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

Анваров Азиз Акмал угли

КУРСОВАЯ РАБОТА

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

по направлению 01.03.02 «Прикладная математика и информатика»

Научный руководитель
м.н.с. Соколов А.П.

«_____»_____ 2018г.

Ташкент 2018 г

Содержание

1. 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 3.6
- Библиотека Keras над Tensorflow
- Библиотеки matplotlib, numpy и другие.

2. Постановка задачи

Классификация изображений из набора данных CIFAR-10 является одной из наиболее распространенной 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 картинки 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 искаженных изображениях.

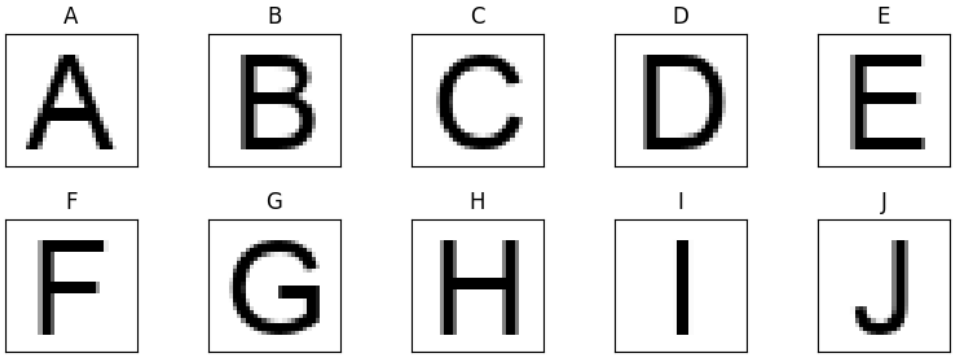
3. Реализация и результаты

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:

```
 7 def render_character(text: str, size: int = 32) -> Image:
 8     image = Image.new(mode='RGB', size=(size, size), color='white')
 9     draw = ImageDraw.Draw(image)
10     font = ImageFont.truetype("arial.ttf", 32)
11     w, h = draw.textsize(text, font=font)
12     draw.text((math.ceil((size - w) / 2), math.floor((size - h) / 2) - 2),
13               text, font=font, fill='black')
14     image.save(fp='gen/{}.png'.format(text))
15     return image
16 
17 
18 def generate_characters(letters: list, size: int = 32) -> np.array:
19     result = []
20     for letter in letters:
21         image = np.array(render_character(letter, size=size))
22         result.append(image.transpose((2, 0, 1)))
23     return np.array(result)
24 
25 
26 def generate_letters(count: int = 10) -> list:
27     if not 0 < count <= 26:
28         raise ValueError("count must be in range (0, 26]")
29     return ascii_uppercase[:count]
30 
31 
32 def generate_data_set(letters_count: int = 10, size: int = 32,
33                       rain_repeat: int = 1, test_repeat: int = 1):
34     letters = generate_letters(count=letters_count)
35     characters = generate_characters(letters, size=size)
36 
37     train_size = letters_count * train_repeat
38     train_shuffle = np.random.permutation(train_size)
39     x_train = characters.repeat(train_repeat, 0)[train_shuffle]
40     y_train = np.arange(letters_count).repeat(train_repeat)[train_shuffle]
41     .reshape(train_size, 1)
42 
43     test_size = letters_count * test_repeat
```

Результат генерации:



На данном наборе была обучена модель четырехслойной сверточной сети:

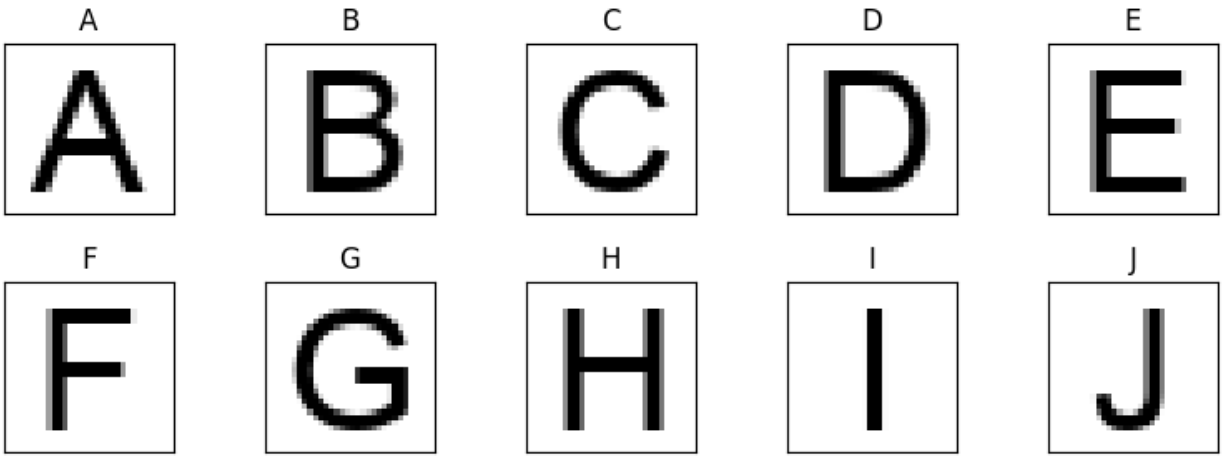
```
OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

              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
```

Обучение длиной в одну эпоху было достаточно для получения точности в 100%.

Первый метод обмана – наложение “шума” на изображения. Шумом назовем пикселы черного цвета, распределенные равномерно по всему изображению с заданной на вход вероятностью.

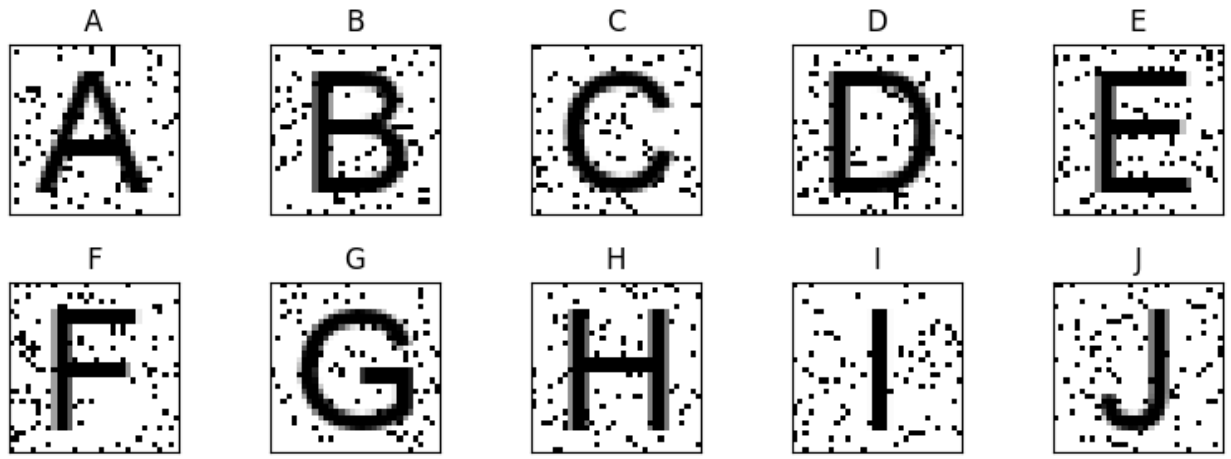
```python
def put_pixel(target: np.array, x: int, y: int, color: tuple = (0., 0., 0.)):
    for i in range(3):
        target[i][y][x] = color[i]


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[0], 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)
```

Результаты работы и классификации изображений:



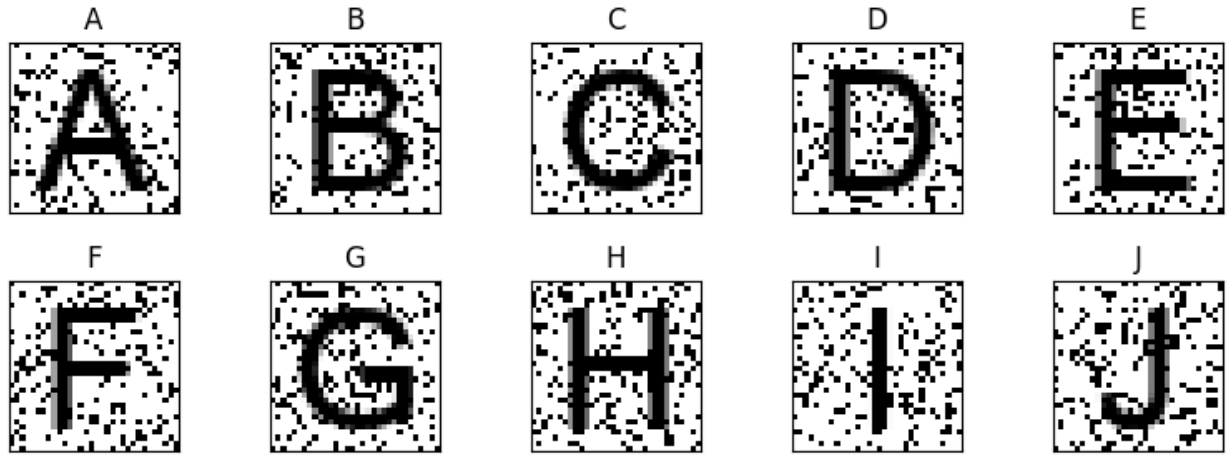

```
Noise level: 0.0
Cls A  Acc:1.0  A:1.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls B  Acc:1.0  A:0.00  B:1.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls C  Acc:1.0  A:0.00  B:0.00  C:1.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls D  Acc:1.0  A:0.00  B:0.00  C:0.00  D:1.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls E  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:1.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls F  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:1.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls G  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:1.00  H:0.00  I:0.00  J:0.00
Cls H  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:1.00  I:0.00  J:0.00
Cls I  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:1.00  J:0.00
Cls J  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:1.00
```

Собственно, сами изображения без искажений. Классификатор определяет все на 100%, что и было ожидаемо.




```
Noise level: 0.1
Cls A  Acc:1.0  A:1.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls B  Acc:1.0  A:0.00  B:1.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls C  Acc:1.0  A:0.00  B:0.00  C:1.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls D  Acc:1.0  A:0.00  B:0.00  C:0.00  D:1.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls E  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:1.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls F  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:1.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls G  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:1.00  H:0.00  I:0.00  J:0.00
Cls H  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:1.00  I:0.00  J:0.00
Cls I  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:1.00  J:0.00
Cls J  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:1.00
```

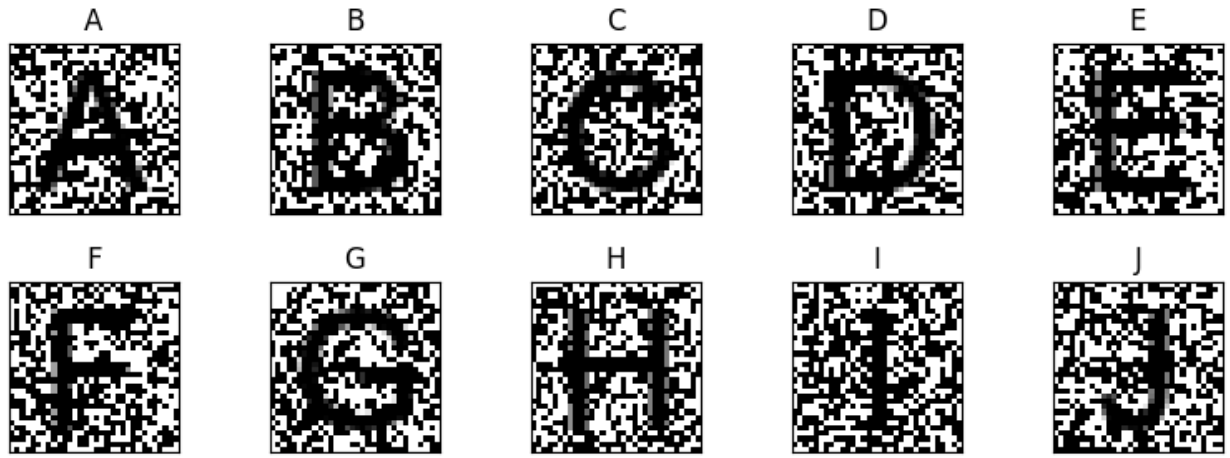
При коэффициенте шума в 10% результаты распознания не были ухудшены.




```
Noise level: 0.2
Cls A  Acc:1.0  A:1.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls B  Acc:1.0  A:0.00  B:1.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls C  Acc:1.0  A:0.00  B:0.00  C:1.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls D  Acc:1.0  A:0.00  B:0.00  C:0.00  D:1.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls E  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:1.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls F  Acc:0.9  A:0.00  B:0.00  C:0.00  D:0.00  E:0.07  F:0.93  G:0.00  H:0.00  I:0.00  J:0.00
Cls G  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:1.00  H:0.00  I:0.00  J:0.00
Cls H  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:1.00  I:0.00  J:0.00
Cls I  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:1.00  J:0.00
Cls J  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:1.00
```

