Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики

АДАПТИВНІ МЕТОДИ АНОНІМІЗАЦІЇ ДАНИХ

Текстова частина до курсової роботи
за спеціальністю 113 „Прикладна математика”

Керівник курсової роботи
к.ф.-м.н., ст. в. Швай Н.О.
_____
(підпис)
“____” _____2021 р.

Виконала студентка МП-1
факультету інформатики
Ронська Д.Р.
“____” _____2021 р.

Київ 2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
проф., д.ф-м.н.
_____ Олійник Б.В.
(підпис)
„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студентки Ронської Дарини Романівни факультету інформатики 5-го курсу
ТЕМА. Адаптивні методи анонімізаціх даних
Вихідні дані:

- прогноз класів для зображень з синтетинчою анонімізацією (розмиття чутливої частини зображення) та прогноз класі після використання градієнтного методу для наближення до оригінального класу тестової частини датасету ImageNet за допомогою моделі MobileNetV2;;
- рохрахунок та порівняння результатів, оцінка використання градієнтного методу для адаптації анонімізованого зображення.

Зміст ТЧ до курсової роботи:

Зміст
Анотація
Вступ
1 Проблема анонімізованих даних
2 Градієнтний метод
3 Адаптивний експеримент на даних ImageNet
Висновки
Список використаних джерел
Додатки

Дата видачі „___” _____ 2021 р.

Керівник
_____Швай Н.О.
(підпис)

Завдання отримала
_____Ронська Д.Р.
(підпис)

Тема: Адаптивні методи анонімізаціх даних.

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	10.10.2020	
2.	Пошук та збір технічної літератури за темою роботи.	15.04. 2021	
3.	Огляд відповідних матеріалів та створення структури роботи.	17.04. 2021	
3.	Написання вступу та плану роботи.	19.04. 2021	
4.	Написання першого розділу.	23.04. 2021	
5.	Написання другого розділу.	29.04. 2021	
6.	Написання коду для практичного досліду.	5.05. 2021	
7.	Написання третього розділу.	8.05. 2021	
8.	Написання висновків.	9.05. 2021	
9.	Оформлення джерел.	11.05. 2021	
10.	Коректне оформлення роботи відповідно до вимог написання курсової роботи.	11.05. 2021	
11.	Створення презентації та написання доповіді для захисту роботи.	17.05. 2021	

Студентка
Ронська Д.Р.

Керівник
Швай Н.О.

“_____”_____

TABLE OF CONTENTS

ABSTRACT

Sometimes it is impossible to use the original image and only anonymized version of it is available (e.g., faces of the people or plate numbers on cars are blurred). In other words, we can use only edited version of the original image. Sometimes the class of the edited image is different from original one and we want to avoid this.

This work is about gradient method which allows to get the class predicted for unchanged image for the one with blurred sensitive part by applying small changes in the edited area only.

INTRODUCTION

Cameras are now commonplace and used for a variety of purposes, ranging from surveillance to information acquisition to the advancement of AI-driven technology. Every day, vast volumes of picture and video data are gathered in public for the advancement of autonomous cars, high-definition maps, and smart retail analytics.

Companies, government organizations, and individuals are expected to encrypt personal information, which includes biometric data in photographs and videos, as a result of increasing regulations across the world, such as the GDPR in the EU, the CCPA in the US, the CSL in China, and the APPI in Japan. Although various regions' privacy laws provide different legal bases for data gathering and distribution, they all share one thing in common: consent.

Data privacy is a big concern in the digital age. Concerns about image data usage affect the way the data is shared and publicized. When it comes to image data, we may want to make some information not visible for others.

There are multiple ways to make some part of the image anonymized, but the most popular are pixelating and blurring. The main problem of these methods is that the classification model may predict the class which is different from the class of the original image if it is not robust (e.g. trained on some augmented data with the particular type of data anonymization). In case some part of the image is blurred we can try to apply gradient method on the edited (blurred) area and achieve the same class that is predicted for the original image.

In this work gradient method and its application for anonymized images adaptation to existing model is explained. Practice part includes experiment with ImageNet test set, where the sensitive part of the image was blurred and fast gradient method was applied to edited images with wrong class. Accuracy of the model on images with gradient method applied is calculated to estimate the results.

The first chapter is about the problem of anonymized images and classification models, also contains some examples.

The second chapter is about Fast Gradient Sign Method (FGSM), which is the main inspiration for this work – it is used for adversarial attacks, which is completely the opposite task, but in both tasks gradient values are used to change the image.

The third chapter is about gradient method application on the ImageNet test set and results estimation.

CHAPTER 1. Anonymized data problem

1.1. Data anonymization definition and reasoning

Data anonymization is the process to protect private or sensitive information by its transformation in such a way that a data subject can not be identified. It refers to data encryption or stripping identifying or personal information from data. In other words, except for the group responsible for the anonymization, it should be difficult to extract insights on a specific person from anonymized data. When handled correctly, such data is not confidential by itself and therefore is not subject to data privacy laws. Anonymization entails the removal of PIIs such as faces and bodies, as well as license plates, from image and video files. This protects publicly identifying details in pictures and videos from being identified by facial or license plate recognition devices.

Healthcare, business, government and other organizations store more and more individuals' information locally or using cloud servers, so data anonymization is crucial to prevent security violation.

Data anonymization is used in most industries, that deal with sensitive data to reduce the risk of unintended personal information disclosure data when sharing data between countries, companies, departments etc.

There is still the possibility that anonymized data will lose its anonymity over time. Any of the methods previously anonymous data sets have been de-anonymized include combining the anonymized dataset with other data, innovative methods, and brute strength. The subjects of the data are no longer anonymous.

Anonymized data is cross-referenced with other data sources to re-identify the anonymous data source, which is known as de-anonymization. The two most common approaches to anonymizing relational data are generalization and perturbation. Pseudonymization is the method of obscuring data with the potential

to re-identify it later, and it is one way for businesses to preserve data in a HIPAA-compliant manner.

Generalization, suppression, anatomization, permutation, and perturbation are the five methods of data anonymization operations.

In order to comply with relevant privacy laws, a variety of technological and operational measures are available. Basic standards of data gathering and processing (e.g. GDPR's purpose restrictions, data minimization, and storage limitation) to specific technical means like encryption and decentralized processing to "classic" TOMs like privacy are all used to improve enforcement.

