

Филиал Московского Государственного Университета
имени М.В.Ломоносова в городе Ташкенте
Факультет прикладной математики и информатики
Кафедра прикладной математики и информатики

Анваров Азизбек Акмал угли

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему: **«Распознавание свёрточными нейронными сетями
ложных изображений»**

по направлению 01.03.02 «Прикладная математика и информатика»

ВКР рассмотрена и рекомендована к защите

зав.кафедрой «ПМИИ», д.ф.-м.н., профессор _____ Кудрявцев В.Б.

Научный руководитель

м.н.с. _____ Соколов А.П.

«__» _____ 2019г.

Ташкент 2019 г

Аннотация

В данной работе мы изучим применение свёрточных нейронных сетей для распознавания искажённых изображений. Для исследования поведения сети, производится создание набора данных из десяти букв латинского алфавита печатного шрифта Arial, и реализуются несколько различных методов искажения изображений.

Annotation

In this paper we will study the use of convolution neural networks for the recognition of distorted images. A data set made of ten letters of the Latin alphabet using the printed font Arial has been created and several different methods of image distortion have been implemented to study the behavior of CNN.

Содержание

Введение.....	3
Основная исследовательская часть	6
Заключение.....	21
Приложения.....	22
Список использованной литературы.....	33

Введение

В рамках данной работы была рассмотрена Свёрточная нейронная сеть (Convolutional neural network, CNN), специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание изображений.

- Была использована модель сети, созданная для классификации изображений на наборе данных CIFAR-10.
- Язык программирования Python 3.7
- Библиотека Keras над Tensorflow
- Использованы библиотеки и программные средства
 - numpy — работа с многомерными массивами и матрицами
 - matplotlib – интерактивная отрисовка графических данных
 - seaborn – статистическая визуализация данных
 - pandas – анализ данных
 - Python Imaging Library – чтение и запись файлов изображений

Классификация изображений из набора данных CIFAR-10 является одной из наиболее распространённой проблемой в мире машинного обучения. Свёрточные нейронные сети показали наиболее высокие результаты в области классификации изображений. Среди преимуществ CNN:

- По сравнению с полносвязной нейронной сетью — гораздо меньшее количество настраиваемых весов, так как одно ядро весов используется целиком для всего изображения, вместо того, чтобы

делать для каждого пикселя входного изображения свои персональные весовые коэффициенты. Это подталкивает нейросеть при обучении к обобщению демонстрируемой информации, а не попиксельному запоминанию каждой показанной картинке в мириадах весовых коэффициентов.

- Удобное распараллеливание вычислений, а следовательно, возможность реализации алгоритмов работы и обучения сети на графических процессорах.
- Относительная устойчивость к повороту и сдвигу распознаваемого изображения.

Целью работы является исследование поведения сверточной сети на искажённых изображениях.

Основная исследовательская часть

В ходе данной работы был создан набор данных, схожий по структуре с набором CIFAR-10, состоящий из изображений букв латинского алфавита.

Шрифт: Arial, размер: 32 pt.

Исходный код генератора:

```
def render_character(out_dir, text: str, size: int = 32) -> Image:
    image = Image.new(mode='RGB', size=(size, size), color='white')
    draw = ImageDraw.Draw(image)
    font = ImageFont.truetype("arial.ttf", 32)
    w, h = draw.textsize(text, font=font)
    draw.text((math.ceil((size - w) / 2), math.floor((size - h) / 2) - 2), text,
              font=font, fill='black')
    image.save(fp=f'{out_dir}/{text}.bmp')
    return image

def generate_characters(out_dir, letters: list, size: int = 32) -> np.array:
    result = []
    for letter in letters:
        image = np.array(render_character(out_dir, letter, size=size))
        result.append(image.transpose((2, 0, 1)))
    return np.array(result)

def generate_letters(count: int = 10) -> list:
    if not 0 < count <= 26:
        raise ValueError("count must be in range (0, 26]")
    return ascii_uppercase[:count]

def generate_data_set(out_dir, letters_count: int = 10, size: int = 32,
                      train_repeat: int = 1, test_repeat: int = 1):
    letters = generate_letters(count=letters_count)
    characters = generate_characters(out_dir, letters, size=size)

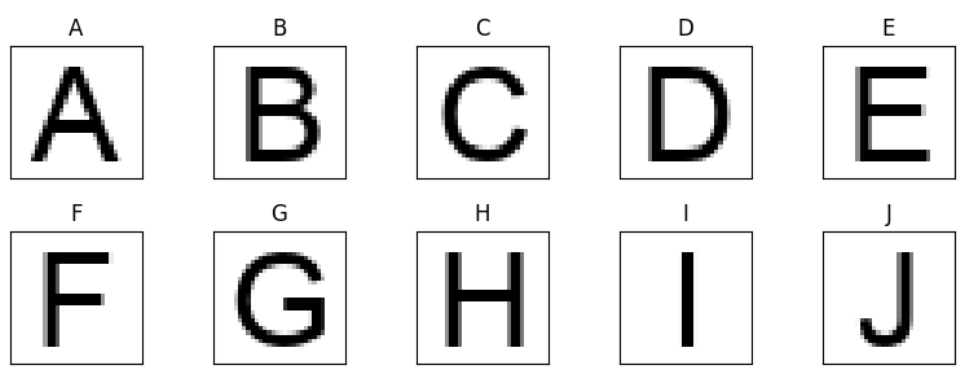
    train_size = letters_count * train_repeat
    train_shuffle = np.random.permutation(train_size)
    x_train = characters.repeat(train_repeat, 0)[train_shuffle]
    y_train = np.arange(letters_count).repeat(train_repeat)[train_shuffle].reshape(train_size, 1)

    test_size = letters_count * test_repeat
    test_shuffle = np.random.permutation(test_size)
    x_test = characters.repeat(test_repeat, 0)[test_shuffle]
    y_test = np.arange(letters_count).repeat(test_repeat)[test_shuffle].reshape(test_size, 1)

    return (x_train, y_train), (x_test, y_test), [c for c in letters]
```

Для упрощения процесса обучения, генератор умеет создавать наборы данных заданного параметрами размера.

Результат генерации:



На данном наборе была обучена модель четырехслойной сверточной сети:

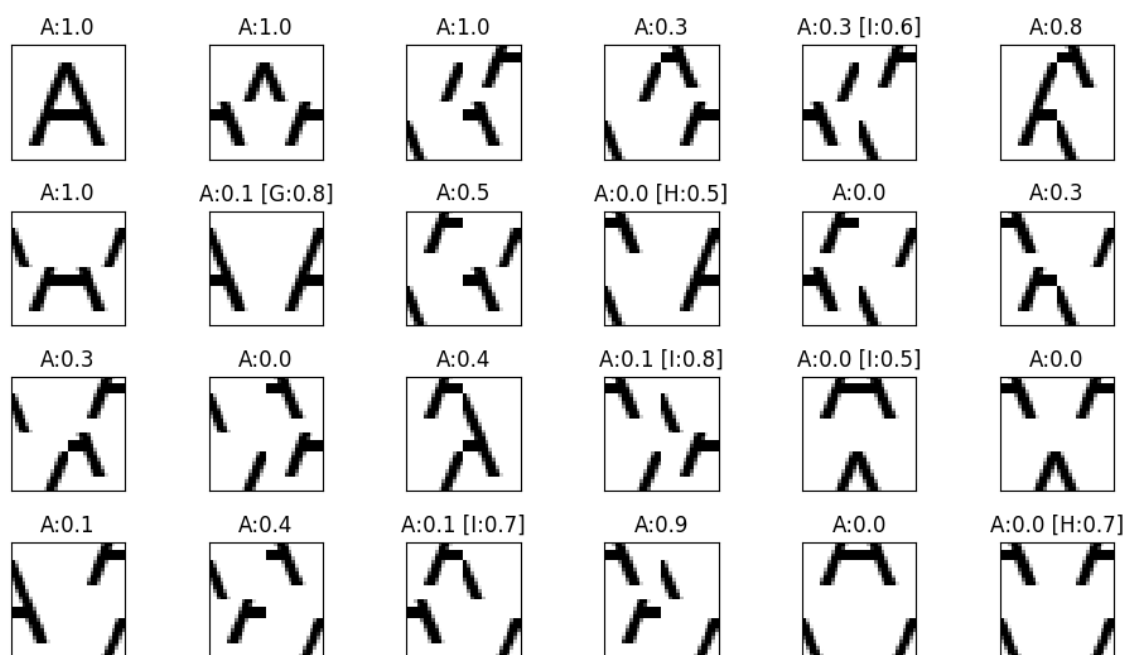
OPERATION		DATA	DIMENSIONS	WEIGHTS(N)	WEIGHTS(%)		
	Input	#####	3	32	32		
Conv2D	\\ /	-----				896	0.1%
relu	#####	32	32	32			
Conv2D	\\ /	-----				9248	0.7%
relu	#####	32	30	30			
MaxPooling2D	Y max	-----				0	0.0%
	#####	32	15	15			
Dropout		-----				0	0.0%
	#####	32	15	15			
Conv2D	\\ /	-----				18496	1.5%
relu	#####	64	15	15			
Conv2D	\\ /	-----				36928	3.0%
relu	#####	64	13	13			
MaxPooling2D	Y max	-----				0	0.0%
	#####	64	6	6			
Dropout		-----				0	0.0%
	#####	64	6	6			
Flatten		-----				0	0.0%
	#####		2304				
Dense	XXXXX	-----				1180160	94.3%
relu	#####		512				
Dropout		-----				0	0.0%
	#####		512				
Dense	XXXXX	-----				5130	0.4%
softmax	#####		10				

Общее число параметров: 1250858

Обучение длиной в одну эпоху было достаточно для получения точности в 100. Для проверки работоспособности сети, можно подать ей на распознавание изображения из набора данных. Основная часть работ проводилась под фиксированным начальным значением генератора случайных чисел для обеспечения воспроизводимости экспериментов.

Первый метод заключался в разделении изображений на 4 равных части и перестановки этих частей местами. Всего таких перестановок 24.

Символ над изображением — буква, которая изначально была изображена. Число после двоеточия – вероятность классификации на выходе нейронной сети для данной буквы. Аналогично для символа и числа в квадратных скобках — если они присутствуют, значит сеть отнесла изображение к указанному символом классу, с вероятностью большей либо равной $\frac{1}{2}$



Полные результаты генераций и соответствующие оценки сети приведены в приложениях к документу.

