Практическая работа №6

Выполнил студент группы БВТ2003 Глазков Даниил

Задание

Необходимо построить сверточную нейронную сеть, которая будет классифицировать черно-белые изображения с простыми геометрическими фигурами на них.

К каждому варианту прилагается код, который генерирует изображения.

Для генерации данных необходимо вызвать функцию gen_data, которая возвращает два тензора:

Тензор с изображениями ранга 3 Тензор с метками классов Обратите внимание:

Выборки не перемешаны, то есть наблюдения классов идут по порядку Классы характеризуются строковой меткой Выборка изначально не разбита на обучающую, контрольную и тестовую Скачивать необходимо оба файла. Подключать файл, который начинается с var (в нем и находится функция gen_data)

In [1]:
```
import tensorflow as tf
physical_device = tf.config.experimental.list_physical_devices('GPU')
print(f'Device found : {physical_device}')
```

```
Device found : [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

In [2]:
```
tf.config.experimental.set_memory_growth(physical_device[0],True)
```

In [3]:
```
import tensorflow as tf
t = tf.config.experimental.get_memory_growth(physical_device[0])
print(t)
```

```
True
```

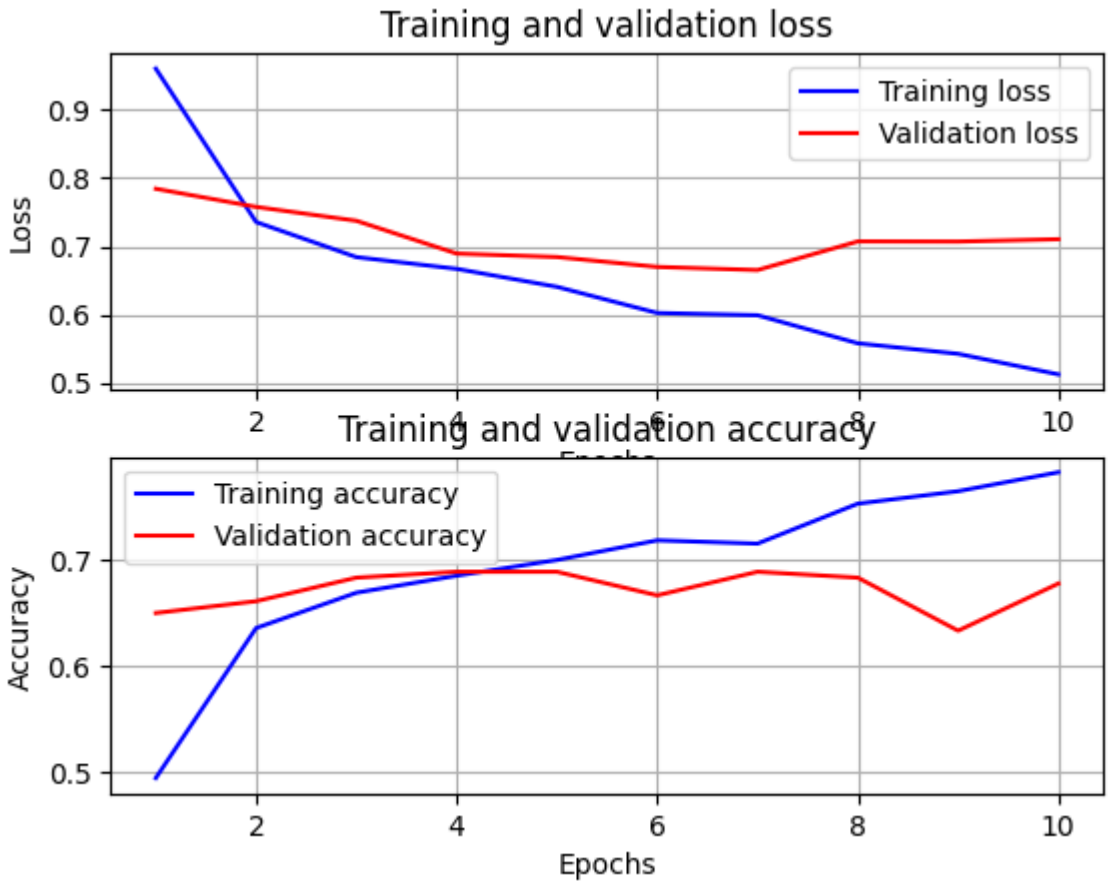
In [4]:
```
import sys
import os
import sklearn

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

sys.path.append(os.path.abspath("C:\\Users\\loprz\\Downloads\\laba6"))
import var7
from var7 import gen_data

import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import LabelEncoder
from tensorflow.python.keras import Input, Sequential
from tensorflow.python.keras.layers import Convolution2D, MaxPooling2D, Flatten,
from tensorflow.python.keras.utils.np_utils import to_categorical
```