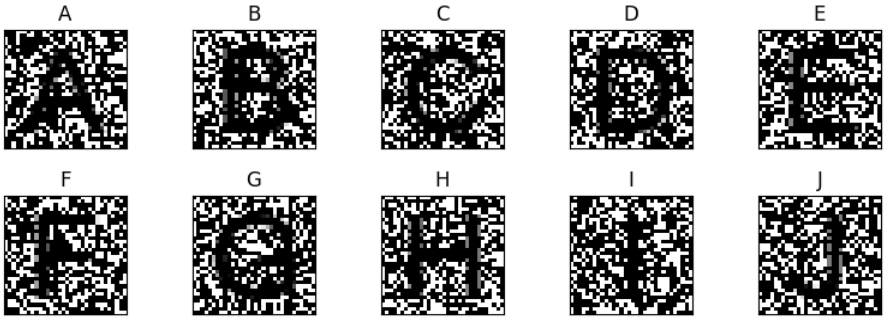
20%: незначительная потеря точности для буквы F. Пикселы попали таким образом, что стали проявлять характерные черты буквы E.

На 30% не было заметных изменений. На 40% - незначительные. Они опущены для краткости.

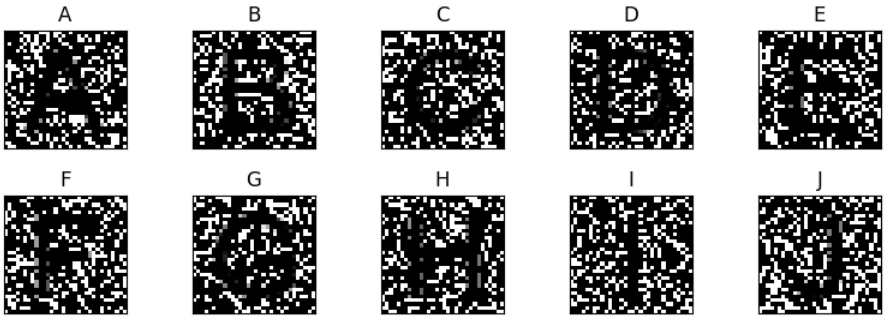



```
Noise level: 0.5
Cls A  Acc:1.0  A:1.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls B  Acc:1.0  A:0.00  B:0.97  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.01  I:0.00  J:0.01
Cls C  Acc:0.9  A:0.00  B:0.01  C:0.92  D:0.01  E:0.00  F:0.00  G:0.06  H:0.00  I:0.00  J:0.00
Cls D  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.99  E:0.00  F:0.00  G:0.00  H:0.00  I:0.00  J:0.00
Cls E  Acc:0.0  A:0.03  B:0.42  C:0.00  D:0.00  E:0.02  F:0.00  G:0.01  H:0.46  I:0.03  J:0.04
Cls F  Acc:0.2  A:0.01  B:0.01  C:0.00  D:0.00  E:0.44  F:0.21  G:0.18  H:0.02  I:0.05  J:0.09
Cls G  Acc:1.0  A:0.00  B:0.00  C:0.01  D:0.00  E:0.00  F:0.00  G:0.98  H:0.01  I:0.00  J:0.00
Cls H  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:1.00  I:0.00  J:0.00
Cls I  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.01  H:0.00  I:0.98  J:0.00
Cls J  Acc:1.0  A:0.00  B:0.00  C:0.00  D:0.00  E:0.00  F:0.00  G:0.00  H:0.00  I:0.01  J:0.99
```