Программа, помимо изображений, выводит подробную таблицу с вероятностями, но они были исключены из данного документа для поддержания краткости содержимого. Вместо этого составлена таблица, показывающая минимум, максимум и арифметическое среднее значение оценок:

Class	Min	Max	Mean
A	0.00	1.00	0.36
B	0.00	1.00	0.21
C	0.00	1.00	0.34
D	0.00	1.00	0.44
E	0.00	1.00	0.31
F	0.00	1.00	0.37
G	0.00	1.00	0.31
H	0.97	1.00	0.99
I	0.17	1.00	0.85
J	0.01	1.00	0.56

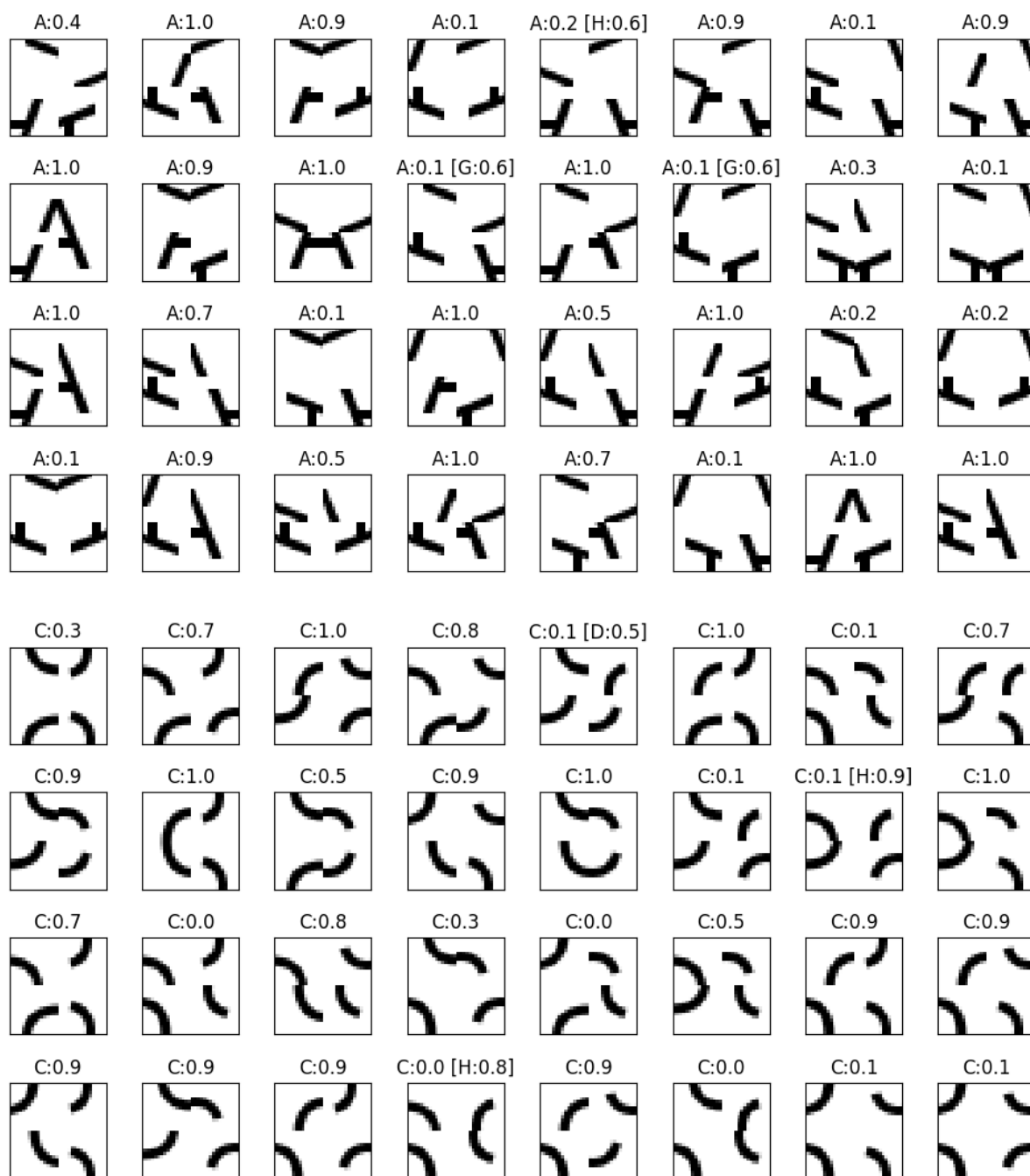
Таблица показывает устойчивость каждого из классов к данному виду искажениям. Среди них стоит отметить изображение буквы «Н», которая оказалась наименее чувствительной к данному рода преобразованиям.

В ходе подобных манипуляций, буква хорошо сохраняет признаки, по которым сеть её идентифицирует.

Хуже всего распознавалась буква «В». Сеть путала ее с буквами «D», «G», «Н».

Второй метод заключается в разделении изображений на 4 равных части и поворот на случайно выбранный угол, кратный 90° . Всего таких вариаций 256, но изобразить все вариации в документе — проблематично. По этой причине привожу по 32 случайно взятых вариаций для каждого класса в приложениях.

Примеры полученных изображений для букв «А» и «С»:



Ниже привожу исходный код классификатора.

Значение `POSITIVE_THRESHOLD` – порог ответа сети, выше которого ответ считается положительным.

Для каждого изображения получаем класс, к которому действительно принадлежит изображение; вероятность, с которой сеть относит изображение к данному классу. Такую же информацию получаем для класса, отличного от действительного, в случае превышения порога `POSITIVE_THRESHOLD` для данного класса. Эта информация поможет нам увидеть причину, по которой сеть выдаёт подобный результат.

```
def classify_images(images, self_letter):
    result = []
    for index, image in enumerate(images):
        image = np.expand_dims(image / 255., axis=0)
        predict = model.predict(image)[0]
        self_index = letters.index(self_letter)
        self_score = predict[self_index]
        top_index = int(np.argmax(predict))
        top_letter = None
        top_score = None
        if self_index != top_index and
            predict[top_index] > POSITIVE_THRESHOLD:
            top_letter = letters[top_index]
            top_score = predict[top_index]

        result.append((self_letter, self_score,
                       top_letter, top_score))
    return result
```

Схожая схема используется и для последующих способов построения ложных изображений.

При полном “перемешивании” изображений модель не могла правильно классифицировать, но если же хоть какая-то часть изображения оставалась нетронутой, наблюдалось обратное. Считаю, что это в основном связано с тем, что вариаций изображений мало.

Привожу аналогичную таблицу, показывающую минимум, максимум и арифметическое среднее значение оценок:

Class	Min	Max	Mean
A	0.03	1.00	0.64
B	0.00	1.00	0.34
C	0.00	1.00	0.51
D	0.00	1.00	0.49
E	0.00	1.00	0.30
F	0.00	1.00	0.46
G	0.00	1.00	0.32
H	0.00	1.00	0.57
I	0.02	1.00	0.60
J	0.01	1.00	0.44

Для данного метода искажения изображений буква «А» оказалась наименее чувствительной к данному рода преобразованиям.

В ходе подобных манипуляций, буква лучше всего сохраняет признаки, по которым сеть её идентифицирует.

Хуже всего распознавались буквы «В», «Е» и «G».

Третий метод обмана – наложение “шума” на изображения. Шумом назовём пиксели чёрного цвета, распределенные равномерно по всему изображению с заданной на вход вероятностью.

Реализовано это следующим образом: на вход функции добавляющей шум подаётся изображение и параметр, отвечающий за интенсивность шума. Назовём его уровнем зашумления. Для каждого пикселя, если значение равномерно распределенной случайной величины меньше значения уровня зашумления, то данный пиксель закрашивается в чёрный цвет.

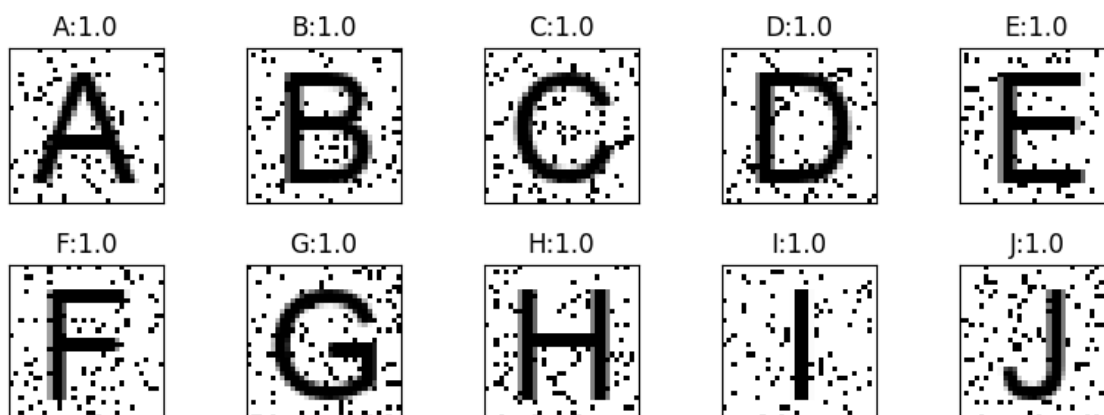
```
def put_pixel(target: np.array, x: int, y: int,
              color: tuple = (0., 0., 0.)):
    for i in range(3):
        target[i][y][x] = color[i]

def noise_image(target: np.array, noise_level: float, size: int = 32):
    for y in range(size):
        for x in range(size):
            if np.random.rand() < noise_level:
                put_pixel(target=target, x=x, y=y)

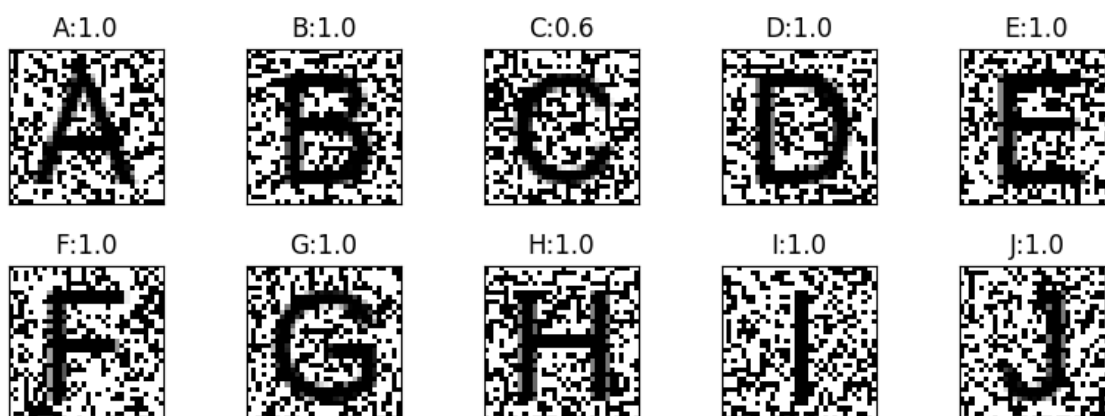
def add_noise(target: np.array, noise_level: float, size: int = 32):
    for image in target:
        noise_image(target=image, noise_level=noise_level, size=size)
```

Приведём примеры изображений для каждого уровня зашумления для того чтобы можно было визуально оценить насколько буквы на изображениях отличимы от шума:

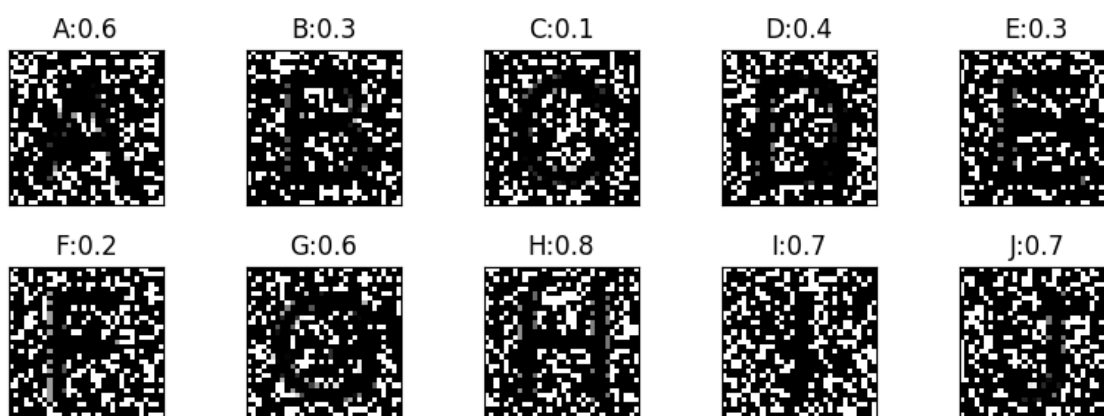
При коэффициенте шума 10% результаты распознавания в данной генерации не были ухудшены.



