Распознавание породы кошек(ML)

Проект включал в себя создание модели машинного обучения для определения породы кошки на основе предоставленных изображений. Основные шаги, которые были успешно пройдены:

1. **Сбор данных:**
    - Собрать изображения кошек разных пород.
2. **Разделение данных:**
    - Разделить данные на обучающий, проверочный и тестовый наборы для обучения.
3. **Подготовка изображений:**
    - Преобразовать изображения в формат, подходящий для обучения (например, изменение размера, нормализация).
4. **Создание модели:**
    - Построить модель машинного обучения для классификации пород кошек.
5. **Обучение модели:**
    - Обучить модель на обучающем наборе и проверить на валидационном наборе.
6. **Оценка модели:**
    - Оценить производительность модели на тестовом наборе.
7. **Сохранение модели:**
    - Сохранить обученную модель для последующего использования без повторного обучения.
8. **Предсказание на новых данных:**
    - Загрузить сохраненную модель и использовать ее для предсказания породы кошки на новых изображениях.

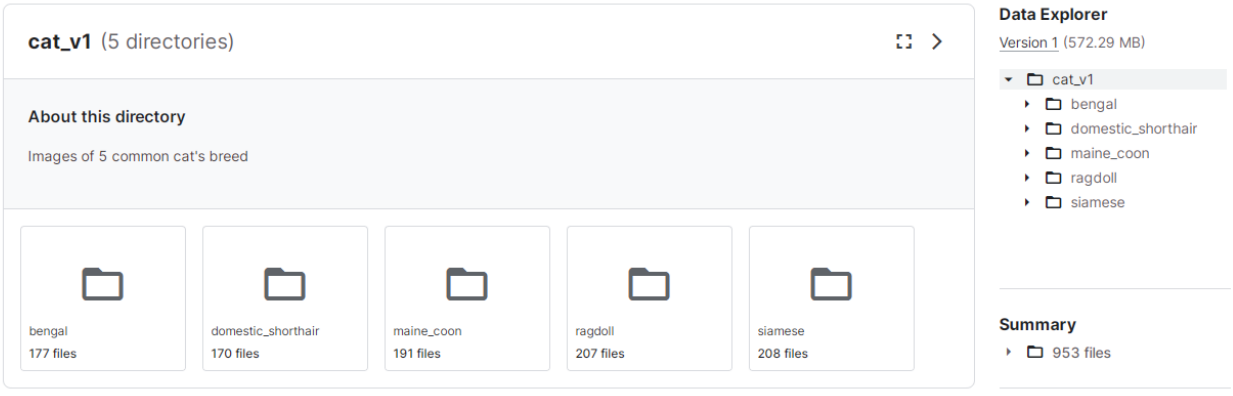
Подробно:

1. Сбор данных:

Данные для обучения модели были взяты из открытых источников(kaggle.com)
Датасет: https://www.kaggle.com/datasets/yapwh1208/cats-breed-dataset?resource=download

Датасет состоял из фотографий пяти различных пород кошек, а именно:

- бенгальская(bengal) - 2362 шт.
- мэйн кун(maine coon) - 1297 шт.
- рэггдолл(ragdoll) - 2511 шт.
- сиамская(siamese) - 2534 шт.
- tortoishell - 3798 шт.



2. Разделение данных

Данные были разделены на три группы: обучающий, проверочный(валидационный) и тестовый.
|-- dataset
|-- train
|-- breed_1
|-- breed_2
|-- ...
|-- validation
|-- breed_1
|-- breed_2
|-- ...
|-- test
|-- breed_1
|-- breed_2
|-- ...

3. Подготовка изображений

На данном шаге была проведена нормализация значений пикселей и приведение изображений к единому формату и размеру.

Первое: нам необходимо гарантировать, чтобы значения пикселей находились в удобном для нас диапазоне для обучения модели. Для этого мы разделили каждый пиксель на 255.
Таким образом, все наши пиксели лежат в диапазоне $[0, 1]$.

Второе: каждое изображение было приведено к единому формату, а именно к 224x224

Далее обработанные изображения были сохранены как массивы в файлах: train_data.npy, test_data.npy, validation_data.npy.

4. Создание модели

Чтобы создать модель для классификации породы кошек была использована библиотека TensorFlow и Keras. Я воспользовался предварительно обученной моделью VGG16.

