МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

**Сверточные нейронные сети. Модели VGG**

Отчет по практической работе №2  
по дисциплине

«Информационные системы искусственного интеллекта в медицине»

Выполнил студент гр. ИВТм-1302-03-00 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Морозов И.В./

(Подпись)

Руководитель к.т.н., доцент  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Крутиков А.К./

(Подпись)

Работа защищена с оценкой «\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2025 г.

Киров 2025

Цель: изучение работы искусственных сверточных нейронных сетей, в частности моделей VGG.

Задание: Для выполнения лабораторной работы необходимо решить задачи с использованием VGG16 и VGG19 моделей нейронный сетей сравнить результаты моделей по каждой из них.

Задача 1: Обучить и использовать модели VGG16 и VGG19 для задач классификации по 6 заданным классам.

Набор исходных данных представляет собой простое медицинское изображение в стиле MNIST размером 64x64;

Набор данных содержит 58954 медицинских изображения, относящихся к 6 классам – КТ грудной клетки (10000 изображений), МРТ молочной железы (8954 изображения), CXR (10000 изображений), руки (10000 изображений), КТ головы (10000 изображений), брюшной полости (10000 изображений). Изображения имеют размеры 64 × 64 пикселя.

Заполнение таблицы для каждой из нейросетевых моделей, в которой включить наборы параметров настройки, характерные для моделей, для каждого из примеров обучения и использования сети (эксперимента) представлено ниже.

Неизменяемые параметры:

* Оптимизатор – Adam.
* Количество эпох – 5.
* Размер бэтча – 32.
* Loss функция – сategorical\_crossentropy.

Таблица 1 – Сравнение моделей VGG16 и VGG19

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | Тип модели | Функции  активации  слоя свертки  Блока №1 | Функции  Активации слоя  свертки  Блока №2 | Функции  активации слоя  свертки  Блока №3 | Функции  активации слоя  свертки  Блока №4 | Функции  активации слоя  свертки  Блока №5 | Метрика (Accuracy) |
| **1** | **VGG16** | **relu** | **relu** | **relu** | **relu** | **relu** | **0.98438** |
| 2 | VGG16 | elu | elu | elu | elu | elu | 0.97542 |
| 3 | VGG16 | gelu | gelu | gelu | gelu | gelu | 0.97229 |
| 4 | VGG19 | relu | relu | relu | relu | relu | 0.88625 |
| 5 | VGG19 | elu | elu | elu | elu | elu | 0.96260 |
| 6 | VGG19 | gelu | gelu | gelu | gelu | gelu | 0.95458 |

Таким образом наилучшую точность показала сеть VGG16 с функцией активации relu, результат проведенных экспериментов продемонстрирован на рисунке 1.

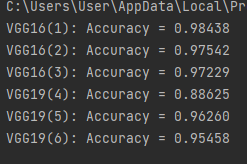


Рисунок 1 – Результат проведения экспериментов

Вывод: В ходе выполнения практической работы были изучены принципы работы сверточных нейронных сетей VGG16 и VGG19, и получены навыки обучения для классификации медицинских изображений по 6 классам.

Рассмотрены различные функции активации в сверточных слоях и их влияние на результаты:

* VGG16 с ReLU показала наибольшую точность.
* VGG19 в данной задаче показала менее стабильные результаты.

Вероятно, для более успешного обучения VGG19 на этом наборе данных требуется больше эпох, изменения размера входных данных или другие модификации гиперпараметров.

**Приложение A**

(обязательное)

Листинг файла task1.py

def limit\_images\_per\_class(dataset\_path, num\_images\_per\_class):

if not os.path.exists(dataset\_path):

print(f"Путь к датасету не найден: {dataset\_path}")

return

for class\_name in os.listdir(dataset\_path):

class\_dir = os.path.join(dataset\_path, class\_name)

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

images = [f for f in os.listdir(class\_dir) if os.path.isfile(os.path.join(class\_dir, f))]

if len(images) > num\_images\_per\_class:

# Сортируем изображения, чтобы удаление было детерминированным (опционально)

images.sort()

for i in range(num\_images\_per\_class, len(images)):

image\_to\_remove = os.path.join(class\_dir, images[i])

try:

os.remove(image\_to\_remove)

# print(f"Удалено: {image\_to\_remove}")

except OSError as e:

print(f"Ошибка при удалении файла {image\_to\_remove}: {e}")

