len(images) * val_size)
train_images = images[:train_len]
val_images = images[train_len:val_len]
test_images = images[val_len:]
```

Получилось три сета данных суммарно на примерно 3000 изображений:

```
Found 2444 images belonging to 6 classes.
Found 302 images belonging to 6 classes.
Found 312 images belonging to 6 classes.
```

Для предотвращения переобучения используется аугментация данных при помощи ImageDataGenerator. Изображение случайным образом может быть повернуто в пределах 10 градусов, приближено в пределах 10%, сдвинуто по ширине или высоте в пределах 20%, а также отражено горизонтально:

```
datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        zoom_range=0.1,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True
)
```

Далее создается модель Sequential для создания сети с последовательным соединением слоев друг за другом :

```
tf.keras.backend.clear_session()
model = Sequential()
model.add(Conv2D(32, 3, activation ='relu', padding='same', input_shape = (IMG_SIZE,IMG_SIZE,3)))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64, 3, padding='same',activation ='relu'))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(128, 3, padding='same',activation ='relu'))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(256, 3, padding='same',activation ='relu'))
model.add(MaxPooling2D((2,2)))

model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(1024, activation = 'relu'))
model.add(Dense(len(labels), activation = "softmax"))

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

Первым слоем идет сверточный слой Conv2D с 32 фильтрами размером 3x3 и функцией активации ReLU. Так как датасеты подготовлены с параметрами (256, 256, 3), для слоя указаны соответствующие параметры входных данных. Значение same параметра padding в данном случае приводит к сохранению размерности данных при проходе через сверточный слой.

Conv2D используется для применения свертки входных данных. Conv2D позволяет извлекать признаки из изображений, что делает их эффективными для обработки изображений и любых других двумерных данных. Conv2D принимает на вход двумерные данные, такие как

изображения, и применяет к ним ядро свертки. Это позволяет находить более высокоуровневые признаки, например границы объектов, что способствует распознаванию объектов.

Из активаторов выбран именно ReLU из-за его относительной простоты в вычислении и при этом достаточной эффективности, что будет проверено на этапе тестирования модели.

После Conv2D находится слой MaxPooling2D для уменьшения размера изображения, что с точки зрения сложности вычислений позволяет использовать следующими слои Conv2D с более большим количеством фильтров.

Аналогично описанная выше комбинация слоев дублируется несколько раз с постепенным увеличением количества фильтров в слое Conv2D до 256.

Затем следует слой Dropout. Аргументом является количество случайно отключаемых нейронов. Засчет случайного отключения нейронов данный слой помогает избежать чрезмерной зависимости от конкретных признаков входных данных, таким образом способствуя развитию возможности модели обрабатывать новые данные

Затем добавляется слой Flatten, преобразующий входные данные в одномерный массив для обработки следующими слоями.

Последующие два слоя Dense сначала с 1024 нейронами и функцией активации ReLU, а затем с количеством нейронов, аналогичным количеству распознаваемых классов, и функцией активации softmax используются для создания предсказания о принадлежности входных данных к классу. Softmax используется для интерпретации результата как распределения вероятности о принадлежности изображения к определяемым классам.

В результате получается данная модель:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 32)      896

 max_pooling2d (MaxPooling2  (None, 128, 128, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 128, 128, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 64, 64, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 64, 64, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 32, 32, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 32, 32, 256)       295168

 max_pooling2d_3 (MaxPoolin  (None, 16, 16, 256)       0
 g2D)

 dropout (Dropout)           (None, 16, 16, 256)       0

 flatten (Flatten)           (None, 65536)             0

 dense (Dense)               (None, 1024)              67109888

 dense_1 (Dense)             (None, 6)                 6150

=================================================================
Total params: 67504454 (257.51 MB)
Trainable params: 67504454 (257.51 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

