МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»

Институт ИТКН

Кафедра инженерной кибернетики

Направление подготовки: «01.03.04 Прикладная математика»

Квалификация: бакалавр

Группа: БПМ-17-2

ОТЧЕТ

ПО КУРСОВОЙ РАБОТЕ «ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ»

на тему: «Сверточные сети. Сеть VGG»

Студент _____/ Скибин К.С.

Руководитель _____/ старший преподаватель, Кондыбаева А.Б.

Оценка: _____

Дата защиты: _____

Москва 2020

Содержание

1. Введение

В течение последних 10 лет с появлением мощных видеокарт, позволяющих производить параллельные расчеты с очень высокой скоростью, интерес исследователей вновь вернулся к нейронным сетям со сверточными архитектурами. Обучение таких моделей наконец-то стало осуществимой задачей. Первые прототипы глубоких сверточных сетей смогли показать ошеломительные результаты в области распознавания образов на изображениях и видеофайлах высокого качества, превзойдя полносвязные модели. Успех сверточных моделей обусловлен возможностью учета двумерной топологии изображения, в отличие от многослойного персептрона.

Сверточные нейронные сети обеспечивают частичную устойчивость к изменениям масштаба, смещениям, поворотам, смене ракурса и прочим искажениям. Сверточные нейронные сети объединяют три архитектурных идеи, для обеспечения инвариантности к изменению масштаба, повороту сдвигу и пространственным искажениям:

- локальные рецепторные поля (обеспечивают локальную двумерную связность нейронов);
- общие синаптические коэффициенты (обеспечивают детектирование некоторых черт в любом месте изображения и уменьшают общее число весовых коэффициентов);
- иерархическая организация с пространственными подвыборками.

Целью данной работы является исследование, разработка и тестирование модели на глубоких сверточных сетях VGG, сумевшей победить в конкурсе “Large Scale Visual Recognition Challenge 2014” в номинации классификации изображений.

Тема данной работы является актуальной, поскольку сверточная архитектура до сих пор остаётся одним из мощнейших инструментов в руках специалистов по машинному обучению и используется в большинстве современных моделей, работающих с изображениями и видеопотоком.

2. Описание модели

VGG16 — модель сверточной нейронной сети, предложенная K. Simonyan и A. Zisserman из Оксфордского университета в статье “Very Deep Convolutional Networks for Large-Scale Image Recognition”. Модель достигает точности 92.7% — топ-5, при тестировании на ImageNet в задаче распознавания объектов на изображении. Этот датасет состоит из более чем 14 миллионов изображений, принадлежащих к 1000 классам [1].

VGG16 — одна из самых знаменитых моделей, отправленных на соревнование ILSVRC-2014. Она является улучшенной версией AlexNet, в которой заменены большие фильтры на несколько фильтров размера 3x3, следующих один за другим. Сеть VGG16 обучалась 3 недель при использовании видеокарт NVIDIA TITAN BLACK.

2.1. Датасет

В качестве корпуса, на котором происходило обучение и тестирование модели VGG, использовался датасет ImageNet. ImageNet — набор данных, состоящий из более чем 15 миллионов размеченных высококачественных изображений, разделенных на 22000 категорий. Изображения были взяты из интернета и размечены вручную людьми-разметчиками с помощью краудсорсинговой площадки Mechanical Turk от Amazon.

В 2010 году, как часть Pascal Visual Object Challenge, началось ежегодное соревнование — ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). В ILSVRC используется подвыборка из ImageNet размером 1000 изображений в каждой из 1000 категорий. Таким образом, тренировочный сет состоял из примерно 1.2 миллионов изображений, проверочный — 50000 изображений, тестовый — 150000 изображений. Так как ImageNet состоит из изображений разного размера, то их необходимо было привести к единому размеру 256x256.

Если изображение представляет из себя прямоугольник, то оно масштабируется и из него вырезается центральная часть размером 256x256.

