**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра вычислительной техники**

**ОТЧЁТ  
по дисциплине «Нейронные сети»**

| Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| --- | --- |
| Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Гатауллин Р.И. |

**Лабораторная работа № 1  
Тема: «Сверточные нейронные сети»**

Санкт-Петербург

2024

**Оглавление**

[Цель работы 3](#_1fob9te)

[Задачи работы 3](#_3znysh7)

[Обоснование выбора набора данных 3](#_2et92p0)

[Краткая теоретическая информация 4](#_tyjcwt)

[Процесс выполнения работы 7](#)

[Вывод 18](#)

[Код программы 19](#_h6sqr2kyabjp)

# **Цель работы**

Эта работа нацелена на изучение сверточных нейронных сетей, приобретение практических навыков в их использовании, понимание архитектуры данных сетей, методов их обучения и применение их в различных задачах.

# **Задачи работы**

В ходе данной работы необходимо выполнить следующие задачи:

1. Провести предварительную обработку данных, включая демонстрацию методов подготовки данных, таких как увеличение выборки и нормализация.

2. Изучить архитектуру сверточных нейронных сетей (CNN) и применить технику трансферного обучения, включая разработку собственных архитектур с последующим обучением "с нуля" нескольких различных моделей с различными комбинациями слоев и функций активации.

3. Обучить сверточные нейронные сети (CNN) на подготовленных данных.

4. Произвести настройку гипер параметров и выбор оптимальной модели для дальнейшего использования.

# **Обоснование выбора набора данных**

<https://www.kaggle.com/datasets/prasunroy/natural-images> – набор данных с 8 видами разных натуральных изображений.

Нейронная сеть, обученная на таком многообразном датасете изображений, содержащем различные классы, может быть использована во многих различных областях и задачах. Например:

1. Классификация изображений: Нейросеть может быть использована для автоматической классификации изображений на один из предсказанных классов.

2. Обнаружение объектов: Нейросеть может быть применена для обнаружения объектов на изображениях. Это может быть полезно, например, для систем наблюдения за обстановкой, систем безопасности или автономных автомобилей.

3. Генерация изображений: С помощью генеративных моделей, на основе нейронной сети, можно создавать синтетические изображения объектов, принадлежащих к любому из классов в датасете.

# **Краткая теоретическая информация**

**Сверточные нейронные сети (CNN)** - это глубокие нейронные сети, которые получили широкое распространение в задачах обработки изображений и видео. Основное их отличие от традиционных нейронных сетей заключается в том, что они автоматически и эффективно могут выявлять важные признаки из изображений, не требуя ручного извлечения характеристик. Это делает CNN идеальными для таких задач, как распознавание лиц, автоматическое управление транспортом, медицинская диагностика и многих других.

**Сверточный слой** – это основной блок сверточной нейронной сети. Слой свертки включает в себя для каждого канала свой фильтр, ядро свертки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты поэлементного произведения для каждого фрагмента).

**Слой “пулинга”** – отвечает за уменьшение пространственного размера свернутого объекта. Это позволяет сократить вычислительную мощность, необходимую для обработки данных. Кроме того, уплотнение изображения полезно для извлечения доминирующих признаков, которые являются ротационными и позиционно-инвариантными. В результате поддерживается процесс эффективного обучения модели. Различают два типа пулинга: *максимальный* – возвращает максимальное значение из фрагмента изображения, охватываемого ядром и *средний* – возвращает среднее всех значений из фрагмента изображения, охватываемого ядром.

**Полносвязный слой** — это слой, в котором каждый нейрон в слое связан с каждым нейроном в предыдущем слое. Каждый нейрон в слое получает входные данные от каждого нейрона в предыдущем слое, а не только от подмножества нейронов.

**Функции активации** – определяют поведение нейронов в ответ на сумму входных сигналов. Они могут быть простыми, например, линейными, или более сложными, например, нелинейными, такими как сигмоидная или ReLU. Главная цель функции активации заключается в добавлении нелинейности в нейронную сеть. Это позволяет модели обучаться и решать более сложные задачи, эмулируя сложные процессы, свойственные человеческому мозгу. Функции активации помогают нормализовать входные данные, приводя их к нужному диапазону. Другими словами, пропуская большие значения через функцию активации, мы получаем выходные значения в удобных для нас пределах.  
 **Регуляризация** – метод добавления некоторых дополнительных ограничений к условию с целью решить некорректно поставленную задачу или предотвратить переобучение. Этого можно добиться с помощью: введения штрафов на высокие значения весов, что помогает управлять переобучением модели или добавления дополнительных слоев для предотвращения переобучения и повышения устойчивости нейронной сети.

Регуляризация способствует улучшению обобщающей способности нейронной сети и повышает ее способность обрабатывать новые данные.

**Алгоритмы оптимизации –** играют ключевую роль в настройке параметров модели, таких как весы и смещения, чтобы обеспечить лучшие результаты при решении конкретной задачи. Adam (Adaptive Moment Estimation) представляет собой один из самых современных и эффективных алгоритмов оптимизации, который широко применяется в области машинного и глубокого обучения. Adam использует момент первого порядка (среднее значение градиентов, аналогичное Momentum) и момент второго порядка (скользящее среднее квадратов градиентов, похожее на RMSprop). Как и RMSprop, Adam адаптирует скорость обучения для каждого параметра индивидуально. Благодаря добавлению момента, Adam ещё более эффективно справляется с различными типами градиентов, будь то разреженные, шумные или иные.

**Трансферное обучение** – это метод обучения нейронной сети, который часто применяется в практике. Он основан на использовании заранее обученной модели на подобных данных или данных схожего характера с теми, на которых будет работать целевая модель, которую мы обучаем.

