МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Факультет прикладной математики и информатики

Кафедра ТПИ

Дисциплина: «Основы теории машинного обучения»

Лабораторная работа №1, модуль «Нейронные сети глубокого обучения»

**СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ**

Факультет: ФПМИ

Группа: ПМИМ-31

Выполнили: Монгуш Н. С., Тарулин М. А., Филипенко Ю. Д.

Проверил: Попов А. А.

Дата выполнения:

Отметка о защите:

1. ЦЕЛЬ РАБОТЫ

Изучить архитектуру и принцип работы сверточной нейронной сети.

1. ЗАДАНИЕ
2. Реализовать структуру сверточной нейронной сети. Визуализировать её.
3. Обучить полученную нейронную сеть на датасете CIFAR10. Количество эпох, итераций и батчей выбирать в зависимости от вычислительных возможностей.
4. ИСХОДНЫЕ ДАННЫЕ

В качестве исходных данных был выбран датасет CIFAR10, откуда выделены 2 класса изображений – изображения котов и собак (рис. 1.1). Изображения имеют размеры 32x32 пикселя, 3 цветовых канала, размер батча – 4.



Рис. 1.1 – Изображения из датасета CIFAR10, где [0. 1.] – классификатор собак, [1. 0.] – классификатор кошек.

1. ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

Архитектура разработанной сверточной нейронной сети представлена на рис. 2.1 Она состоит из следующих слоев:

1. InputLayer – входной слой, входные данные представляют из себя нормализованные цветные изображения JPEG с размером 32x32 пикселя. Каждое изображение разбивается на 3 канала: красный, синий, зеленый.
2. Conv2D – сверточные слои с количеством фильтров от 32 до 4 в зависимости от слоя. Размер ядра свертки равен 3, с функцией активацией «relu».
3. GlobalAveragePooling2D – слой пуллинга по максимальному значению.
4. Flatten – слой который, который многомерный массив представляет в виде одномерного.
5. Dense – полносвязный слой с функцией активации «softmax» для двухклассовой классификации.

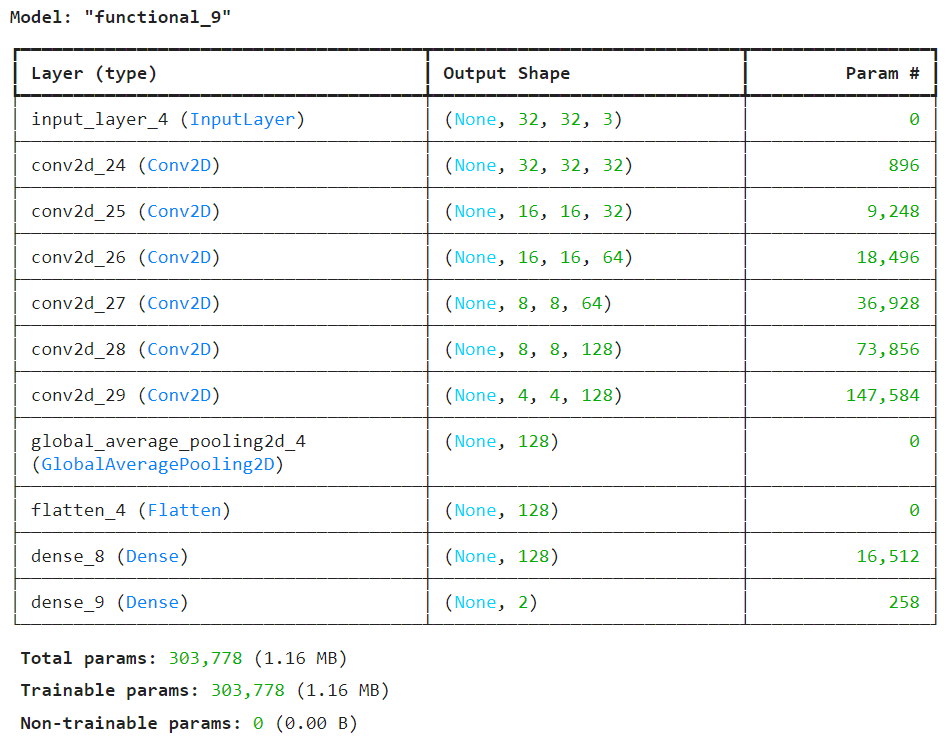


Рис. 2.1 – Архитектура сверточной нейронной сети.

Представленная архитектура была скомпилирована и обучена в течении 20 эпох. На рис. 2.2 можно увидеть график изменения функции ошибки в процессе обучения. По этому графику можно сказать, что имеется тенденция на снижения ошибки с каждой эпохой. Отсутствуют выбросы на графике. Это говорит о правильно подобранной архитектуре сверточной нейронной сети.