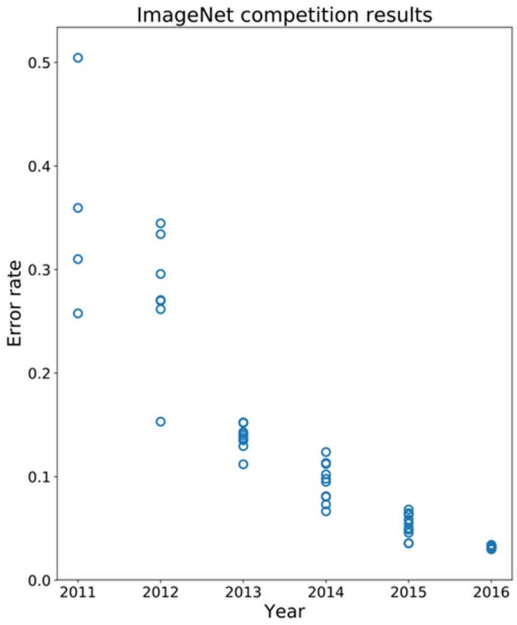

Рисунок 1 – Ошибка моделей, обученных на ImageNet, в зависимости от года их создания

2.2. Архитектура

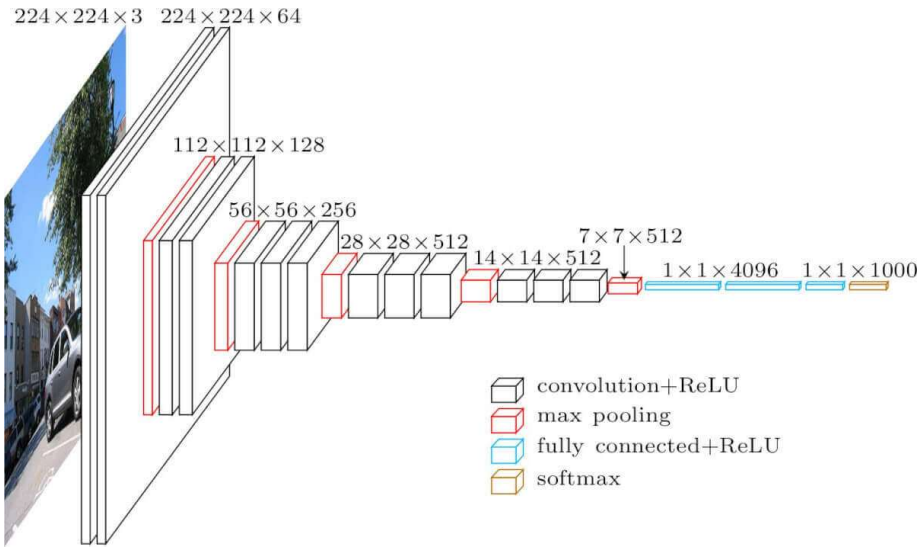

Рисунок 2 – Архитектура модели VGG

Модель VGG является последовательной и глубокой. В целом её архитектура довольно проста и состоит из сверточных и полносвязных слоёв.

На вход модели подаётся RGB изображение размером 224x224 пикселя. Требованием по предобработке изображения является его нормализация. Из каждого пикселя изображения на входе необходимо вычесть средние значения по R, G, B каналам, полученные из изображений обучающей выборки:

$$R_{mean} = 103.939,$$
$$G_{mean} = 116.779,$$
$$B_{mean} = 123.680.$$

Далее изображения проходят через стек сверточных слоев, в которых используются фильтры с очень маленьким рецептивным полем размера 3x3 (который является наименьшим размером для получения представления о том,где находится право/лево, верх/низ, центр).