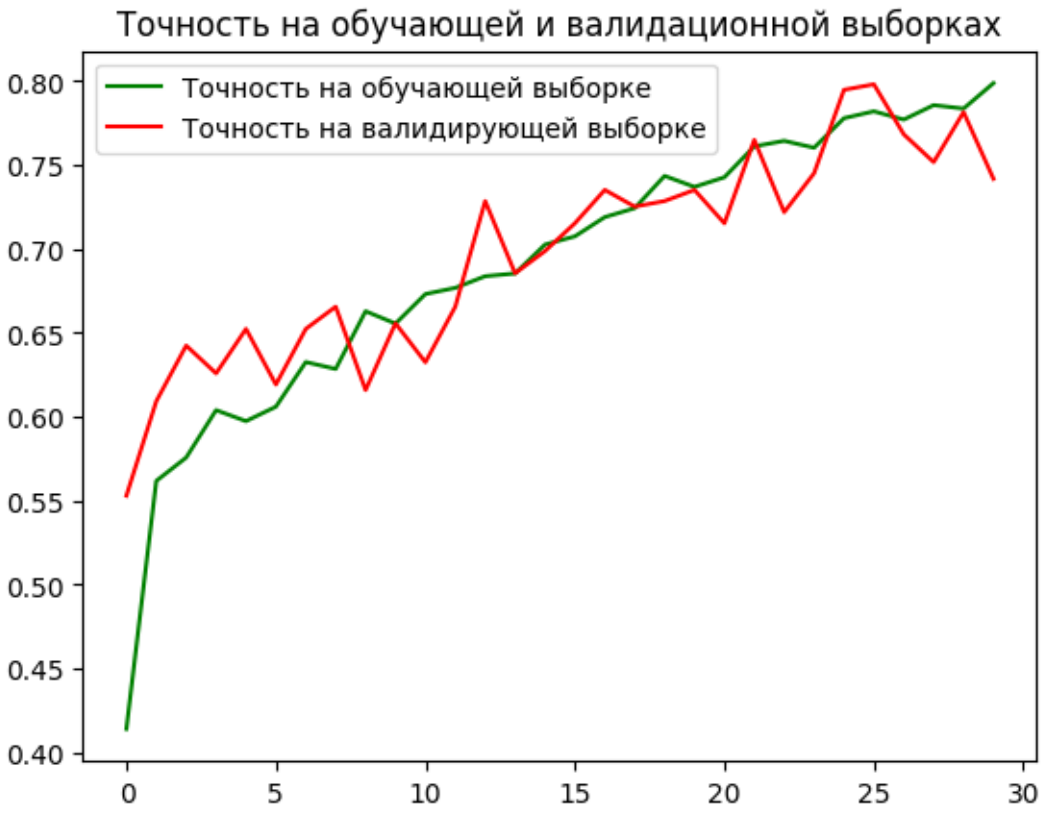
За созданием модели следует ее компиляция и обучение с использованием подготовленных ранее датасетов train и val в течение 30 эпох:

```
Epoch 1/30
77/77 [==============================] - 54s 572ms/step - loss: 1.5064 - accuracy: 0.4141 - val_loss: 1.1192 - val_accuracy: 0.5530
Epoch 2/30
77/77 [==============================] - 44s 565ms/step - loss: 1.1051 - accuracy: 0.5618 - val_loss: 1.0265 - val_accuracy: 0.6093
Epoch 3/30
77/77 [==============================] - 44s 566ms/step - loss: 1.0472 - accuracy: 0.5757 - val_loss: 0.9427 - val_accuracy: 0.6424
Epoch 4/30
77/77 [==============================] - 43s 563ms/step - loss: 1.0056 - accuracy: 0.6039 - val_loss: 0.9970 - val_accuracy: 0.6258
Epoch 5/30
77/77 [==============================] - 43s 564ms/step - loss: 0.9997 - accuracy: 0.5974 - val_loss: 0.9240 - val_accuracy: 0.6523
Epoch 6/30
77/77 [==============================] - 42s 551ms/step - loss: 0.9464 - accuracy: 0.6060 - val_loss: 1.0727 - val_accuracy: 0.6192
Epoch 7/30
77/77 [==============================] - 43s 555ms/step - loss: 0.9195 - accuracy: 0.6326 - val_loss: 0.8705 - val_accuracy: 0.6523
Epoch 8/30
77/77 [==============================] - 42s 541ms/step - loss: 0.9003 - accuracy: 0.6285 - val_loss: 0.8625 - val_accuracy: 0.6656
Epoch 9/30
77/77 [==============================] - 43s 551ms/step - loss: 0.8566 - accuracy: 0.6628 - val_loss: 1.0153 - val_accuracy: 0.6159
Epoch 10/30
77/77 [==============================] - 42s 544ms/step - loss: 0.8507 - accuracy: 0.6555 - val_loss: 0.8129 - val_accuracy: 0.6556
Epoch 11/30
77/77 [==============================] - 43s 552ms/step - loss: 0.8307 - accuracy: 0.6731 - val_loss: 0.9782 - val_accuracy: 0.6325
Epoch 12/30
77/77 [==============================] - 43s 556ms/step - loss: 0.8121 - accuracy: 0.6768 - val_loss: 0.8329 - val_accuracy: 0.6656
Epoch 13/30
77/77 [==============================] - 43s 555ms/step - loss: 0.7908 - accuracy: 0.6837 - val_loss: 0.7404 - val_accuracy: 0.7285
Epoch 14/30
77/77 [==============================] - 43s 558ms/step - loss: 0.7828 - accuracy: 0.6854 - val_loss: 0.8179 - val_accuracy: 0.6854
Epoch 15/30