**Батч** – представляет собой группу обучающих примеров из входных данных. Нейронная сеть обновляет свои веса после обработки всех примеров в батче. Этот процесс называется итерацией - успешная обработка всех обучающих примеров из батча с последующим обновлением весов в сети. Когда все обучающие примеры из входных данных обрабатываются с периодическим обновлением весов, это называется эпохой.

# **Процесс выполнения работы**

**Подготовка**

Так как изначальный датасет содержит в себе только разделение на классы, необходимо разделить данные на обучающий, валидационный и тестовый наборы, в соотношении ~70:15:15 (%). После данных преобразований, в исходном каталоге проекта программы, появились папки с обучающим, валидационным и тестовым наборами.

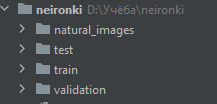


Рис. 1 – наборы данных

Для дальнейшего обучения нужно подготовить исходные изображения:

train\_gen = ImageDataGenerator(rescale = 1.0/255.0)

train\_image\_generator = train\_gen.flow\_from\_directory(

train\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

val\_gen = ImageDataGenerator(rescale = 1.0/255.0)

val\_image\_generator = train\_gen.flow\_from\_directory(

validation\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

test\_gen = ImageDataGenerator(rescale = 1.0/255.0)

test\_image\_generator = train\_gen.flow\_from\_directory(

test\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

* ImageDataGenerator – создает генератор данных, который обрабатывает изображения для аугментации и предварительной обработки перед их передачей в нейронную сеть. Параметр **(rescale = 1.0/255.0)** нормализует значения пикселей изображений, приводя их к диапазону от 0 до 1, улучшая процесс обучения.
* train\_image\_generator – создает генератор изображений для обучения нейронной сети с помощью метода flow\_from\_directory, извлекая изображения из указанной директории (train\_path), изменяя их размер до 150x150 пикселей, формируя пакеты из 32 изображений, и используя категориальный формат для меток классов (если имеются несколько классов, представленных в виде категорий).

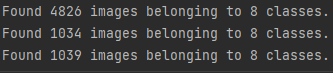


Рис. 2 – кол-во данных, разделенных на разные наборы обучения.

**Обучение на 3-х разных архитектурах**

Архитектуры CNN:

* Архитектура №1

В данной архитектуре используются следующие методы обучения: **Conv2D –** один сверточный слой, функция активации – **ReLU**, приведение матрицы к виду вектора – Reshape (в нашем случае **Flatten**), создание полносвязного слоя – **Dense** и функция **Softmax**, преобразующая наш вектор таким образом, чтобы все его значения удовлетворяли определению вероятности.

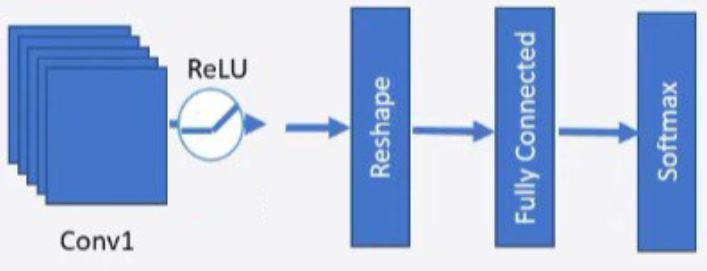


Рис. 3 – Архитектура №1.

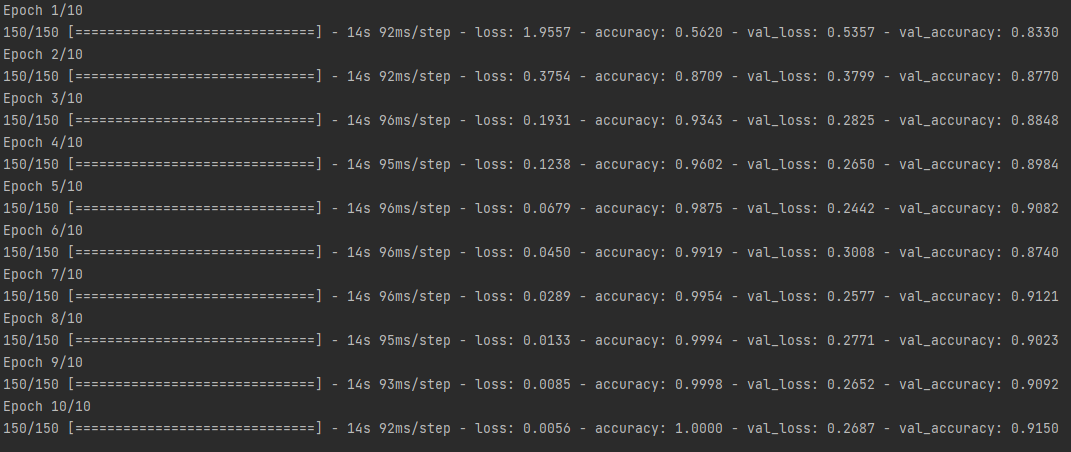
На данной архитектуре нейронной сети получаются следующие результаты:

Рис. 4 – Обучение нейронной сети на архитектуре №1

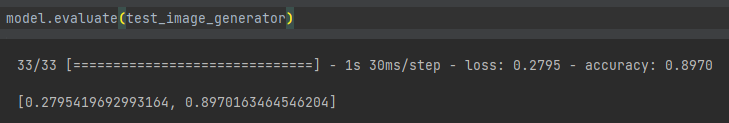


Рис. 5 – Потери и точность работы нейронной сети на тестовом наборе

архитектуры №1

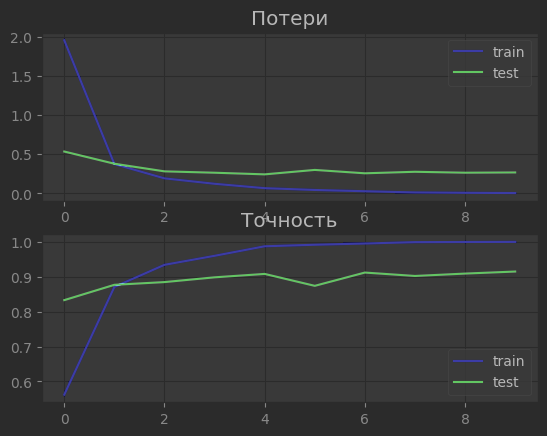


Рис. 6 – Зависимости потерь и точности модели от кол-ва эпох обучения

на архитектуре №1

Из данных графиков можно сделать вывод, что на данной архитектуре модель обучается достаточно точно, однако, скачки 5-ой эпохи в кривых тестов, как в точности обучения так и в потерях, указывают на то, что модель может переобучаться. Это обусловлено нехваткой сверточных слоев и активационных функций, а также самим исходным датасетом, кол-вом данных в нем и разницей в классах обучения.

* Архитектура №2

В данной архитектуре используются следующие методы обучения: **Conv2D –** два сверточных слоя, функции активации – **ReLU** после каждого свертывания, функции подвыборки (пулинга) – **MaxPooling2D**, приведение матрицы к виду вектора – Reshape (в нашем случае **Flatten**), создание полносвязного слоя – **Dense** и функция **Softmax**, преобразующая наш вектор таким образом, чтобы все его значения удовлетворяли определению вероятности.

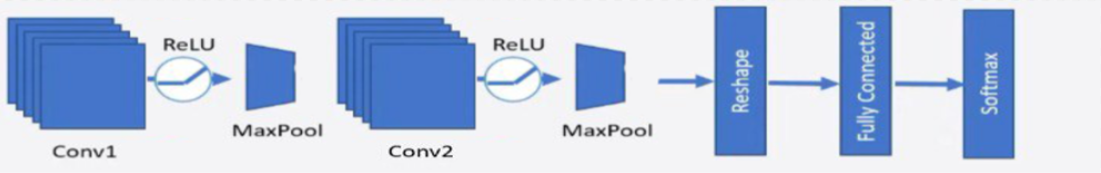


Рис. 7 – Архитектура №2.

На данной архитектуре нейронной сети получаются следующие результаты:

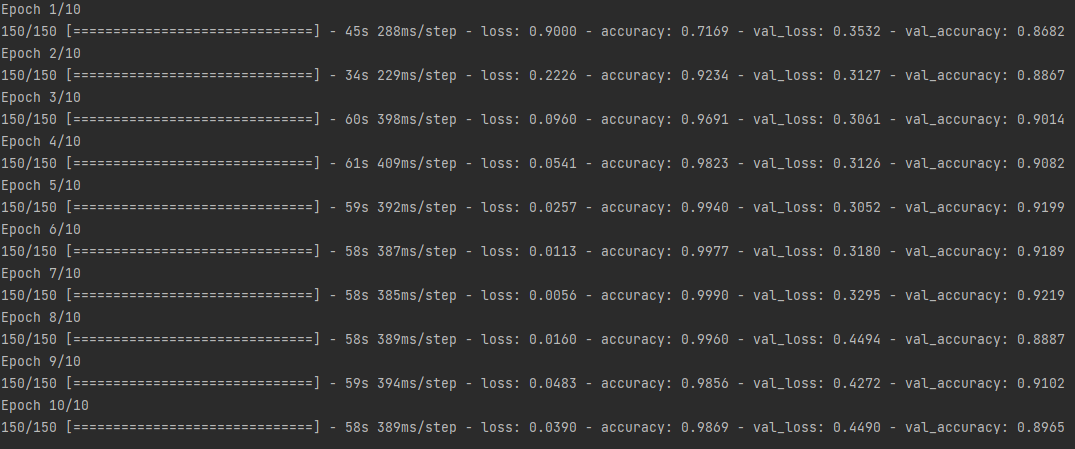


Рис. 8 – Обучение нейронной сети на архитектуре №2

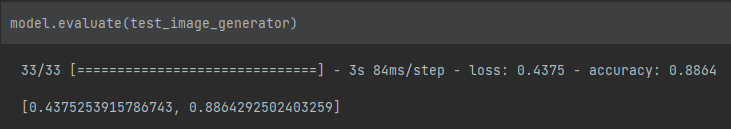


Рис. 9 – Потери и точность работы нейронной сети на тестовом наборе

архитектуры №2

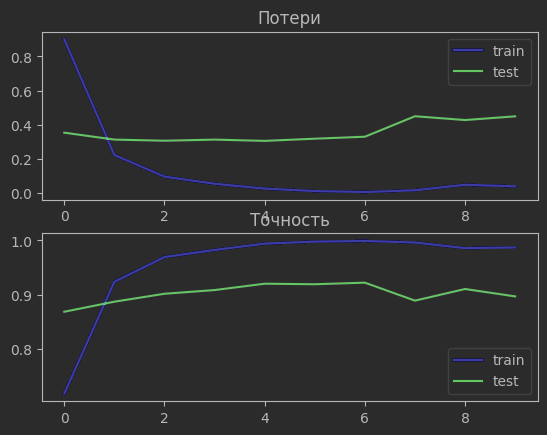


Рис. 10 – Зависимости потерь и точности модели от кол-ва эпох обучения

на архитектуре №2

В данной архитектуре были добавлены функции подвыборки после каждой функции активации, данный метод способствуем выделению основных признаков используя максимальное значение в окне каждого свертывания, в отличие от самостоятельной работы функции **Conv2D**. Это помогает нейронной сети точнее отделять одни классы от других. Однако недостатком данной архитектуры становится сильное переобучение на поздних эпохах, приводящее значение потерь на них к наибольшим значениям в архитектуре №1.