В одной из конфигураций используется сверточный фильтр размера 1x1, который может быть представлен как линейная трансформация входных каналов (с последующей нелинейностью). Сверточный шаг фиксируется на значении 1 пиксель. Пространственное дополнение (padding) входа сверточного слоя выбирается таким образом, чтобы пространственное разрешение сохранялось после свертки, то есть дополнение равно 1 для 3x3 сверточных слоев. Пространственный пулинг осуществляется при помощи пяти max-pooling слоев, которые следуют за одним из сверточных слоев (не все сверточные слои имеют последующие max-pooling). Операция max-pooling выполняется на окне размера 2x2 пикселей с шагом 2.

После стека сверточных слоев (который имеет разную глубину в разных архитектурах) идут три полносвязных слоя: первые два имеют по 4096 каналов, третий — 1000 каналов (так как в соревновании ILSVRC требуется классифицировать объекты по 1000 категориям; следовательно, классу соответствует один канал). Последним идет soft-max слой.

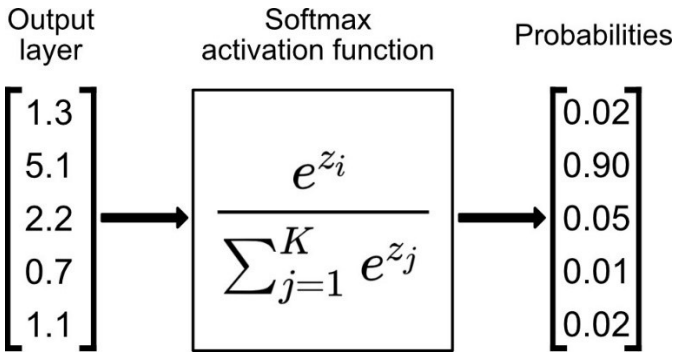

Рисунок 3 – Формула активационной функции SoftMax

В качестве активацонных функций во всех слоях, кроме последнего, а также max-pooling, используется ReLU, обеспечивающий достаточный уровень нелинейности.

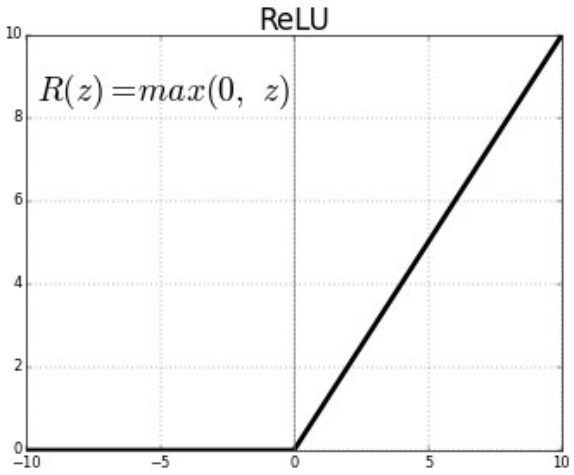

Рисунок 4 – Функция активации Rectified Linear Unit

Архитектура VGG не содержит слоя нормализации (Local Response Normalisation) в отличие от модели AlexNet, так как после проведенного тестирования было выяснено, что нормализация не улучшает результата на датасете ILSVRC, а ведет к увеличению потребления памяти и времени исполнения кода.

2.3. Конфигурации сети

В ходе разработки модели VGG разработчиками было представлено множество конфигураций архитектуры сети. Каждая сеть соответствует своему имени (A-E). Все конфигурации имеют общую конструкцию, представленную в архитектуре, и различаются только глубиной: от 11 слоев с весами в сети A (8 сверточных и 3 полносвязных слоя) до 19 (16 сверточных и 3 полносвязных слоя). Ширина сверточных слоев (количество каналов) относительно небольшая: от 64 в первом слое до 512 в последнем с увеличением количества каналов в 2 раза после каждого max-pooling слоя. Наивысшие результаты на конкурсе ILSVRC продемонстрировала архитектура E, что объясняется её увеличенной глубиной по сравнению с остальными моделями.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Рисунок 5 – Конфигурации архитектуры сети модели VGG

2.4. Оценка качества работы

К недостаткам модели VGG можно отнести следующее:

- очень медленная скорость обучения, объясняемая обилием сверточных слоёв в архитектуре;
- большой размер модели (файла с весами).