77/77 [==============================] - 43s 559ms/step - loss: 0.7358 - accuracy: 0.7025 - val_loss: 0.7157 - val_accuracy: 0.6987
```

```
Epoch 16/30
77/77 [==============================] - 43s 562ms/step - loss: 0.7413 - accuracy: 0.7074 - val_loss: 0.8077 - val_accuracy: 0.7152
Epoch 17/30
77/77 [==============================] - 44s 568ms/step - loss: 0.7097 - accuracy: 0.7189 - val_loss: 0.7133 - val_accuracy: 0.7351
Epoch 18/30
77/77 [==============================] - 43s 561ms/step - loss: 0.6883 - accuracy: 0.7242 - val_loss: 0.7157 - val_accuracy: 0.7252
Epoch 19/30
77/77 [==============================] - 44s 565ms/step - loss: 0.6523 - accuracy: 0.7435 - val_loss: 0.6853 - val_accuracy: 0.7285
Epoch 20/30
77/77 [==============================] - 43s 563ms/step - loss: 0.6633 - accuracy: 0.7369 - val_loss: 0.6876 - val_accuracy: 0.7351
Epoch 21/30
77/77 [==============================] - 42s 550ms/step - loss: 0.6558 - accuracy: 0.7426 - val_loss: 0.6795 - val_accuracy: 0.7152
Epoch 22/30
77/77 [==============================] - 42s 543ms/step - loss: 0.6074 - accuracy: 0.7610 - val_loss: 0.6628 - val_accuracy: 0.7649
Epoch 23/30
77/77 [==============================] - 44s 563ms/step - loss: 0.5934 - accuracy: 0.7643 - val_loss: 0.7289 - val_accuracy: 0.7219
Epoch 24/30
77/77 [==============================] - 43s 559ms/step - loss: 0.5911 - accuracy: 0.7602 - val_loss: 0.6437 - val_accuracy: 0.7450
Epoch 25/30
77/77 [==============================] - 43s 559ms/step - loss: 0.5640 - accuracy: 0.7778 - val_loss: 0.5816 - val_accuracy: 0.7947
Epoch 26/30
77/77 [==============================] - 43s 564ms/step - loss: 0.5593 - accuracy: 0.7819 - val_loss: 0.5992 - val_accuracy: 0.7980
Epoch 27/30
77/77 [==============================] - 44s 568ms/step - loss: 0.5545 - accuracy: 0.7770 - val_loss: 0.6411 - val_accuracy: 0.7682
Epoch 28/30
77/77 [==============================] - 43s 563ms/step - loss: 0.5360 - accuracy: 0.7856 - val_loss: 0.6878 - val_accuracy: 0.7517
Epoch 29/30
77/77 [==============================] - 43s 565ms/step - loss: 0.5458 - accuracy: 0.7836 - val_loss: 0.6006 - val_accuracy: 0.7815
Epoch 30/30
77/77 [==============================] - 44s 573ms/step - loss: 0.5098 - accuracy: 0.7987 - val_loss: 0.6651 - val_accuracy: 0.7417
```

