Abstract

Detkov Nikita Sergeevich, “Influence of the label smoothing of pseudo-labeled data on training Convolutional Neural Networks”, the paper contains 33 pages, 10 figures,7 tables, 19 sources.

Key words: skin cancer, melanoma, convolutional neural networks, computer vision, machine learning, deep learning, label smoothing, pseudo-labeling, under-sampling.

The present paper aims to build a machine learning model, which identifies melanoma by image of mole or skin damage, and to research the applicability of label smoothing methods to pseudo-labeled data to improve this model.

We have carried out a lot of experiments and presented several hypotheses, which demonstrated us the fact that even a task of this complexity with certain negative factors can be solved with better results.

Table of contents

Key terminology

Artificial perceptron – a model that describes the brain's perception, response, and processing of information in terms of mathematics and computer computations.

Convolutional Neural Network – a function that takes a tensor as an input and performs conversion using convolutions, activation functions, and other mathematical functions. It can also be defined as a multi-layer perceptron.

Artificial neuron is one of the nodes of a neural network, which is represented as a mathematical function similar to a neuron – an electrically excitable cell of the nervous system.

Loss function is a function that describes the disagreement between the actual and predicted values.

Backpropagation algorithm is an iterative method for calculating the gradient that is used when updating the weights of a neural network in order to minimize its loss function. It is applicable only in the case of differentiability of activation functions of neurons and other components of the neural network.

Neural network training is the process of training a neural network, which is possible due to the backpropagation algorithm.

Neural network fine-tuning is a process of additional training of a neural network, in which usually not all layers are trained, but only a part.

Neural network inference – the process of getting the result of the work of the neural network.

Binary classification – the task of classifying an object into one of two classes.

Under-Sampling – the process of changing the distribution of objects of different classes by reducing the number of objects of the prevailing class.

Label Smoothing – the process of "smoothing" discrete values of class labels into a continuous values.

Stratification – the process of selecting a set from the population, in which the distribution of a feature within the selected set is kept as close as possible to the population.

Pseudo-labeling – the process of obtaining is a class labels from a neural network by inference on data without labels or obtained labels themselves.

Augmentation is a method of data transformations which allows to increase the generalization ability of the model that will be trained on it.

Test Time Augmentation (TTA) – augmentation applied to test data to increase prediction accuracy.

Models ensemble – a set of models that can perform a task and combine their predictions.

Optimizer is a gradient descent algorithm. For example, stochastic gradient descent.

Learning rate is a parameter of the optimizer that determines the step size in the direction of the gradient vector.

Learning rate scheduler – a "scheduler" that determines the change in the learning rate during training.

Train set – a set of labeled data.

Validation set – a set of labeled data, used for validating the quality of training. It is usually a subset of the train set.

Test set – a set of unlabeled data.

Hold-out set – a set of labeled data, used for building ensembles and other ways to combine the results of models.

Folds – disjoint sets that the train set can be divided into.

Out Of Fold (OOF) Predictions – predictions for a test set given by models after training on all but one of the folds.

Threshold – a threshold or decision-making boundary.

A target, target variable, or class label are synonyms for the value to predict.

Introduction

Skin cancer is the most prevalent type of cancer. Melanoma, specifically, is responsible for 75% of skin cancer deaths, despite being the least common skin cancer. The American Cancer Society estimates over 100,000 new melanoma cases will be diagnosed in 2020. It's also expected that almost 7,000 people will die from the disease. As with other cancers, early and accurate detection, potentially aided by data science, can make treatment more effective.

Currently, dermatologists evaluate every one of a patient's moles to identify outlier lesions or “ugly ducklings” that are most likely to be melanoma. Existing AI approaches have not adequately considered this clinical frame of reference. Dermatologists could enhance their diagnostic accuracy if detection algorithms consider “contextual” images within the same patient to determine which images represent a melanoma. If successful, classifiers would be more accurate and could better support dermatological clinic work.

Melanoma is a deadly disease, but if caught early, most melanomas can be cured with minor surgery. Image analysis tools that automate the diagnosis of melanoma will improve dermatologists' diagnostic accuracy. Better detection of melanoma has the opportunity to positively impact millions of people. [1]

The first task of this project is to detect melanoma in images of moles and skin lesions. In particular, images of the skin of the same patient will be used to determine which of them may represent melanoma.

The aim of the second part of the paper is to study the possibility of improving this solution using the insufficiently studied label smoothing technique of pseudo-labeled data, which is the subject of this work. This is a technique that allows you to calibrate the confidence of a mathematical model's prediction by adding more uncertainty to it and thus improve its generalization ability on real-world data with imperfect labels and data for training and inference. One of these relevant tasks in the real world is to determine melanoma from a photo of a mole or skin lesion.

This method can be applied to this type of problem, because it is extremely difficult to achieve a unified method of shooting: there will always be different cameras, shooting conditions, and so on. Moreover, melanoma has the ability to develop, and its visual signs can be expressed differently at different stages of the disease, and in this data, as in many subsequent ones that will be collected, the sign "there is a melanoma or not" will only take the value "Yes" and " No". Thus, it is important to try to apply this method and find out the results with different approaches.

The main part

1. Dataset description

The ISIC archive contains the largest publicly available collection of quality-controlled dermoscopic images of skin lesions.

The images are provided in DICOM format. It can be accessed using commonly available libraries such as *pydicom*, and contains both image and metadata. This is a widely used format for medical imaging data.

Images are also provided in JPEG and TFRecord format (in the *jpeg* and *tfrecords* directories, respectively). Images in TFRecord format were resized to a uniform 1024x1024. Metadata is also provided outside of DICOM format, in CSV files. There are 44,108 images in total, including 33,126 in the train set and 10,982 in the test set.

1.1 Overview of available tabular data

- *train.csv* – train set;
- *test.csv* – test set.

Columns description:

- *image_name* – unique identifier, points to filename of related DICOM image;
- *patient_id* – unique patient identifier;
- *sex* – пол the sex of the patient (when unknown, will be blank);
- *age_approx* – approximate age of the patient at the time of imaging (about 20, 25, etc.);
- *anatom_site_general_challenge* – location of imaged site;
- *diagnosis* – detailed diagnosis information (only in the train set);
- *benign_malignant* – indicator of malignancy of imaged lesion;
- *target* – binarized version of the target variable (0 and 1).