Из-за глубины и количества полносвязных узлов, VGG16 весит более 533 МБ. Это делает процесс развертывания VGG утомительной задачей. Хотя VGG16 и используется для решения многих проблем классификации при помощи нейронных сетей, меньшие архитектуры более предпочтительны (SqueezeNet, GoogLeNet и другие). Несмотря на недостатки, данная архитектура является отличным строительным блоком для обучения, так как её легко реализовать.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

Рисунок 6 – Ошибки предсказаний различных моделей на конкурсе ILSVRC

Также модель VGG на время создания показывала очень высокие результаты в задаче классификации изображений. VGG16 существенно превосходит в производительности прошлые поколения моделей в соревнованиях ILSVRC-2012 and ILSVRC-2013. Достигнутый VGG16 результат сопоставим с победителем соревнования по классификации (GoogLeNet с ошибкой 6.7%) в 2014 году и значительно опережает результат Clarifai

победителя ILSVRC-2013, который показал ошибку 11.2% с внешними тренировочными данными и 11.7% без них. Что касается одной сети, архитектура VGG16 достигает наилучшего результата (7.0% ошибки на тесте), опережаю одну сеть GoogLeNet на 0.9%.

Было показано, что глубина представления положительно влияет на точность классификации, и наилучший результат на соревновательном датасете ImageNet может быть достигнут с помощью обычной сверточной нейронной сети с значительно большей глубиной.

3. Реализация модели

3.1. Используемые программные средства

Реализация модели в рамках данной работы производилась на облачной платформе Colab, на языке Python. Список используемых библиотек:

- Keras – построение архитектуры модели, осуществление предсказаний, просмотр карт характеристик (feature maps) для анализа работы модели;
- OpenCV – загрузка и предобработка изображений;
- Matplotlib – построение графиков и диаграмм;
- NumPy – работа с векторами и массивами.

3.2. Разработанные классы

Для достижения поставленной цели работы был разработан класс VGG_16, инкапсулирующий в себе логику работы модели VGG. В классе реализовано 4 метода:

- __init__ – конструктор класса, в котором производится создание модели загрузка заранее скаченных весов модели (самостоятельное обучение модели не осуществлялось по соображениям экономии времени), компиляция модели, создание отдельной модели для просмотра карт характеристик (см. Приложение А);
- summary – метод, выводящий основную информацию о модели;
- inference – метод, принимающий в качестве аргумента путь img_path к изображению, которое необходимо классифицировать. Метод осуществляет загрузку, предобработку изображения, а также предсказание модели на нём и возвращает вероятностное распределение принадлежности изображения к одному из 1000 классов;