* Архитектура 3

В данной архитектуре используются те же методы что и в архитектуре №2, однако с учетом добавления функции **DropOut** – метода регуляризации нейронных сетей.

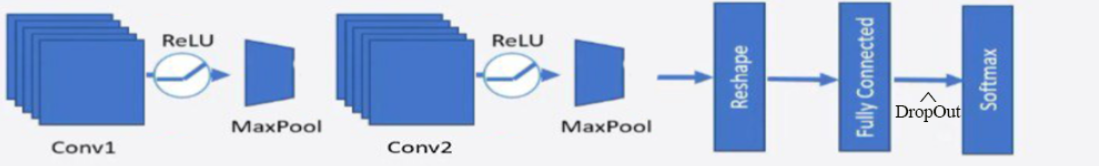


Рис. 11 – Архитектура №3

На данной архитектуре нейронной сети получаются следующие результаты:

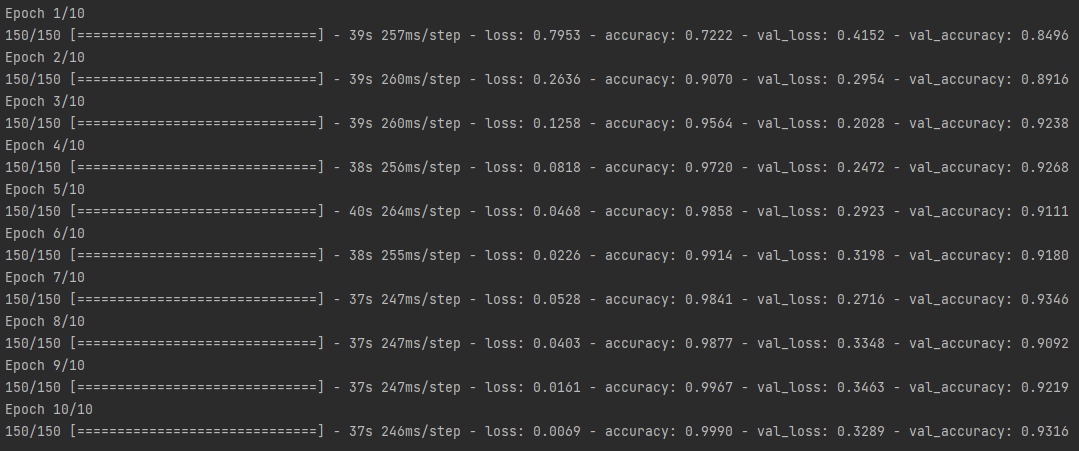


Рис. 12 – Обучение нейронной сети на архитектуре №3

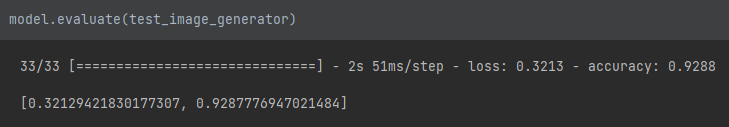


Рис. 13 –Потери и точность работы нейронной сети на тестовом наборе

архитектуры №3

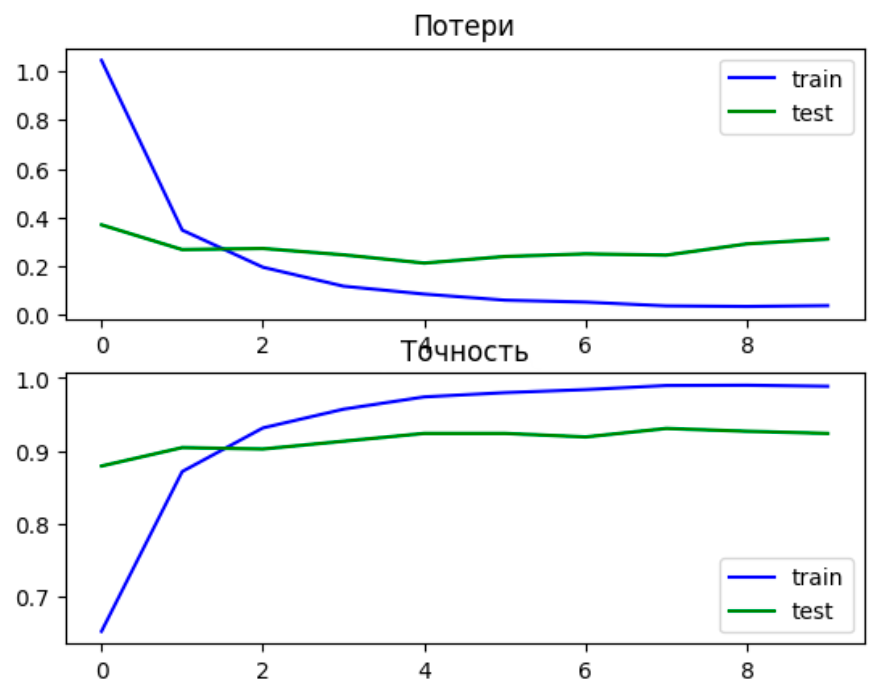


Рис. 13 – Зависимости потерь и точности модели от кол-ва эпох обучения

на архитектуре №3

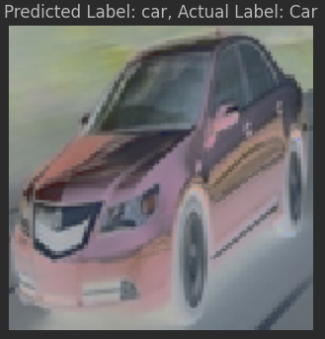


Рис. 14 – Проверка нейронной сети на тестовом наборе

Данная архитектура показала лучшие результаты, по сравнению с первыми двумя. Она лишена всех предыдущих недостатков. Функция **DropOut** уменьшает степень переобучения нейронной сети, за счет предотвращения сложных коадаптаций отдельных нейронов на тренировочных данных во время обучения. За счет этого на графиках уже не видно сильных скачков как в потерях так и в точности работы сети на тестовом наборе данных.

Также при ручном подборе гипер параметров, добавлении большего кол-ва слоев на данном датасете точность может, как улучшится так и ухудшится, порождая больше потерь. Для этого существуют уже готовые модели обучения, одна из которых описана в следующем пункте.

**Transfer learning**

Трансферное обучение – это мощный метод обучения глубоких нейронных сетей, который позволяет взять знания, полученные при обучении похожих моделей на похожих наборах данных.

В нашем случае использовалась готовая архитектура InceptionV3 состоящая из следующих 42-ух слоев:

* Conv 3×3/2 299×299×3
* Conv 3×3/1 149×149×32
* Conv padded 3×3/1 147×147×32
* Pool 3×3/2 147×147×64
* Conv 3×3/1 73×73×64
* Conv 3×3/2 71×71×80
* Conv 3×3/1 35×35×192
* 3 × Inception Module 1 35×35×288
* 5 × Inception Module 2 17×17×768
* 2 × Inception Module 3 8×8×1280
* Pool 8 × 8 8 × 8 × 2048
* Linear Logits 1 × 1 × 2048
* Softmax Classifier 1 × 1 × 1000

InceptionV3 — это модель глубокого обучения, предназначенная для распознавания и классификации изображений. Она является частью семейства моделей Inception и известна своей способностью эффективно анализировать сложные визуальные данные и предоставлять точные результаты. Компания Google разработала модель InceptionV3 в рамках своей платформы TensorFlow.