#print(f"Класс '{class\_name}': осталось {len(os.listdir(class\_dir))} изображений.")

def split\_dataset(source\_dir, train\_dir, test\_dir, test\_size=0.2, random\_state=42):

if not os.path.exists(source\_dir):

print(f"Исходный путь к датасету не найден: {source\_dir}")

return

# Удаление тренировочных и тестовых папок, если они существуют

if os.path.exists(train\_dir):

shutil.rmtree(train\_dir)

if os.path.exists(test\_dir):

shutil.rmtree(test\_dir)

# Создание тренировочных и тестовых папок

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

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

# Проходим по каждой подпапке (классу) в исходной директории

for class\_name in os.listdir(source\_dir):

class\_source\_dir = os.path.join(source\_dir, class\_name)

# Убедимся, что это директория

if os.path.isdir(class\_source\_dir):

# Создаем подпапки для этого класса в тренировочной и тестовой директориях

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

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

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

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

# Получаем список всех файлов (изображений) в текущем классе

images = [f for f in os.listdir(class\_source\_dir) if os.path.isfile(os.path.join(class\_source\_dir, f))]

# Разделяем список изображений на тренировочную и тестовую части

if len(images) > 0:

train\_images, test\_images = train\_test\_split(images, test\_size=test\_size, random\_state=random\_state)

# Копируем тренировочные изображения в соответствующую папку

for img in train\_images:

src\_path = os.path.join(class\_source\_dir, img)

dest\_path = os.path.join(class\_train\_dir, img)

shutil.copy(src\_path, dest\_path)

# Копируем тестовые изображения в соответствующую папку

for img in test\_images:

src\_path = os.path.join(class\_source\_dir, img)

dest\_path = os.path.join(class\_test\_dir, img)

shutil.copy(src\_path, dest\_path)

#print(f"Разделен класс '{class\_name}': {len(train\_images)} для тренировки, {len(test\_images)} для теста.")

def load\_dataset(train\_dataset\_path, test\_dataset\_path, image\_size=(224, 224), batch\_size=32):

try:

datagen = ImageDataGenerator(rescale=1./255)

train\_generator = datagen.flow\_from\_directory(

train\_dataset\_path,

target\_size=image\_size,

batch\_size=batch\_size,

class\_mode='categorical',

color\_mode='rgb')

test\_generator = datagen.flow\_from\_directory(

test\_dataset\_path,

target\_size=image\_size,

batch\_size=batch\_size,

class\_mode='categorical',

color\_mode='rgb')

return train\_generator, test\_generator

except Exception as e:

print(f"Ошибка при загрузке датасета: {e}")

return None, None

def create\_model\_vgg16(classes, input\_shape=(224, 224, 3), activation\_block1='relu', activation\_block2='relu',

activation\_block3='relu', activation\_block4='relu', activation\_block5='relu',

activation\_dense='relu'):

# Построение и компиляция модели VGG16

model = Sequential()

# Блок 1: два слоя свёртки + pooling

model.add(Input(input\_shape))

model.add(Conv2D(64, (3, 3), activation=activation\_block1, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(64, (3, 3), activation=activation\_block1, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Блок 2: два слоя свёртки + pooling

model.add(Conv2D(128, (3, 3), activation=activation\_block2, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(128, (3, 3), activation=activation\_block2, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Блок 3: три слоя свёртки + pooling

model.add(Conv2D(256, (3, 3), activation=activation\_block3, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(256, (3, 3), activation=activation\_block3, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(256, (3, 3), activation=activation\_block3, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Блок 4: три слоя свёртки + pooling

model.add(Conv2D(512, (3, 3), activation=activation\_block4, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block4, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block4, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Блок 5: три слоя свёртки + pooling

model.add(Conv2D(512, (3, 3), activation=activation\_block5, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block5, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block5, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Полносвязная сеть

model.add(Flatten())

model.add(Dense(4096, activation=activation\_dense))

model.add(Dropout(0.5))

model.add(Dense(4096, activation=activation\_dense))

model.add(Dropout(0.5))

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

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy', 'mse'])

return model

def create\_model\_vgg19(classes, input\_shape=(224, 224, 3), activation\_block1='relu', activation\_block2='relu',

activation\_block3='relu', activation\_block4='relu', activation\_block5='relu',

activation\_dense='relu'):

# Построение и компиляция модели VGG19

model = Sequential()