1.2 Image overview

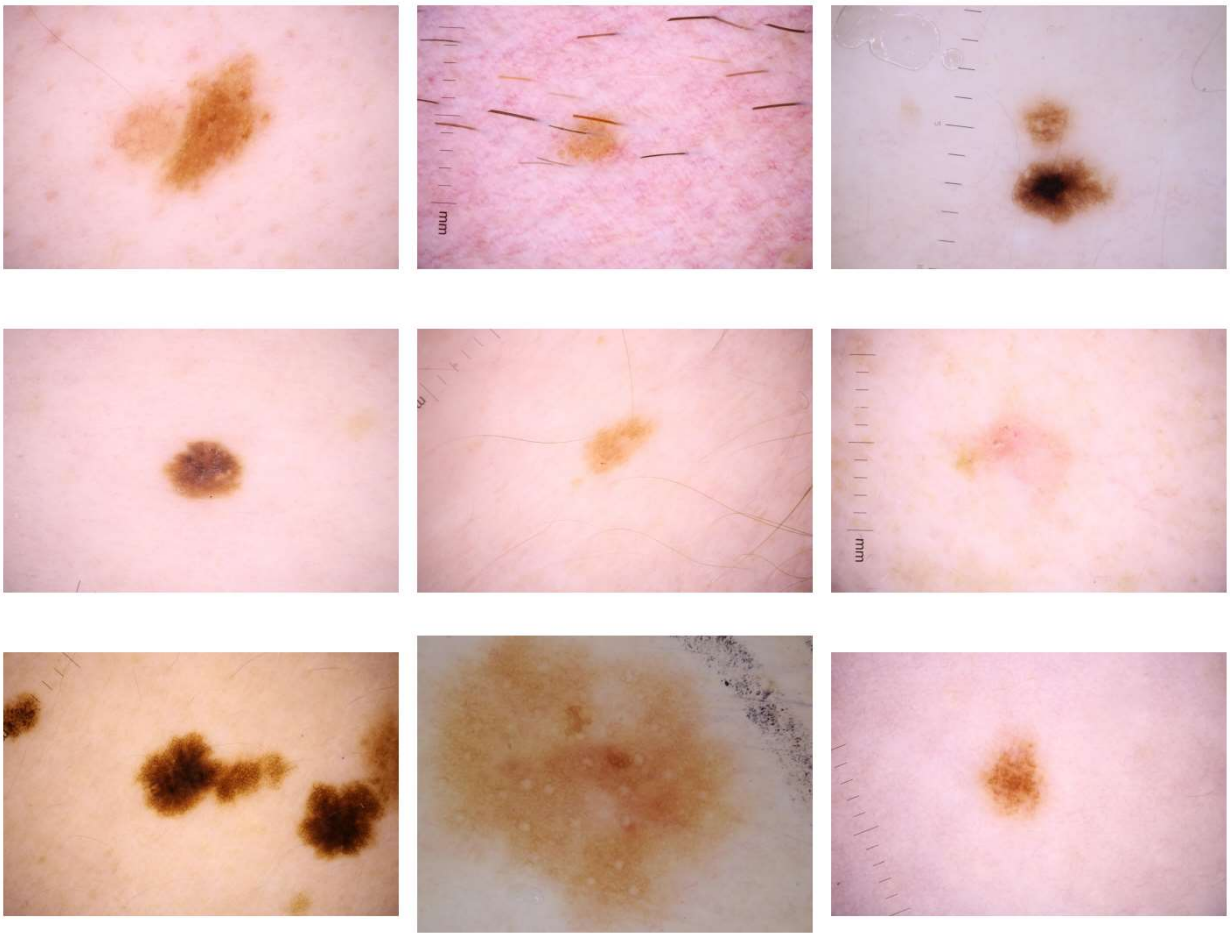

Figure 1.2.1 – Random images

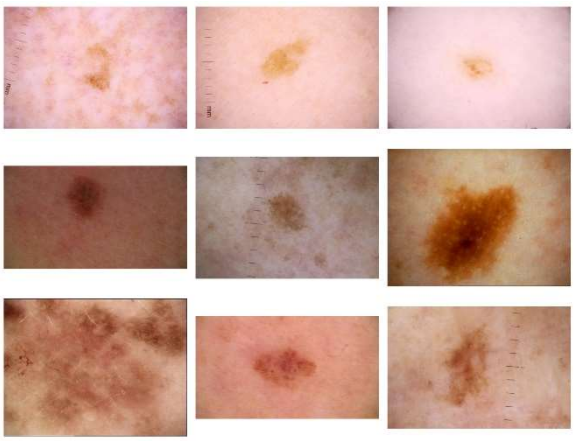

Figure 1.2.2 – Images with a benign label

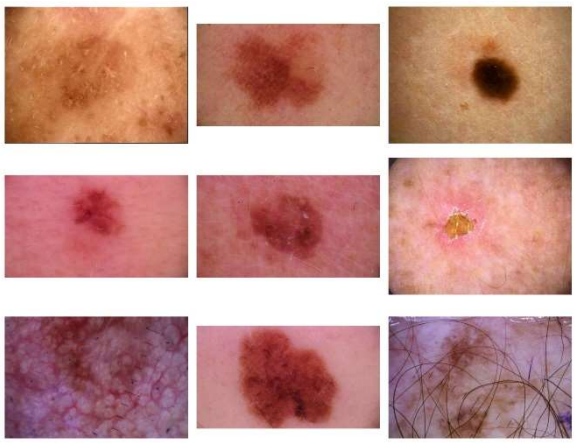

Figure 1.2.3 – Images with a malignancy label

Note that images have different resolutions and shapes.

2. Metric

In a supervised machine learning task, metric evaluates the quality of the algorithm by comparing predictions and true labels.

In this problem, the metric is the ROC curve (receiver operating characteristic), namely, the AUC (Area Under ROC Curve) – the area under the ROC curve [2]. The ROC curve is plotted as a curve line with the TPR (True Positive Rate) and FPR (False Positive Rate) [3] axes with the calculation of these statistical values, based on the consequent use of different thresholds.

The metric has the following visual representation (the area under the blue line):

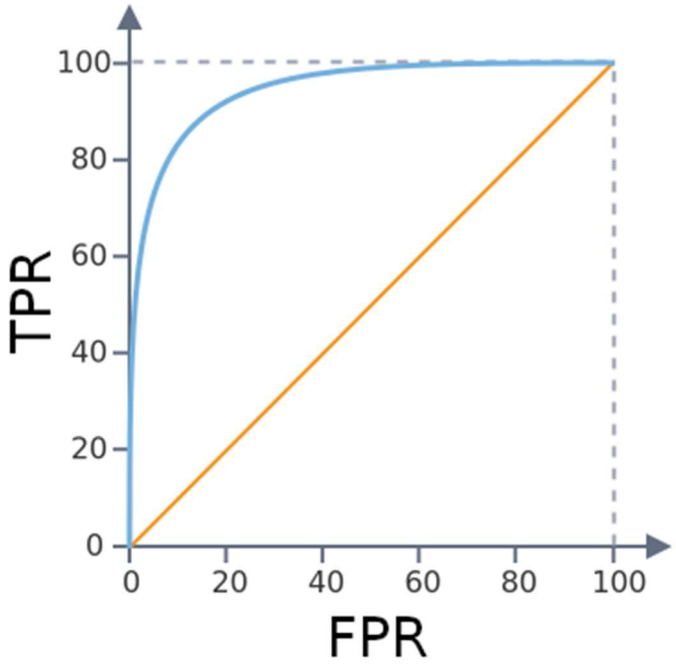

Figure 2.1 – the ROC curve plot

3. Creating a model for determining melanoma

3.1 Data overview and solution strategy

The executable code in this header is simplified to avoid overloading the document. You can view the full-fledged code in the project repository, which has a link to it at the end of the Conclusion chapter.

First, we read the data in CSV files:

```
train = pd.read_csv('../input/train.csv')
test  = pd.read_csv('../input/test.csv')
```

From the very beginning, we look at the target feature and by applying the `value_counts()` function to it, we see that there are only 584 images with a malignant tumor, while 32,543 images without it:

```
train['target'].value_counts()
```

We also learn that in the train set there are 1077 men and 977 women, and in the test set 364 men and 326 women:

```
train.groupby('patient_id')['sex'].first().value_counts()
test.groupby('patient_id')['sex'].first().value_counts()
```

It would be interesting to know, since we were given the *age_approx*, how many years the patients contained in the data were observed. The feature is represented by discrete values that are multiples of 5, so you can only roughly know how old a person was who had this feature equal to, say, 30.

To begin with, let's look at the age at which patients who have had pictures of moles taken, both benign and malignant:

```
import seaborn as sns

sns.countplot(train['age_approx'],
              hue=train['benign_malignant'],
              palette=['g', 'r'])
```