- show_feature_maps – метод, принимающий в качестве аргумента путь img_path к изображению и индекс слоя модели fmap_index, по которому необходимо построить карты характеристик, осуществляя прогонку изображения по пути img_path.

```
Model: "sequential_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

Рисунок 7 – Вывод метода summary

При помощи метода summary можно детально ознакомиться со структурой нейронной сети, используемой в модели. Также благодаря этому методу была получена информация о количестве тренируемых весов модели. Их более 130 миллионов. Наибольшее количество весов содержится в полносвязных слоях.

3.3. Тестирование

Для тестирования модели были скачаны три изображения из сети, найденные по запросам: сэндвич, пляж, письменный стол. Использовался метод inference из объекта класса VGG_16.

Рисунок 8 – Изображения, использованные для тестирования

Для построения гистограмм распределения вероятности распределения по классам использовались методы pyploy.plot и ravel:

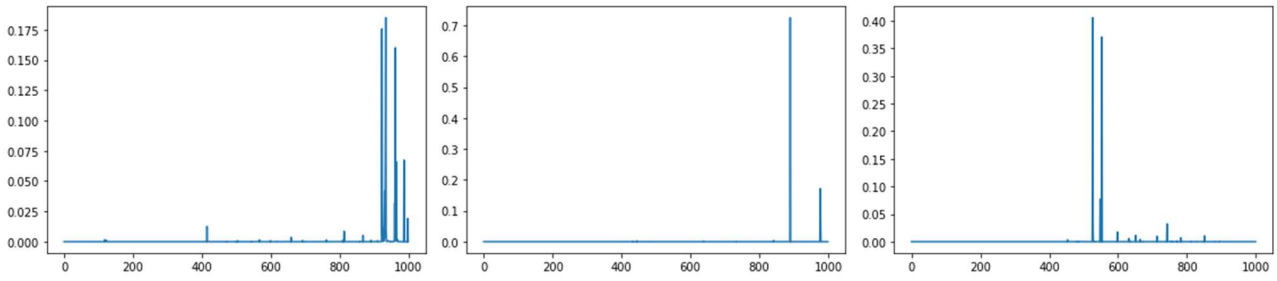

Рисунок 9 – Полученные гистограммы распределений классов для данных изображений

С точки зрения пользователя номера классов малоинформативны. Поэтому результаты предсказаний были переведены в текстовые метки на английском

языке при помощи метода decode_predictions из keras.applications.imagenet_utils. Топ-5 меток для каждого из изображений, вместе с вероятностями приведены на таблице ниже:

Таблица 1 – Топ-5 меток для изображений с рис. 8

Сэндвич		Пляж		Письменный стол	
Метка	Вероятность	Метка	Вероятность	Метка	Вероятность
Хот-дог	18%	Волейбол	72%	Стол	40%
Тарелка	17%	Песок	17%	Файл	37%
Мясо	16%	Берег	9%	ТРЦ	8%
Чизбургер	11%	Плавки	0.4%	Принтер	3%
Кукуруза	6%	Бикини	0.2%	Компьютер	3%

Полученные предсказания по изображению пляжа с первого взгляда являются совершенно нелогичными. Однако такое поведение модели можно объяснить предположением, что данные обучающей выборки, где фигурировали пляж и люди, чаще всего были фотографиями пляжного волейбола.

В целом результаты работы модели можно считать удовлетворительными, так как в топ-5 классов, как минимум 3 класса, определенные моделью, действительно имели отношение к изображению.