При данном подходе нейронная сеть показала следующие результаты:



Рис. 15 – Обучение нейронной сети на архитектуре InceptionV3

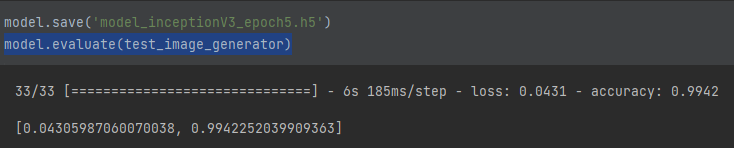


Рис. 16 –Потери и точность работы нейронной сети на тестовом наборе

архитектуры InceptionV3

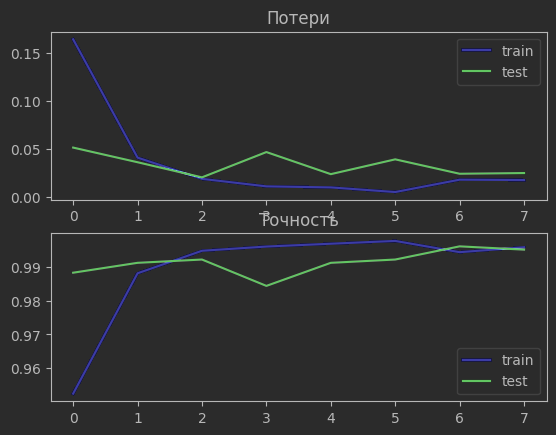


Рис. 17 – Зависимости потерь и точности модели от кол-ва эпох обучения

на архитектуре InceptionV3

# 

Рис. 18 – Проверка нейронной сети на тестовом наборе

Данная архитектура показала лучшие результаты, по сравнению с “самодельными”. Данная архитектура имеет 42 слоя, вследствие чего имеет большой процент точности обучения с малыми потерями, нейронная сеть на такой архитектуре будет исправно работать с такими же показателями и на более больших наборах входных данных, что ставит ее в приоритет при выборе модели для нейронной сети, для решения задач распознавания тех или иных объектов. На графиках заметны некоторые потери, это обусловлено большим числом эпох обучения. Для такого относительно небольшого набора данных хватит и 3-х эпох для распознавания объектов с большой точностью обучения.

# **Вывод**

В рамках проведенной лабораторной работы были разработаны и протестированы три собственные архитектуры сверточных нейронных сетей, а также применено трансферное обучение на базе готовой модели *InceptionV3*. Было отмечено, что изменение параметров слоев и функция *Dropout* влияют на результаты обучения.

Первые две разработанные архитектуры показали точность ниже 90%, что, возможно, связано с проблемой переобучения. Тем не менее, первая архитектура обладает значительно более высокой скоростью обучения, из-за малого кол-ва гипер параметров.

Применение методов предотвращения переобучения позволило увеличить точность предсказаний на тренировочном наборе данных до 93%, что на 5% выше, чем при наличии переобучения. Данный высокий показатель в основном обусловлен относительно малой выборкой входных данных, а также разностью изображений разных классов. Однако если же набор изначальных данных, даже при использовании такого деления на классы, будет в разы больше, архитектуры разработанные в данной лабораторной работе будут обучаться хуже.