40% зашумления. Сеть пока ещё устойчива к искажениям.

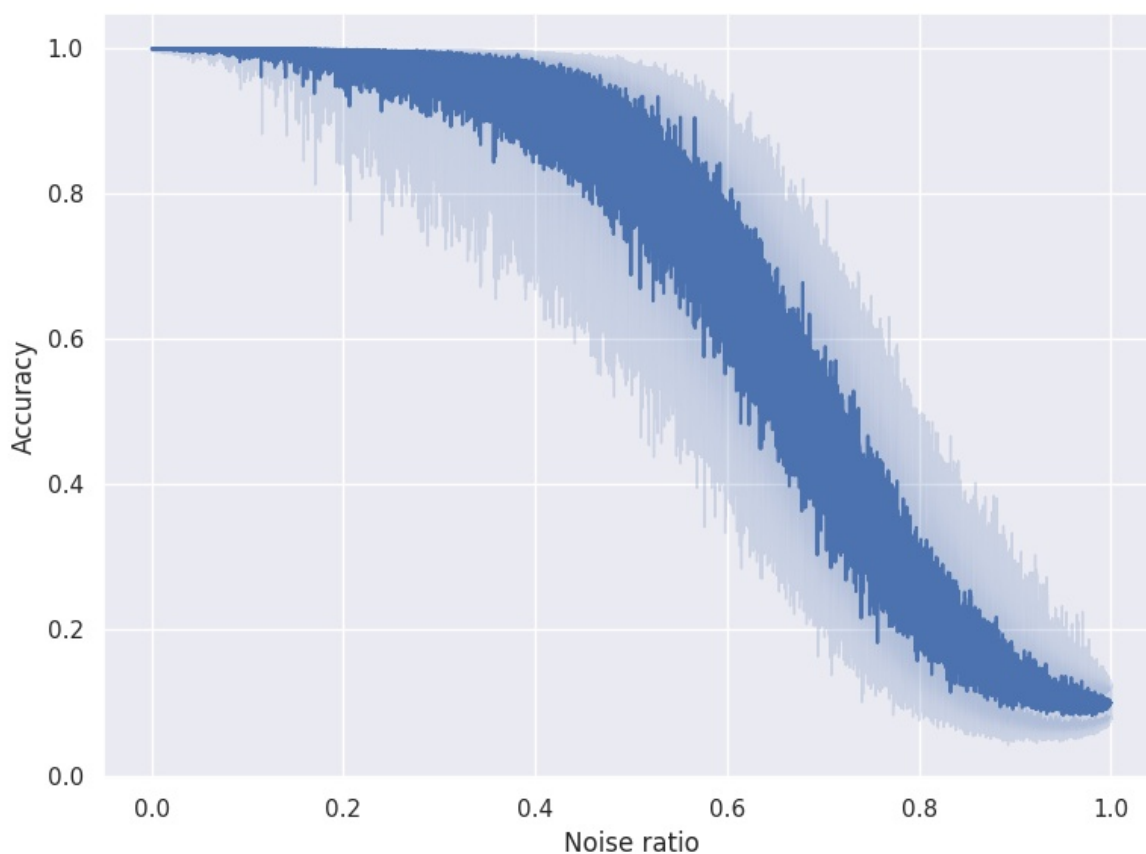


70% зашумления. Точность сети заметно снижается.



Модель на данных примерах показывает неплохие результаты, но для более качественной проверки необходимо рассмотреть гораздо больше случаев наложения шума.

По этой причине, я провёл анализ поведения сети на более широкой выборке. Для этого построим график зависимости точности сети от уровня зашумления. Ниже приведёт график, состоящий из 5,000,000 данных для уровня зашумления на интервале $[0, 1]$:



Из графика видно, что сеть довольно стойкая к подобным искажениям.

Считаю разумным предполагать, что ответ сети более $\frac{1}{2}$ для данного класса является положительным ответом при распознавании. При 40% зашумлении сеть стабильно распознает изображения.

Для оценки искаженности изображений можно использовать отношение сигнала к шуму (peak signal to noise ratio, PSNR) – является инженерным термином, означающим соотношение между максимумом возможного значения сигнала и мощностью шума, искажающего значения сигнала.

PSNR наиболее часто используется для измерения уровня искажений при сжатии изображений. Проще всего его определить через среднеквадратичную ошибку (mean square error , MSE)

В случае использования MSE этот показатель для двух монохромных изображений I и K размера $m \times n$, одно из которых считается зашумленным приближением другого, вычисляется так:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I(i, j) - K(i, j)|^2$$

PSNR определяется так:

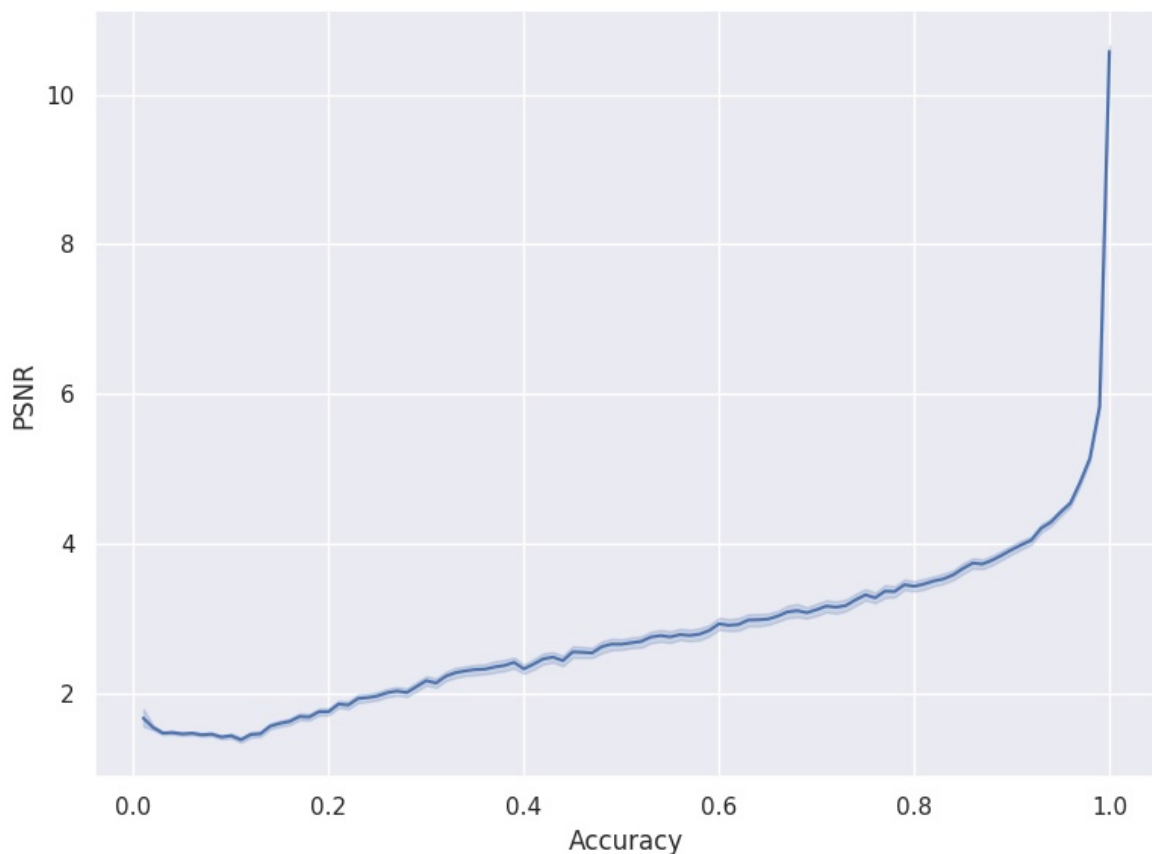
$$PSNR = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20 \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right)$$

где MAX_I — это максимальное значение, принимаемое пикселем изображения, равное 255 в нашем случае.

Реализация вычисления PSNR при помощи библиотеки Numpy:

```
def psnr(original_image, altered_image):  
    mse = np.mean((original_image - altered_image) ** 2)  
    if mse == 0:  
        return 100 # undefined  
    return 20 * math.log10(255. / math.sqrt(mse))
```

Ниже приведём график отношение точности сети к PSNR для выборки из 1,000,000 данных:



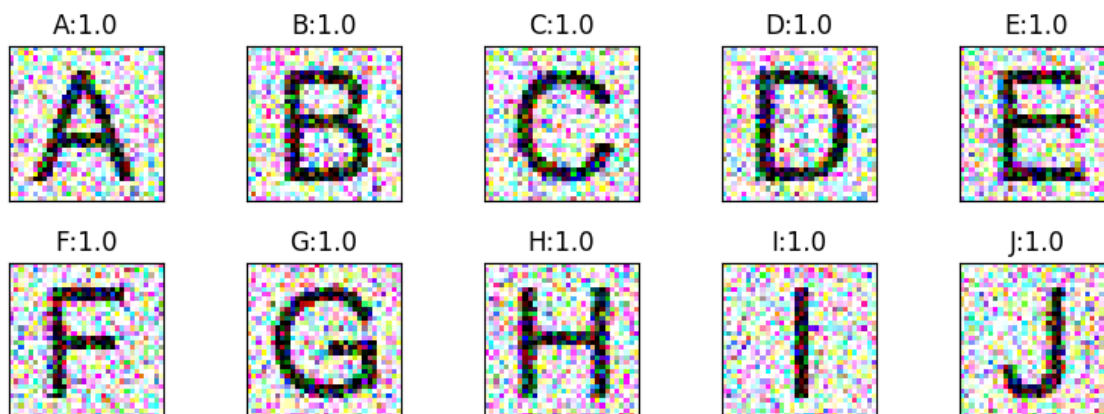
При стремлении среднеквадратичной ошибки MSE к нулю, PSNR стремится к бесконечности. MSE равная нулю означает что два изображения абсолютно идентичны.

Далее изучим поведение сети при воздействии гауссовского шума (Gaussian noise). Гауссов шум, названный в честь Карла Фридриха Гаусса, является статистическим шумом, имеющим плотность распределения вероятностей, равную плотности нормального распределения, которое также известно как гауссовское распределение. Другими словами, значения, которые может принимать шум, являются гауссово распределенными.