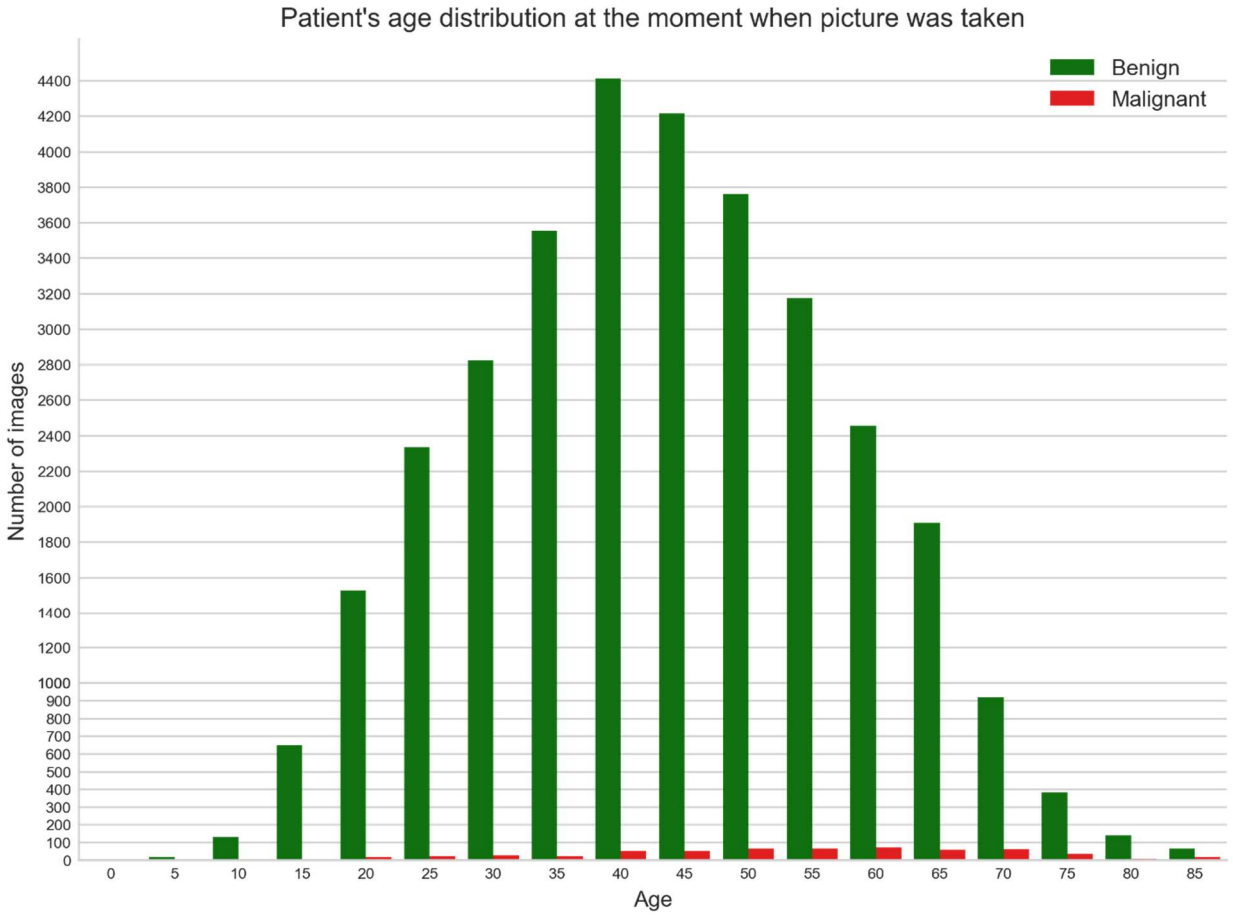

Figure 3.1.1 – Distribution of the patient's age at the time of the mole image

According to the plot, you can see that malignant moles are more common in adults, which seems logical from the point of view of biology.

We were also wondering how long these patients could have been observed, so I looked at the number of different *age_approx* patients had:

```
train.groupby(['patient_id'])['age_approx'].nunique()
                                           .value_counts()
                                           .plot(kind='bar')
```

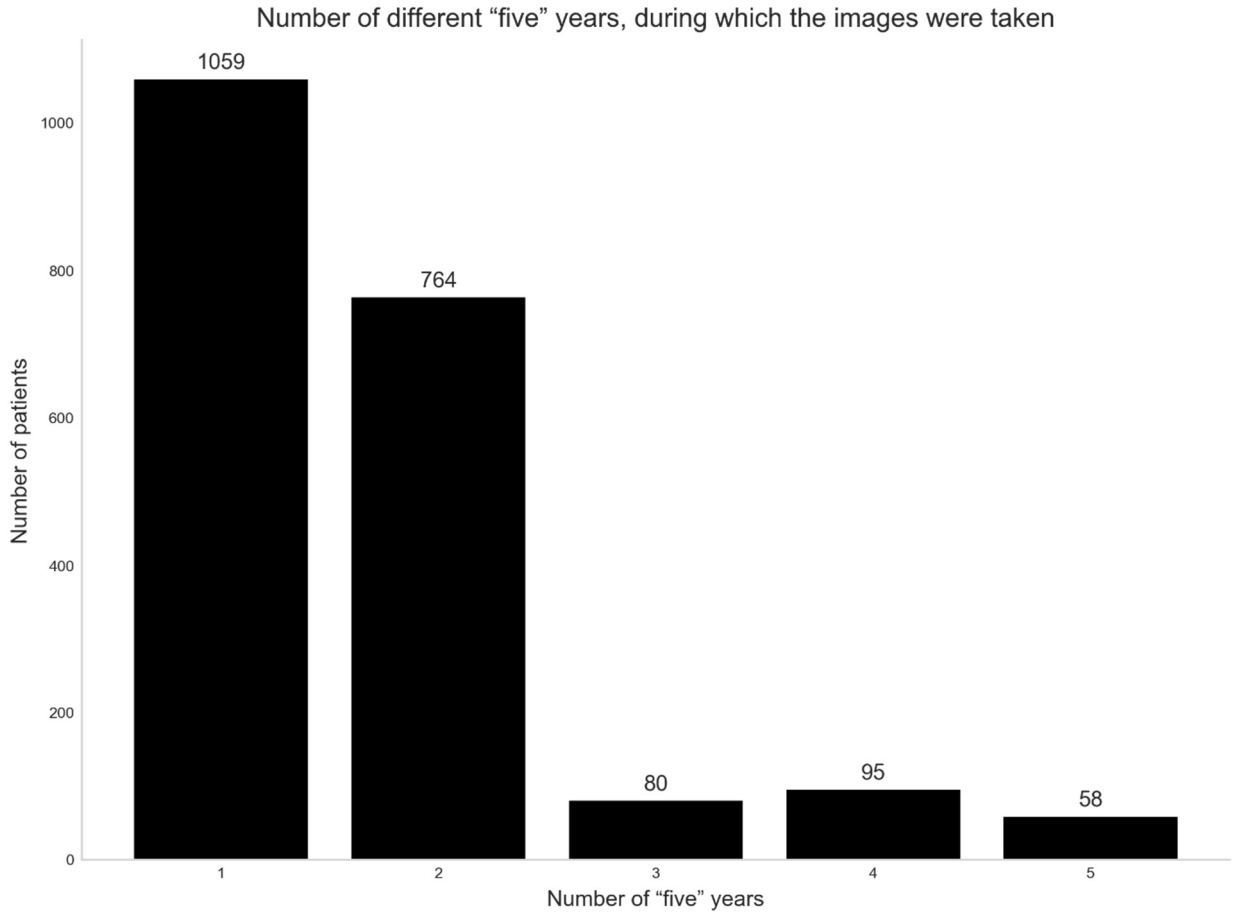

Figure 3.1.2 – The number of “five” years during which images of moles were taken

It is easy to see that most patients were observed for a maximum of 5 years, while the longest-observed patient could have been observed for more than 25 years. Meanwhile, only 11% of patients were observed for at least 10 years.

By the way, it is quite interesting, how many images are there for each of the patients? It is not difficult to find out the exact number for each one:

```
train.groupby(['patient_id']).size()
```

It turns out that the smallest number is two images, and only one patient has them, while the others have three or more. Four patients have the maximum number – 115.

Find out the average:

```
train.groupby(['patient_id']).size().mean()
```

The average number of images is 16, but we would like to get a closer look at the statistics for these values, and to get them in a visual representation, we can use the box plot [4]:

```
import matplotlib.pyplot as plt

plt.boxplot(train.groupby(['patient_id']).size(),
            showfliers=False, vert=False)
```

It was decided to remove a few outliers, as they will be superfluous.