3.4. Визуализация работы

Для визуализации работы разработанной модели было принято решение использовать карты характеристик. Карты характеристик позволяют понять, какие именно паттерны изображения выделяют фильтры на различных сверточных слоях. Для просмотра карт характеристик используется метод show_feature_maps. Всего можно проследить за работой 6 слоёв, выбирая их при помощи аргумента функции fmap_index. Ниже приведена таблица соответствий между индексами fmap_index и слоями модели:

Таблица 2 – Соответствие между индексами и солями модели с рис. 7

fmap_index	Наименование слоя модели
0	block1_conv1
1	block2_conv1
2	block3_conv1
3	block4_conv1
4	block5_conv1
5	block5_conv3

Ниже приведены карты характеристик письменного слоя у слоёв block1_conv1, block3_conv1, block5_conv3:

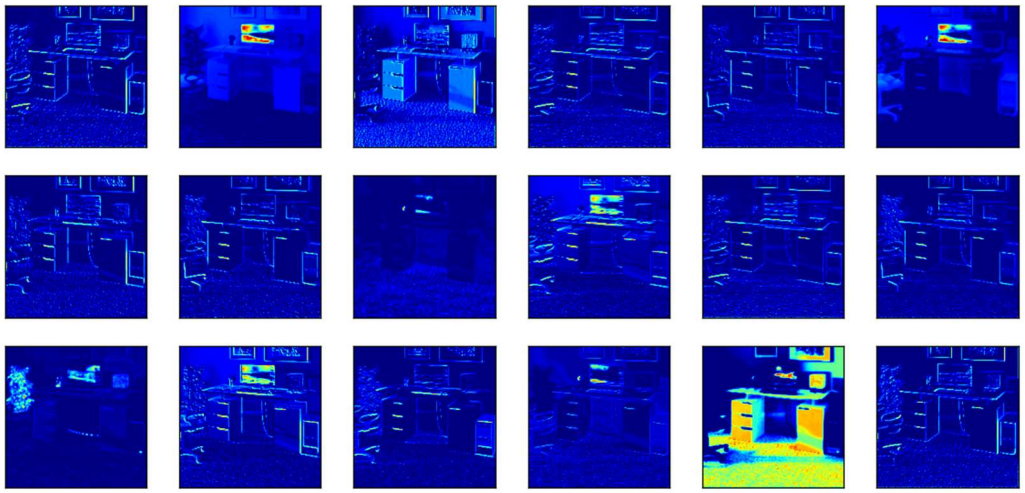

Рисунок 10 – Первые 36 карт характеристик block1_conv1

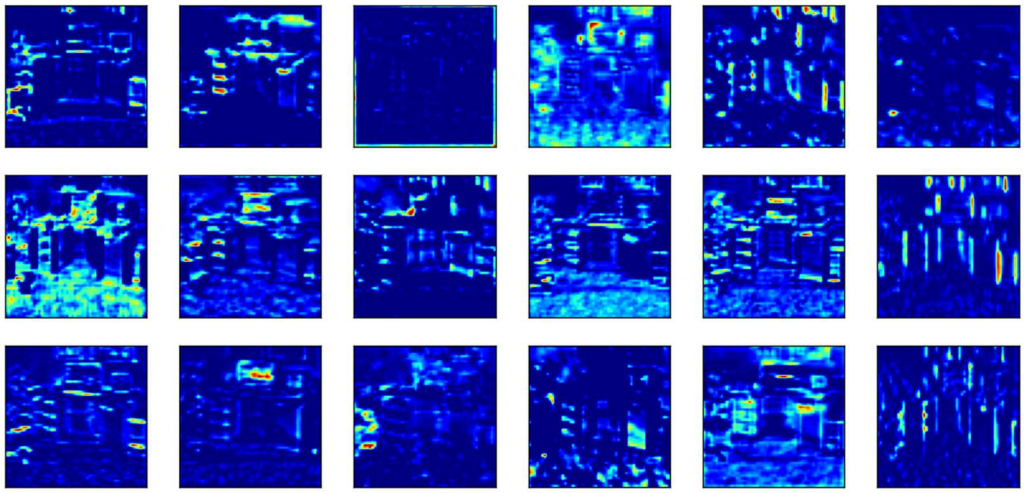

Рисунок 11 – Первые 36 карт характеристик block3_conv1

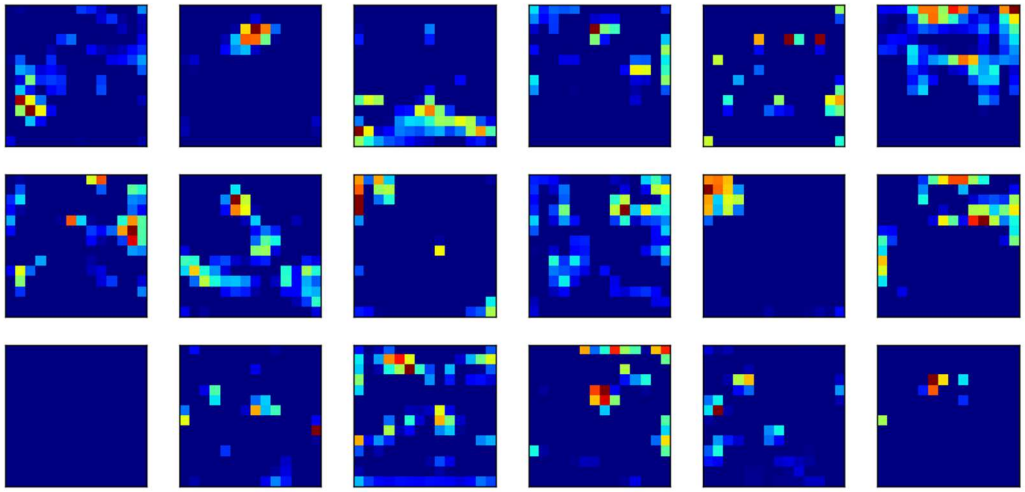

Рисунок 12 – Первые 36 карт характеристик block5_conv3

На картах характеристик поверхностных слоёв обычно выделяются однородные цвета и простая геометрия. С глубиной сети уровень абстракций, с которыми работает модель увеличивается, последние слои обычно выделяют сложные формы и композиции.

4. Заключение

Таким образом, в рамках данной работы была исследована архитектура модели VGG, проанализированы её сильные и слабые стороны. С использованием программных средств данная модель была с нуля реализована и протестирована на различных примерах изображений. Модель была проанализирована при помощи визуализации работы её сверточных слоёв на разных уровнях. По результатам тестирования сделан вывод, что модель даёт прогнозы удовлетворительного уровня, несмотря на свою простую структуру. Тем не менее данная модель в настоящее время является устаревшей и нерелевантной.

5. Список использованных источников

1. Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition – 2014.

Приложение. Класс VGG_16