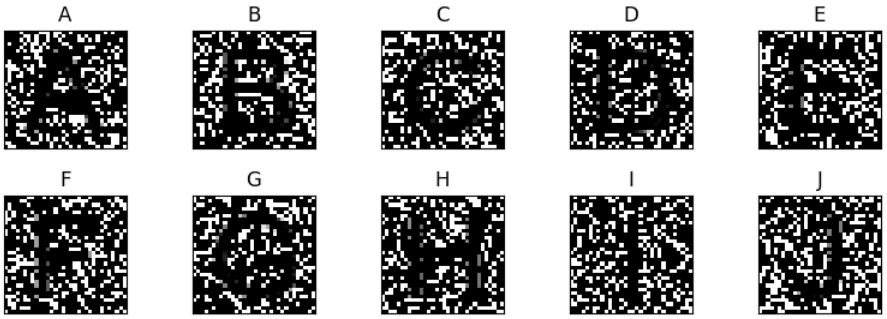
50%: модель перестает видеть некоторые буквы. Также усложняется визуальное восприятие.




```
Noise level: 0.6
Cls A  Acc:1.0
Cls B  Acc:0.6
Cls C  Acc:0.5
Cls D  Acc:0.5
Cls E  Acc:0.6
Cls F  Acc:0.0
Cls G  Acc:0.9
Cls H  Acc:1.0
Cls I  Acc:0.8
Cls J  Acc:1.0
```



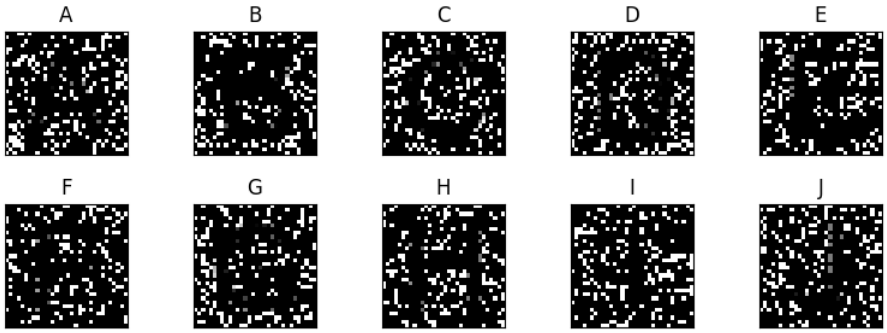

```
Noise level: 0.7
Cls A  Acc:0.8
Cls B  Acc:0.7
Cls C  Acc:0.0
Cls D  Acc:0.4
Cls E  Acc:0.4
Cls F  Acc:0.1
Cls G  Acc:0.8
Cls H  Acc:0.9
Cls I  Acc:0.8
Cls J  Acc:0.7
```




```
Noise level: 0.8
Cls A  Acc:0.4
Cls B  Acc:0.2
Cls C  Acc:0.0
Cls D  Acc:0.4
Cls E  Acc:0.1
Cls F  Acc:0.0
Cls G  Acc:0.5
Cls H  Acc:0.8
Cls I  Acc:0.4
Cls J  Acc:0.4
```

Модель оказалась заметно стойкой к зашумлению даже там, где визуально сложно понять какая из букв изображена.