In order to comply with relevant privacy laws, a variety of technological and operational measures are available. Basic standards of data gathering and processing (e.g. GDPR's purpose restrictions, data minimization, and storage limitation) to specific technical means like encryption and decentralized processing to "classic" TOMs like privacy are all used to improve compliance.

1.2 Image data anonymization and classification issue

Image data anonymization in most cases is about blurring, pixelating or cutting out some sensitive part of the image (Figure 1.1).

If classification model is not trained on data with particular type of anonymization, then it is likely to struggle with anonymized images (Figure 1.2.). Model can be robust to such images if it is trained on augmented data, but it is not always possible and we should deal with the classification model given as it is. In this case some additional pre- or postprocessing of the input image or its model output is needed.

In this work gradient method is used to change the image with blur anonymization in such a way that model will predict the class of the original image, nevertheless the image remains anonymized.

Figure 1.1. Types of image data anonymization

Figure 1.2. MobileNetV2 predict for original and anonymized images

CHAPTER 2. Gradient method

2.1. Gradient Descent

Gradient Descent is a very general optimization algorithm that can find optimal solutions to a wide variety of problems. Gradient Descent's basic concept is to iteratively tweak parameters in order to minimize a cost function.

To implement Gradient Descent, you must first calculate the cost function's gradient with respect to each model parameter θ_j. In other words, you must determine how much the cost function would change if θ_j is changed slightly. A partial derivative is what this is called.

Instead of computing partial derivatives separately, using equation above they can be computed in one go. The gradient vector $\nabla_\theta J(\theta)$ contains partial derivates for all θ_j of the cost function $J(\theta)$:

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} J(\theta) \\ \frac{\partial}{\partial \theta_1} J(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\theta) \\ \end{pmatrix}$$

Once the gradient vector is calculated, which points uphill and we need to go in the opposite direction (minimizing loss function). This is done by subtracting $\nabla_\theta J(\theta)$ multiplied by learning rate η from θ iteratively:

$$\theta^{(next\ step)} = \theta - \eta \nabla_\theta J(\theta)$$

2.2. Fast Gradient Sign Method (FGSM)

Several machine learning models, like neural networks, routinely misclassify adversarial examples—inputs created by adding tiny yet deliberately worst-case perturbations to dataset examples, resulting in the algorithm producing an incorrect response with high confidence. Nonlinearity and overfitting were initially proposed as explanations for this phenomenon. Instead, it is argued that the linear structure of neural networks is the primary cause of their susceptibility to adversarial perturbation. This theory is backed up by recent quantitative findings, and it explains for the first time the most intriguing aspect of them: their ability to generalize through architectures and training sets. This viewpoint leads to a quick and easy way to generate adversarial examples.

Adversarial examples make many machine learning models, including state-of-the-art neural networks, vulnerable. That is, these machine learning models misclassify data distribution examples that are only slightly different from correctly categorized examples. Many models with different architectures trained on different subsets of the training data misclassify the same adversarial example in many situations. As a result, adversarial examples can reveal fundamental flaws in our training algorithms.

The cause of these adversarial examples was unknown, but speculative theories indicated it was due to deep neural network nonlinearity, perhaps coupled with inadequate model averaging and regularization of the strictly supervised learning problem. These speculative theories are shown to be unnecessary. In high-dimensional spaces, linear behavior is enough to provide adversarial examples. This perspective allows one to implement a fast method for producing adversarial examples, making adversarial training feasible.

Adversarial images are created with purpose to confuse neural network, so the image is misclassified by the model. The changes applied to the image are indistinguishable to the human, but they make the model fail to predict correctly

the contents of the image. There are a few types of such attacks and FGSM is one of them. It is white box attack, which means that there should complete access to the model attacked. The example of such attack is Figure 2.1. Attacker adds perturbations to the input image with panda, and the model predicts this transformed image as gibbon.

The fast gradient sign method (FGSM) uses the gradient values of the neural network to create adversarial example. The method uses input image gradients of the loss to create a new image with maximized loss, which is called the adversarial image. The expression to achieve adversarial image is the following:

$$adv\_x = x + \epsilon * sign(\nabla_x J(\theta, x, y)),$$

where

- *adv_x* – adversarial image,
- *x* – original input image,
- *y* – original input label,
- ϵ – multiplier to ensure the perturbations are small,
- θ – parameters of the model,
- *J* – loss.

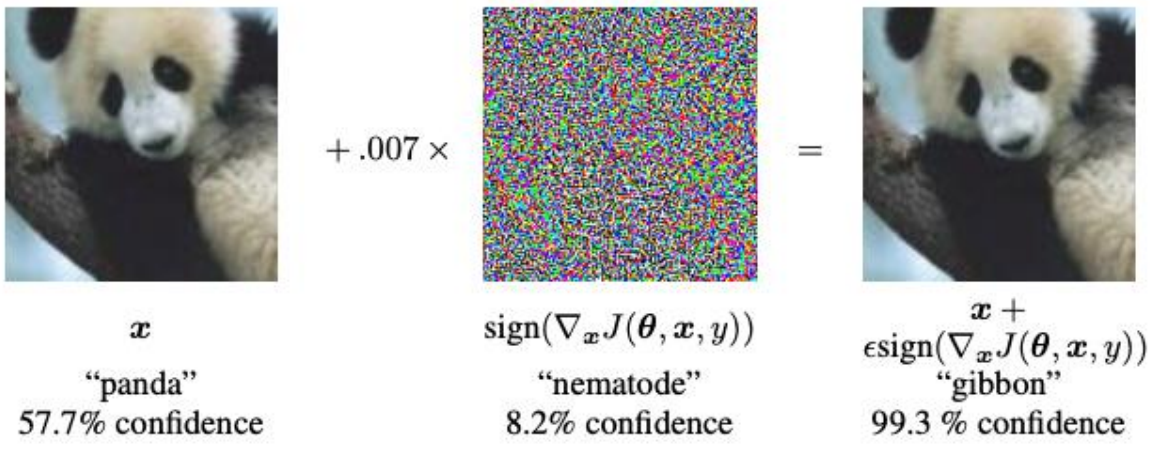

Figure 2.1. Example of using FGSM to create adversarial image

In this case, gradients are taken respectively to the input image, and the objective of the method is to create the image that maximizes the loss. Perturbations are calculated from how much each pixel in the input image contributes to the loss value. This method is called fast because it is relatively easy to find how much each pixel contributes to the loss, and using chain rule it is easy to find required gradients. Fast gradient sign method is used to fool already trained model only – it does not change the parameters (weights) of the model.

