

Лабораторная работа №5 – Применение искусственных нейронных сетей: в играх, обработке естественного языка, распознавании образов

5.1 Реализация нейронной сети для обработки естественного языка или

5.2 Реализация нейронной сети для распознавания образов

Практическая реализация:

1. Создание сверточной нейронной сети для распознавания изображений и их классификации.

Шаг 1: Разархивирование папки с изображениями и загрузка необходимых библиотек.

```
In [2]: !unzip -q "/content/kagglecatsanddogs_3367a.zip"
```

```
In [3]: !ls "PetImages/"
```

```
Cat Dog
```

```
In [1]: import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pathlib
import os
from tqdm import tqdm
```

```
In [2]: # Number of Dog images
len(os.listdir("PetImages/Dog"))
```

```
Out[2]: 12501
```

```
In [3]: # Number of Cat images
len(os.listdir("PetImages/Cat"))
```

```
Out[3]: 12501
```

```
In [4]: DataPath = pathlib.Path("PetImages")
all_paths = DataPath.glob("*/*.jpg")
all_paths
```

```
Out[4]: <generator object Path.glob at 0x7f4d1d675af0>
```

Шаг 2. Чтение первых 10 изображений из файла.

```
In [5]: all_paths = list(all_paths)
all_paths[:10]
```

```
Out[5]: [PosixPath('PetImages/Cat/4969.jpg'),
PosixPath('PetImages/Cat/5682.jpg'),
PosixPath('PetImages/Cat/10285.jpg'),
PosixPath('PetImages/Cat/9575.jpg'),
PosixPath('PetImages/Cat/11089.jpg'),
PosixPath('PetImages/Cat/733.jpg'),
PosixPath('PetImages/Cat/3968.jpg'),
PosixPath('PetImages/Cat/10998.jpg'),
PosixPath('PetImages/Cat/11012.jpg'),
PosixPath('PetImages/Cat/9741.jpg')]
```

```
In [6]: all_paths = list(map(lambda x : str(x) , all_paths))
all_paths[:10]
```

```
Out[6]: ['PetImages/Cat/4969.jpg',
'PetImages/Cat/5682.jpg',
'PetImages/Cat/10285.jpg',
'PetImages/Cat/9575.jpg',
'PetImages/Cat/11089.jpg',
'PetImages/Cat/733.jpg',
'PetImages/Cat/3968.jpg',
'PetImages/Cat/10998.jpg',
'PetImages/Cat/11012.jpg',
'PetImages/Cat/9741.jpg']
```

Шаг 3. Перемешивание изображений в файле для предотвращения обучения сети только на одном типе изображений.

```
In [7]: from random import shuffle

shuffle(all_paths)
all_paths[:10]
```

```
Out[7]: ['PetImages/Dog/3583.jpg',
'PetImages/Cat/11799.jpg',
'PetImages/Dog/8758.jpg',
'PetImages/Dog/6731.jpg',
'PetImages/Cat/317.jpg',
'PetImages/Cat/2640.jpg',
'PetImages/Cat/4956.jpg',
'PetImages/Dog/9749.jpg',
'PetImages/Cat/10977.jpg',
'PetImages/Dog/12187.jpg']
```

Шаг 4. Декодирование изображений.

```
In [8]: def TestImageQuality(all_paths):
new_all_paths = []
for path in tqdm(all_paths):
    try :
        image = tf.io.read_file(path)
        image = tf.io.decode_jpeg(image , channels = 3)
    except :
        continue
    new_all_paths.append(path)
return new_all_paths

all_paths = TestImageQuality(all_paths)
all_paths[:10]
```

```
100%|██████████| 25000/25000 [00:29<00:00, 842.68it/s]
```

```
Out[8]: ['PetImages/Dog/3583.jpg',
        'PetImages/Cat/11799.jpg',
        'PetImages/Dog/8758.jpg',
        'PetImages/Dog/6731.jpg',
        'PetImages/Cat/317.jpg',
        'PetImages/Cat/2640.jpg',
        'PetImages/Cat/4956.jpg',
        'PetImages/Dog/9749.jpg',
        'PetImages/Cat/10977.jpg',
        'PetImages/Dog/12187.jpg']
```