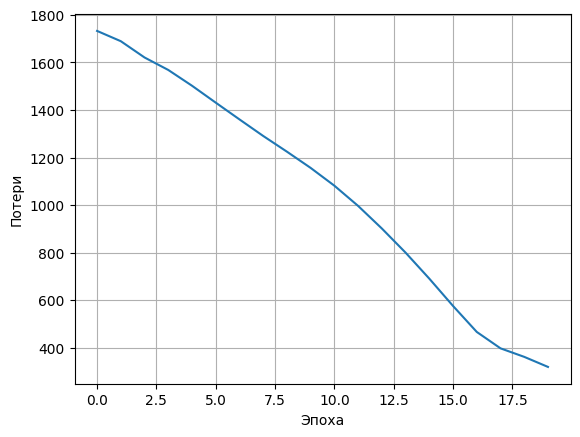
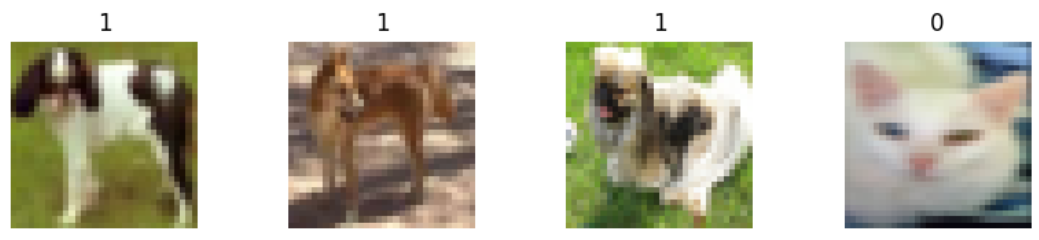
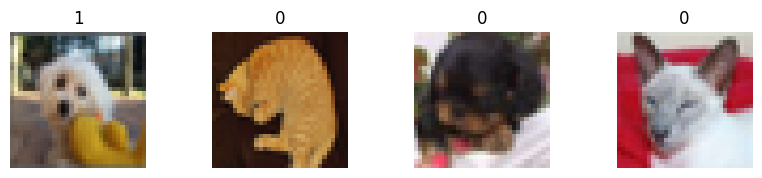


Рис. 2.2 – График изменения функции ошибки в процессе обучения.

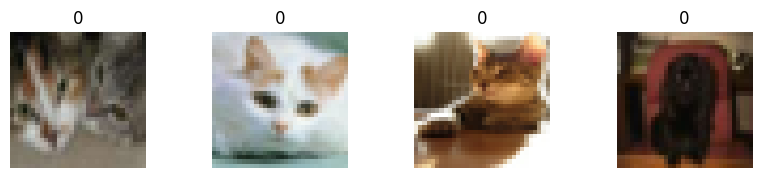
Полученная модель совершила 2 ошибки классификации из 20 представленных изображений.











1. ВЫВОД

В результате работы реализована структура сверточной нейронной сети, которая была обучена на датасете CIFAR10, допущено всего лишь 2 ошибок классификации из 20 представленных изображений.

Анализ графика изменения функции ошибки в процессе обучения показал, что модель стабильно улучшалась с каждой эпохой, без выбросов, что свидетельствует о правильно подобранной архитектуре сети и эффективном процессе обучения.

1. КОД ПРОГРАММЫ

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.layers import Dense, Flatten, Reshape, Input, Conv2D, GlobalAveragePooling2D

import matplotlib.pyplot as plt

import numpy as np

from IPython.display import clear\_output

from typing import NamedTuple

class TrainData(NamedTuple):

train\_images: np.ndarray

train\_labels: np.ndarray

class TestData(NamedTuple):

test\_images: np.ndarray

test\_labels: np.ndarray

class DataSet(NamedTuple):

TrainData: TrainData

TestData: TestData

def get\_2\_class\_raw\_dataset(class\_names: list[str]=[3, 5]) -> DataSet:

(train\_images, train\_labels), (test\_images, test\_labels) = tf.keras.datasets.cifar10.load\_data()

train\_mask = np.isin(train\_labels, class\_names).reshape(-1)

train\_images = train\_images[train\_mask]

train\_labels = train\_labels[train\_mask]

test\_mask = np.isin(test\_labels, class\_names).reshape(-1)

test\_images = test\_images[test\_mask]

test\_labels = test\_labels[test\_mask]

train\_labels = np.where(train\_labels == 3, 0, 1)

test\_labels = np.where(test\_labels == 3, 0, 1)

return (train\_images, train\_labels), (test\_images, test\_labels)

def \_process\_x(x):

img = tf.cast(x, tf.float32) / 255

return img

def \_process\_y(y):

y = tf.one\_hot(tf.squeeze(y), 2)

return y

def get\_train\_dataset():