2.3 Relevance to the problem and changes

The idea of FGSM for adversarial attacks is completely the opposite to the adaptation of the anonymized image, nevertheless the method used for this is pretty similar to FGSM attack (Figure 2.2.).

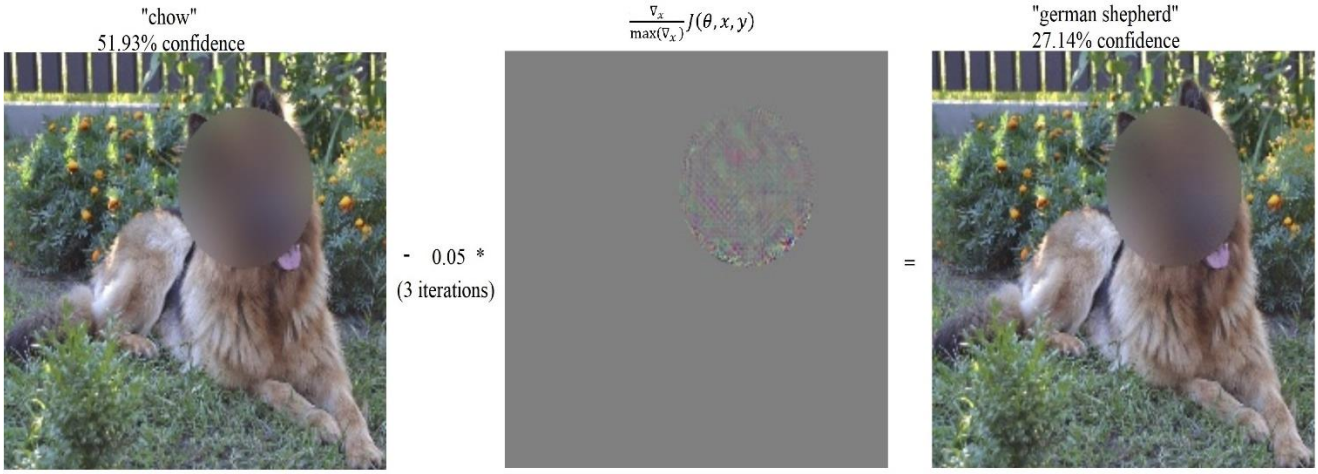

Figure 2.2. Gradient method applied on blur anonymized image iteratively

To change the image with the blur anonymized part we should change the formula in the following way:

$$adapt\_x = clip(x[blur] - \epsilon * \frac{\nabla_x}{\max(\nabla_x)} J(\theta, x, y)[blur]),$$

where

- *adapt_x* – adapted blurred part of the image,
- *blur* – blurred part mask of the image,

- x – original input image with blurred part,
- y – original input label,
- ϵ – multiplier to ensure the perturbations are small,
- *clip(t)* – function *max(min(t, -1), 1)*,
- θ – parameters of the model,
- J – loss.

This process should be done iteratively until the max number of iterations performed or target class is achieved.

Notice, that in this case we change the image in the blur anonymized part only.

CHAPTER 3. Adaptation experiment on ImageNet

For the experiment ImageNet test set is used and model MobileNetV2 is used for image classification.

3.1. ImageNet description

For the practice part ImageNet test set is used. The ImageNet is a massive graphic library created to aid in the development of visual object recognition application. The project has hand-annotated over 14 million photos to indicate the objects depicted, with bounding boxes given in at least one million of the images.

There are over 20,000 categories in ImageNet, with a common category like "balloon" or "strawberry" containing several hundred images. ImageNet provides a free archive of annotations for third-party image URLs, but the images themselves are not owned by ImageNet.

Since 2010, the ImageNet project has hosted the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an international software competition in which software programs compete to correctly identify and detect objects and scenes. A "trimmed" list of 1,000 non-overlapping groups is included in the challenge.

3.2. MobileNet description

MobileNet is a convolutional neural network which is used for image classification and mobile vision. Other models exist, but what makes MobileNet unique is that it needs relatively little computational resources to run or implement

transfer learning (Figure 3.1).