Реализация была выполнена следующим образом: `target` — изображение, на которое будет наложен шум, `mean` — среднее значение, `stddev` — среднеквадратическое отклонение.

```
def add_noise(target: np.array, mean: float, stddev: float):  
    for index, image in enumerate(target):  
        image += np.random.normal(mean, stddev, image.shape)  
        target[index] = np.clip(image, 0, 255)
```

Пример изображения полученного при помощи наложения гауссовского шума при задании среднеквадратического отклонения 100:



Пример изображения полученного при помощи наложения гауссовского шума при задании среднеквадратического отклонения 500:

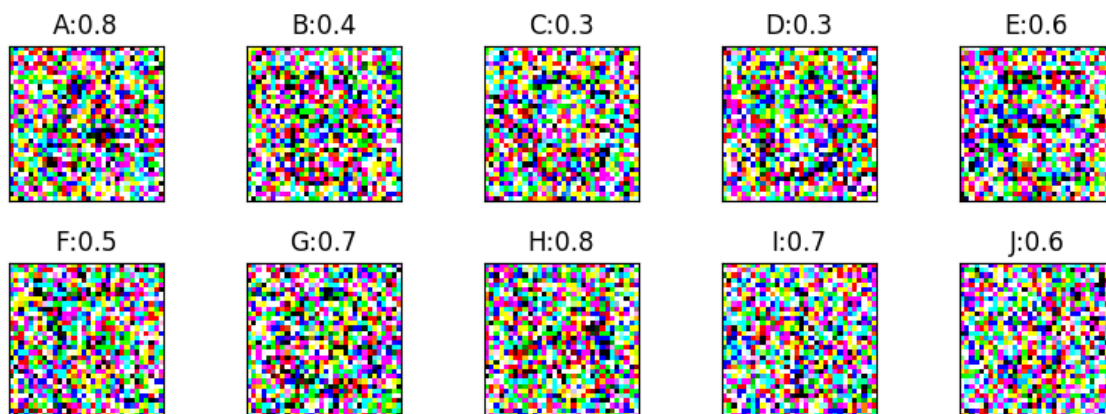
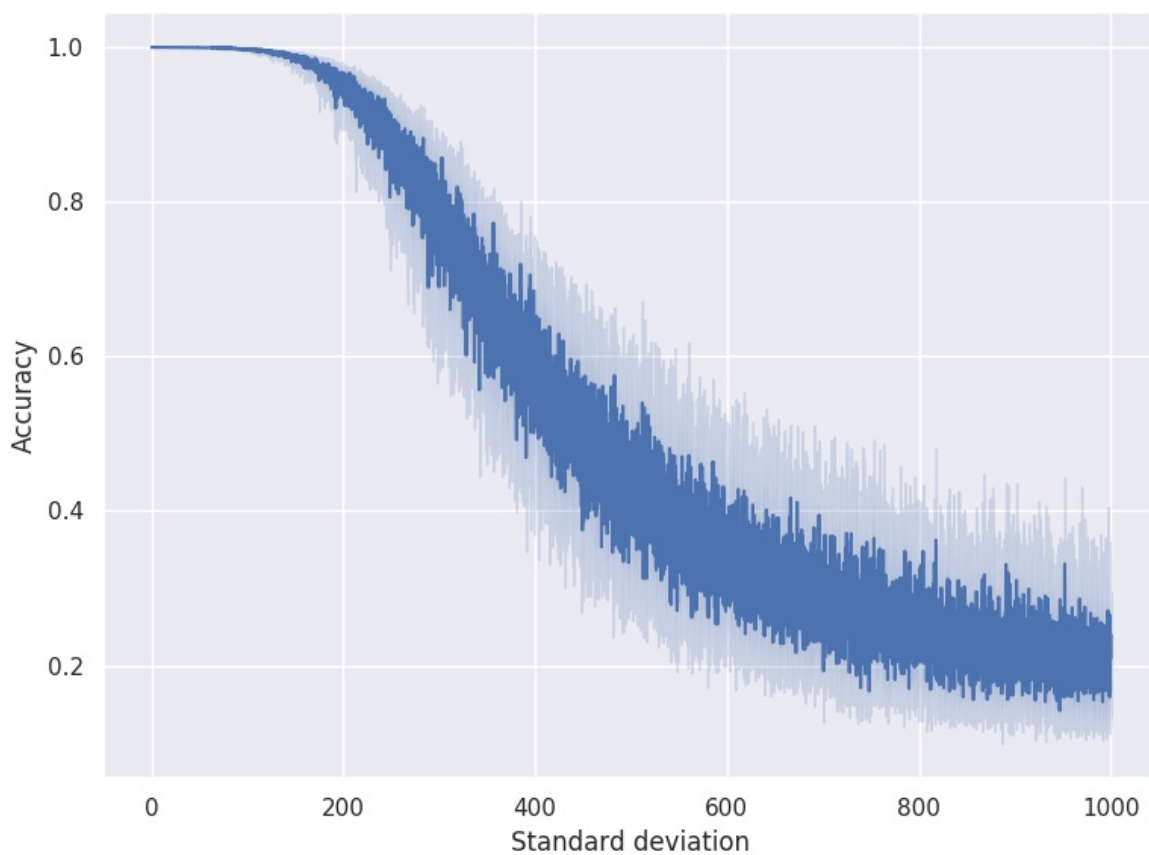
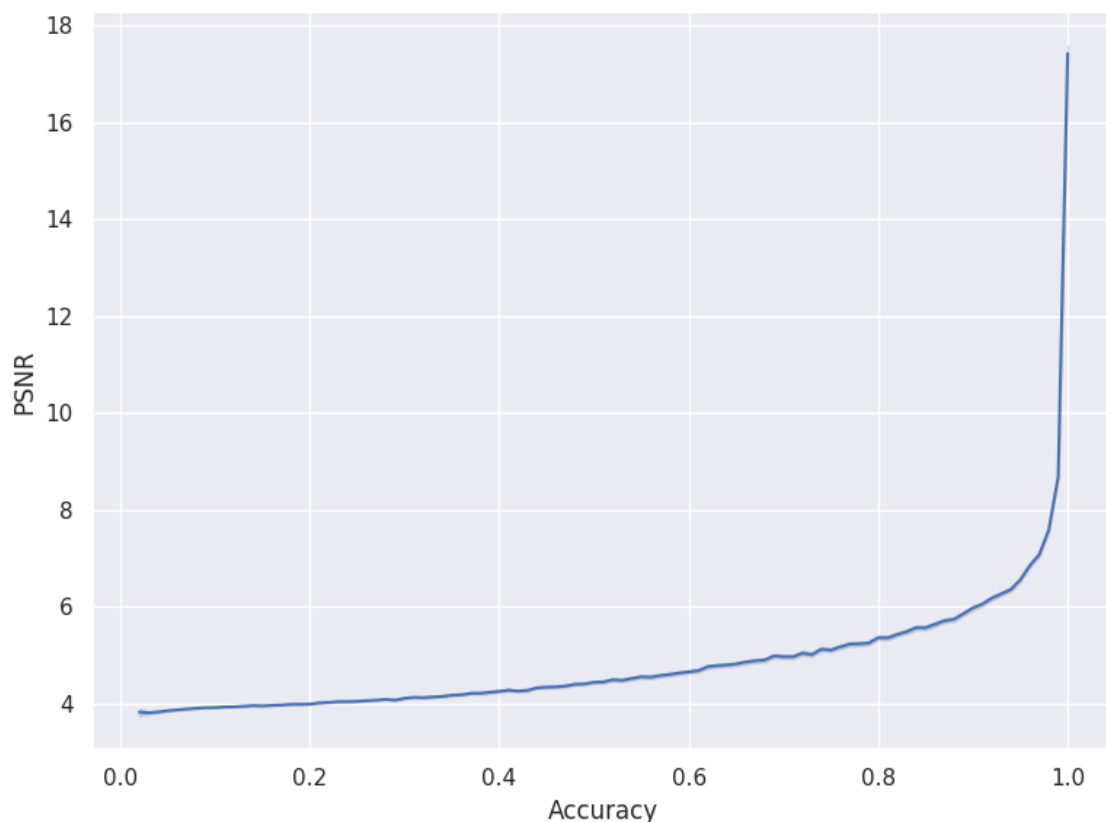


График поведения сети в зависимости среднеквадратического отклонения к точности сети для гауссовского шума:



На самом деле, сравнивать с предыдущим аналогичным графиком, график, приведённый выше было бы неправильным. Гауссовский шум не путает сеть больше, чем шум, который накладывался ранее. Из соображений визуальной различимости изображений все получается иначе: при 50% коэффициенте шума для первого метода зашумления изображения получаются чётче и понятнее визуально, нежели, при значении среднеквадратического отклонения 500, которая находится в той же точке на оси уровня зашумления графика.

Графики поведения сети в зависимости от отношения точности сети к PSNR для гауссовского шума:



По графику можно заметить более низкий уровень PSNR в тестах с гауссовским зашумлением, что и подтверждает вышесказанное.

Заключение

Свёрточная нейронная сеть показала хорошую стойкость к обману. В большинстве случаев, где сеть не могла классифицировать изображение – не получалось сделать без значительных усилий визуально. Поведение сети достаточно стабильное, это следует из проделанных экспериментов. В дальнейшем можно рассматривать другие методы «обмана», расширив исследования изменением способов обучения и набора данных. Также, от изменения модели сети есть возможность обнаружения новых результатов.

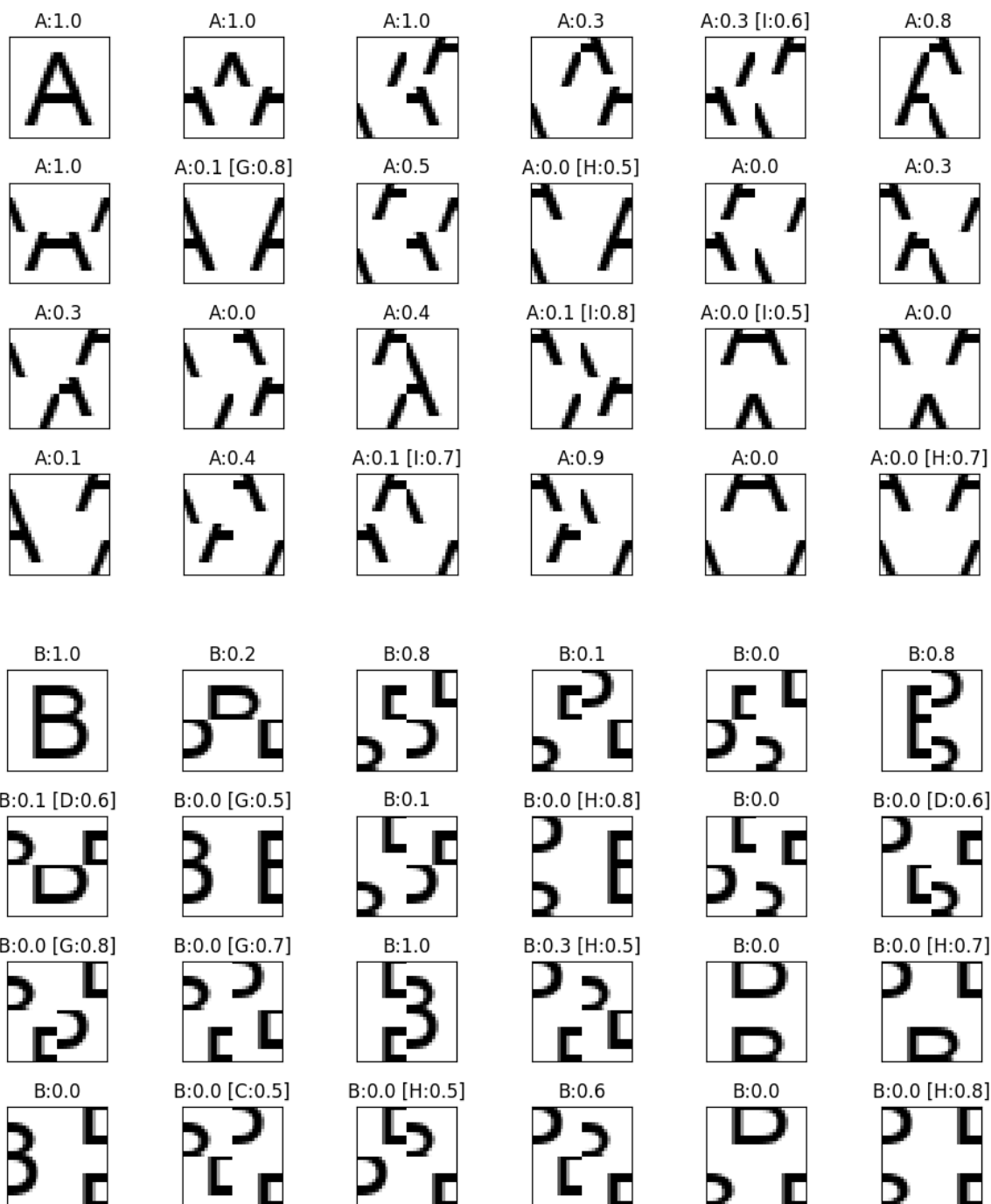
Все исследования проводились при фиксированном значении инициализации генератора случайных чисел для возможности повторного воспроизведения экспериментов, что безусловно влечёт за собой возможность потери важных результатов.