# Блок 1: два слоя свёртки + pooling

model.add(Input(input\_shape))

model.add(Conv2D(64, (3, 3), activation=activation\_block1, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(64, (3, 3), activation=activation\_block1, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Блок 2: два слоя свёртки + pooling

model.add(Conv2D(128, (3, 3), activation=activation\_block2, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(128, (3, 3), activation=activation\_block2, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Блок 3: четыре слоя свёртки + pooling

model.add(Conv2D(256, (3, 3), activation=activation\_block3, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(256, (3, 3), activation=activation\_block3, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(256, (3, 3), activation=activation\_block3, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(256, (3, 3), activation=activation\_block3, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Блок 4: четыре слоя свёртки + pooling

model.add(Conv2D(512, (3, 3), activation=activation\_block4, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block4, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block4, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block4, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Блок 5: четыре слоя свёртки + pooling

model.add(Conv2D(512, (3, 3), activation=activation\_block5, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block5, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block5, padding='same'))

model.add(BatchNormalization())

model.add(Conv2D(512, (3, 3), activation=activation\_block5, padding='same'))

model.add(BatchNormalization())

model.add(MaxPooling2D((2, 2), strides=(2, 2)))

# Полносвязная сеть

model.add(Flatten())

model.add(Dense(4096, activation=activation\_dense))

model.add(Dropout(0.5))

model.add(Dense(4096, activation=activation\_dense))

model.add(Dropout(0.5))

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

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy', 'mse'])

return model

def train\_model(model, train\_generator, test\_generator, epochs, batch\_size, verbose):

history = model.fit(

train\_generator,

epochs=epochs,

batch\_size=batch\_size,

verbose=verbose,

validation\_data=test\_generator,

)

return history, model

# Ограничение количества изображений на класс

limit\_images\_per\_class('labs/lab2/task1/content', 2000)

# Деление выборки на тренировочную и тестовую

split\_dataset('labs/lab2/task1/content', 'labs/lab2/task1/dataset/train', 'labs/lab2/task1/dataset/test', test\_size=0.2, random\_state=42)

# Создание генераторов для тренировки и тестирования

train\_generator, test\_generator = load\_dataset('labs/lab2/task1/dataset/train', 'labs/lab2/task1/dataset/test', image\_size=(64, 64))

# Создание папки для сохранения обученных моделей

os.makedirs('labs/lab2/task1/models', exist\_ok=True)

# Тренировка и вывод точности моделей VGG16

hidden\_layer\_activations = ['relu', 'leaky\_relu', 'elu', 'gelu', 'swish']

models\_vgg16 = []

trained\_models\_vgg16 = []

historys\_vgg16 = []

for i in range(len(hidden\_layer\_activations)):

model = create\_model\_vgg16(6, (64, 64, 3), hidden\_layer\_activations[i], hidden\_layer\_activations[i], hidden\_layer\_activations[i], hidden\_layer\_activations[i], hidden\_layer\_activations[i])

models\_vgg16.append(model)

history, trained\_model = train\_model(model, train\_generator, test\_generator, 5, 32, 0)

trained\_models\_vgg16.append(trained\_model)

historys\_vgg16.append(history)

trained\_model.save(f'labs/lab2/task1/models/model\_vgg16({i+1}).keras')

print('VGG16(' + str(i+1) + '): ' + 'Accuracy = ' + '{:.5f}'.format(historys\_vgg16[i].history['accuracy'][-1]))

# Тренировка и вывод точности моделей VGG19

models\_vgg19 = []

trained\_models\_vgg19 = []

historys\_vgg19 = []

for i in range(len(hidden\_layer\_activations)):

model = create\_model\_vgg19(6, (64, 64, 3), hidden\_layer\_activations[i], hidden\_layer\_activations[i], hidden\_layer\_activations[i], hidden\_layer\_activations[i], hidden\_layer\_activations[i])

models\_vgg19.append(model)

history, trained\_model = train\_model(model, train\_generator, test\_generator, 5, 32, 0)

trained\_models\_vgg19.append(trained\_model)

historys\_vgg19.append(history)

trained\_model.save(f'labs/lab2/task1/models/model\_vgg19({i+1}).keras')

print('VGG19(' + str(i+1) + '): ' + 'Accuracy = ' + '{:.5f}'.format(historys\_vgg19[i].history['accuracy'][-1]))