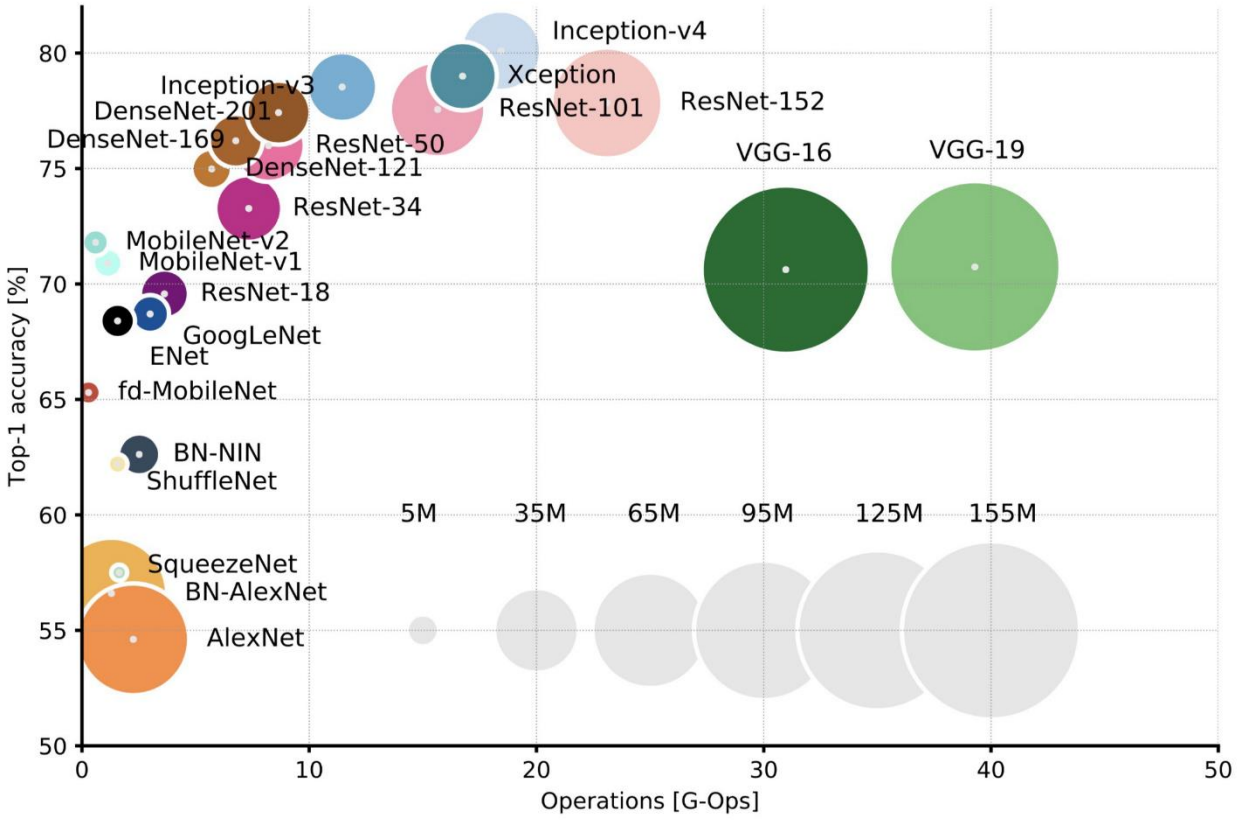

Figure 3.1. Ball chart reporting the Top-1 accuracy vs. computational complexity

This makes it ideal for mobile devices, embedded systems, and computers with poor computing performance or no GPU, without compromising the precision of the results substantially. It's also ideal for web browsers, which have limitations in terms of computing, graphics processing, and storage.

MobileNets, which are built on a streamlined architecture that uses depthwise separable convolutions to create light weight deep neural networks, are proposed for smartphone and embedded vision applications. Two basic global hyper-parameters that effectively trade off latency and accuracy are introduced.

Depthwise separable filters, also known as Depthwise Separable Convolution, are core layer of MobileNet. Another aspect that improves efficiency is the network structure. Finally, the width and resolution of the image can be adjusted to balance latency and accuracy.

In MobileNetV2, a better module is introduced with inverted residual structure (Figure 3.2.). Non-linearities in narrow layers are removed from the structure.

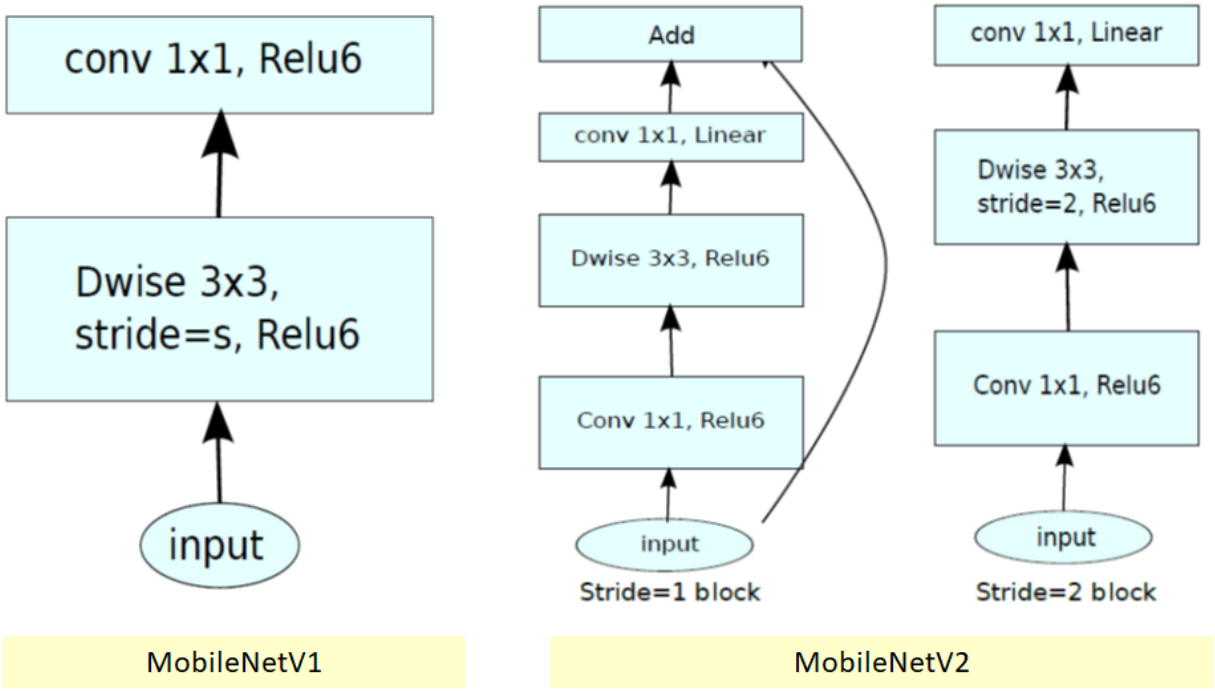

Figure 3.2. The architectures of MobileNetV1 and MobileNetV2.

3.3. Gaussian blur

The Gaussian blur functionality is obtained by blurring (smoothing) an image using a Gaussian function to minimize the noise level. It can be thought of as a nonuniform low-pass filter that maintains low spatial frequency while reducing image noise and minor information. It's usually done by using a Gaussian kernel to convolve a picture.

This Gaussian kernel in 2-D form is expressed as:

$$G_{2D}(x, y, \theta) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}},$$

Where σ is the standard deviation of distribution and x and y are the position indices. The value σ of determines the magnitude of the blurring effect around a pixel by controlling the variation around a mean value of the Gaussian distribution.

3.4. Blur anonymization application

ImageNet is not already anonymized, so synthetic way of anonymization was used which is explained here.

Firstly, gradient is computed, so we can define the most sensitive part of the image as the area that has the highest absolute gradient values (Figure 3.3).

The size and shape of the area is fixed in this experiment – circle with radius 40, given the size of the image is reshaped to 224x224 for this model.