Шаг 5. Получение меток и их преобразование в 0 и 1.

```
In [9]: # Getting their respective labels
def get_label(image_path):
    return image_path.split("/")[-2]

all_labels = list(map(lambda x : get_label(x) , all_paths))
all_labels[:10]
```

```
Out[9]: ['Dog', 'Cat', 'Dog', 'Dog', 'Cat', 'Cat', 'Cat', 'Dog', 'Cat', 'Dog']
```

```
In [10]: from sklearn.preprocessing import LabelEncoder

Le = LabelEncoder()
all_labels = Le.fit_transform(all_labels)

all_labels[:10]
```

```
Out[10]: array([1, 0, 1, 1, 0, 0, 0, 1, 0, 1])
```

Шаг 6. Создание обучающего набора данных.

```
In [11]: from sklearn.model_selection import train_test_split

Train_paths , Val_paths , Train_labels , Val_labels = train_test_split(all_paths , all_labels)
```

```
In [12]: Train_paths[:10] , Train_labels[:10]
```

```
Out[12]: (['PetImages/Dog/12437.jpg',
          'PetImages/Cat/12464.jpg',
          'PetImages/Dog/6869.jpg',
          'PetImages/Dog/4825.jpg',
          'PetImages/Dog/2074.jpg',
          'PetImages/Dog/7479.jpg',
          'PetImages/Cat/945.jpg',
          'PetImages/Cat/11412.jpg',
          'PetImages/Dog/5897.jpg',
          'PetImages/Cat/6329.jpg'],
         array([1, 0, 1, 1, 1, 1, 0, 0, 1, 0]))
```

```
In [13]: # Function used to decode jpeg into Tensor

def load(image , label):
    image = tf.io.read_file(image)
    image = tf.io.decode_jpeg(image , channels = 3)
    return image , label
```

```
In [14]: # Define IMAGE SIZE and BATCH SIZE
IMG_SIZE = 224
BATCH_SIZE = 128

# Basic Transformation
resize = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.Resizing(IMG_SIZE, IMG_SIZE)
])

# Data Augmentation
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
    tf.keras.layers.experimental.preprocessing.RandomZoom(height_factor = (-0.3, -0.2))
])
```

```
In [15]: # Function used to Create a Tensorflow Data Object
AUTOTUNE = tf.data.experimental.AUTOTUNE
def get_dataset(paths , labels , train = True):
    image_paths = tf.convert_to_tensor(paths)
    labels = tf.convert_to_tensor(labels)

    image_dataset = tf.data.Dataset.from_tensor_slices(image_paths)
    label_dataset = tf.data.Dataset.from_tensor_slices(labels)

    dataset = tf.data.Dataset.zip((image_dataset , label_dataset))

    dataset = dataset.map(lambda image , label : load(image , label))
    dataset = dataset.map(lambda image, label: (resize(image), label) , num_parallel_calls=AUTOTUNE)
    dataset = dataset.shuffle(1000)
    dataset = dataset.batch(BATCH_SIZE)

    if train:
        dataset = dataset.map(lambda image, label: (data_augmentation(image), label) , num_parallel_calls=AUTOTUNE)
        dataset = dataset.repeat()
    return dataset
```

```
In [16]: # Creating Train Dataset object and Verifying it
%time train_dataset = get_dataset(Train_paths , Train_labels)

image , label = next(iter(train_dataset))
print(image.shape)
print(label.shape)

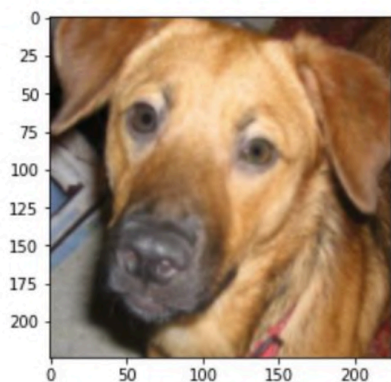
CPU times: user 289 ms, sys: 4.94 ms, total: 294 ms
Wall time: 294 ms
(128, 224, 224, 3)
(128,)
```