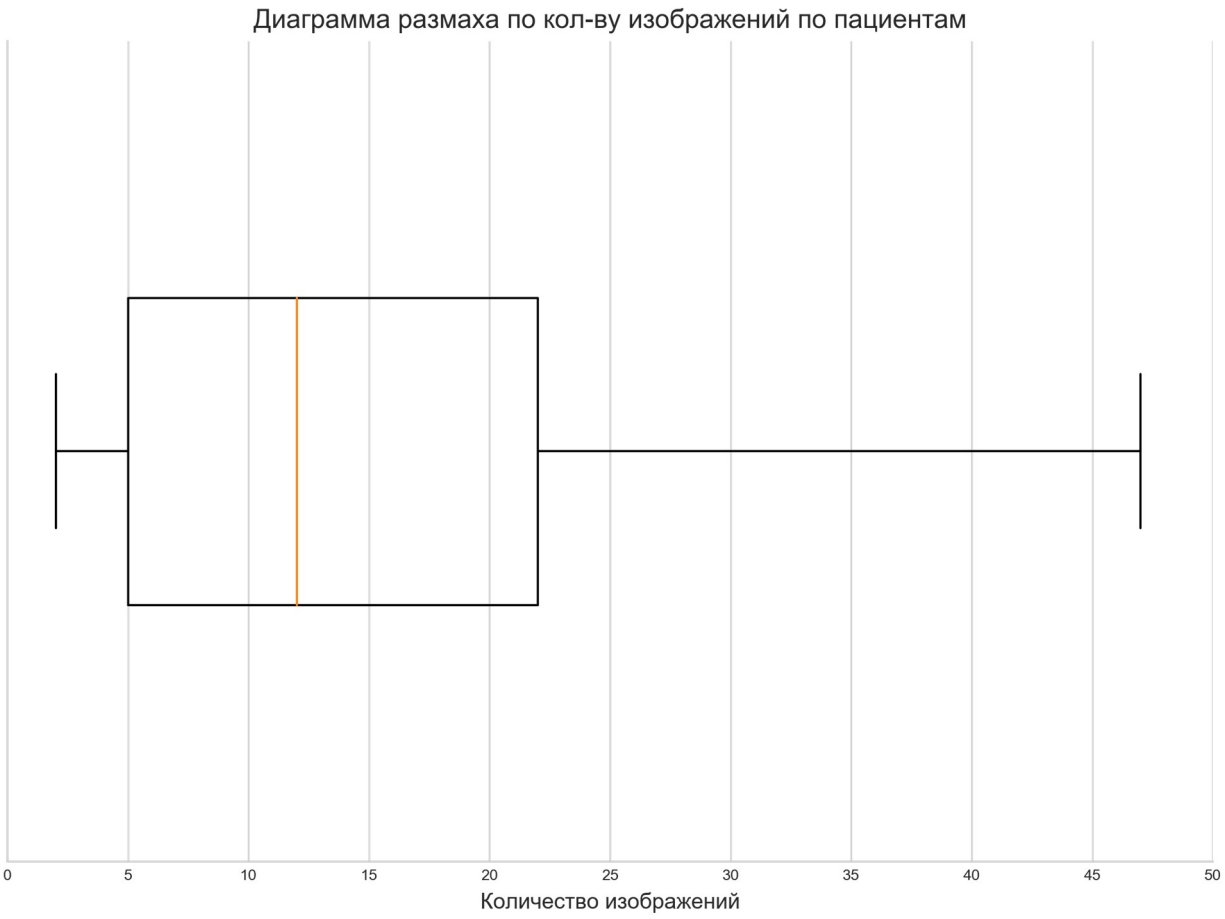

Figure 3.1.3 – Box plot of patients' number of images

The median number of images per patient is 12, and the 75th percentile is 22. It can be concluded that people either photographed moles once or maintained supervision of their development over the years.

To consolidate the understanding, here is another visualization:

```
import matplotlib.pyplot as plt

image_freq_per_patient = train.groupby(['patient_id']).size()
plt.hist(image_freq_per_patient,
         image_freq_per_patient.nunique(), color='black')
```

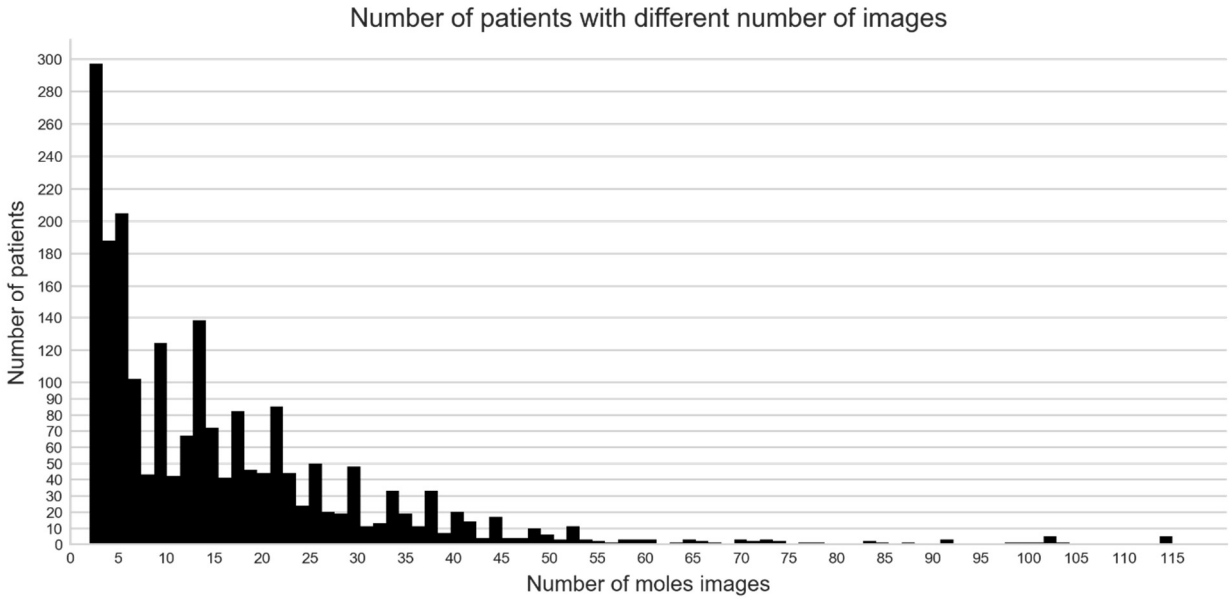

Figure 3.1.4 – Number of patients with different number of mole images

Now you can look at the *anatom_site_general_challenge* feature, which tells us about the location of the mole on the body:

```
import matplotlib.pyplot as plt
import numpy as np
site_vs_diagnosis = (train.groupby(
    ['anatom_site_general_challenge',
    'benign_malignant']
    ).count()['patient_id'])
labels = (train['anatom_site_general_challenge']
          .value_counts(normalize=True)
          .sort_values().index)
benign_data = site_vs_diagnosis[0:12:2]
maglignant_data = site_vs_diagnosis[1:12:2]
x = np.arange(len(labels))

width = 0.35
fig, ax = plt.subplots(figsize=(8,4))
ax.bar(x-width/2, benign_data, width, color='g')
ax.bar(x+width/2, maglignant_data, width, color='r')
```

Figure 3.1.5 – Number of patients by tumor type and location

Unfortunately, nothing can be figured out from the presented plot except the knowledge of numerical values: there are no discoveries in this feature.

The information received has definitely given us an understanding of what data we are dealing with, and so we can proceed to solve the problem. And what is the problem? Taking into account all the above, we see that we are faced with the imbalanced binary classification problem with images with metadata about patients.

Problems related to images are usually solved by using convolutional neural networks, and we will go the same way. Initially, it was decided to use only images and data about their belonging to patients, although, obviously, other tabular data can also help. Therefore, we are talking about building a convolutional neural network that accepts only the image as input. Now we will describe the steps we took to get a solution and start conducting research, which is dedicated to, but is not limited to, the topic of this work.

3.2 Data processing

The very first steps were downloading and processing the dataset. The processing consists in an obvious decrease in resolution because the original images had a resolution even up to 4000x6000 pixels. The question is, can non-square images be converted to square format in terms of changing the shape of the skin defect in the image? Is it possible to assume from the modified

image that it has a different class than before the conversion? The answer can be given by finding out what characteristics of a mole can indicate malignancy, and they are, as applicable to this problem, approximately the following: asymmetry, irregularity, raggedness, notchiness or blurring of borders, as well as the presence of several colors [5]. Thus, we can conclude that we cannot apply elastic transformations. Further, this knowledge will help us in making correct augmentations.