The sum of absolute gradient values for the area with randomly selected center coordinates on the image are calculated. This random picking of the center and gradient computation is done 100 times, and then area with highest sum of absolute gradient values is selected as area to apply gaussian blur with factor 3 (Figure 3.4.).

Figure 3.3. Left: original image, right: gradient values of the image

Figure 3.4. Area with highest absolute gradient values

3.5. Experiment description

In 2.2. gradient method was applied to the blur anonymized image with a german shepherd on it which was misclassified as chow. After 3 iterations with epsilon equals to 0.05 the predicted class was correct.

ImageNet test set contains 100,000 of images, so applying the gradient method to all of them will give us a possibility to estimate the results.

The steps for each image were followed:

1) Predicting class for the original image.
2) Applying blur anonymization, as described in 3.3.
3) Predicting class for the image with blurred area.
4) If the classes predicted in steps 1) and 3) are different then gradient method with epsilon and number of iterations equal to 0.05 and 10 respectively is applied to get the correct class.

Sometimes it is not possible to get the correct result with this number of iterations, but more iterations are applied more distinguishable to the human eye

those changes are. To keep this method fast and make changes with the image relatively not visible small number of iterations is used.

Python is used as a programming language and TensorFlow framework is used for modeling. Some other packages are used for image preprocessing and results visualization are pandas, numpy, cv2, matplotlib.

3.5. Experiment results

After blurring the most sensitive part of the image (as described in 3.3.) 47,679 out of 100,000 images was predicted as class, which is different from the class of original image. Gradient method is applied only to those images which changed their class after blur anonymization application.

After gradient method was applied, 45,990 out of 47,679 anonymized images (96.46%) had the same class as before anonymization added. 1689 out of 47679 images (3.54%) did not change the class same to anonymized image (Table 3.1.).

Result	Number of images	Ratio
Successful (label is corrected)	45,990	96.46%
Successful (label is not corrected)	1,689	3.54%

Table 3.1. Blur anonymized images adaptation using gradient method on ImageNet test set

Overall distribution of resulting labels is on the Figure 3.5. This shows that 52,321 (52.3%) images have their classes changed by blur anonymization, 45,990 (46%) images have their classes successfully corrected by gradient method and only 1,689 (1.7%) images remained incorrectly classified after gradient method.

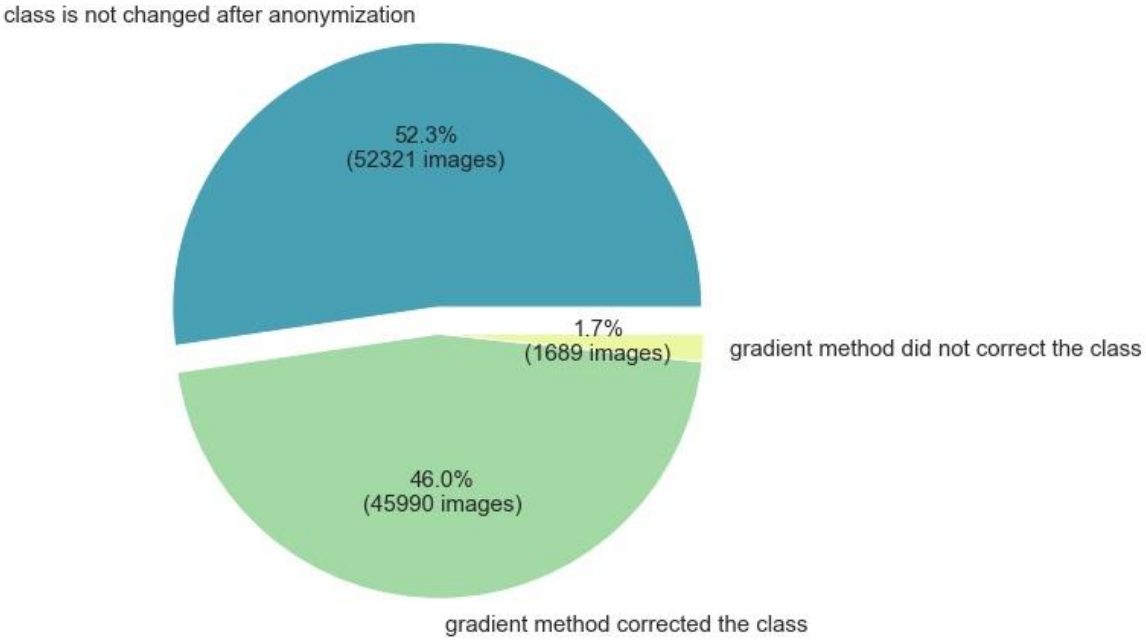

Figure 3.5. Pie chart of test set distribution

The most difficult class to adapt is “basset” – only 28 out of 43 images (65.11%) of this class were correctly predicted gradient method application. Some other classes that are hard to adapt are “standard schnauzer” (40 out of 54 (74.07%)) are correct and “hippopotamus” (15 out of 19 (78.95%)).

Overall class accuracies histogram look as follows (Figure 3.6.)

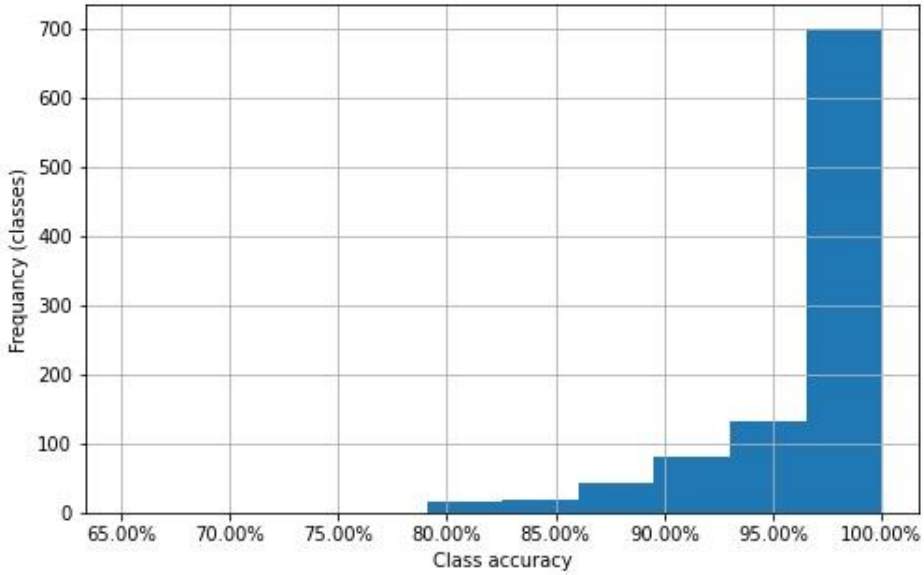

Figure 3.6. Class accuracies histogram

SUMMARY

Data anonymization is essential nowadays. The amount of publicly opened information is growing, so appears more restrictions and attention is more pointed than before.