Следует отметить, что точность на тренировочном датасете несильно превышает точность на валидационном датасете в течение всего обучения, а в некоторых эпохах точность на валидационном датасете даже превышает точность на тренировочном датасете, например в 26 эпохе. Это показывает достаточность предпринятых против переобучения мер.

Процесс обучения описывается следующими графиками точности и потерь:



Финальная оценка точности модели производится с использованием тестового датасета:

```
_, accuracy = model.evaluate(test_ds)
print("Test accuracy:", accuracy)
```

```
10/10 [==============================] - 2s 170ms/step - loss: 0.6280 - accuracy: 0.7756
Test accuracy: 0.7756410241127014
```

В результате полученной cnn моделью достигнута точность 77.5%. Данный результат можно превзойти с помощью предобученных моделей.

Предобученная модель

Для данной работы выбрана модель ResNet50V2. Предобучение на датасете imagenet из более чем 14 миллионов подписанных изображений должно положительно сказаться на способности нейронной сети классифицировать изображения морских животных.

Рассмотрим процесс создания модели:

```
base_model = ResNet50V2(include_top=False, weights='imagenet', input_shape=(IMG_SIZE,IMG_SIZE,3))
base_model.trainable = False
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(len(labels), activation='softmax')
])
model.summary()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Сначала загружается предобученная модель ResNet50V2 с указанием весов, полученных при обучении на датасете imagenet. В встроенном внутрь модели слой Dense нет необходимости, так как в дальнейшем мы используем свои слои Dense под специфику поставленной задачи, поэтому аргумент include_top передается со значением False. Параметры размеры изображения остаются неизменными из процесса создания сверточной сети. Также стоит отметить отключение обучения предобученной модели – в процессе обучения будут обучаться только необученные слои Dense. Пересечение процессов обучения предобученных слоев ResNet50V2 и необученных слоев может привести к ухудшению эффективности работы предобученных слоев.

В итоговой модели после ResNet50V2 следует слой GlobalAveragePooling2D, необходимый для конвертации формата выходных данных из предобученной модели в двумерный.

Затем добавляются слой Dropout для предотвращения переобучения и слои Dense для непосредственно предсказания о пренадлежности изображения к классу. Предназначение данных слоев аналогично уже рассмотренным выше в процессе создания сверточной нейронной сети.

Итоговая модель выглядит следующим образом:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50v2 (Functional)     (None, 8, 8, 2048)        23564800

 global_average_pooling2d_4  (None, 2048)              0
  (GlobalAveragePooling2D)

 dense_12 (Dense)            (None, 256)               524544

 dropout_6 (Dropout)         (None, 256)               0

 dense_13 (Dense)            (None, 6)                 1542

=================================================================
Total params: 24090886 (91.90 MB)
Trainable params: 526086 (2.01 MB)
Non-trainable params: 23564800 (89.89 MB)
_________________________________________________________________
```

После создания модель проходит обучение в течение 10 эпох:

```
model.fit(train_ds, epochs=10, validation_data=val_ds)
```

```
Epoch 1/10
77/77 [==============================] - 49s 585ms/step - loss: 0.3630 - accuracy: 0.8768 - val_loss: 0.2464 - val_accuracy: 0.9338
Epoch 2/10
77/77 [==============================] - 49s 639ms/step - loss: 0.1699 - accuracy: 0.9349 - val_loss: 0.2297 - val_accuracy: 0.9305
Epoch 3/10
77/77 [==============================] - 45s 583ms/step - loss: 0.1413 - accuracy: 0.9472 - val_loss: 0.1801 - val_accuracy: 0.9437
Epoch 4/10
77/77 [==============================] - 46s 592ms/step - loss: 0.1016 - accuracy: 0.9595 - val_loss: 0.1304 - val_accuracy: 0.9536
Epoch 5/10
77/77 [==============================] - 45s 586ms/step - loss: 0.0931 - accuracy: 0.9587 - val_loss: 0.1494 - val_accuracy: 0.9570
Epoch 6/10
77/77 [==============================] - 45s 581ms/step - loss: 0.0887 - accuracy: 0.9648 - val_loss: 0.1638 - val_accuracy: 0.9603
Epoch 7/10
77/77 [==============================] - 44s 566ms/step - loss: 0.0648 - accuracy: 0.9738 - val_loss: 0.1144 - val_accuracy: 0.9636
Epoch 8/10
77/77 [==============================] - 45s 579ms/step - loss: 0.0789 - accuracy: 0.9742 - val_loss: 0.1298 - val_accuracy: 0.9669
Epoch 9/10
77/77 [==============================] - 45s 582ms/step - loss: 0.0629 - accuracy: 0.9771 - val_loss: 0.1350 - val_accuracy: 0.9669
Epoch 10/10
77/77 [==============================] - 45s 581ms/step - loss: 0.0568 - accuracy: 0.9775 - val_loss: 0.1236 - val_accuracy: 0.9603
```

Оценим эффективность работы модели на тестовом датасете:

```
_, accuracy = model.evaluate(test_ds)
print("Test accuracy:", accuracy)
```

```
10/10 [==============================] - 4s 373ms/step - loss: 0.1349 - accuracy: 0.9359
Test accuracy: 0.9358974099159241
```

Точность на тестовом датасете составила 93.5%, что значительно превышает результат полученной ранее сверточной нейронной сети без использования предобученной модели.

Вывод

В данной работе была решена задача классификации морских животных с использованием двух способов. Первым способом является сверточная нейронная сеть без использования предобученной модели. Данный вариант смог достичь точности в 77.5%. Вторым способом является сеть с использованием предобученной модели. Этот способ оказался более эффективным и за меньшее количество эпох смог достичь точности в 93.5%. Также на обучение второй сети понадобилось меньше времени с учетом использования тех же вычислительных ресурсов, что и для первой сети. Таким образом можно сделать вывод, что использование предобученной модели в задаче классификации морских животных является более эффективным подходом.