```
class VGG_16:
    def __init__(self, weights_path=None):
        # модель - последовательная
        # у каждого слоя один вход и один выход
        self.model = Sequential()

        # вход - изображения размером 224x224, 3 канала для цветов
        # окружаем нулями с каждой стороны при помощи ZeroPadding2D

        ## 1 блок
        # 2 сверточных слоя по 64 каналов с фильтрами 3x3 и паддингом 1
        # активационная функция - ReLU
        # в конце - извлечение максимумов из ячеек 2x2 с шагом 2
        self.model.add(Convolution2D(64, (3, 3), activation='relu', padding='same',
                                     name='block1_conv1', input_shape=(224,224,3)))
        self.model.add(Convolution2D(64, (3, 3), activation='relu', padding='same',
                                     name='block1_conv2'))
        self.model.add(MaxPooling2D((2,2), strides=(2,2), padding='same',
                                    name='block1_pool'))

        ## 2 блок
        # 2 сверточных слоя по 128 каналов с фильтрами 3x3 и паддингом 1
        # активационная функция - ReLU
        # в конце - извлечение максимумов из ячеек 2x2 с шагом 2
        self.model.add(Convolution2D(128, (3, 3), activation='relu', padding='same',
                                     name='block2_conv1'))
        self.model.add(Convolution2D(128, (3, 3), activation='relu', padding='same',
                                     name='block2_conv2'))
        self.model.add(MaxPooling2D((2,2), strides=(2,2), padding='same',
                                     name='block2_pool'))

        ## 3 блок
        # 3 сверточных слоя по 256 каналов с фильтрами 3x3 и паддингом 1
        # активационная функция - ReLU
        # в конце - извлечение максимумов из ячеек 2x2 с шагом 2
        self.model.add(Convolution2D(256, (3, 3), activation='relu', padding='same',
                                     name='block3_conv1'))
        self.model.add(Convolution2D(256, (3, 3), activation='relu', padding='same',
                                     name='block3_conv2'))
```

```
self.model.add(Convolution2D(256, (3, 3), activation='relu', padding='same',
                    name='block3_conv3'))
self.model.add(MaxPooling2D((2,2), strides=(2,2), padding='same',
                    name='block3_pool'))