When it comes to image data there are plenty of techniques to make sensitive parts of the image anonymized. Most popular simple techniques are blob, blur and pixelate anonymization, but also some other AI-driven techniques are used to make some parts of image anonymized. Some of them are used to solve the problem of image segmentation, localization or/and recognition to identify the sensitive area of the image and then some simple techniques mentioned above, others use auto-encoders to change the image sensitive part, so the anonymized part does not change the class, but looks unrecognizable (e.g., a face is transformed is such a way that it looks as face of another person or a numberplate's letters and digits are substituted by other characters).

Gradient descent is an optimization algorithm to train a machine learning model. It's built on a convex function that iteratively tweaks its parameters to reduce a function to its local minimum. Fast Gradient Sign Method (FGSM) is an algorithm to create adversarial images – images that have been altered with small values, such that image looks almost the same to the human eye but is misclassified by the model. The FGSM creates an adversarial example by using the neural network's gradients. The method creates a new image that maximizes the loss for an input image by using the gradients of the loss with respect to the input image.

In this work gradient method for blur anonymized images adaptation is being presented. It works somewhat similar to FGSM and the main difference is that we move against the direction of the gradient. Also, instead of sign function normalization is used for gradient values. Computed gradient values multiplied by small slope are subtracted from the blurred area of the image only, to ensure that

only anonymized part of the image is being changed. Doing this process iteratively the class similar to the class of the original image can be achieved.

The gradient method for blur anonymized images adaptation is proved on ImageNet test set using MobileNetV2 model for image classification. The most sensitive part of each image was identified by the largest sum of absolute gradient values in the area. After this the area is anonymized using gaussian blur. Then gradient method is used to revert class of the newly created image to the original one it is changed.

For the experiment on ImageNet test set gradient method is applied iteratively with maximum number of iterations equals to 10 and epsilon (multiplier to subtract small values from blurred image) equals to 0.05. After gradient method was applied, 45,990 out of 47,679 anonymized images (96.46%) had the same class as before anonymization added and only 1689 out of 47679 images (3.54%) did not change the class same to the anonymized image. The results of the experiment prove the effectiveness of the gradient method used for blur anonymized images adaptation.

REFERENCES

1. Goodfellow, Ian J., et al. “Explaining and Harnessing Adversarial Examples.” *ICLR*, 2015, arxiv.org/pdf/1412.6572.pdf.
2. Géron, Aurélien. “Gradient Descent.” Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 1st ed., O’Reilly Media, 2017, pp. 150–56.
3. Bianco, Simone, et al. “Benchmark Analysis of Representative Deep Neural Network Architectures.” IEEE Access, vol. 6, 2018, pp. 64270–77. Crossref, doi:10.1109/access.2018.2877890.
4. Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.” Communications of the ACM, vol. 60, no. 6, 2017, pp. 84–90. Crossref, doi:10.1145/3065386.
5. “Adversarial Example Using FGSM | TensorFlow Core.” TensorFlow, www.tensorflow.org/tutorials/generative/adversarial_fgsm. Accessed 11 May 2021.

APPLICATION A. Gradient method application (Python code)

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import os
import random
import numpy as np
import pandas as pd

import tensorflow as tf
import matplotlib as mpl
import matplotlib.pyplot as plt
import cv2


# In[2]:


mpl.rcParams['figure.figsize'] = (8, 8)


# In[3]:


def set_seed(seed = 42):
    """Set seed for reproducibility.
    """
    os.environ['PYTHONHASHSEED']=str(seed)
    os.environ['TF_CUDNN_DETERMINISTIC'] = '1'  # new flag present in tf 2.0+
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
set_seed()


# In[4]:


pretrained_model = tf.keras.applications.MobileNetV2(include_top=True,
                                                      weights='imagenet')
pretrained_model.trainable = False

# ImageNet labels
decode_predictions = tf.keras.applications.mobilenet_v2.decode_predictions
```

```
# In[5]:


# Helper function to preprocess the image so that it can be inputted in MobileNetV2
def preprocess(image):
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, (224, 224))
    image = tf.keras.applications.mobilenet_v2.preprocess_input(image)
    image = image[None, ...]
    return image

# Helper function to extract labels from probability vector
def get_imagenet_label(probs):
    return decode_predictions(probs, top=1)[0][0]


# In[6]:


def predict_and_show(image):
    image_probs = pretrained_model.predict(image)
    plt.figure()
    plt.imshow(image[0] * 0.5 + 0.5)  # To change [-1, 1] to [0,1]
    _, image_class, class_confidence = get_imagenet_label(image_probs)
    plt.title('{} : {:.2f}% Confidence'.format(image_class, class_confidence*100))
    plt.show()


# In[7]:


def save_image(filename, tensor, save_dir = './results/'):
    return cv2.imwrite(
        os.path.join(save_dir, filename),
        cv2.convertScaleAbs(tensor.numpy()[:, :,::-1] * 0.5 + 0.5, alpha=(255.0))
    )



# ### Read image

# In[9]:
```

```
# image_path = tf.keras.utils.get_file('YellowLabradorLooking_new.jpg',
'https://storage.googleapis.com/download.tensorflow.org/example_images/YellowLabradorL
ooking_new.jpg')
image_path = 'C:/Users/daryn/Downloads/IMG_4953.JPG'
image_raw = tf.io.read_file(image_path)
image = tf.image.decode_image(image_raw, channels=3)
image = preprocess(image)

predict_and_show(image)


# In[10]:


# cv2.imwrite('./results/original_Prada.jpg', cv2.convertScaleAbs(image[0].numpy()[:,
:,::-1] * 0.5 + 0.5, alpha=(255.0)))
save_image('original_Prada.jpg', image[0])