(train\_images, train\_labels), (\_, \_) = get\_2\_class\_raw\_dataset()

x\_dataset = tf.data.Dataset.from\_tensor\_slices(train\_images)

y\_dataset = tf.data.Dataset.from\_tensor\_slices(train\_labels)

x\_dataset = x\_dataset.map(\_process\_x)

y\_dataset = y\_dataset.map(\_process\_y)

dataset = tf.data.Dataset.zip((x\_dataset, y\_dataset))

dataset = dataset.shuffle(1000)

dataset = dataset.batch(4)

AUTOTUNE = tf.data.AUTOTUNE

dataset = dataset.cache().prefetch(buffer\_size=AUTOTUNE)

return dataset

def check\_train\_dataset() -> None:

dataset = get\_train\_dataset()

for x, y in dataset.take(1):

print("Shape of x:", x.shape, "Shape of y:", y.shape) # должно быть что-то вроде (4, 32, 32, 3) и (4, 2)

check\_train\_dataset()

def show\_dataset(plot\_number: int=4) -> None:

dataset = get\_train\_dataset()

plt.figure(figsize=(10, 6))

for images, labels in dataset.take(1):

for i in range(plot\_number):

img = images[i]

label = labels[i]

ax = plt.subplot(3, plot\_number, i + 1 + plot\_number)

plt.imshow(img)

plt.title(str(labels.numpy()[i]))

plt.axis('off')

ax.get\_yaxis().set\_visible(False)

plt.show()

show\_dataset()

def get\_model\_layers():

inputs = Input(shape=(32, 32, 3))

x = inputs

x = Conv2D(32, 3, activation='relu', padding='same')(x)

x = Conv2D(32, 3, activation='relu', strides=2, padding='same')(x)

x = Conv2D(64, 3, activation='relu', padding='same')(x)

x = Conv2D(64, 3, activation='relu', strides=2, padding='same')(x)

x = Conv2D(128, 3, activation='relu', padding='same')(x)

x = Conv2D(128, 3, activation='relu', strides=2, padding='same')(x)

x = GlobalAveragePooling2D()(x)

x = Flatten()(x)

x = Dense(128, activation='relu')(x)

x = Dense(2, activation='softmax')(x)

outputs = x

cnn = keras.Model(inputs, outputs)

return cnn

get\_model\_layers().summary()

class Model(tf.keras.Model):

def \_\_init\_\_(self, nn):

super(Model, self).\_\_init\_\_()

self.nn = nn

self.optimizer = tf.keras.optimizers.Adam()

def get\_loss(self, y, preds):

loss = tf.keras.losses.CategoricalCrossentropy()(y, preds)

return loss

@tf.function

def training\_step(self, x, y):

with tf.GradientTape() as tape:

preds = self.nn(x)

loss = self.get\_loss(y, preds)

gradients = tape.gradient(loss, self.nn.trainable\_variables)

self.optimizer.apply\_gradients(zip(gradients, self.nn.trainable\_variables))

return tf.reduce\_mean(loss)

def compile\_model():

cnn = get\_model\_layers()

model = Model(cnn)

model.compile(optimizer=tf.keras.optimizers.Adam(1e-4))

return model

compile\_model()

def check\_teaching\_model():

dataset = get\_train\_dataset()

model = compile\_model()

for x, y in dataset.take(1):

print(model.training\_step(x, y))

check\_teaching\_model()

def teach\_model(epochs: int):

hist = np.array(np.empty([0]))

dataset = get\_train\_dataset()

model = compile\_model()

for epoch in range(1, epochs + 1):

loss = 0

for step, (x, y) in enumerate(dataset):

loss += model.training\_step(x, y)

clear\_output(wait=True)

print(epoch)

hist = np.append(hist, loss)

plt.plot(np.arange(0, len(hist)), hist)

plt.xlabel('Эпоха')

plt.ylabel('Потери')

plt.grid(True)

plt.show()

teach\_model(20)

def get\_model(epochs=20):

dataset = get\_train\_dataset()

model = compile\_model()

for epoch in range(1, epochs + 1):

loss = 0

for step, (x, y) in enumerate(dataset):

loss += model.training\_step(x, y)

return model

model = get\_model()

def pred\_dataset(n: int=4):

plt.figure(figsize=(10, 6))

dataset = get\_train\_dataset()

for images, labels in dataset.take(1):

for i in range(n):

img = images[i]

img\_tensor = tf.expand\_dims(img, axis=0)

pred = model.nn(img\_tensor)

pred = tf.squeeze(pred, axis=0)

pred = pred.numpy()

ax = plt.subplot(3, n, i + 1 + n)

plt.imshow(img)

ma = pred.max()

res = np.where(pred == ma)

plt.title(res[0][0])

plt.axis('off')

ax.get\_yaxis().set\_visible(False)

plt.show()

pred\_dataset()