## 4 блок
# 3 сверточных слоя по 512 каналов с фильтрами 3x3 и паддингом 1
# активационная функция - ReLU
# в конце - извлечение максимумов из ячеек 2x2 с шагом 2
self.model.add(Convolution2D(512, (3, 3), activation='relu', padding='same',
                    name='block4_conv1'))
self.model.add(Convolution2D(512, (3, 3), activation='relu', padding='same',
                    name='block4_conv2'))
self.model.add(Convolution2D(512, (3, 3), activation='relu', padding='same',
                    name='block4_conv3'))
self.model.add(MaxPooling2D((2,2), strides=(2,2), padding='same',
                    name='block4_pool'))

## 5 блок
# 3 сверточных слоя по 512 каналов с фильтрами 3x3 и паддингом 1
# активационная функция - ReLU
# в конце - извлечение максимумов из ячеек 2x2 с шагом 2
self.model.add(Convolution2D(512, (3, 3), activation='relu', padding='same',
                    name='block5_conv1'))
self.model.add(Convolution2D(512, (3, 3), activation='relu', padding='same',
                    name='block5_conv2'))
self.model.add(Convolution2D(512, (3, 3), activation='relu', padding='same',
                    name='block5_conv3'))
self.model.add(MaxPooling2D((2,2), strides=(2,2), padding='same',
                    name='block5_pool'))

# объединяем все каналы в одномерный вектор
self.model.add(Flatten(name='flatten'))
# 3 полносвязных слоя
self.model.add(Dense(4096, activation='relu', name='fc1'))
self.model.add(Dense(4096, activation='relu', name='fc2'))
# softmax на выходе для получения распределения по 1000 классам
self.model.add(Dense(1000, activation='softmax', name='predictions'))

if weights_path:
   self.model.load_weights(weights_path)
```

```
  # настриваем оптимизатор и собираем модель
  sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
  self.model.compile(optimizer=sgd, loss='categorical_crossentropy')

  # создадим модель для фич,
  # содержащую выходы некоторых светочных слоёв основной модели
  ixs = [0, 3, 6, 10, 14, 16]
  outputs = [self.model.layers[i].output for i in ixs]
  self.model_for_features = Model(inputs=self.model.inputs, outputs=outputs)

# показать общую инофрмацию о модели
def summary(self):
  self.model.summary()

# прогнать изображение по пути img_path через модель
def inference(self, img_path):
  # загрузка и предобработка изображения
  im_original = cv2.resize(cv2.imread(img_path), (224, 224)).astype(np.float32)
  # нормализуем изображение
  im_original[:,:,0] -= 103.939
  im_original[:,:,1] -= 116.779
  im_original[:,:,2] -= 123.68

  im = im_original.transpose((0,1,2))
  im = np.expand_dims(im, axis=0)

  out = self.model.predict(im)

  return out

def show_feature_maps(self, img_path, fmap_index):
  # загрузка и предобработка изображения
  im_original = cv2.resize(cv2.imread(img_path), (224, 224)).astype(np.float32)
  # нормализуем изображение
  im_original[:,:,0] -= 103.939
  im_original[:,:,1] -= 116.779
  im_original[:,:,2] -= 123.68

  im = im_original.transpose((0,1,2))
  im = np.expand_dims(im, axis=0)

  feature_maps = self.model_for_features.predict(im)
```

```
fmap = feature_maps[fmap_index]

plt.figure(num=None, figsize=(12, 12), dpi=120, facecolor='w', edgecolor='k')
ix = 1
for _y in range(6):
  for _x in range(6):

    if(ix-1 < fmap.shape[3]) :
      ax = plt.subplot(6, 6, ix)
      ax.set_xticks([])
      ax.set_yticks([])
      # выводим в палитре jet
      plt.imshow(fmap[0, :, :, ix-1], cmap='jet')
    ix += 1
plt.show()
```