Среди полностью самостоятельно созданных архитектур лучшие результаты продемонстрировала третья архитектура: она показала наилучшие метрики качества. Исходя из этого, если же перед пользователем стоит выбор между тремя данными архитектурами, рекомендуется остановить свой выбор именно на третьей архитектуре.

Однако уже готовые и протестированные модели, такие как: InceptionV3, VGG16/19 или другие известные архитектуры общего доступа, будут выдавать лучшие результаты в ходе трансферного обучения. Так модель на основе InceptionV3 продемонстрировала лучшие показатели эффективности по сравнению с моделями сделанными в ходе лабораторной работы.

# **Код программы**

**Разделение датасета на наборы**

import os

import shutil

import random

from collections import defaultdict

data\_dir = 'D:/Учёба/neironki/natural\_images/'

train\_dir = 'train'

val\_dir = 'validation'

test\_dir = 'test'

os.makedirs(train\_dir, exist\_ok=True)

os.makedirs(val\_dir, exist\_ok=True)

os.makedirs(test\_dir, exist\_ok=True)

train\_percent = 0.7

val\_percent = 0.15

test\_percent = 0.15

class\_images = defaultdict(list)

for class\_folder in os.listdir(data\_dir):

class\_path = os.path.join(data\_dir, class\_folder)

if os.path.isdir(class\_path):

class\_images[class\_folder] = [os.path.join(class\_path, img) for img in os.listdir(class\_path)]

for class\_name, images in class\_images.items():

random.shuffle(images)

class\_train\_dir = os.path.join(train\_dir, class\_name)

os.makedirs(class\_train\_dir, exist\_ok=True)

class\_val\_dir = os.path.join(val\_dir, class\_name)

os.makedirs(class\_val\_dir, exist\_ok=True)

class\_test\_dir = os.path.join(test\_dir, class\_name)

os.makedirs(class\_test\_dir, exist\_ok=True)

train\_images = images[:int(train\_percent \* len(images))]

val\_images = images[int(train\_percent \* len(images)):int((train\_percent + val\_percent) \* len(images))]

test\_images = images[int((train\_percent + val\_percent) \* len(images)):]

for img in train\_images:

shutil.copy(img, os.path.join(class\_train\_dir, os.path.basename(img)))

for img in val\_images:

shutil.copy(img, os.path.join(class\_val\_dir, os.path.basename(img)))

for img in test\_images:

shutil.copy(img, os.path.join(class\_test\_dir, os.path.basename(img)))

**Архитектура №1**

#%%

import numpy as np

import matplotlib.pyplot as plt

from tensorflow import keras

from keras.layers import \*

from keras.models import \*

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

import os

import warnings

warnings.filterwarnings('ignore')

#%%

train\_path = "D:/Учёба/neironki/train"

validation\_path = "D:/Учёба/neironki/validation"

test\_path = "D:/Учёба/neironki/test"

image\_categories = os.listdir('D:/Учёба/neironki/train')

def plot\_images(image\_categories):

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

for i, title in enumerate(image\_categories):

image\_path = train\_path + '/' + title

images\_in\_folder = os.listdir(image\_path)

first\_image\_of\_folder = images\_in\_folder[0]

first\_image\_path = image\_path + '/' + first\_image\_of\_folder

img = image.load\_img(first\_image\_path)

img\_arr = image.img\_to\_array(img)/255.0

plt.subplot(8, 8, i+1)

plt.imshow(img\_arr)

plt.title(title)

plt.axis('off')

plt.show()

plot\_images(image\_categories)

#%%

train\_gen = ImageDataGenerator(rescale = 1.0/255.0)

train\_image\_generator = train\_gen.flow\_from\_directory(

train\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

val\_gen = ImageDataGenerator(rescale = 1.0/255.0)

val\_image\_generator = train\_gen.flow\_from\_directory(

validation\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

test\_gen = ImageDataGenerator(rescale = 1.0/255.0)

test\_image\_generator = train\_gen.flow\_from\_directory(

test\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

#%%

class\_map = dict([(v, k) for k, v in train\_image\_generator.class\_indices.items()])

print(class\_map)

#%%

model = Sequential()

model.add(Conv2D(filters=32, kernel\_size=3, strides=2, padding='same', activation='relu', input\_shape=[150, 150, 3]))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(8, activation='softmax'))

model.summary()

#%%

early\_stopping = keras.callbacks.EarlyStopping(patience=7)

model.compile(optimizer='Adam', loss='categorical\_crossentropy', metrics='accuracy')

hist = model.fit(train\_image\_generator,

epochs=10,

batch\_size= 32,

verbose=1,

validation\_data=val\_image\_generator,

steps\_per\_epoch = 4826//32,

validation\_steps = 1034//32,

callbacks=early\_stopping)

#%%

model.evaluate(test\_image\_generator)

#%%

test\_image\_path = 'D:/Учёба/neironki/test/airplane/airplane\_0005.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Airplane')

#%%

test\_image\_path = 'D:/Учёба/neironki/test/car/car\_0009.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Car')

#%%

test\_image\_path = 'D:/Учёба/neironki/test/motorbike/motorbike\_0006.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Motorbike')

#%%

model.save('Model-CNN-1.h5')

#%%

h = hist.history

plt.subplot(211)

plt.title('Потери')

plt.plot(h['loss'], color='blue', label='train')

plt.plot(h['val\_loss'], color='green', label='test')

plt.legend()

plt.subplot(212)

plt.title('Точность')

plt.plot(h['accuracy'], color='blue', label='train')

plt.plot(h['val\_accuracy'], color='green', label='test')

plt.legend()

plt.show()