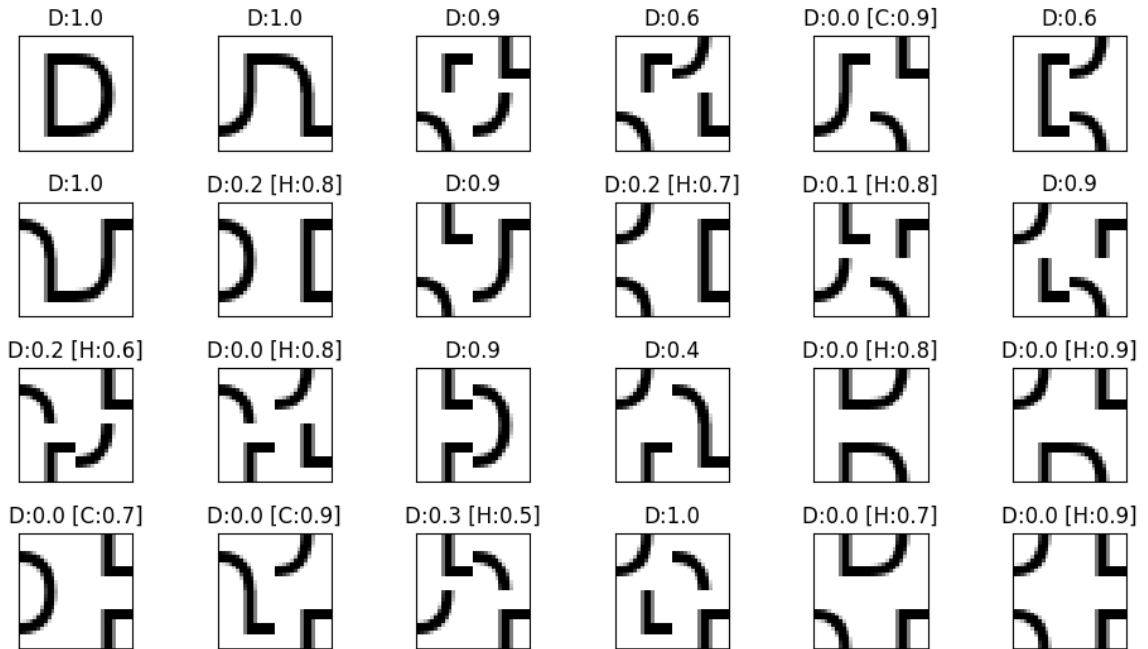
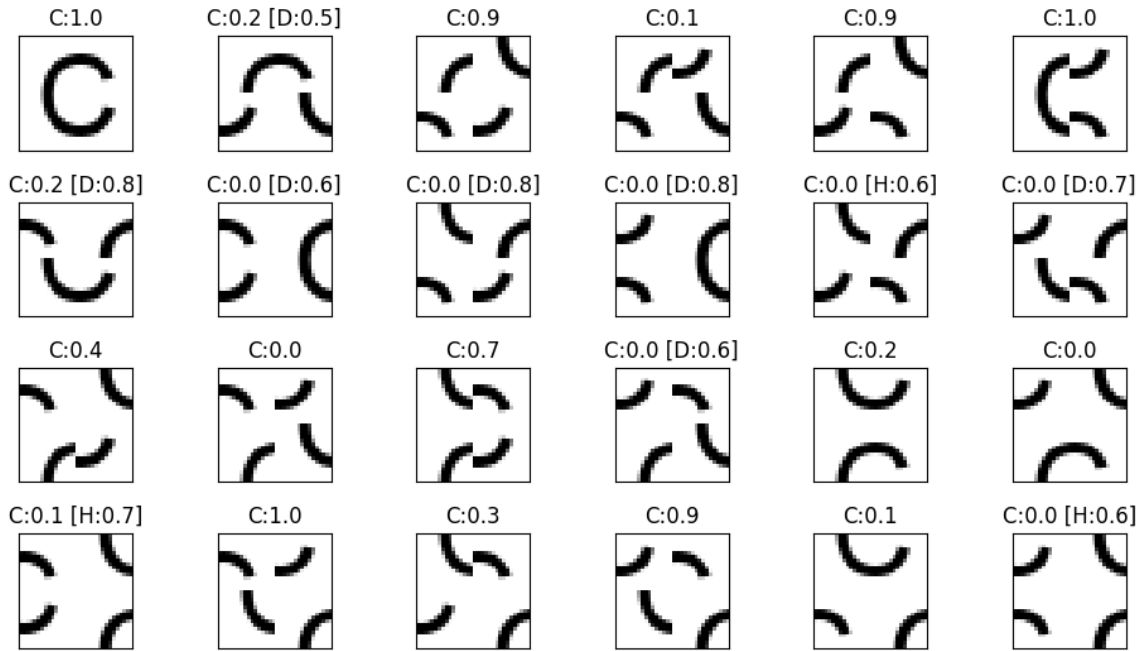
Приложения

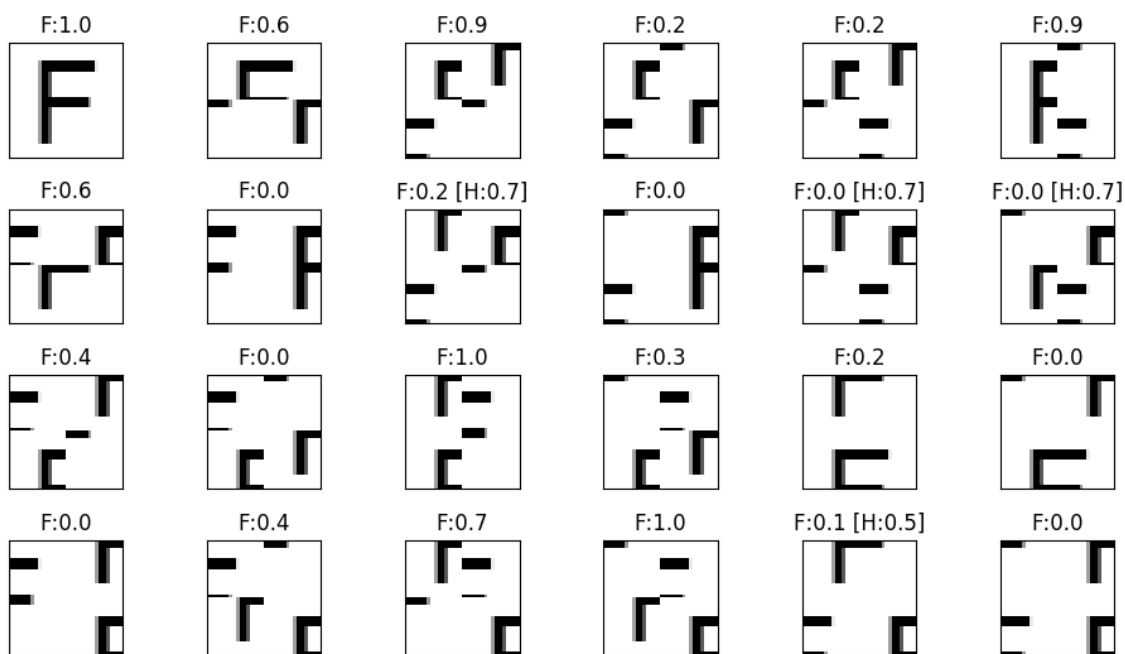
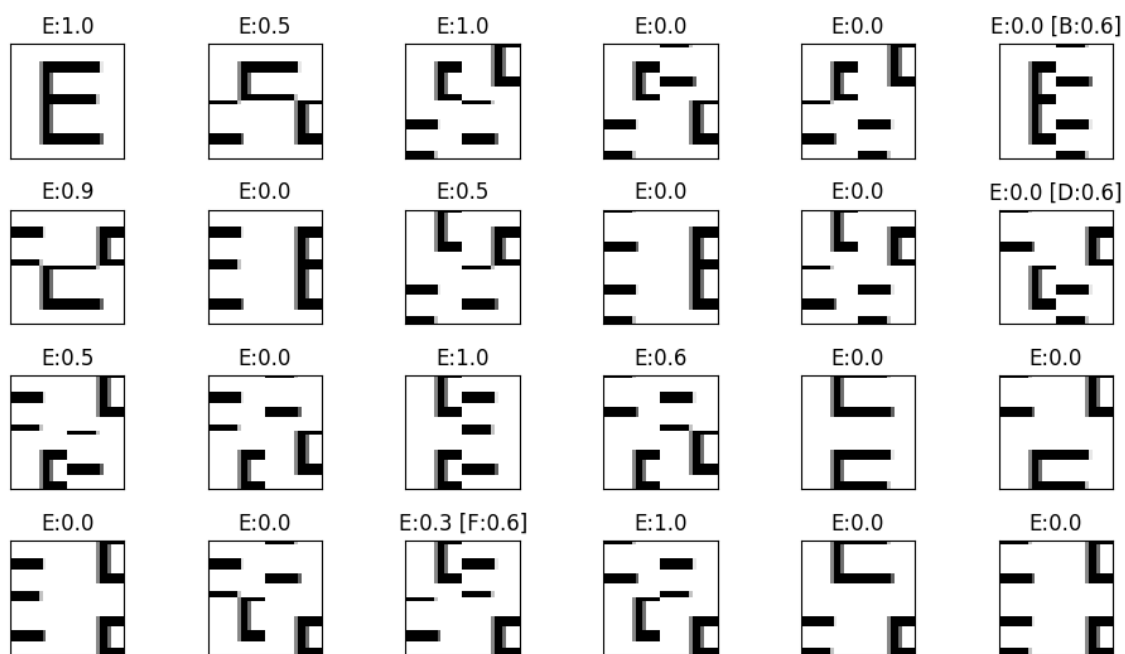
Исходные коды, со всеми необходимыми ресурсами и полученными результатами расположены по адресу:

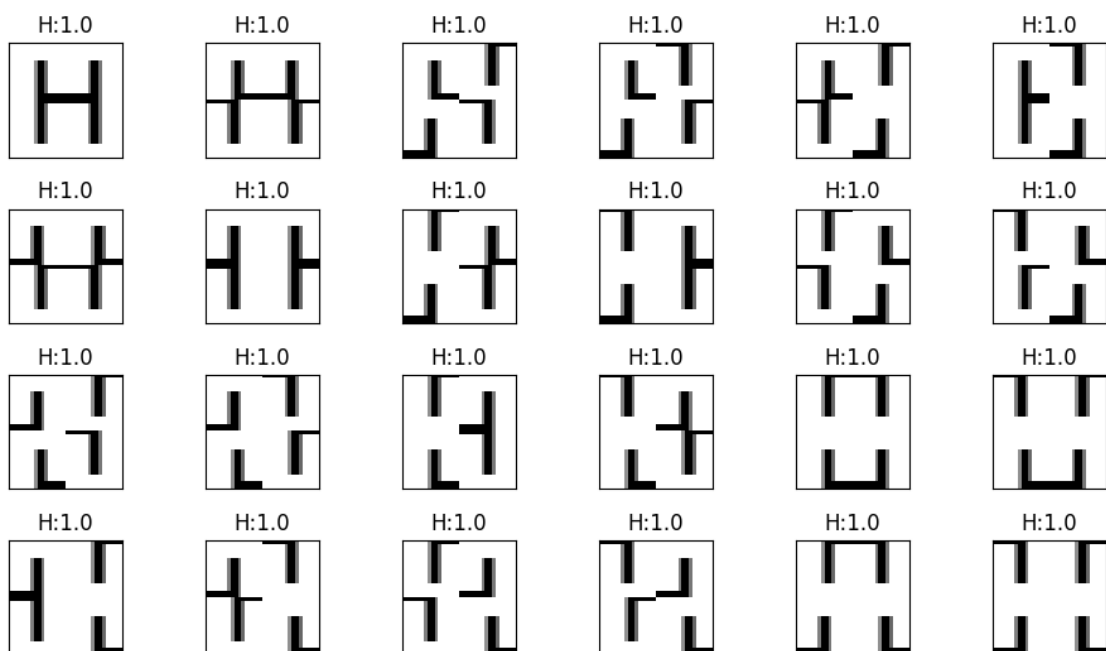
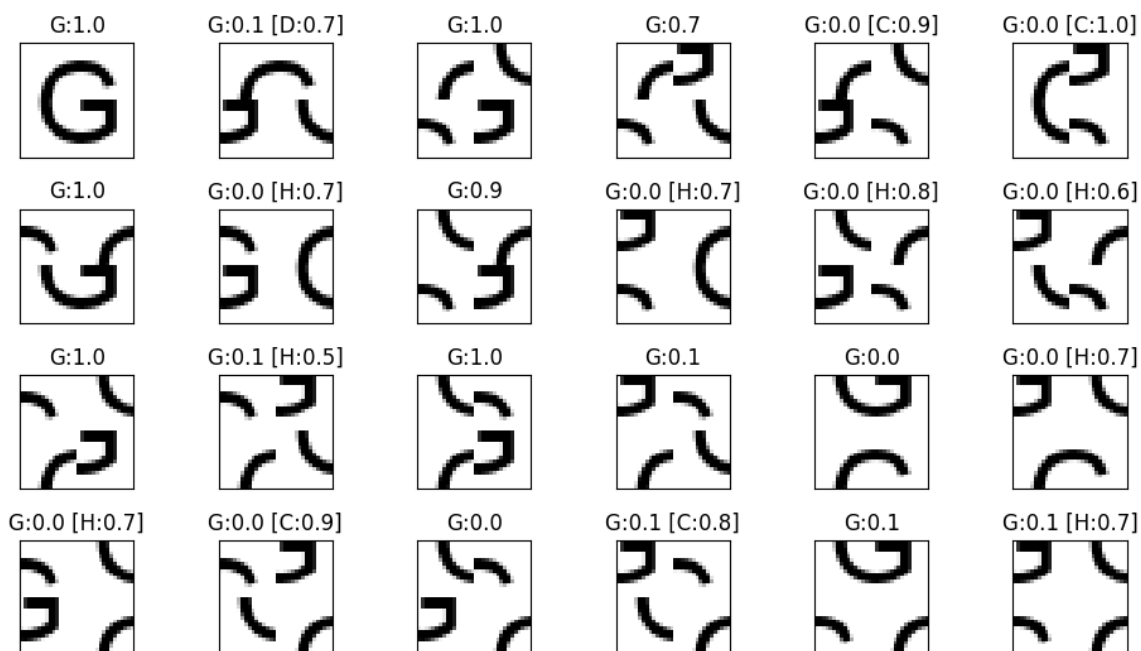
<https://github.com/drunckoder/ThesisWork>

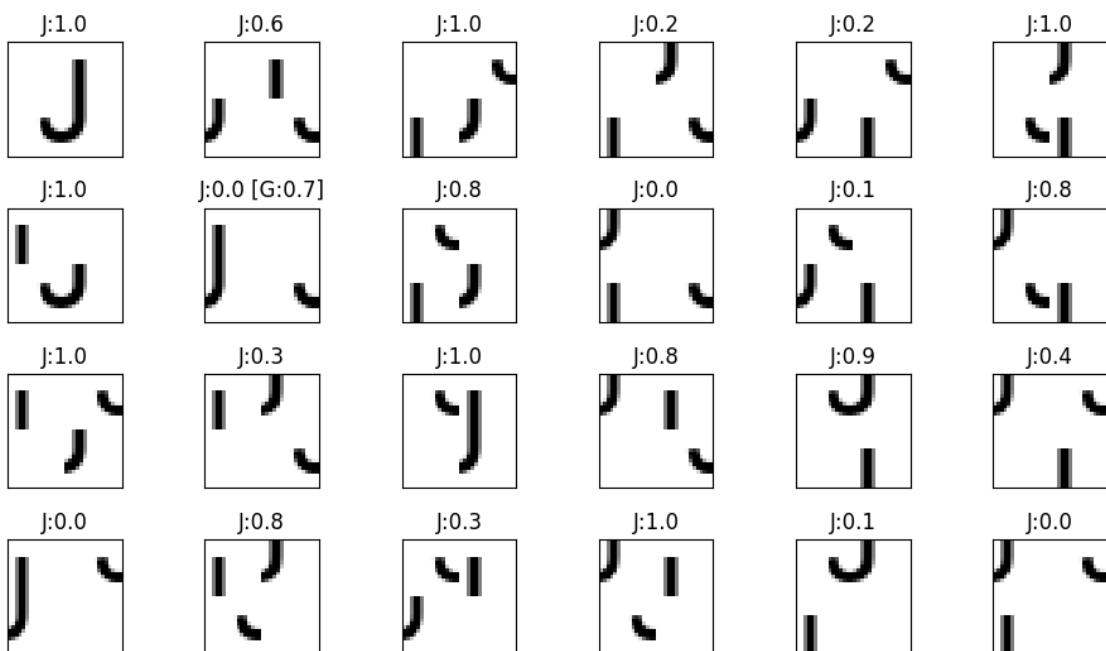
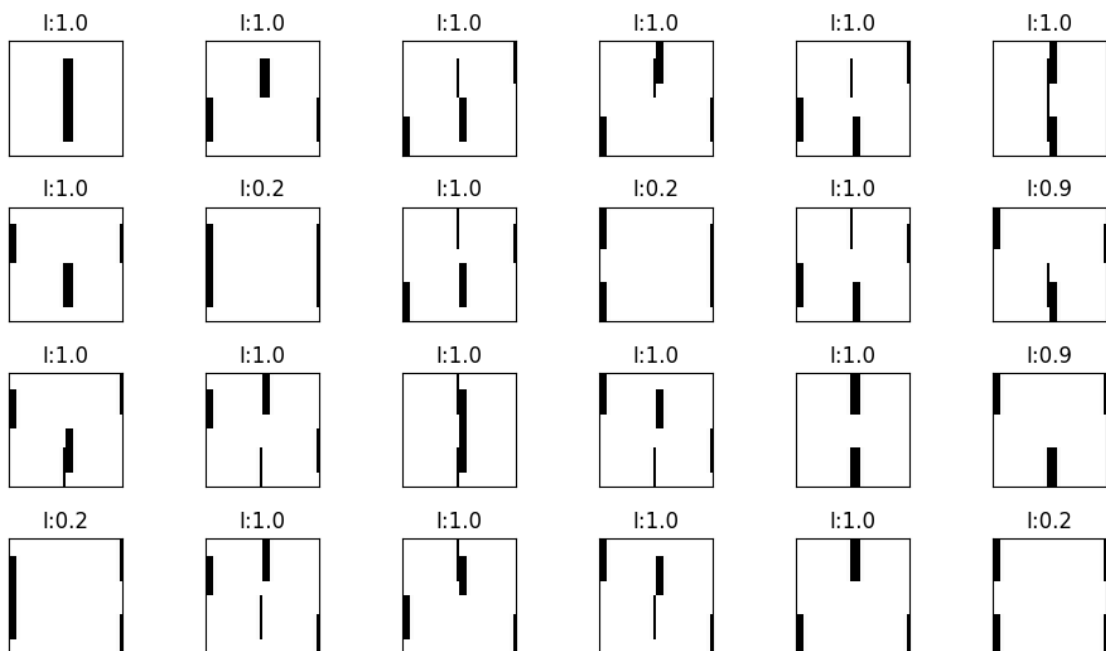
Результаты генераций для метода с перестановками квадрантов:



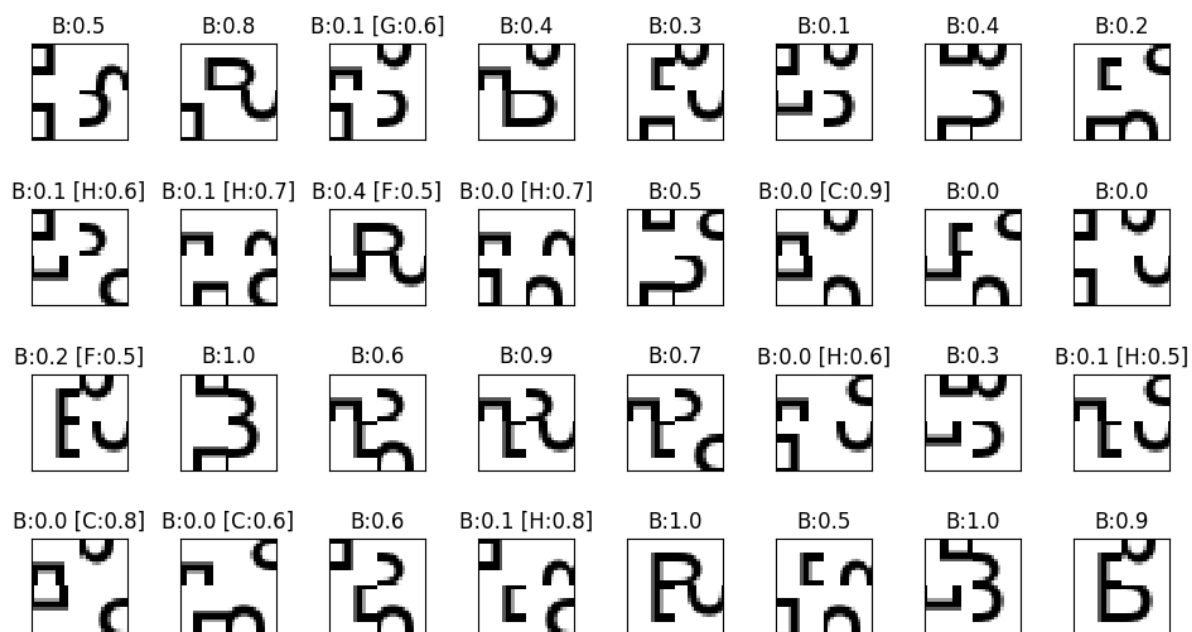
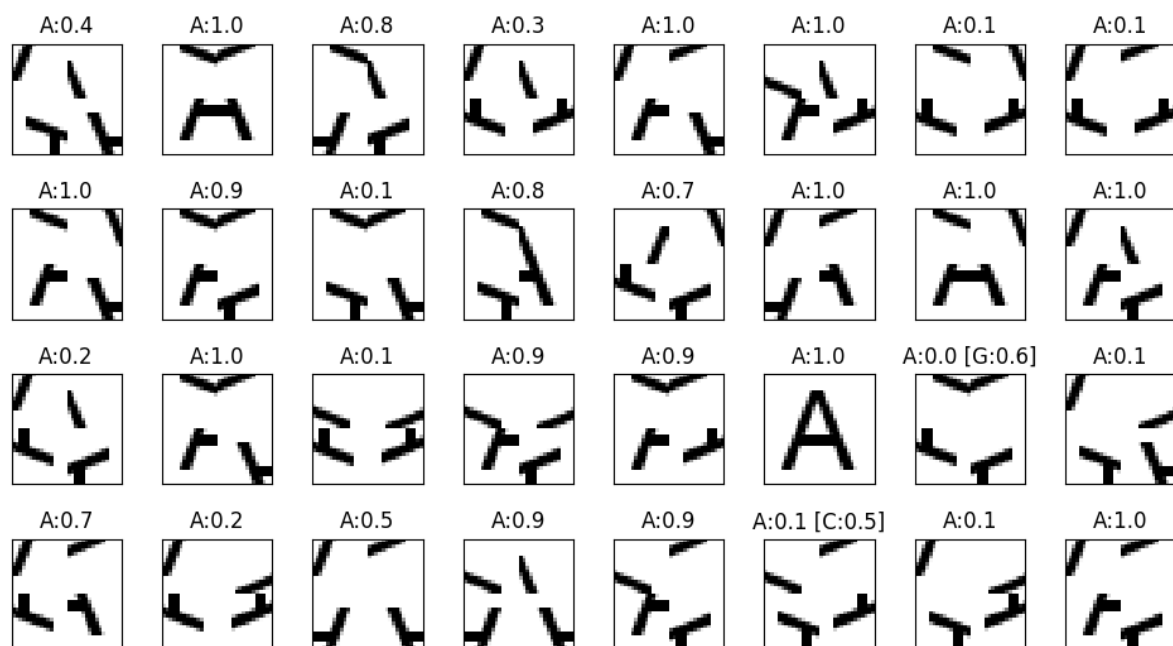


