FACULTY OF SCIENCES OF THE UNIVERSITY OF PORTO

Research on Deep Augmentation for Ordinal Regression

Internship Report

Beatriz Marques de Sá

&

Bachelor's Degree in Computer Science

FCUP Supervisor: Prof. Rolando da Silva Martins

Company Supervisor: Ricardo Cruz e Prof. Jaime S. Cardoso

July 3, 2024

Contents

1 Introduction **1**

2 Related work **2**

2.1 Machine Learning 2

2.2 Data Augmentation 3

2.2.1 Classical techniques 3

2.2.2 Advance techniques 3

2.2.2.1 Cutout 3

2.2.2.2 Mixup 5

2.2.2.3 Cutmix 6

2.3 Tools used 8

2.3.1 PyTorch 8

2.3.2 Argparse 8

2.3.3 Time 8

2.3.4 Matplotlib 8

3 Experimental Details **9**

3.1 Proposed Methods 9

3.1.1 Ordinal Mixup 9

3.1.2 Ordinal Cutmix 11

3.1.2.1 OSequential Ordinal Cutmix 11

3.1.2.2 Paired Ordinal Cutmix 12

3.2 Datasets 14

3.2.1 Pap Smear (SMEAR2005) 15

3.2.2 Dating historical color images (DHCI) 16

3.3 Experimental Protocol 17

3.4 Evaluation Metrics 17

3.5 Results 18

4 Conclusion **20**

References **21**

List of Figures

1 Illustration of a Cutout example 4

2 Illustration of a Mixup example 5

3 Illustration of a Cutmix example 7

4 Illustration of a *Ordinal Mixup* exemple with $\tau = 0.5$ 10

5 Illustration of a *Sequential Ordinal Cutmix* exemple with $\tau = 0.5$ 12

6 Illustration of a *Paired Ordinal Cutmix* example with $\tau = 0.7$ 14

7 Illustration of a Example of SMEAR2005 classes 15

8 Illustration of a Example of DHCI classes 16

List of Tables

1 Information of the datasets used for this experimentation 15
2 Results for the Dataset DHCI 19
3 Results for the Dataset SMEAR2005 20

Listings

1 Ordinal Mixup 10
2 Sequential Ordinal Cutmix 11
3 Paired Ordinal Cutmix 12

1 Introduction

Deep learning has stood out as a powerful approach in various fields, especially those involving large amounts of data, such as computer vision and natural language processing. However, a fundamental challenge in training neural networks is the need for large volumes of labeled data to achieve robust and generalizable performance. In many practical scenarios, obtaining sufficient labeled data can be costly and time-consuming, making the use of data augmentation techniques crucial.

Data augmentation involves artificially creating new samples from existing ones, thereby increasing the variability of the training dataset without the need to collect more data. Common techniques for data augmentation in computer vision include geometric transformations, changes in color and noise, as well as more advanced methods such as mixup [1] and cutmix [2]. These methods have shown significant improvements in model performance, promoting greater generalization.

Specifically, in ordinal tasks where there is an intrinsic order between classes (e.g., healthy < mild < moderate < severe < proliferative), it is essential that data augmentation techniques respect this ordering. Recently, the mixup method has been adapted for ordinal tasks [3], allowing weighted linear combinations of images and their respective classes to preserve the underlying order.

In this report, our main objective is to adapt the cutmix method for ordinal tasks. Cutmix is a data augmentation technique that involves combining parts of different images to create a new sample that is a blend of the original images and their respective classes. Adapting cutmix for ordinal tasks requires a careful approach to ensure that the generated new sample accurately reflects the order relationship between classes.The code is available at [4]

2 Related work

2.1 Machine Learning

Machine Learning is a subset of artificial intelligence that focuses on developing algorithms that enable computers to learn from and make predictions or decisions based on data. Rather than being explicitly programmed to perform a task, machine learning algorithms identify patterns and infer rules from the data they are trained on. This process can be broadly categorized into three main types:

- ***Supervised Learning:*** In supervised learning, the algorithm is trained on a labeled dataset, which means that each training example is paired with an output label. The goal is to learn a mapping from inputs to outputs that can be used to predict the output for new, unseen inputs.
  - **Regression:** In regression tasks, the output is a continuous value. Examples include predicting house prices, stock prices, or any other quantity that varies continuously. Linear regression, decision trees, and support vector regression are common techniques.
  - **Classification:** In classification tasks, the output is a discrete label. Examples include email spam detection, image classification, and medical diagnosis. Techniques include logistic regression, support vector machines (SVM), k-nearest neighbors (KNN), and neural networks.