Hence, you can either simply make it square with a size reducing or cut out 2 intersecting squares from images that are not square and change their size. The second method is difficult from the point of view of implementation and may be incorrect from the point of view of assigning class labels, since it may turn out that there will be nothing at all on one of the squares. As a result, we made 2 datasets consisting of images with resolution 512x512 and 256x256.

3.3 Splitting data for training

Training requires data for validation that can be used to evaluate the quality of the model, so the standard technique is to split the data into N folds. After that, training on different sets of folds and cross-validation is performed in turn [6].

In this case, the data is imbalanced relative to the class labels: malignancy was noted on 1.76% of all images in the train set. Therefore, it is worth investigating the stratified splitting into folds, that is, in the received folds, the distribution of the target variable will be as close as possible to the distribution of the entire train set (population).

On the other hand, every patient has at least 2 images in the train set, and, out of 2056 patients, 427 have images with and without melanoma. Accordingly, it makes sense to do the partitioning with non-overlapping groups of patients. If translated into a clear language, this sets the condition that each patient is fully contained in only one fold. Interestingly, with this type of splitting, the target distribution is kept close to the train set.

Also, in order to be able to test hypotheses and ideas, we need to select a hold-out set that the neural network would not “see” during training, and for which the class labels would be known. Accordingly, you need to take it from the train set. We decided to take 10% of the entire train set for a hold-out set, and the remaining 90% for training and for splitting by folds. Of course, this hold-out set should have the same distribution as the train set.

Thus, both types of splitting were created with the putting aside 10% for the hold-out set (according to the corresponding splitting rule). The target's fold distributions in both splits and the

situation in the train set are shown below. Target distribution in train set and both types of splits are shown below.

Table 3.3.1 – Target distribution in train set

	All the data
% of class 0	98.237034
% of class 1	1.762966

Table 3.3.2 – Target distribution within folds with stratification

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Hold-out
% of class 0	98.2391	98.2391	98.2388	98.2388	98.2388	98.2196
% of class 0	1.7608	1.7608	1.7611	1.7611	1.7611	1.7803

Table 3.3.3 – Target distribution within groups-splitted folds

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Hold-out
% of class 0	98.4236	98.4739	98.0543	97.9033	98.3227	98.2498
% of class 0	1.5763	1.5260	1.9456	2.0966	1.6772	1.7501

As we can see, when divided into groups, the distribution of an underrepresented class can vary up to 20% of the distribution of the train set.

3.4 Augmentation

In order to improve the generalization ability of the network, it is necessary to train it on various variations of the predicted class. For computer vision problems, these different variations can be images of the research object from different angles, states, etc.

As noted in the section “Data processing”, only affine transformations are suitable for this problem, i.e. all types of rotations, flips, shifts, zooming, etc. There are also many augmentation options related to changing the color, brightness, contrast, and other parameters of images. Applicable to this task, we can say that it is undesirable to change the original colors, since one of the main sign of melanoma is its color. At the same time, the images were taken in different conditions, including illumination, different camera parameters, hand shaking, and many other

factors. Based on this, we decided that in this case, you can change the brightness, contrast, and illumination.

The final list of augmentations used for training is as follows:

```
import albumentations as A

A.Compose([
    A.OneOf([
        A.ShiftScaleRotate(rotate_limit=90, p=1.0),
        A.HorizontalFlip(p=1.0),
        A.VerticalFlip(p=1.0),
        A.RandomRotate90(p=1.0),
    ], p=0.5),

    A.OneOf([
        A.RandomBrightness(p=1.0),
        A.RandomBrightnessContrast(p=1.0),
        A.RandomGamma(p=1.0),
    ], p=0.5)
 ])
```

There is also a technique called Test Time Augmentation, or just TTA. The name makes it clear that it is used during inference to improve the quality of prediction. The point is simple: during the prediction, the input of the neural network is not a picture, but its augmented states, for each of which a prediction is made. Further, these predictions are averaged or mixed in some other way.

We selected 0, 90, 180, and 270 degree rotations and horizontal flip, and later averaging predictions from them.

3.5 Training experiments

To conduct the experiments and increase the reproducibility of the results, the following was done:

- Fixed seeds (random states) for all functions that have a random number generation process, including for GPU calculations;
- Each training process is accompanied by a YAML configuration file that contains all the experiment parameters;
- In each of the experiments, the weights of trained networks and files with the prediction of the test set were saved, and the metric value on the hold-out set was calculated.

To work on the project, it was decided to use convolutional neural networks of the EfficientNet architecture [7]. After a few experiments and comparisons, it became clear that the

type of EfficientNet-B3 pre-trained on the ImageNet dataset [8] using the so-called noisy student [9], is the most optimal. Binary cross-entropy was chosen as the loss function [10], and AdamW was chosen as the optimizer [11, 12]. ReduceLROnPlateau was chosen as the scheduler, but StepLR was present in several experiments – both are present in the PyTorch framework [13]. The full list of hyperparameters and algorithms can be found in the repository. 5 OOF models were trained, and their predictions were averaged on the test set. At the same time, these predictions themselves were obtained using the TTA. Predictions for the hold-out set were also obtained using TTA by each model separately, and then averaged.

In total, 7 large training experiments were fully conducted, a summary of which is provided in the table below.

Table 3.5.1 – Results and conditions of training experiments

Number of experiment	Using splitting strategy	Images resolution	Batch size	Number of epochs	Initial learning rate	Metric on 30% of test set[1]	Metric on hold-out set
2	Страт.	512x512	10	15	0.0003	0.901	0.8263
3	Страт.	256x256	32	20	0.001	0.868	0.7252
4	Группы	256x256	32	20	0.001	0.884	0.8620
5	Группы	512x512	10	25	0.001	0.891	0.8892
6	Группы	512x512	10	25	0.0003	0.914	0.9040
7	Страт.	512x512	10	25	0.0003	0.911	0.8529

A few comments about the decisions made in terms of changing hyperparameters:

- Changing the resolution of images from 512x512 to 256x256 and back is associated with the fact that the second resolution of the learning process lasts about 3-4 times faster, but, at the same time, get worse results. In other words, if the experiment with 512x512 lasted from 12 hours, then with 256x256 it could be completed in 4;
- The transition from stratified splitting to groups and back is due to the fact that the idea of grouping appeared only during the fourth experiment, while the seventh experiment was only intended to test the result;

[1]Predictions for the test set were uploaded to Kaggle, during the competition a metric for 30% of predictions with only 3 digits after the decimal point is shown there.

- The change in the number of epochs is continuously associated with an early stopping in the absence of improvements, ReduceLROnPlateau with its change in the learning rate and with its factor by which the learning rate is multiplied.

There were also a lot of small, one-fold experiments, based on which other hyperparameters were selected. The very first big experiment was skipped because the solution could not be restored after refactoring the code.

From all the above, it can be established that the best strategy for learning in these conditions is to train on high-resolution images, starting with the learning rate of the order of ten thousandth, the scheduler ReduceLROnPlateau, and the AdamW optimizer.

3.6 Results analysis

Table 3.5.1 shows the metric values. We can't say for sure which model is objectively better from the point of view of metrics. It is also impossible to choose which model should be used in the future for conducting the main research. To do this, you need to conduct a more in-depth study of the test set predictions, paying attention to the training environment and the general population.

To begin with, you can look at the target distribution and compare it with the train set, making a logical, and most likely correct, assumption that the class balance is the same there. Recall that in train set, 98.237034% of data belongs to class 0, and 1.762966% belongs to class 1.