```
def build_model(layers):
    model = Sequential()
    for layer in layers:
        model.add(layer)
    model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["a
    return model


def show_image(image, label):
    plt.imshow(image.reshape(image.shape[0], image.shape[0]), cmap=plt.cm.binary
    plt.show()
    print(label)


def plot(epochs, train, validation, metrics):
    plt.plot(epochs, train, 'b', label=f'Training {metrics}')
    plt.plot(epochs, validation, 'r', label=f'Validation {metrics}')
    plt.title(f'Training and validation {metrics}')
    plt.xlabel('Epochs')
    plt.ylabel(metrics.capitalize())
    plt.grid(True)
    plt.legend()


def plot_history(history):
    loss = history['loss']
    val_loss = history['val_loss']
    acc = history['accuracy']
    val_acc = history['val_accuracy']
    epochs = range(1, len(loss) + 1)

    plt.figure()
    plt.subplot(211)
    plot(epochs, loss, val_loss, "loss")
    plt.subplot(212)
    plot(epochs, acc, val_acc, "accuracy")
    plt.show()


def normalize(fn):
    def wrapper():
        x, y = fn()
        x /= np.max(x)
        return x, y

    return wrapper


def one_hot_y(fn):
    def wrapper():
        x, y = fn()
        return x, to_categorical(y)

    return wrapper


def encode_labels(fn):
    def wrapper():
        x, y = fn()
```

```
        return x, LabelEncoder().fit_transform(y)

    return wrapper


def shuffle(fn):
    def wrapper():
        x, y = fn()
        x, y = sklearn.utils.shuffle(x, y)
        return x, y

    return wrapper


def train_test(fn):
    def wrapper():
        global train_size, image_side
        x, y = fn()
        x = x.reshape(-1, image_side, image_side, 1)
        train_len = int(x.shape[0] * train_size)
        return x[:train_len, :], y[:train_len, :], x[train_len:, :], y[train_len

    return wrapper


image_side = 50
n_samples = 2000
train_size = 0.9


@train_test
@normalize
@one_hot_y
@shuffle
@encode_labels
def prepare_data():
    global n_samples, image_side
    return gen_data(n_samples, image_side)


train_x, train_y, test_x, test_y = prepare_data()


layers = [
    Input(shape=(image_side, image_side, 1)),
    Convolution2D(filters=32, kernel_size=(7, 7), padding="same", activation="re
    MaxPooling2D(pool_size=(5, 5), padding="same"),
    Convolution2D(filters=64, kernel_size=(7, 7), padding="same", activation="re
    MaxPooling2D(pool_size=(5, 5), padding="same"),
    Flatten(),
    Dense(512, activation="relu"),
    Dropout(0.25),
    Dense(256, activation="relu"),
    Dropout(0.15),
    Dense(3, activation="softmax")
]

model = build_model(layers)
history = model.fit(train_x, train_y, batch_size=20, epochs=10, validation_split
```

```
model.evaluate(test_x, test_y)
plot_history(history.history)
```

```
C:\Users\loprz\miniconda3\envs\tf\lib\site-packages\sklearn\preprocessing\_label.
py:116: DataConversionWarning: A column-vector y was passed when a 1d array was e
xpected. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
WARNING:tensorflow:Please add `keras.layers.InputLayer` instead of `keras.Input`
to Sequential model. `keras.Input` is intended to be used by Functional model.
Epoch 1/10
81/81 [==============================] - 8s 12ms/step - loss: 0.9604 - accuracy:
0.4944 - val_loss: 0.7844 - val_accuracy: 0.6500
Epoch 2/10
81/81 [==============================] - 1s 9ms/step - loss: 0.7356 - accuracy:
0.6358 - val_loss: 0.7578 - val_accuracy: 0.6611
Epoch 3/10
81/81 [==============================] - 1s 9ms/step - loss: 0.6842 - accuracy:
0.6691 - val_loss: 0.7375 - val_accuracy: 0.6833
Epoch 4/10
81/81 [==============================] - 1s 9ms/step - loss: 0.6670 - accuracy:
0.6852 - val_loss: 0.6894 - val_accuracy: 0.6889
Epoch 5/10
81/81 [==============================] - 1s 9ms/step - loss: 0.6407 - accuracy:
0.7000 - val_loss: 0.6843 - val_accuracy: 0.6889
Epoch 6/10
81/81 [==============================] - 1s 9ms/step - loss: 0.6023 - accuracy:
0.7185 - val_loss: 0.6698 - val_accuracy: 0.6667
Epoch 7/10
81/81 [==============================] - 1s 9ms/step - loss: 0.5990 - accuracy:
0.7154 - val_loss: 0.6652 - val_accuracy: 0.6889
Epoch 8/10
81/81 [==============================] - 1s 9ms/step - loss: 0.5579 - accuracy:
0.7531 - val_loss: 0.7072 - val_accuracy: 0.6833
Epoch 9/10
81/81 [==============================] - 1s 9ms/step - loss: 0.5425 - accuracy:
0.7648 - val_loss: 0.7070 - val_accuracy: 0.6333
Epoch 10/10
81/81 [==============================] - 1s 9ms/step - loss: 0.5124 - accuracy:
0.7827 - val_loss: 0.7104 - val_accuracy: 0.6778
7/7 [==============================] - 0s 18ms/step - loss: 0.6449 - accuracy: 0.
6950
```



In [13]:
```
X, y = gen_data()
X, y = sklearn.utils.shuffle(X, y)
for i in range(222, 260):
    plt.imshow(X[i], cmap=plt.cm.binary)
```