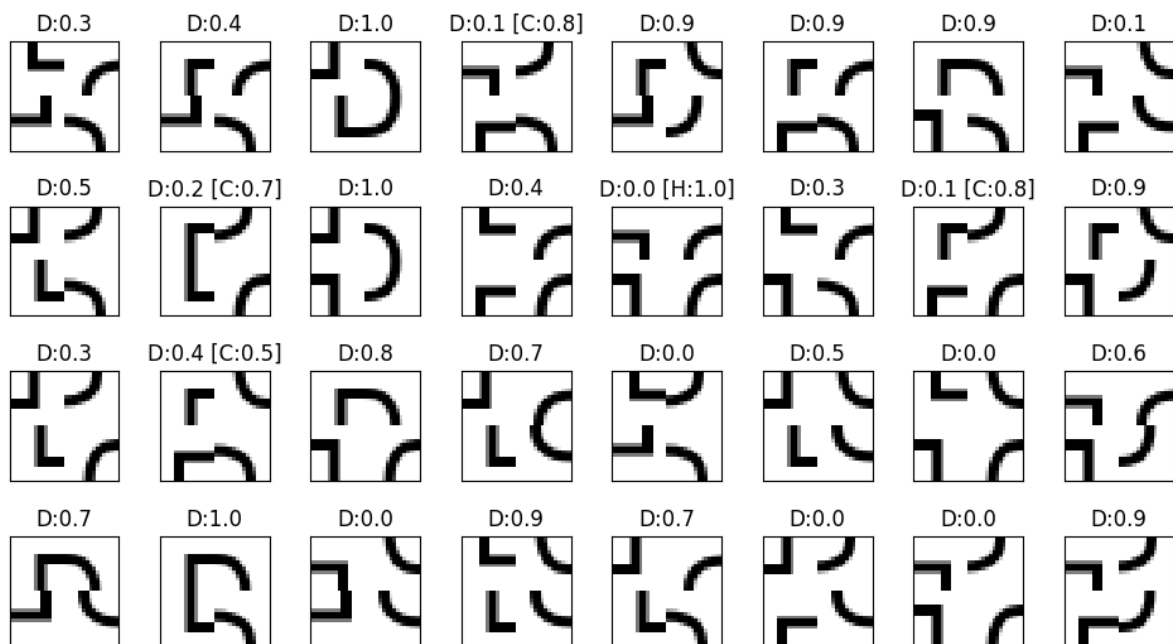
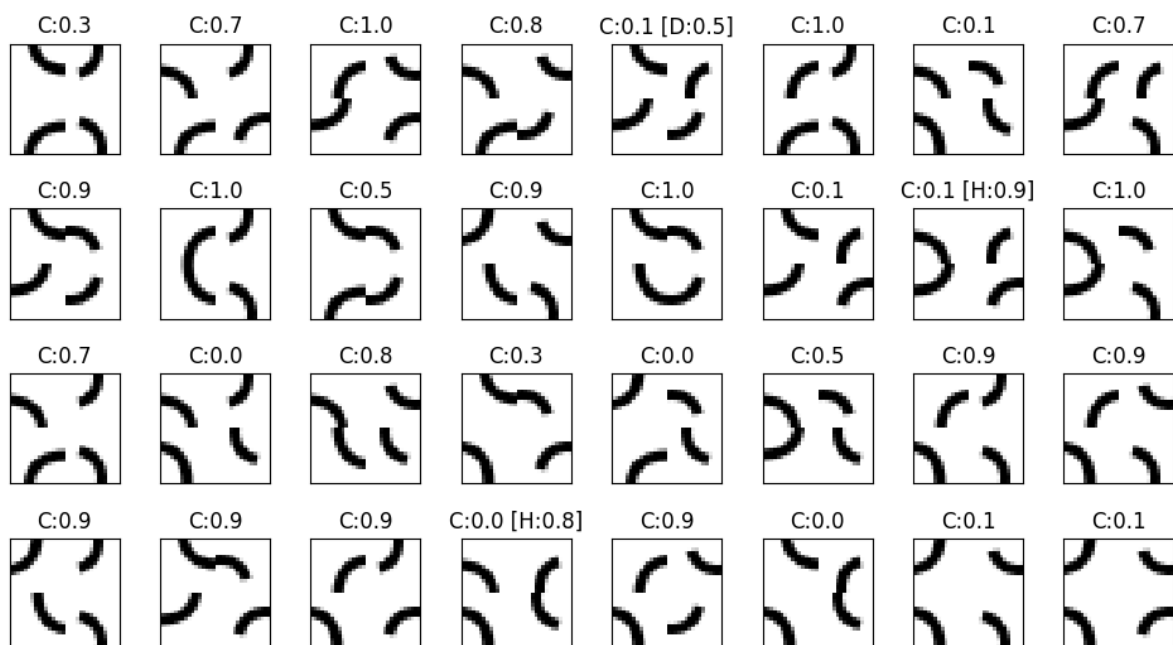


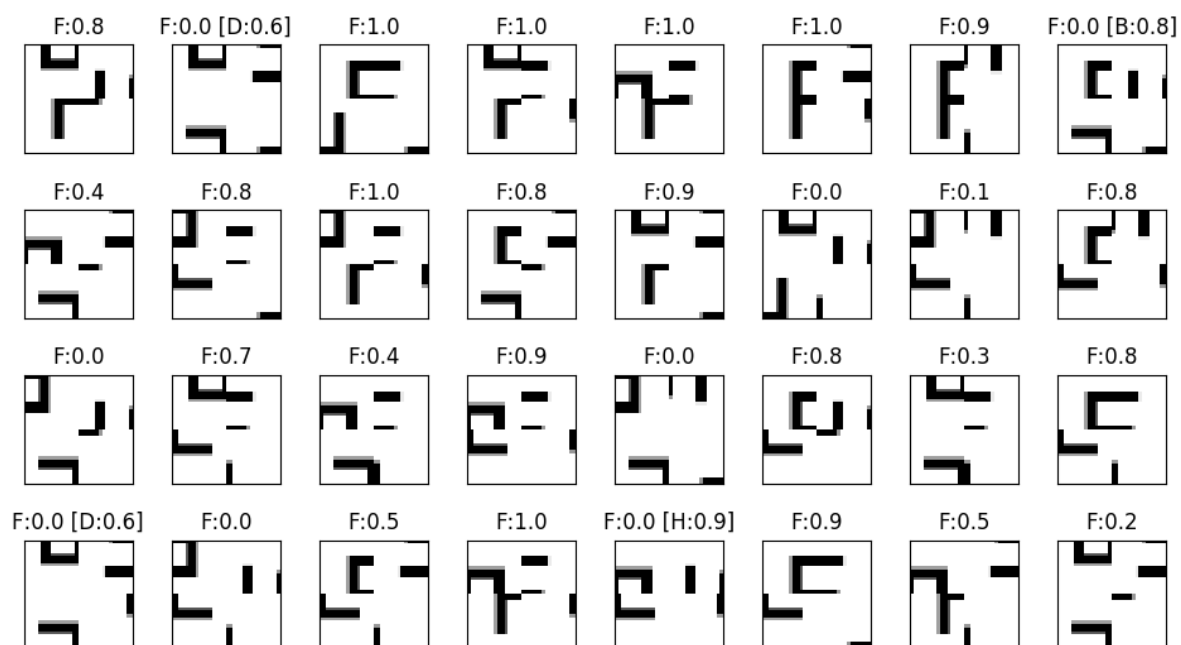
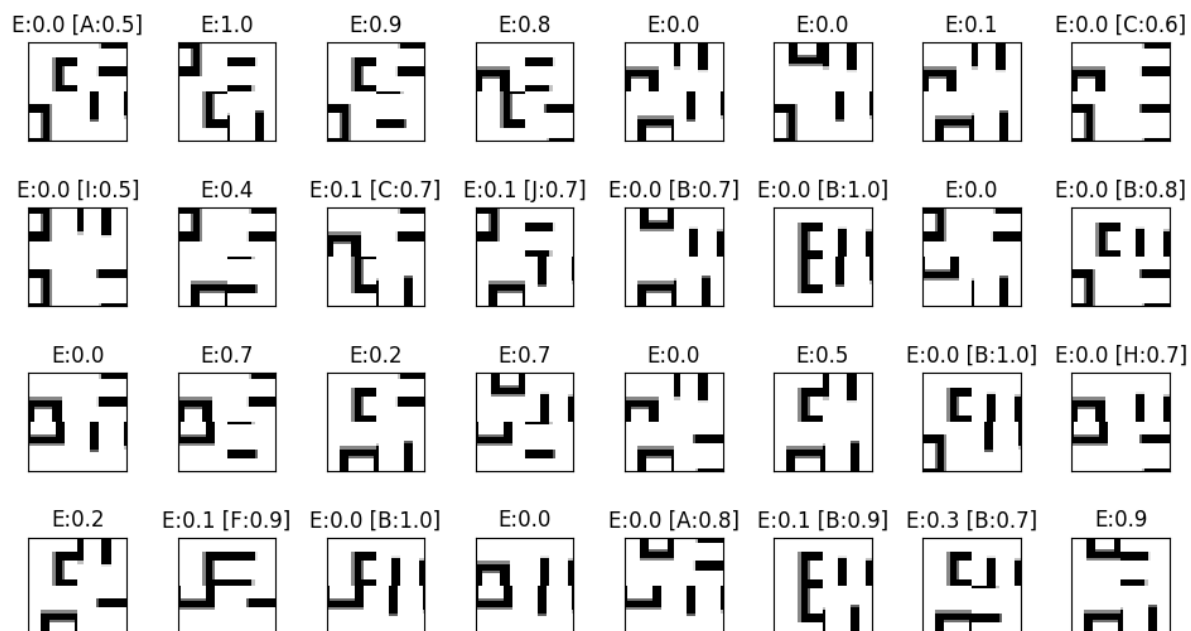