Table 3.6.1 – Target distribution in predicting various experiments

The number of the experiment	**2**	**3**	**4**	**5**	**6**	**7**
% of class 0	99.4810	99.9454	99.9454	99.9545	99.9180	99.7359
% of class 1	0.5190	0.0546	0.0546	0.0455	0.0820	0.2641

The results that are visible in the table clearly give us an understanding that none of the distributions is close to the distribution of the general population, that is, the train set. Looking at the 6th experiment, it becomes surprising that with such a class distribution, it has the best results on the test set and hold-out set – this may indicate that all predictions of the class 1 must be correct and should take approximately the same value (based on how the ROC curve is constructed). The 3rd, 4th, and 5th experiments with a critically low percentage of class 1 predictions also attract attention. If for the 2nd experiment – with the highest percentage – it is lower than the percentage in the general population by more than 3 times, then for the 3rd, 4th and 5th it is lower by more

than 30 times. Such models will not be used further, at least because of this point. The 7th experiment lies approximately in the middle between all parameters, so for further consideration we will leave only the 2nd, 6th and 7th.

Now it is necessary to find out the confidence of model predictions, and to do this, you can see how many predictions lies in different intervals from 0 to 1 in absolute value and percentage. Also, if we assume that the class 0 is set in the case when the prediction is less or equal 0.5, and otherwise – 1, then we want to understand what proportion of the values of "it's" class lies in different intervals. All this was combined in the table below:

Table 3.6.2 – Detailed target distribution of different experiments predictions

The number of the experiment	2			6			7		
Interval	Abs. N	N %	N % of "it's" class	Abs. N	N %	N % of "it's" class	Abs. N	N %	N % of "it's" class
[0.00, 0.05]	10057	91,57	92,05	9579	87,23	87,30	10185	92,74	92,99
(0,05, 0,10]	373	3,40	3,41	780	7,10	7,11	381	3,47	3,48
(0.10, 0.15]	169	1,54	1,55	256	2,33	2,33	149	1,36	1,36
(0.15, 0.20]	113	1,03	1,03	136	1,24	1,24	83	0,76	0,76
(0.20, 0.50]	213	1,94	1,95	222	2,02	2,02	155	1,41	1,42
(0.50, 0.80)	52	0,47	91,23	9	0,08	100,00	29	0,26	100,00
[0.80, 0.85)	4	0,04	7,02	0	0,00	0,00	0	0,00	0,00
[0.85, 0.90)	1	0,01	1,75	0	0,00	0,00	0	0,00	0,00
[0.90, 0.95)	0	0,00	0,00	0	0,00	0,00	0	0,00	0,00
[0.95, 1.00]	0	0,00	0,00	0	0,00	0,00	0	0,00	0,00

It is immediately obvious that the 6th and 7th experiments have extremely low confidence in predictions of class 1, while the 2nd experiment's confidence is slightly higher, and there are more such predictions in principle. Again, if you look at the 6th, it becomes unclear why it received the best metric values, as opposed to the 2nd and 7th. We can only cite some differences in their training:

- 5 models from the ensemble of the 6th experiment were trained in an average of 8.2 epochs, while models from the ensemble of the 2nd and 7th experiments were trained in an average of 14 epochs;
- The 6th experiment was based on splitting folds into groups, while the 2nd and 7th experiments were based on stratified splitting;
- Also, the early stop parameter for the 6th experiment was set to 4 epochs, while for the 2nd and 7th, it was set to 3 and 5, respectively.

The last thing that has not been used so far is predictions on the hold-out set, which, by the way, can give us more insight due to the fact that there we know the true values and can observe what specific samples the models are wrong on. The prediction distribution plot obtained using Kernel Density Estimation [14] is ideal for this purpose, it is shown below:

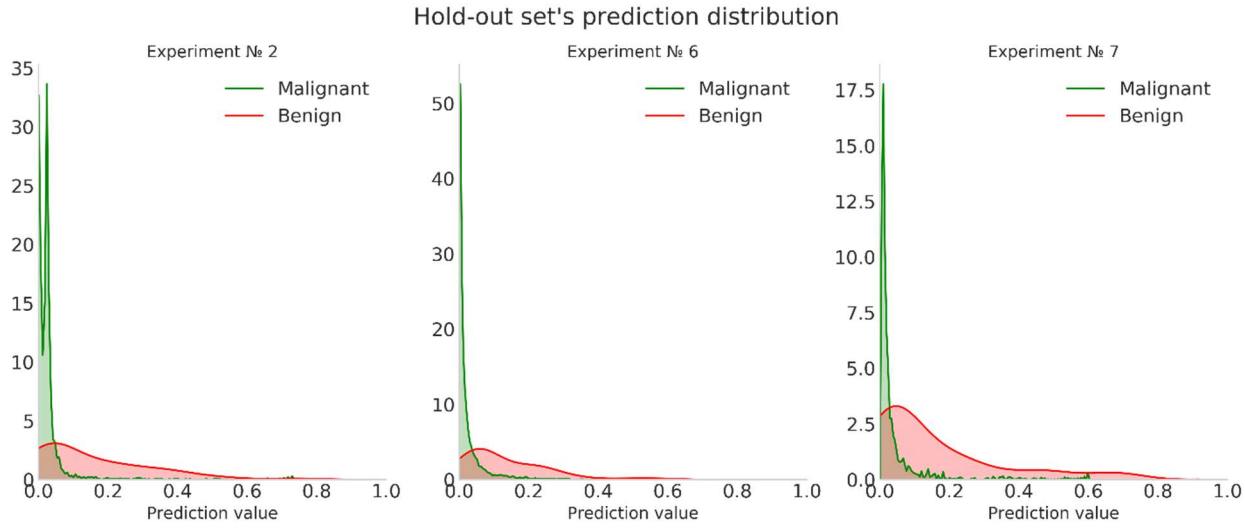

Figure 3.6.1 – Prediction distribution on a hold-out set

The plot shows that the distributions of the 2nd and 7th experiments are very similar to each other, but the 2nd is less sure of class 0 than the first – this is expressed by the fact that the peak on the left side is far from the zero border.

From all of the above, we can conclude that, perhaps, the models of the 6th experiment were underfitted in the ability to identify the first class, and do this rarely, but aptly, while the models from the 2nd and 7th experiments have been overfitted and determine a relatively large number of zero-class samples incorrectly.

After considering and analyzing all the advantages and disadvantages, we decided to take the models and prediction of the 2nd experiment as the main ones for the study.

Further work will be continued as part of the fight against the complicating factor in the form of an extremely imbalanced dataset and as part of an attempt to use given and smoothed pseudo-labeled data for fine-tuning the neural network.

4. Research on the impact of Label Smoothing

4.1 What is Pseudo-labeling?

Pseudo-labeling – the process of obtaining is a class labels from a neural network by inference on data without labels or obtained labels themselves [15]. It can be used for further fine-tuning of the neural network or just for labeling data with some given confidence. This method is successfully used both in research projects [16] and in various machine learning competitions. It is enough to read reviews of problem solutions and make sure that almost always training on pseudo-labeled data is an advantage and improves the generalization ability of models. Moreover, the author himself used this technique and got a better result than before.

4.2 What is Label Smoothing?

Label Smoothing is the process of "smoothing" discrete values of class labels into a continuous values [17, 18, 19]. If we talk about how this idea was initially presented [17], let's imagine the activation function of the penultimate layer of the neural network that solves the classification problem:

$$p_k = \frac{e^{(x^T w_k)}}{\sum_{l=1}^{L} e^{(x^T w_l)}}$$

, where

p_k – probability that the model will assign a k-th class,

w_k – weights and bias of the last layer,

x – vector containing the values of the activations in the penultimate layer.

Then we use cross-entropy as a loss function and try to minimize it to maximize the likelihood.

The problem is with discrete, "hard" labels. The model must produce a large logit[2] value for the correct label. This causes the differences between the largest logit and all the others to become large, which, combined with a limited gradient, reduces the model's ability to adapt, resulting in the model being too confident in its forecasts. This, in turn, can lead to overfitting[3].

[2] A logit is a continuous value of a label that is predicted by the model

[3] Overfitting is a phenomenon in which the model has adapted too much to the data it was trained on and is not able to generalize data that was not available during the training.

But this does not mean that such “hard” labels are bad labels. It all depends on what goals you are pursuing. If you just want to maximize likelihood, then discrete labels are not a bad choice, and we know and have seen that this works. But if your motivation is to create a stable model, it can help.

Thus, the original work introduces a smoothing parameter that changes the target as follows:

$$y_k^{LS} = y_k(1 - \alpha) + \frac{\alpha}{K}$$

Now, instead of minimizing cross-entropy with discrete, “hard” labels, we minimize cross-entropy with continuous, “soft” labels.