#%%

**Архитектура №2**

#%%

import numpy as np

import matplotlib.pyplot as plt

from tensorflow import keras

from keras.layers import \*

from keras.models import \*

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

import os

import warnings

warnings.filterwarnings('ignore')

#%%

train\_path = "D:/Учёба/neironki/train"

validation\_path = "D:/Учёба/neironki/validation"

test\_path = "D:/Учёба/neironki/test"

image\_categories = os.listdir('D:/Учёба/neironki/train')

def plot\_images(image\_categories):

plt.figure(figsize=(12, 12))

for i, cat in enumerate(image\_categories):

image\_path = train\_path + '/' + cat

images\_in\_folder = os.listdir(image\_path)

first\_image\_of\_folder = images\_in\_folder[0]

first\_image\_path = image\_path + '/' + first\_image\_of\_folder

img = image.load\_img(first\_image\_path)

img\_arr = image.img\_to\_array(img)/255.0

plt.subplot(4, 4, i+1)

plt.imshow(img\_arr)

plt.title(cat)

plt.axis('off')

plt.show()

plot\_images(image\_categories)

#%%

train\_gen = ImageDataGenerator(rescale = 1.0/255.0)

train\_image\_generator = train\_gen.flow\_from\_directory(

train\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

val\_gen = ImageDataGenerator(rescale = 1.0/255.0)

val\_image\_generator = train\_gen.flow\_from\_directory(

validation\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

test\_gen = ImageDataGenerator(rescale = 1.0/255.0)

test\_image\_generator = train\_gen.flow\_from\_directory(

test\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

#%%

class\_map = dict([(v, k) for k, v in train\_image\_generator.class\_indices.items()])

print(class\_map)

#%%

model = Sequential()

model.add(Conv2D(filters=32, kernel\_size=3, padding='same', activation='relu', input\_shape=[150, 150, 3]))

model.add(MaxPooling2D(pool\_size=(2,2), strides=2))

model.add(Conv2D(filters=64, kernel\_size=3, padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2), strides=2))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(8, activation='softmax'))

model.summary()

#%%

early\_stopping = keras.callbacks.EarlyStopping(patience=8)

model.compile(optimizer='Adam', loss='categorical\_crossentropy', metrics='accuracy')

hist = model.fit(train\_image\_generator,

epochs=10,

batch\_size= 32,

verbose=1,

validation\_data=val\_image\_generator,

steps\_per\_epoch = 4826//32,

validation\_steps = 1034//32,

callbacks=early\_stopping)

#%%

model.evaluate(test\_image\_generator)

#%%

h = hist.history

plt.subplot(211)

plt.title('Потери')

plt.plot(h['loss'], color='blue', label='train')

plt.plot(h['val\_loss'], color='green', label='test')

plt.legend()

plt.subplot(212)

plt.title('Точность')

plt.plot(h['accuracy'], color='blue', label='train')

plt.plot(h['val\_accuracy'], color='green', label='test')

plt.legend()

plt.show()

#%%

test\_image\_path = 'D:/Учёба/neironki/test/airplane/airplane\_0005.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Airplane')

#%%

test\_image\_path = 'D:/Учёба/neironki/test/car/car\_0009.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Car')

#%%

test\_image\_path = 'D:/Учёба/neironki/test/motorbike/motorbike\_0006.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Motorbike')

#%%

model.save('Model-CNN-2.h5')

**Архитектура №3**

#%%

import numpy as np

import matplotlib.pyplot as plt

from tensorflow import keras

from keras.layers import \*

from keras.models import \*

from keras.preprocessing import image

from keras.preprocessing.image import ImageDataGenerator

import os

import warnings

warnings.filterwarnings('ignore')

#%%

train\_path = "D:/Учёба/neironki/train"

validation\_path = "D:/Учёба/neironki/validation"

test\_path = "D:/Учёба/neironki/test"

image\_categories = os.listdir('D:/Учёба/neironki/train')

def plot\_images(image\_categories):

plt.figure(figsize=(12, 12))

for i, cat in enumerate(image\_categories):

image\_path = train\_path + '/' + cat

images\_in\_folder = os.listdir(image\_path)

first\_image\_of\_folder = images\_in\_folder[0]

first\_image\_path = image\_path + '/' + first\_image\_of\_folder

img = image.load\_img(first\_image\_path)

img\_arr = image.img\_to\_array(img)/255.0

plt.subplot(4, 4, i+1)

plt.imshow(img\_arr)

plt.title(cat)

plt.axis('off')

plt.show()

plot\_images(image\_categories)

#%%

train\_gen = ImageDataGenerator(rescale = 1.0/255.0)

train\_image\_generator = train\_gen.flow\_from\_directory(

train\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

val\_gen = ImageDataGenerator(rescale = 1.0/255.0)

val\_image\_generator = train\_gen.flow\_from\_directory(

validation\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

test\_gen = ImageDataGenerator(rescale = 1.0/255.0)

test\_image\_generator = train\_gen.flow\_from\_directory(

test\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

#%%

class\_map = dict([(v, k) for k, v in train\_image\_generator.class\_indices.items()])

print(class\_map)

#%%

model = Sequential()

model.add(Conv2D(filters=32, kernel\_size=3, padding='same', activation='relu', input\_shape=[150, 150, 3]))

model.add(MaxPooling2D(pool\_size=(2,2), strides=2))

model.add(Conv2D(filters=64, kernel\_size=3, padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2), strides=2))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.23))

model.add(Dense(8, activation='softmax'))

model.summary()