Анализ результатов

Каким образом происходил сбор данных:
70 % - обучающих
15 % - тестовых
15 % - проверочных

Количество данных: 2250

Модель

`base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))`

```
custom_model = Sequential([
        base_model,
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(5, activation='softmax')  # 5 - количество классов (пород кошек)

    ])
```

Полученные результаты

> 50/50 [==============================] - 343s 7s/step - loss: 1.6124 - accuracy: 0.3130 - val_loss: 1.1369 - val_accuracy: 0.5765
> Epoch 2/10
> 50/50 [==============================] - 353s 7s/step - loss: 1.1076 - accuracy: 0.5695 - val_loss: 0.9458 - val_accuracy: 0.6118
> Epoch 3/10
> 50/50 [==============================] - 355s 7s/step - loss: 0.9249 - accuracy: 0.6438 - val_loss: 0.8904 - val_accuracy: 0.6647
> Epoch 4/10
> 50/50 [==============================] - 354s 7s/step - loss: 0.7643 - accuracy: 0.7048 - val_loss: 0.8407 - val_accuracy: 0.6882
> Epoch 5/10
> 50/50 [==============================] - 352s 7s/step - loss: 0.6391 -

```
accuracy: 0.7695 - val_loss: 0.8816 - val_accuracy: 0.6618
Epoch 6/10
50/50 [----------] - 354s 7s/step - loss: 0.5276 -
accuracy: 0.7987 - val_loss: 0.9365 - val_accuracy: 0.6353
Epoch 7/10
50/50 [----------] - 355s 7s/step - loss: 0.5137 -
accuracy: 0.8057 - val_loss: 1.0016 - val_accuracy: 0.6294
11/11 [----------] - 60s 6s/step - loss: 1.0301 -
accuracy: 0.5881
Test Accuracy: 58.81%
```

Исходя из полученных результатов, можно сделать вывод о том, что модель достигла максимальной точности на обучающем наборе данных в пределах 80 %, но на проверочном и тестовом наборах точность значительно ниже.

Проверочная точность до 4 эпохи растет, затем падает(переобучение)

Добавляем такой параметр, как BatchNormalization().

```
50/50 [----------] - 353s 7s/step - loss: 1.4904 -
accuracy: 0.4565 - val_loss: 1.3150 - val_accuracy: 0.5412
Epoch 2/10
50/50 [----------] - 345s 7s/step - loss: 0.8321 -
accuracy: 0.6813 - val_loss: 1.2049 - val_accuracy: 0.5147
Epoch 3/10
50/50 [----------] - 348s 7s/step - loss: 0.6076 -
accuracy: 0.7714 - val_loss: 0.9605 - val_accuracy: 0.6294
Epoch 4/10
50/50 [----------] - 343s 7s/step - loss: 0.4103 -
accuracy: 0.8571 - val_loss: 1.2037 - val_accuracy: 0.6000
Epoch 5/10
50/50 [----------] - 345s 7s/step - loss: 0.2660 -
accuracy: 0.9092 - val_loss: 1.4788 - val_accuracy: 0.5382
Epoch 6/10
50/50 [----------] - 344s 7s/step - loss: 0.1935 -
accuracy: 0.9397 - val_loss: 1.3233 - val_accuracy: 0.5853
11/11 [----------] - 61s 6s/step - loss: 1.0204 -
accuracy: 0.6388
Test Accuracy: 63.88%
```

Использование оптимизатора Nadam:

```
Nadam(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
```

> 50/50 [----------] - 364s 7s/step - loss: 1.4072 - accuracy: 0.4616 - val_loss: 1.3936 - val_accuracy: 0.4853
> Epoch 2/10
> 50/50 [----------] - 355s 7s/step - loss: 0.8330 - accuracy: 0.6927 - val_loss: 1.3595 - val_accuracy: 0.4971
> Epoch 3/10
> 50/50 [----------] - 356s 7s/step - loss: 0.5584 - accuracy: 0.8019 - val_loss: 1.1208 - val_accuracy: 0.6118
> Epoch 4/10
> 50/50 [----------] - 357s 7s/step - loss: 0.3912 - accuracy: 0.8610 - val_loss: 1.4338 - val_accuracy: 0.5588
> Epoch 5/10
> 50/50 [----------] - 360s 7s/step - loss: 0.2907 - accuracy: 0.9016 - val_loss: 2.0664 - val_accuracy: 0.5588
> Epoch 6/10
> 50/50 [----------] - 360s 7s/step - loss: 0.1884 - accuracy: 0.9410 - val_loss: 1.7902 - val_accuracy: 0.5941
> 11/11 [----------] - 61s 6s/step - loss: 1.2226 - accuracy: 0.6090
> Test Accuracy: 60.90%

Попробуем использовать другую предварительно обученную модель, такую как EfficientNet.