This is an original approach, and it has undergone many changes and modifications in the 5 years since its publication. In principle, label smoothing is now called almost any action that “softens” class labels. This study will use a different approach, which will be discussed later.

4.3 Applicability to the problem

To begin with, it is worth noting that successful articles with good results on the topic of label smoothing improve already excellent results, in our case, the task is extremely difficult and initially, we do not have a great solution. We need to understand what actions need to be taken in order to somehow bring our conditions closer to sufficient to obtain reliable results.

4.3.1 The problem of equalizing the distributions

The problem of “shortage” of class 1 labels, which was described earlier, is a big obstacle to fine-tuning models, because if they had poor predictive ability before, then providing data for training, which also has several times fewer representatives of class 1 than it was before, the results will not be better.

Therefore, we need to artificially solve the problem of equalizing the target distribution in the predicted test set and the original train set by some kind of post-processing of the predicted labels. A very common name that meets our requirements is Under-Sampling.

Under-Sampling – the process of changing the distribution of objects of different classes by reducing the number of objects of the prevailing class. In our case, we need to somehow reduce the number of representatives of the class 0 in the test set predictions in such a way as to equalize the distributions. We came up with two ways: a random selection of a subset of the desired size, or a selection of a subset so that the distribution of segments obtained by dividing a segment from 0 to 0.5 by intervals is kept – stratified under-sampling.

It is also possible to approach this issue not by removing representatives of the prevailing class, but by changing the border of the label assignment. You can understand this as follows: now, the border for assigning a label is 0.5: if the prediction is greater than 0.5, it is rounded to 1, and if it is not less than 0.5, it is rounded to 0. What if we move this border? Obviously, the number of class representatives will change. Therefore, as a third option to solve the problem with target distribution, it is proposed to "accumulate" representatives of class 0 and assign them to class 1 until the class distribution reaches the required value. Thus, we do not reduce the amount of data for fine-tuning, but simply re-assign a certain number of labels.

In total, we have 3 ways to solve the above problem.

4.3.2 Definition of "smoothing" functions

Now we need to determine which logit "smoothing" functions we need to try. Definitely, in order to have something to compare it with, we must enter a function that does nothing, that is, leave the logit as it is. Further, just for the sake of experiment and understanding the effect on the result, we want to smooth the labels a little and map them from 0 and 1 to the values 0.05 and 0.95, respectively. The same with the mapping to 0.10 and 0.90.

The following functions will be more reasonable and based on the following hypothesis: what if it is possible to affect the overall confidence of the neural network in predicting the selected class by lowering the initial value of the label, and maximizing all the other labels? This action allows you to bring the prevailing class closer to the border of the label assignment, so more samples will have a chance to "pass" 0.5. This sounds logical, so we would like to try this approach with respect to class 0. Thus, we add 3 more functions that map 0 and 1 to 0.05 and 1, 0.10 and 1 and 0.20 and 1 – that is, we leave the value of the class 1 label equal to 1, but increase the value of the class 0 label, bringing it closer to the boundary of the label assignment, to improve the generalization ability of the model.

4.3.2 Determining subset of data for training

Since we do not have a very stable and high-quality model in this experiment, it makes sense not to completely trust the model's predictions, but to take only those predictions in which it is the most confident.

We decided to make 3 datasets consisting of 50, 80 and 100 percents of the data. When we talk about N % of the data, for example, 50%, we are talking about taking 50% of the most confident predictions for all classes. Therefore, on the one hand, we reduce the amount of data for training, and on the other, we increase their reliability.

Also, there is the question of which splitting of the data into folds to choose. Since the second experiment was performed on stratified folds, the experiments with fine-tuning will be performed by splitting into stratified folds.

4.3.2 Fine-Tuning strategy

There is no clear answer of which type of fine-tuning is better. As is often the case in deep learning, everything is learned in experiments, but since we have limited time and computational resources, we will choose only two learning strategies.

Some options for how to fine-tune: fine-tune the entire network, or its last or last few layers, complete training on the train set with the addition of pseudo-labels, or only on pseudo-labels (in this case, you will have to do several experiments with balancing the loss function), be validated on pseudo-labels or not, change the training parameters to more regularized to avoid overfitting, or not, and many other options.

We chose 2 strategies: training the entire network and its head[4] as 2 different approaches. The other training parameters are the same: training without validation on pseudo-labels for 3 epochs with the same hyperparameters as during the training.

4.4 Conducting experiments

Summing up all the above, we have 3 ways to equalize the distributions, 6 “smoothing” functions, 3 different subsets of the data and 2 types of fine-tuning. In total, we got $3\times6\times3\times2=108$ unique experiments.

At the end of the project, 105 experiments were conducted. 3 more experiments, which were not conducted, were as follows: training the entire network on 100% of the data with the function leaving values as it is on all three methods of equalizing distribution.

4.5 Analysis of the obtained results

According to the table Applications.1, which contains the results of all experiments, we are ready to provide the main conclusions on the conducted experiments. Note that all conclusions will be provided based on the metric value on the hold-out set. Recall that the metric value of the 2nd experiment is 0.8263.

To begin with, the first question is: did the pseudo-labeling help? Yes, it is, in 2 out of 3 cases. If we take a “raw” pseudo-labels, then depending on the methods of equalizing the

[4] The head is a synonym for the last layer of the neural network.

distribution, the metric takes the following values: 0.8313, 0.8259, 0.8387. The stratified method showed worse results than without fine-tuning.

Next, is it better to fine-tune only the last layer or the entire network? Based on the average, the training of the last layer has a significant advantage: 0.7712 for the entire network versus 0.8333 for the head of the network. This is a critical difference, and we can't answer the question of why this result was obtained, other than the assumption of overfitting. What is interesting is that if you leave only the training with the head in the results, the answer to the previous paragraph changes from 2/3 of cases to 3/3 of cases. Therefore, we will only consider statistics with training the head of the network.

Now, what percentage of the most confident predictions is better to take? The average metric values were as follows: 0.8001 for 100%, 0.8485 for 80%, and 0.8520 for 50%. Therefore, we can conclude that using 50 to 80% of the most confident predictions of all classes can significantly improve the results.

If you look at the results for the "smoothing" function, the best function was the one that left the data as is, it gave the average value of the metric equal to 0.8439, followed by the function (0, 1) → (0.05, 1) from 0.8353 onwards to 0.8275. This is a rather interesting result, which suggests that the pseudo-labeling was not accurate enough for the functions to perform what they were theoretically supposed to do. And because of the insufficient quality of pseudo-labels, they just made more noise on target.

In conclusion, we would like to say that out of 54 experiments with head training only, 40 received a better metric value on the hold-out set than the original experiment. And, that 14 of the 14 experiments with the worst value took 100% of the data.

Thus, based on all the above, we conclude that even in such a complex task, with all the negative factors, the proposed methods still allow us to improve the results of the neural network.

5. Technology stack

Computing power used:

- Desktop computer: NVIDIA RTX 2080Ti 11Gb VRAM, Intel i7-9700k, 32Gb RAM, Ubuntu;
- Laptop: NVIDIA GeForce MX230 2Gb VRAM, Intel i7-10510U, 16Gb RAM, Ubuntu;
- Google Colab: NVIDIA Tesla K80 12Gb VRAM / NVIDIA Tesla P100 16Gb VRAM, Intel Xeon, 13Gb RAM, дистрибутив Linux.

Software used:

- GitKraken Pro paired with github.com as a version control system;
- VS Code, Jupyter Lab with Jupyter Notebook for coding.

Development technologies used:

- Main programming language – Python;
- The main framework for deep learning – PyTorch;
- Scripts on bash;
- Configuration files in the YAML format;
- Description of the solution progress, ideas, hypotheses, and registering via MARKDOWN files;
- Storing tabular data in CSV;
- Modules: pandas, numpy, sklearn, albumentations, ttach, opencv-python, tqdm, PIL, multiprocessing, etc.