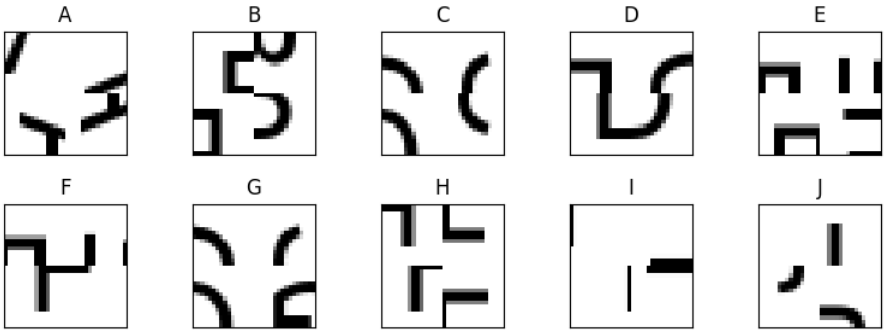
Полная потеря способности распознания была получена при уровне зашумления в 90%:



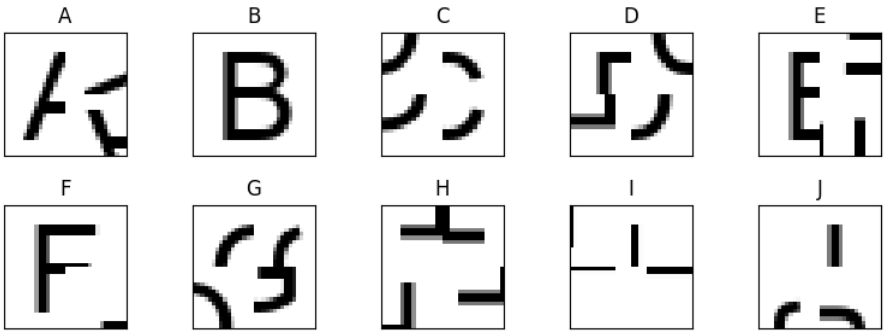

```
Noise level: 0.9
Cls A  Acc:0.1
Cls B  Acc:0.1
Cls C  Acc:0.0
Cls D  Acc:0.0
Cls E  Acc:0.0
Cls F  Acc:0.0
Cls G  Acc:0.2
Cls H  Acc:0.2
Cls I  Acc:0.2
Cls J  Acc:0.2
```

Второй метод заключался в разделении изображений на 4 равных части и поворот на случайно выбранный угол, кратный 90°.

```
def apply_transform(target: np.array, k: int):
    return np.rot90(target, k=k)
def process_image(target: np.array, size: int = 32):
    def process_quadrant(clip):
        _result = []
        k = random.randint(0, 3)
        for channel in target:
            _result.append(apply_transform(channel[clip(size)], k))
        return np.array(_result)
    def concat(im1, im2, axis):
        _result = []
        for i in range(len(im1)):
            _result.append(np.concatenate((im1[i], im2[i]), axis))
        return np.array(_result)
    ul = process_quadrant(up_left)
    ur = process_quadrant(up_right)
    dl = process_quadrant(down_left)
    dr = process_quadrant(down_right)
    up = concat(ul, ur, axis=1)
    dn = concat(dl, dr, axis=1)
    return concat(up, dn, axis=0)
def add_effect(targets: np.array, size: int = 32):
    result = []
    for image in targets:
        result.append([process_image(target=image[0], size=size)])
    result = np.array(result)
    return result
```




```
Cls A  Acc:0.0
Cls B  Acc:0.0
Cls C  Acc:0.0
Cls D  Acc:0.2
Cls E  Acc:0.0
Cls F  Acc:0.0
Cls G  Acc:0.0
Cls H  Acc:0.9
Cls I  Acc:0.5
Cls J  Acc:1.0
```




```
Cls A  Acc:1.0
Cls B  Acc:1.0
Cls C  Acc:0.1
Cls D  Acc:1.0
Cls E  Acc:1.0
Cls F  Acc:1.0
Cls G  Acc:1.0
Cls H  Acc:0.5
Cls I  Acc:1.0
Cls J  Acc:1.0
```

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

4. 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.

5. 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

[4] https://docs.scipy.org/doc/ - документация numpy, matplotlib