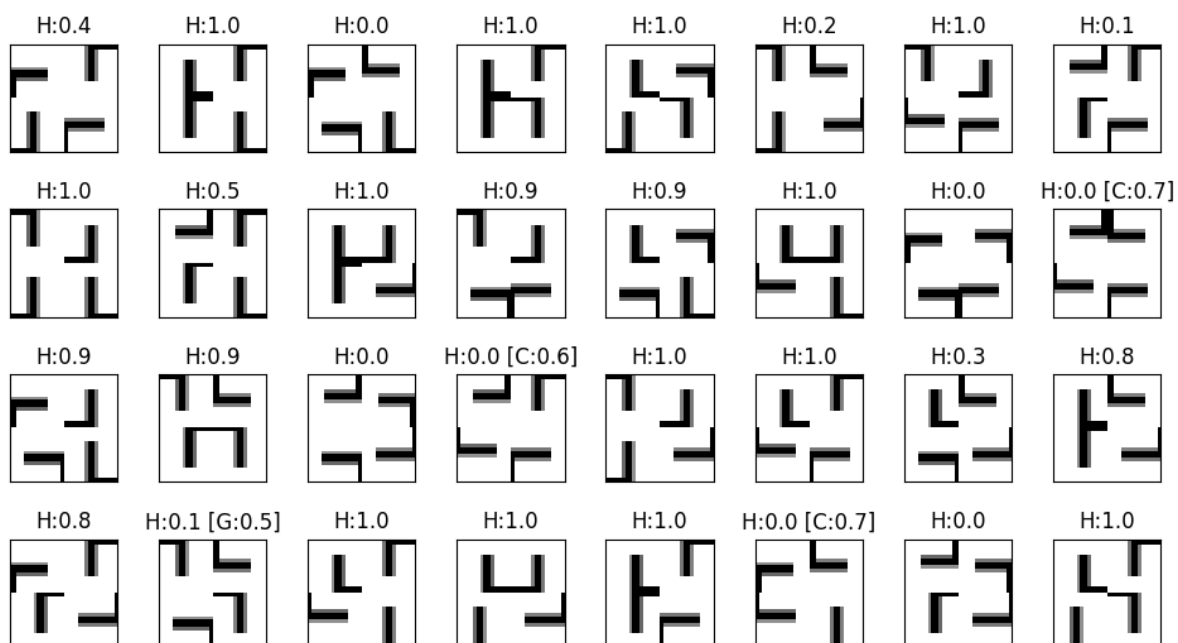
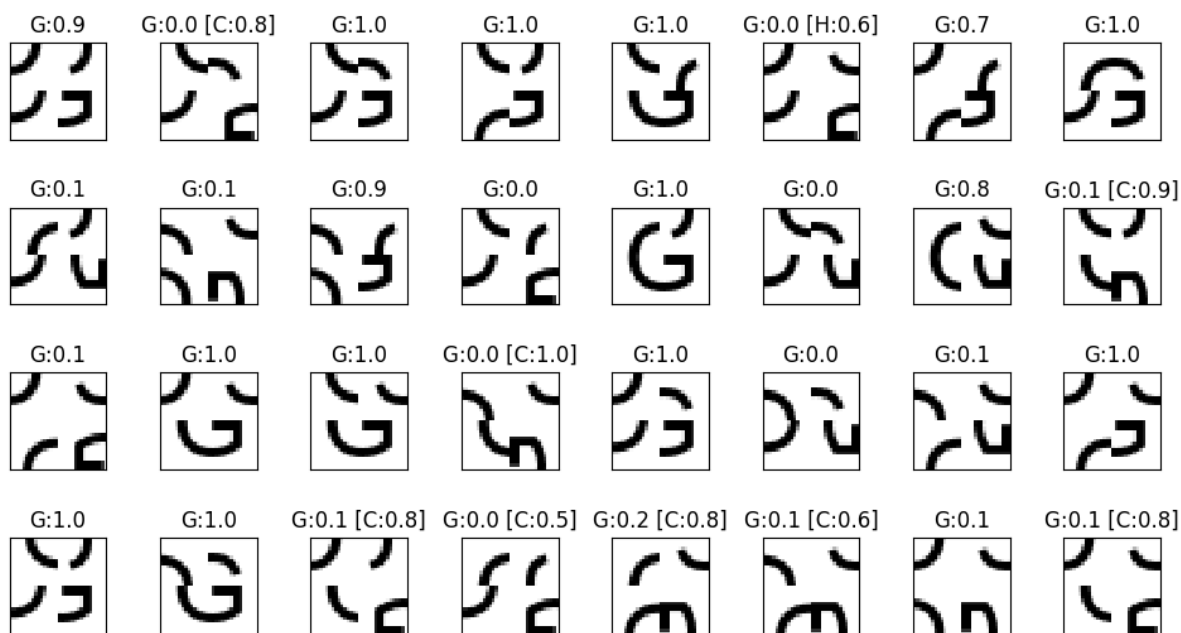


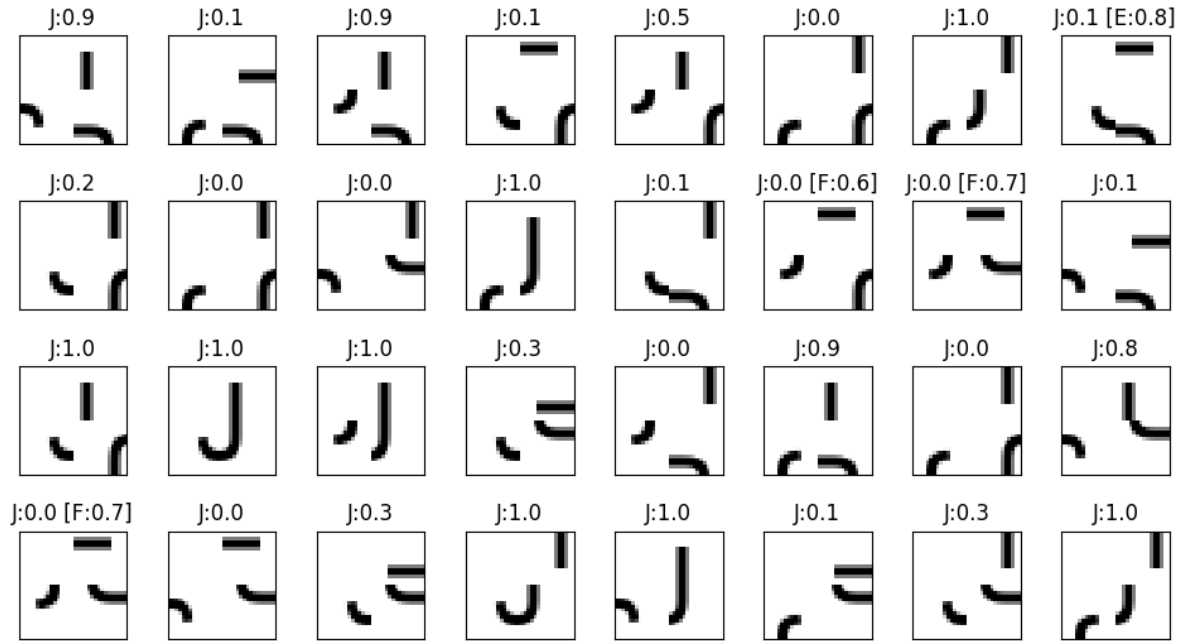
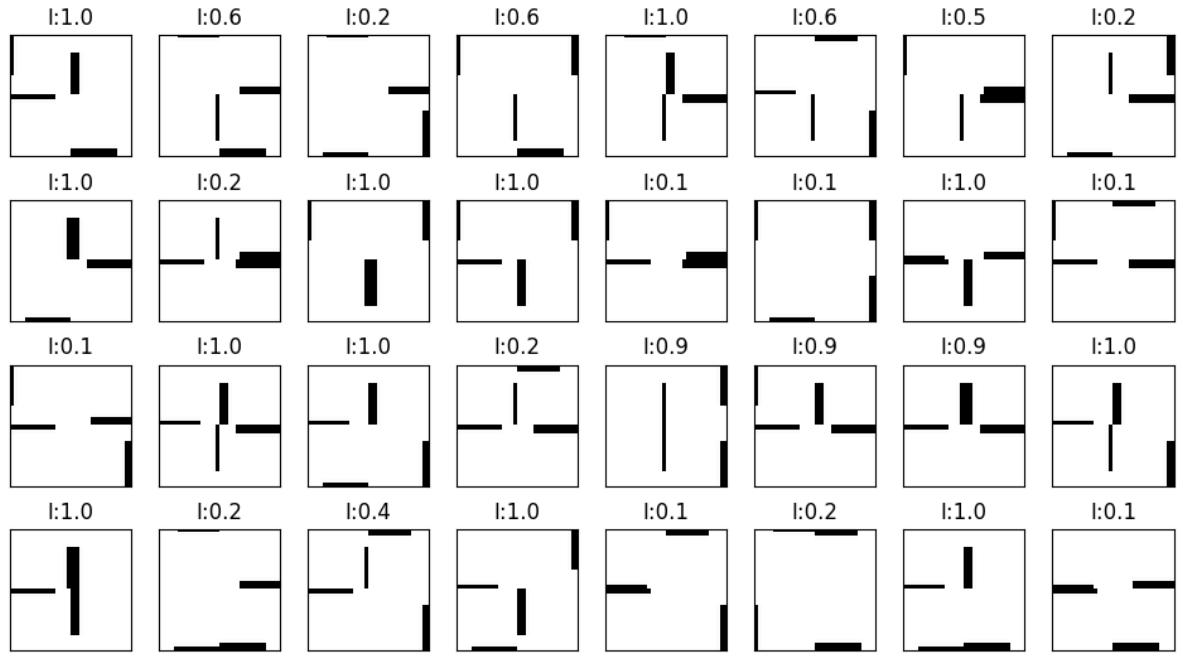
Результаты генераций для метода с вращениями квадрантов:







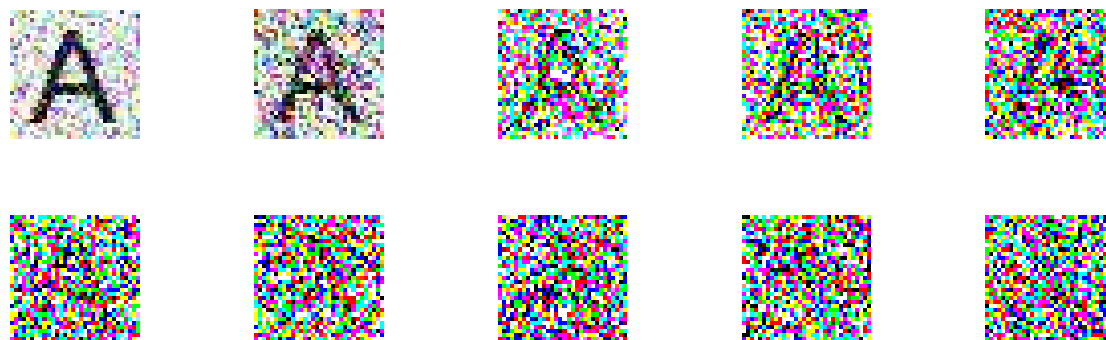




Результаты генераций для случая буквы «А» первым методом зашумления:



Результаты генераций для случая буквы «А» гауссовским зашумлением:



Список использованной литературы

- [1] <https://keras.io/> - документация Keras
- [2] <https://blog.plon.io/tutorials/cifar-10-classification-using-keras-tutorial/>
- [3] <https://docs.python.org/> - документация Python 3.7
- [4] <https://docs.scipy.org/doc/> - документация numpy, matplotlib
- [5] D.H. Johnson and D.E. Dudgeon, Array Signal Processing: Concepts and Techniques. – SNR
- [6] <https://www.mathworks.com/help/vision/ref/psnr.html> – PSNR
- [6] <http://pandas.pydata.org/pandas-docs/stable/> – документация Pandas
- [7] <https://seaborn.pydata.org/api.html> — документация Seaborn