# ## Blur

# In[11]:


loss_object = tf.keras.losses.CategoricalCrossentropy()


# In[12]:


def create_adversarial_pattern(input_image, input_label):
  with tf.GradientTape() as tape:
    tape.watch(input_image)
    prediction = pretrained_model(input_image)
    loss = loss_object(input_label, prediction)

  # Get the gradients of the loss w.r.t to the input image.
  gradient = tape.gradient(loss, input_image)
  # Get the sign of the gradients to create the perturbation
#   signed_grad = tf.sign(gradient)

  return gradient / tf.math.reduce_max(gradient)


# In[13]:
```

```
label_idx = pretrained_model.predict(image)
gradient = create_adversarial_pattern(image, label_idx)
plt.imshow(gradient[0]*0.5 + 0.5)


# In[14]:


save_image('grad_Prada.jpg', gradient[0])
# save_image('grad_blur_Prada.jpg', perturbations[0])


# In[15]:


def create_circular_mask(h, w, center=None, radius=None, channels=3):

    if center is None: # use the middle of the image
        center = (int(w/2), int(h/2))
    if radius is None: # use the smallest distance between the center and image walls
        radius = min(center[0], center[1], w-center[0], h-center[1])

    Y, X = np.ogrid[:h, :w]
    dist_from_center = np.sqrt((X - center[0])**2 + (Y-center[1])**2)

    mask = dist_from_center <= radius
    mask_d = np.zeros(shape = (h, w, channels), dtype=bool)
    for channel in range(channels):
        mask_d[:,:,channel] = mask
    return mask_d


# In[16]:


def find_best_mask(gradient, radius=40, n=100):
    radius=40
    n = 200
    grad_max, mask_blur = 0, None
    for i in range(n):
        center = random.randint(0, 224), random.randint(0, 224)
        mask = create_circular_mask(224, 224, center, radius)
        grad_cur = np.sum(np.abs(gradient[mask]))
        if grad_cur > grad_max:
            grad_max = grad_cur
            mask_blur = mask
    return mask_blur
```

```
# In[17]:


def apply_blur(img, mask, factor=3.):
    (h, w) = img.shape[:2]
    kW = int(w / factor)
    kH = int(h / factor)
    # ensure the width of the kernel is odd
    if kW % 2 == 0:
        kW -= 1
    # ensure the height of the kernel is odd
    if kH % 2 == 0:
        kH -= 1
    blurred_img = cv2.GaussianBlur(img, (kW, kH), 0)
    out = np.where(mask, blurred_img, img)
    return out


# In[18]:


mask_blur = find_best_mask(gradient[0])
image_blur = tf.convert_to_tensor([apply_blur(image[0].numpy(), mask_blur)])

predict_and_show(image_blur)


# In[19]:


save_image('blur_Prada.jpg', image_blur[0])


# In[21]:


# Get the input label of the image.
# https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a
# labrador_retriever_index = 208
# german_shepherd_index = 235
target_index = np.argmax(pretrained_model.predict(image), axis=1)[0]
image_probs = pretrained_model.predict(image_blur)

label = tf.one_hot(target_index, image_probs.shape[-1])
label = tf.reshape(label, (1, image_probs.shape[-1]))
```

```
perturbations = create_adversarial_pattern(image_blur, label)
perturbations = perturbations.numpy()
perturbations[0][~mask_blur] = 0
perturbations = tf.convert_to_tensor(perturbations)
plt.imshow(perturbations[0] * 0.5 + 0.5);  # To change [-1, 1] to [0,1]


# In[22]:


predict_and_show(perturbations)


# In[23]:


save_image('grad_blur_Prada.jpg', perturbations[0])


# In[25]:


gradient = gradient.numpy()
gradient[0][~mask_blur] = 0
gradient = tf.convert_to_tensor(gradient)
save_image('grad_area_Prada.jpg', gradient[0])


# In[33]:


def display_images(image, description):
    _, label, confidence = get_imagenet_label(pretrained_model.predict(image))
    plt.figure()
    plt.imshow(image[0]*0.5+0.5)
    plt.title('{} \n {} : {:.2f}% Confidence'.format(description,
                                                     label, confidence*100))

    print()
    plt.show()


# In[34]:


perturbations.shape
```

```
# ##  Apply FGSM to blurred image

# In[35]:


def apply_fgsm(img, mask, correct_idx, eps=0.05, max_iter=10, display_correct=True):

    image_probs = pretrained_model.predict(img)
    label = tf.one_hot(correct_idx, image_probs.shape[-1])
    label = tf.reshape(label, (1, image_probs.shape[-1]))

    for i in range(max_iter):
        perturbations = create_adversarial_pattern(img, label)
        adv_x = img.numpy()
        adv_x[0][mask] -= eps*perturbations[0][mask]
        adv_x = tf.clip_by_value(adv_x, -1, 1)
        img = tf.convert_to_tensor(adv_x)
        probs = pretrained_model.predict(img)
        label_idx = np.argmax(probs, axis=1)[0]
        label_prob = np.max(probs, axis=1)[0]
        if label_idx == correct_idx:
            if display_correct:
                display_images(img, f'Epsilon={eps}, step={i+1}')
            return img, label_idx, label_prob, i+1
    return img, label_idx, label_prob, i+1


# In[37]:


res = apply_fgsm(image_blur, mask_blur, target_index)
fgsm_image = res[0]


# In[38]:


save_image('adapted_Prada.jpg', fgsm_image[0])


# ## ImageNet

# In[22]:
```

```
from tqdm.notebook import tqdm


# In[23]:


IMAGENET_PATH = './ILSVRC2012_img_test_v10102019/test'


# In[24]:


len(os.listdir(IMAGENET_PATH))


# In[25]:


from IPython.display import display, Image
display(Image(filename=(os.path.join(IMAGENET_PATH,
'ILSVRC2012_test_00000001.JPEG'))))


# In[26]:


image_blur_masks = dict()
res_all = []
items = sorted(os.listdir(IMAGENET_PATH))[:50000]
pbar = tqdm(total=len(items))

def worker(image_name):
    global image_blur_mask, res_all
    try:
        res = {}
        image_path = os.path.join(IMAGENET_PATH, image_name)
        image_raw = tf.io.read_file(image_path)
        image = tf.image.decode_image(image_raw, channels=3)
        image = preprocess(image)