Шаг 7. Визуализация по одному примеру из каждого класса (cat, dog) из обучающего набора, чтобы убедиться в корректности чтения данных.

```
In [17]: # View a sample Training Image
print(Le.inverse_transform(label)[0])
plt.imshow((image[0].numpy()/255).reshape(224 , 224 , 3))

Dog
```


Out[17]: <matplotlib.image.AxesImage at 0x7f4ca53a5240>



```
In [18]: %time val_dataset = get_dataset(Val_paths , Val_labels , train = False)
```

```
image , label = next(iter(val_dataset))  
print(image.shape)  
print(label.shape)
```

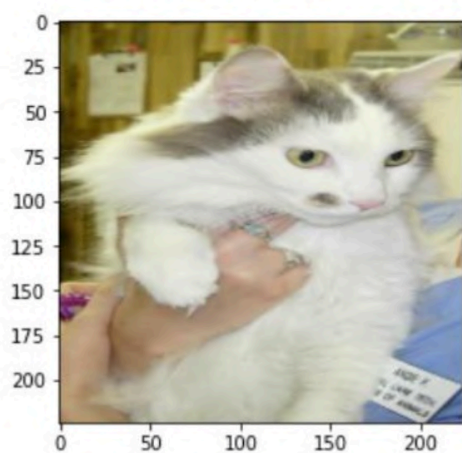
```
CPU times: user 22.9 ms, sys: 888 µs, total: 23.8 ms  
Wall time: 33.7 ms  
(128, 224, 224, 3)  
(128,)
```

```
In [19]: # View a sample Training Image
```

```
print(Le.inverse_transform(label)[0])  
plt.imshow((image[0].numpy()/255).reshape(224 , 224 , 3))
```

Cat

Out[19]: <matplotlib.image.AxesImage at 0x7f4ca6cb2978>



Шаг 8. Создание модели сверточной нейронной сети.

```

In [20]: # Model part
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatten, Dropout,
        BatchNormalization, GlobalMaxPool2D

model = Sequential()

# Block 1
model.add(Conv2D(input_shape=(224 , 224 , 3), padding='same',filters=32, kernel_size=(7, 7)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Block 2
model.add(Conv2D(filters=64, padding='valid', kernel_size=(5, 5)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Block 3
model.add(Conv2D(filters=128, padding='valid', kernel_size=(3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Dropout(0.2))

# Block 4
model.add(Conv2D(filters=256, padding='valid', kernel_size=(3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())

model.add(Conv2D(filters=256 , kernel_size=(3, 3)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(GlobalMaxPool2D())

model.add(Dense(units=256))
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(units=1))
model.add(Activation('sigmoid'))

```

Шаг 9. Визуализация информации о созданной модели.

```
In [21]: print(model.summary())
```

Model: "sequential_2"

| Layer (type) | Output Shape | Param # |
|------------------------------|----------------------|---------|
| conv2d (Conv2D) | (None, 224, 224, 32) | 4736 |
| activation (Activation) | (None, 224, 224, 32) | 0 |
| batch_normalization (BatchNo | (None, 224, 224, 32) | 128 |
| max_pooling2d (MaxPooling2D) | (None, 112, 112, 32) | 0 |
| dropout (Dropout) | (None, 112, 112, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 108, 108, 64) | 51264 |