> 50/50 [----------] - 54s 946ms/step - loss: 1.7094 - accuracy: 0.2114 - val_loss: 1.6949 - val_accuracy: 0.2000
> Epoch 2/10
> 50/50 [----------] - 42s 840ms/step - loss: 1.6325 - accuracy: 0.1924 - val_loss: 1.6532 - val_accuracy: 0.2000
> Epoch 3/10
> 50/50 [----------] - 42s 846ms/step - loss: 1.6143 - accuracy: 0.1924 - val_loss: 1.6397 - val_accuracy: 0.2000
> Epoch 4/10
> 50/50 [----------] - 43s 865ms/step - loss: 1.6179 - accuracy: 0.1962 - val_loss: 1.6173 - val_accuracy: 0.2000
> Epoch 5/10

```
50/50 [----------] - 42s 845ms/step - loss: 1.6158 - accuracy: 0.2019 - val_loss: 1.6165 - val_accuracy: 0.2000
Epoch 6/10
50/50 [----------] - 43s 872ms/step - loss: 1.6195 - accuracy: 0.1879 - val_loss: 1.6206 - val_accuracy: 0.2000
Epoch 7/10
50/50 [----------] - 44s 873ms/step - loss: 1.6174 - accuracy: 0.1917 - val_loss: 1.6111 - val_accuracy: 0.2000
Epoch 8/10
50/50 [----------] - 44s 875ms/step - loss: 1.6116 - accuracy: 0.2127 - val_loss: 1.6112 - val_accuracy: 0.2000
Epoch 9/10
50/50 [----------] - 42s 848ms/step - loss: 1.6098 - accuracy: 0.2019 - val_loss: 1.6094 - val_accuracy: 0.2000
Epoch 10/10
50/50 [----------] - 44s 879ms/step - loss: 1.6103 - accuracy: 0.2038 - val_loss: 1.6095 - val_accuracy: 0.2000
11/11 [----------] - 7s 643ms/step - loss: 1.6095 - accuracy: 0.2000
Test Accuracy: 20.00%
```

Мдаа, оставим без комментариев.

Количество данных: 3000

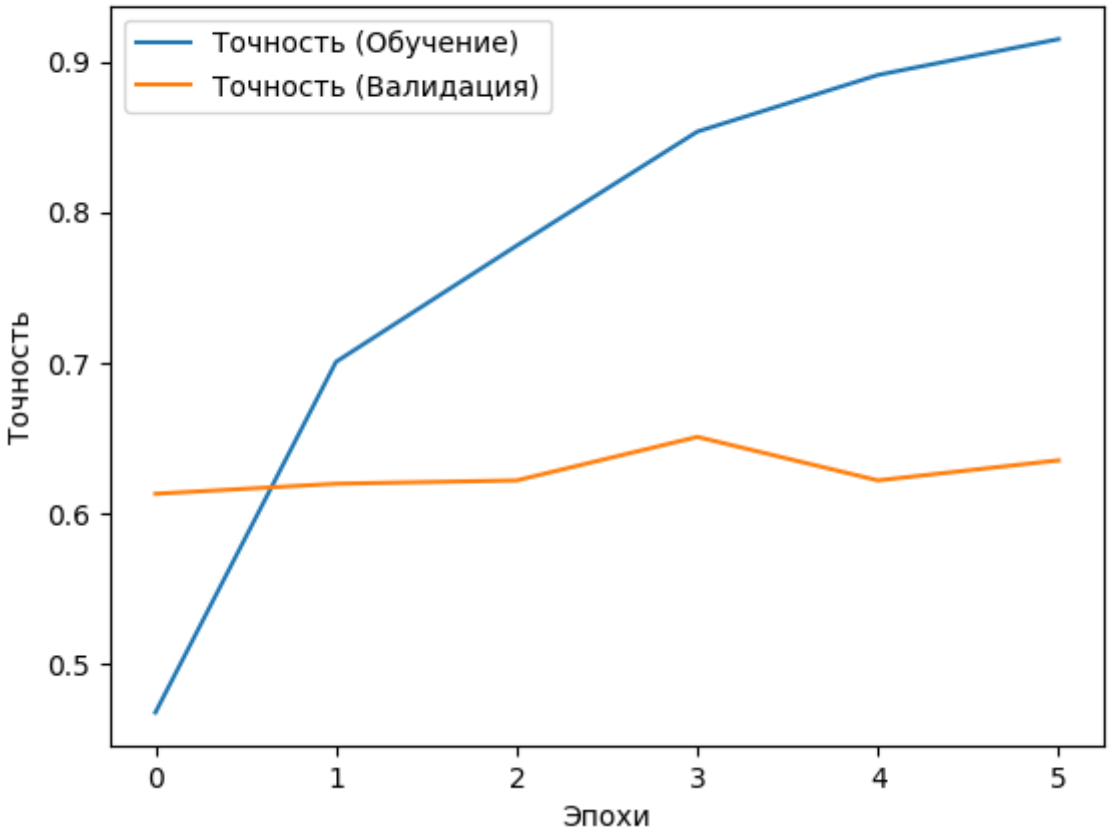
Попробуем увеличить количество входных данных. Пусть теперь для каждой породы будет 600 фотографий. Остальные параметры оставить