    #     predict_and_show(image)

        probs = pretrained_model.predict(image)
        label_idx = np.argmax(probs, axis=1)[0]
        label_prob = np.max(probs, axis=1)[0]

        gradient = create_adversarial_pattern(image, probs)
```

```
    #     plt.imshow(gradient[0]*0.5 + 0.5)

        mask_blur = find_best_mask(gradient[0])
        image_blur = tf.convert_to_tensor([apply_blur(image[0].numpy(), mask_blur)])
        probs_blur = pretrained_model.predict(image_blur)
        label_idx_blur = np.argmax(probs_blur, axis=1)[0]
        label_prob_blur = np.max(probs_blur, axis=1)[0]

        image_blur_masks[image_name] = mask_blur

        res = {
            'image_name': image_name,

            'initial_label_idx': label_idx,
            'initial_label_prob':label_prob,

            'blurred_label_idx': label_idx_blur,
            'blurred_label_prob': label_prob_blur,
        }

        if label_idx == label_idx_blur:
            res_all.append(res)
            return

    #     predict_and_show(image_blur)

        res_fgsm = apply_fgsm(image_blur, mask_blur, label_idx, display_correct=False)

        res.update({
            'fgsm_label_idx': res_fgsm[1],
            'fgsm_prob': res_fgsm[2],
            'fgsm_iter': res_fgsm[3]
#             'correct': label_idx == res_fgsm[1]
        })

        res_all.append(res)
    except Exception as e:
        print(image_name, e)
    finally:
        pbar.update(1)



# In[ ]:
```

```
# from tqdm.contrib.concurrent import process_map
# items = os.listdir(IMAGENET_PATH)
# process_map(worker, items, max_workers=10)

from multiprocessing.pool import ThreadPool as Pool
pool_size = 12  # your "parallelness"

pool = Pool(pool_size)
for item in items:
    pool.apply_async(worker, (item,))


pool.close()
pool.join()


# In[ ]:


res_all = pd.DataFrame(res_all)
res_all['is_correct'] = None
res_all.loc[res_all['initial_label_idx'] != res_all['blurred_label_idx'],
'is_correct'] = res_all['initial_label_idx']==res_all['fgsm_label_idx']
res_all


# In[ ]:


RESULTS_DIR = './results'


# In[ ]:


filename_identificator = '1-50000'


# In[ ]:


res_all.to_csv(os.path.join(RESULTS_DIR, f'results_{filename_identificator}.csv'),
index=False)


# In[ ]:
```

```
import json
with open(os.path.join(RESULTS_DIR, f'blur_masks_{filename_identificator}.json'), 'w')
as f:
    json.dump({k:v.tolist() for k, v in image_blur_masks.items()}, f)
```

APPLICATION B. Results estimation (Python code)

```
#!/usr/bin/env python
# coding: utf-8

# In[52]:


import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# import tensorflow as tf
import matplotlib.pyplot as plt


# In[29]:


RESULTS_DIR = './results'


# In[30]:


mapping = pd.read_csv('./mapping/map_clsloc.txt', header=None, sep=' ', names=['n',
'idx', 'label'])
# mapping = dict(zip(mappin))
mapping = dict(zip(mapping['idx'], mapping['label']))
mapping


# In[31]:


df1 = pd.read_csv(os.path.join(RESULTS_DIR, 'results_1-50000.csv'))
df2 = pd.read_csv(os.path.join(RESULTS_DIR,'results_50000-100000.csv'))

df = pd.concat([df1, df2])


# In[32]:


df['is_changed_after_blur'] = df["initial_label_idx"] != df["blurred_label_idx"]
```

```
# In[33]:


df


# In[34]:


df['initial_label'] = df['initial_label_idx'].map(mapping)
df['blurred_label'] = df['blurred_label_idx'].map(mapping)
df['fgsm_label'] = df['fgsm_label_idx'].map(mapping)


# In[40]:


sum(df['is_correct'].astype(bool))


# In[85]:


# !pip install seaborn


# In[115]:


import seaborn as sns
sns.set(palette='Spectral_r')

def func(pct, n):
    absolute = int(round(pct/100.*n))
    return "{:.1f}%\n({:d} images)".format(pct, absolute)

df['is_correct'].map({np.nan: "class is not changed after anonymization", True:
'gradient method corrected the class',  False: 'gradient method did not correct the
class'}).value_counts().plot.pie(
    autopct= lambda pct: func(pct, df.shape[0]),
    figsize=(8, 8),
    explode = (0.1, 0, 0),
    fontsize=15
)
plt.ylabel('')
plt.savefig('./results/pie_blur.jpg',  bbox_inches = "tight")
```

```
# In[49]:


df_label_rest = pd.DataFrame()
df_label_rest['correct'] =
df[df['is_correct'].notna()].groupby('initial_label').sum(numeric_only=False)['is_corr
ect']
df_label_rest['all'] =
df[df['is_correct'].notna()].groupby('initial_label').count()['is_correct']
df_label_rest['correct_pct'] = df_label_rest['correct'] / df_label_rest['all']
df_label_rest.sort_values(['correct_pct'])


# In[67]:


ax = df_label_rest['correct_pct'].hist(figsize=(8, 5))
vals = ax.get_xticks()

ax.set_xticklabels(['{:,.2%}'.format(x) for x in vals])
ax.set_ylabel('Frequancy (classes)')
ax.set_xlabel('Class accuracy')
plt.savefig('./results/class_acc_hist.jpg')


# In[47]:


correct_label_count


# In[48]:


changed_label_count


# In[ ]:


df.groupby('initial_label').sum('is_correct')


# In[9]:
```

```
df.shape
print(f'Test set: {df.shape[0]}')

n = sum(df["is_changed_after_blur"])
p_n = df["is_correct"].sum()
q_n = (df["is_correct"] == False).sum()

print(f'Class changed after blurring part of image: {n}')
print(f'Class is correct after FGSM applied: {p_n}/{n} ({p_n/n:.2%})')
print(f'Class is wrong after FGSM applied: {q_n}/{n} ({q_n/n:.2%})')
```