| | | |
|---|----------------------|--------|
| activation_1 (Activation) | (None, 108, 108, 64) | 0 |
| batch_normalization_1 (Batch Normalization) | (None, 108, 108, 64) | 256 |
| max_pooling2d_1 (MaxPooling2D) | (None, 54, 54, 64) | 0 |
| dropout_1 (Dropout) | (None, 54, 54, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 52, 52, 128) | 73856 |
| activation_2 (Activation) | (None, 52, 52, 128) | 0 |
| batch_normalization_2 (Batch Normalization) | (None, 52, 52, 128) | 512 |
| max_pooling2d_2 (MaxPooling2D) | (None, 26, 26, 128) | 0 |
| dropout_2 (Dropout) | (None, 26, 26, 128) | 0 |
| conv2d_3 (Conv2D) | (None, 24, 24, 256) | 295168 |
| activation_3 (Activation) | (None, 24, 24, 256) | 0 |
| batch_normalization_3 (Batch Normalization) | (None, 24, 24, 256) | 1024 |
| conv2d_4 (Conv2D) | (None, 22, 22, 256) | 590080 |
| activation_4 (Activation) | (None, 22, 22, 256) | 0 |
| batch_normalization_4 (Batch Normalization) | (None, 22, 22, 256) | 1024 |
| max_pooling2d_3 (MaxPooling2D) | (None, 11, 11, 256) | 0 |
| dropout_3 (Dropout) | (None, 11, 11, 256) | 0 |
| global_max_pooling2d (Global Max Pooling2D) | (None, 256) | 0 |
| dense (Dense) | (None, 256) | 65792 |
| activation_5 (Activation) | (None, 256) | 0 |
| dropout_4 (Dropout) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 1) | 257 |
| activation_6 (Activation) | (None, 1) | 0 |
| ===== | | |
| Total params: 1,084,097 | | |
| Trainable params: 1,082,625 | | |
| Non-trainable params: 1,472 | | |
| ===== | | |
| None | | |

Шаг 10. Компиляция и обучение модели.

```
In [22]: model.compile(
          loss='binary_crossentropy',
          optimizer='adam',
          metrics=['accuracy']
        )
```



```
In [23]: # Train the model
history = model.fit(
    train_dataset,
    steps_per_epoch=len(Train_paths)//BATCH_SIZE,
    epochs=10,
    validation_data=val_dataset,
    validation_steps = len(Val_paths)//BATCH_SIZE,
)
```

```
Epoch 1/10
145/145 [=====] - 153s 1s/step - loss: 1.3252 - accuracy: 0.5397 - val_loss: 0.6580 - val_accuracy: 0.6162
Epoch 2/10
145/145 [=====] - 151s 1s/step - loss: 0.6303 - accuracy: 0.6545 - val_loss: 0.6232 - val_accuracy: 0.6616
Epoch 3/10
145/145 [=====] - 149s 1s/step - loss: 0.5782 - accuracy: 0.6990 - val_loss: 0.8106 - val_accuracy: 0.5033
Epoch 4/10
145/145 [=====] - 149s 1s/step - loss: 0.5484 - accuracy: 0.7241 - val_loss: 0.7540 - val_accuracy: 0.5684
Epoch 5/10
145/145 [=====] - 149s 1s/step - loss: 0.5307 - accuracy: 0.7353 - val_loss: 0.6003 - val_accuracy: 0.6974
Epoch 6/10
145/145 [=====] - 149s 1s/step - loss: 0.4903 - accuracy: 0.7624 - val_loss: 0.6985 - val_accuracy: 0.6442
Epoch 7/10
145/145 [=====] - 149s 1s/step - loss: 0.4679 - accuracy: 0.7753 - val_loss: 0.5867 - val_accuracy: 0.7095
Epoch 8/10
```

Шаг 11. Оценка модели.

```
In [24]: # Evaluating the loaded model
loss, acc = model.evaluate(val_dataset)

print(" Testing Acc : " , acc)
print(" Testing Loss : " , loss)
```

```
49/49 [=====] - 12s 212ms/step - loss: 0.4526 - accuracy: 0.7734
Testing Acc : 0.7734450697898865
Testing Loss : 0.45264872908592224
```