#%%

early\_stopping = keras.callbacks.EarlyStopping(patience=8)

model.compile(optimizer='Adam', loss='categorical\_crossentropy', metrics='accuracy')

hist = model.fit(train\_image\_generator,

epochs=10,

batch\_size= 32,

verbose=1,

validation\_data=val\_image\_generator,

steps\_per\_epoch = 4826//32,

validation\_steps = 1034//32,

callbacks=early\_stopping)

#%%

h = hist.history

plt.subplot(211)

plt.title('Потери')

plt.plot(h['loss'], color='blue', label='train')

plt.plot(h['val\_loss'], color='green', label='test')

plt.legend()

plt.subplot(212)

plt.title('Точность')

plt.plot(h['accuracy'], color='blue', label='train')

plt.plot(h['val\_accuracy'], color='green', label='test')

plt.legend()

plt.show()

#%%

model.evaluate(test\_image\_generator)

#%%

test\_image\_path = 'D:/Учёба/neironki/test/airplane/airplane\_0005.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Airplane')

#%%

test\_image\_path = 'D:/Учёба/neironki/test/car/car\_0009.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Car')

#%%

test\_image\_path = 'D:/Учёба/neironki/test/motorbike/motorbike\_0006.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Motorbike')

#%%

model.save('Model-CNN-3.h5')

**TransferLearning**

#%%

import numpy as np

import matplotlib.pyplot as plt

from tensorflow import keras

import os, pathlib, warnings

warnings.filterwarnings("ignore")

import tensorflow.keras.backend as K

from tensorflow.keras.models import Model

from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.inception\_v3 import InceptionV3

from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout

K.clear\_session()

#%%

train\_path = os.path.join("D:/Учёба/neironki/train")

test\_path = os.path.join("D:/Учёба/neironki/test")

validation\_path = os.path.join("D:/Учёба/neironki/validation")

#%%

def count\_files(rootdir):

for path in pathlib.Path(rootdir).iterdir():

if path.is\_dir():

print("There are " + str(len([name for name in os.listdir(path) \

if os.path.isfile(os.path.join(path, name))])) + " files in " + \

str(path.name))

count\_files(os.path.join(test\_folder))

#%%

images = []

for food\_folder in sorted(os.listdir(train\_folder)):

food\_items = os.listdir(train\_folder + '/' + food\_folder)

food\_selected = np.random.choice(food\_items)

images.append(os.path.join(train\_folder,food\_folder,food\_selected))

fig=plt.figure(1, figsize=(15, 10))

for subplot,image\_ in enumerate(images):

category=image\_.split('/')[-2]

imgs = plt.imread(image\_)

a,b,c=imgs.shape

fig=plt.subplot(3, 5, subplot+1)

fig.set\_title(category, pad = 10,size=18)

plt.imshow(imgs)

plt.tight\_layout()

#%%

IMAGE\_SIZE = [150, 150]

inception = InceptionV3(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)

for layer in inception.layers:

layer.trainable = False

x = inception.output

x = GlobalAveragePooling2D()(x)

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

x = Dropout(0.2)(x)

prediction = Dense(8, activation='softmax')(x)

model = Model(inputs=inception.input, outputs=prediction)

model.compile(

loss='categorical\_crossentropy',

optimizer='adam',

metrics=['accuracy']

)

#%%

train\_gen = image.ImageDataGenerator(rescale = 1.0/255.0)

train\_image\_generator = train\_gen.flow\_from\_directory(

train\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

val\_gen = image.ImageDataGenerator(rescale = 1.0/255.0)

val\_image\_generator = train\_gen.flow\_from\_directory(

validation\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

test\_gen = image.ImageDataGenerator(rescale = 1.0/255.0)

test\_image\_generator = train\_gen.flow\_from\_directory(

test\_path,

target\_size=(150, 150),

batch\_size=32,

class\_mode='categorical')

#%%

class\_map = training\_set.class\_indices

class\_map

#%%

early\_stopping = keras.callbacks.EarlyStopping(patience=5)

r = model.fit\_generator(

train\_image\_generator,

validation\_data=val\_image\_generator,

epochs=10,

verbose=1,

steps\_per\_epoch = 4826//32,

validation\_steps = 1034//32,

callbacks=early\_stopping

)

#%%

model.save('model\_inceptionV3\_epoch5.h5')

model.evaluate(test\_image\_generator)

#%%

class\_map = dict([(v, k) for k, v in train\_image\_generator.class\_indices.items()])

print(class\_map)

test\_image\_path = 'D:/Учёба/neironki/test/person/person\_0004.jpg'

def generate\_predictions(test\_image\_path, actual\_label):

test\_img = image.load\_img(test\_image\_path, target\_size=(150, 150))

test\_img\_arr = image.img\_to\_array(test\_img)/255.0

test\_img\_input = test\_img\_arr.reshape((1, test\_img\_arr.shape[0], test\_img\_arr.shape[1], test\_img\_arr.shape[2]))

predicted\_label = np.argmax(model.predict(test\_img\_input))

predicted\_vegetable = class\_map[predicted\_label]

plt.figure(figsize=(4, 4))

plt.imshow(test\_img\_arr)

plt.title("Predicted Label: {}, Actual Label: {}".format(predicted\_vegetable, actual\_label))

plt.grid()

plt.axis('off')

plt.show()

generate\_predictions(test\_image\_path, actual\_label='Person')

#%%

h = r.history

plt.subplot(211)

plt.title('Потери')

plt.plot(h['loss'], color='blue', label='train')

plt.plot(h['val\_loss'], color='green', label='test')

plt.legend()

plt.subplot(212)

plt.title('Точность')

plt.plot(h['accuracy'], color='blue', label='train')

plt.plot(h['val\_accuracy'], color='green', label='test')

plt.legend()

plt.show()

#%%

#%%