Conclusion

In the course of the work, information about melanoma, skin lesions, and other issues related to this area was studied. Then, using the obtained knowledge, correct augmentations and data transformations reflecting real-world situations were selected for neural network training. We also conducted several experiments related to data representation, different neural network architectures, and the choice of relevant approaches and optimization algorithms and their hyperparameters.

This project opened the way to research such complex topics as extremely imbalanced datasets, calibration of model confidence, pseudo-labeling, label smoothing, and others. Therefore, the main achievement of the work can be considered the study of an extremely poorly studied area and its relevance in the current development of neural networks and the appearance of new problems that they will need to solve sustainably.

All the developments and code base for reproducing the results can be found in the project repository on GitHub https://github.com/detkov/LSoPLD.

Bibliography

1. Melanoma description. URL:https://www.kaggle.com/c/siim-isic-melanoma-classification.
2. ROC curve. URL:https://en.wikipedia.org/wiki/Receiver_operating_characteristic.
3. Confusion matrix. URL:https://en.wikipedia.org/wiki/Confusion_matrix.
4. Box plot. URL:https://en.wikipedia.org/wiki/Box_plot.
5. Signs and Symptoms of Melanoma Skin Cancer. URL:https://www.cancer.org/cancer/melanoma-skin-cancer/detection-diagnosis-staging/signs-and-symptoms.html.
6. Cross-validation. URL:https://en.wikipedia.org/wiki/Cross-validation_(statistics).
7. Mingxing Tan, Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv preprint arXiv:1905.11946, 2019.
8. ImageNet Dataset. URL:http://www.image-net.org/.
9. Qizhe Xie, Minh-Thang Luong, Eduard Hovy, Quoc V. Le. Self-training with Noisy Student improves ImageNet classification. arXiv preprint arXiv:1911.04252, 2019.
10. Binary cross-entropy and log-loss. URL:https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.
11. Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980, 2014.
12. Ilya Loshchilov, Frank Hutter. Decoupled Weight Decay Regularization. arXiv preprint arXiv:1711.05101, 2017.
13. PyTorch framework implementing various optimization algorithms. URL:https://pytorch.org/docs/stable/optim.html.
14. Kernel Density Estimation. URL: https://en.wikipedia.org/wiki/Kernel_density_estimation.
15. Lee, Dong-Hyun. (2013). Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. ICML 2013 Workshop : Challenges in Representation Learning (WREPL).
16. Longlong Jing, Yingli Tian. Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey. arXiv preprint arXiv:1902.06162, 2019.
17. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. arXiv preprint arXiv:1512.00567, 2015.
18. Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, Geoffrey Hinton. Regularizing Neural Networks by Penalizing Confident Output Distributions. arXiv preprint arXiv:1701.06548, 2017.

19. Rafael Müller, Simon Kornblith, Geoffrey Hinton. When Does Label Smoothing Help? arXiv preprint arXiv:1906.02629, 2019.

Appendix

Method for equalizing the distribution	% of data used	№ of the smoothing function	Training only the last layer	Hold-Out AUC-ROC
rdf_bnd	50	1	true	0.855
rnd_undspml	50	4	true	0.8541
rnd_undspml	50	5	true	0.8539
str_undspml	50	5	true	0.8538
rnd_undspml	50	3	true	0.8538
str_undspml	50	4	true	0.8536
str_undspml	80	4	true	0.8535
str_undspml	50	3	true	0.8535
rnd_undspml	50	6	true	0.8533
str_undspml	50	2	true	0.8533
rnd_undspml	50	1	true	0.8533
rnd_undspml	50	2	true	0.8532
str_undspml	50	6	true	0.8532
str_undspml	80	3	true	0.853
str_undspml	80	1	true	0.851
rdf_bnd	80	4	true	0.8507
str_undspml	80	6	true	0.8507
rdf_bnd	50	4	true	0.8505
str_undspml	80	5	true	0.8504
rdf_bnd	50	3	true	0.8504
str_undspml	50	1	true	0.8504
str_undspml	80	2	true	0.85
rdf_bnd	80	3	true	0.85
rnd_undspml	80	4	true	0.8499
rnd_undspml	80	3	true	0.8496
rdf_bnd	50	5	true	0.8492
rnd_undspml	80	5	true	0.8492
rnd_undspml	80	2	true	0.8483
rdf_bnd	50	2	true	0.8478
rdf_bnd	80	1	true	0.8474
rnd_undspml	80	6	true	0.8473
rdf_bnd	80	5	true	0.8453
rdf_bnd	80	2	true	0.8439
rdf_bnd	50	6	true	0.8437
rnd_undspml	80	1	true	0.8424
rdf_bnd	80	6	true	0.8405
rnd_undspml	100	1	true	0.8387
rdf_bnd	100	1	true	0.8313
rdf_bnd	100	3	true	0.8309

rdf_bnd	100	4	true	0.8307
				0.8263
str_undspml	100	1	true	0.8259
rdf_bnd	80	5	false	0.8215
rdf_bnd	100	5	true	0.82
rdf_bnd	50	4	false	0.8153
rdf_bnd	100	2	true	0.8129
rnd_undspml	50	2	false	0.8105
rdf_bnd	80	1	false	0.8081
rdf_bnd	80	4	false	0.8078
rdf_bnd	50	5	false	0.8034
rdf_bnd	50	3	false	0.803
str_undspml	80	4	false	0.8027
rdf_bnd	80	2	false	0.8021
str_undspml	80	1	false	0.8015
rdf_bnd	80	6	false	0.7998
rdf_bnd	50	2	false	0.7995
rnd_undspml	50	1	false	0.7987
rdf_bnd	50	1	false	0.7987
rdf_bnd	100	2	false	0.7979
rnd_undspml	100	4	true	0.7968
rdf_bnd	50	6	false	0.7961
rdf_bnd	100	6	true	0.796
rnd_undspml	100	3	true	0.7938
rdf_bnd	100	3	false	0.7917
rdf_bnd	100	5	false	0.7901
rnd_undspml	80	2	false	0.79
rnd_undspml	50	6	false	0.7885
rnd_undspml	100	6	true	0.7883
str_undspml	50	4	false	0.7879
str_undspml	50	2	false	0.7874
str_undspml	80	3	false	0.7867
str_undspml	50	3	false	0.7859
rnd_undspml	100	5	true	0.785
rnd_undspml	80	1	false	0.7837
rdf_bnd	80	3	false	0.7835
str_undspml	80	6	false	0.7832
str_undspml	50	5	false	0.7825
rnd_undspml	50	3	false	0.7796
rnd_undspml	100	2	true	0.7791
str_undspml	100	4	true	0.7778
rnd_undspml	50	5	false	0.7773
rnd_undspml	100	2	false	0.7771
str_undspml	100	5	false	0.7762
str_undspml	100	3	true	0.7759
str_undspml	100	6	true	0.7748
str_undspml	100	3	false	0.7748
rnd_undspml	80	3	false	0.7728

str_undspml	100	5	true	0.7726
str_undspml	80	2	false	0.7711
str_undspml	100	2	true	0.7711
rnd_undspml	50	4	false	0.7696
rdf_bnd	100	4	false	0.7678
rnd_undspml	100	4	false	0.7678
rnd_undspml	80	5	false	0.7665
str_undspml	100	2	false	0.7663
str_undspml	100	4	false	0.7662
str_undspml	50	6	false	0.7623
str_undspml	80	5	false	0.7623
rnd_undspml	80	6	false	0.7612
str_undspml	50	1	false	0.7267
rnd_undspml	100	5	false	0.7221
rdf_bnd	100	6	false	0.6946
rnd_undspml	100	3	false	0.6907
rnd_undspml	80	4	false	0.6548
rnd_undspml	100	6	false	0.6426
str_undspml	100	6	false	0.5725

Table Attachments.1 – a complete table with experiments' results, where in the "Method for equalizing the distribution" column, the value "rdf_bnd" is a method with a shift in the label assignment boundary, "rnd_undspml" is a method of random under-sampling, "str_undspml" is a method of stratified under-sampling, and "№ of the smoothing function" corresponds to the ordinal number of function appearing in the text. A row with only the metric value shows the original 2nd experiment.