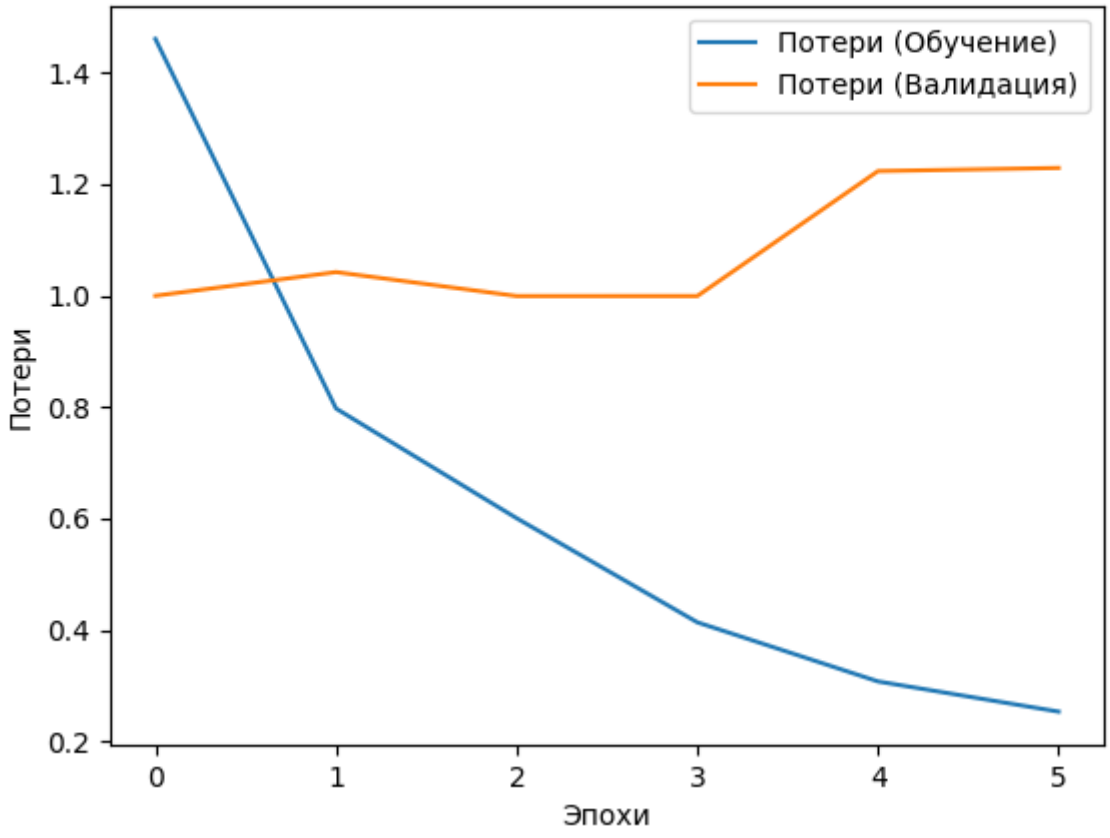
неизменными.

```
66/66 [----------] - 474s 7s/step - loss: 1.4597 - accuracy: 0.4681 - val_loss: 0.9994 - val_accuracy: 0.6133
Epoch 2/10
66/66 [----------] - 470s 7s/step - loss: 0.7972 - accuracy: 0.7010 - val_loss: 1.0418 - val_accuracy: 0.6200
Epoch 3/10
66/66 [----------] - 468s 7s/step - loss: 0.6006 - accuracy: 0.7781 - val_loss: 0.9989 - val_accuracy: 0.6222
Epoch 4/10
66/66 [----------] - 468s 7s/step - loss: 0.4140 - accuracy: 0.8538 - val_loss: 0.9989 - val_accuracy: 0.6511
Epoch 5/10
66/66 [----------] - 467s 7s/step - loss: 0.3080 - accuracy: 0.8914 - val_loss: 1.2229 - val_accuracy: 0.6222
Epoch 6/10
66/66 [----------] - 470s 7s/step - loss: 0.2539 - accuracy: 0.9152 - val_loss: 1.2285 - val_accuracy: 0.6356
15/15 [----------] - 78s 5s/step - loss: 1.0317 - accuracy: 0.6111
Test Accuracy: 61.11%
```

Как мы видим, результаты не сильно улучшилось. Для дальнейшего улучшения производительности модели можно попробовать провести аугментацию данных, подобрать другие архитектуры модели, но делать этого я не буду.

Ниже приведены график точности и потерь.


Точность (Обучение)
Точность (Валидация)
Точность
Эпохи
0.5
0.6
0.7
0.8
0.9
0
1
2
3
4
5


Потери (Обучение)
Потери (Валидация)
Потери
Эпохи
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
1
2
3
4
5