- ***Unsupervised Learning:*** Unsupervised learning deals with unlabeled data, and the goal is to identify patterns or structures within the data. The algorithm tries to learn the underlying distribution without any guidance on what the outputs should be.
  - **Clustering:** The task of grouping similar data points together. Common algorithms include K-means clustering, hierarchical clustering, and DBSCAN (Density-Based Spatial Clustering of Applications with Noise).
  - **Dimensionality Reduction:** The task of reducing the number of variables under consideration. Principal component analysis (PCA), t-distributed stochastic neighbor embedding (t-SNE), and autoencoders are widely used techniques.

- ***Reinforcement Learning:*** In reinforcement learning, an agent interacts with an environment and learns to make decisions by receiving rewards or penalties. The goal is to learn a policy that maximizes the cumulative reward over time.
  - **Value-Based Methods:** Algorithms like Q-learning estimate the value of taking a certain action in a given state and update these values iteratively based on the rewards received.
  - **Policy-Based Methods:** Algorithms like policy gradient methods directly optimize the policy, which is the mapping from states to actions, to maximize the expected reward.
  - **Actor-Critic Methods:** These methods combine value-based and policy-based approaches to achieve more stable and efficient learning.

2.2 Data Augmentation

Data augmentation is a technique used in deep learning and machine learning referred to as *"the process of generating new data examples for training a model."* [5] It *"is a useful technique for providing more information from less data."* [5] The following is an overview of both classical and advanced techniques used in data augmentation..

2.2.1 Classical techniques

Classical image augmentation techniques remain relevant and widely used in computer vision due to their simplicity and effectiveness in enhancing the size and diversity of training datasets. These methods are easy to implement and computationally efficient, making them suitable for large-scale datasets and real-time applications. Here is an examination of where each technique is most effectively applied:

- **Flipping and Rotating:** These techniques are ideal for making models robust to changes in object orientation.
- **Cropping and Resizing:** Useful for addressing variations in object scale and position within the image.
- **Color Jittering:** Helps handle changes in lighting conditions and color variations in the dataset.
- **Adding Noise:** Beneficial for managing variations in image quality and simulating noisy environments.
- **Image Warping:** Enhances a model's ability to handle changes in object pose and simulates variations in camera viewpoint.
- **Random Erasing:** Makes models more robust to occlusions or clutter in images and simulates scenarios with missing data.

2.2.2 Advance techniques

The advanced techniques not only increase the variability of the training data but also enhance the model's ability to generalize to unseen data and improve robustness. Each method has specific advantages [6] depending on the task and dataset characteristics, allowing practitioners to tailor augmentation strategies to their specific needs in computer vision tasks.

2.2.2.1 Cutout

Cutout [7] works by randomly selecting a square region of a specified size within an image and setting the pixel values in this region to zero (or another constant value). This process can be formally described as follows:

- **Image Selection:** For each image in the training dataset, a square region with side length l is randomly chosen.
- **Region Masking:** The pixel values in the chosen region are set to zero (or another constant value), effectively "cutting out" that part of the image.

Mathematically, we can represent an original image X with the cutout region as X':

$$X' = X \odot M$$

where M is a binary mask of the same size as X, with values 0 in the selected square region and 1 elsewhere. $\odot$ denotes element-wise multiplication.(See Figure 1).

Image 1 Result of Cutout

Figure 1: Illustration of a Cutout example

Advantages:

- **Occlusion Robustness:** By training on images with missing regions, the model learns to recognize objects even when parts of them are occluded.
- **Reduced Overfitting:** Cutout acts as a regularizer by forcing the model to rely on the entire image rather than memorizing specific parts.
- **Feature Utilization:** By masking out parts of the image, Cutout encourages the model to utilize all available features, leading to better feature representation and utilization.
-

Limitations:

- **Loss of Information:** By masking parts of the image, Cutout can remove critical information that could be useful for the learning task, potentially harming performance if the cutout regions contain important features.
- **Dependence on Cutout Size:** The choice of the cutout region size is crucial. A very large cutout size can remove too much information, while a very small cutout size might not provide the desired regularization benefits.

- **Random Positioning:** Although the random selection of cutout regions contributes to the diversity of augmented data, there may be instances where the cutout region negatively affects training quality, especially if it systematically cuts out important regions.

2.2.2.2 Mixup

Mixup [1] is a data augmentation technique introduced by Zhang et al. (2018) that creates new training examples by linearly interpolating pairs of examples and their corresponding labels. Specifically, given two training examples (x_i, y_i) and (x_j, y_j), this method generates a new exemple $(\tilde{x}, \tilde{y})$ as follows:

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

where λ is a mixing coefficient sampled from a beta distribution Beta(α, α), with α being a hyperparameter that controls the strength of interpolation.(See Figure 2)

Image 1
Label [1.0, 0.0]

Image 2
Label [0.0, 1.0]

Result of Mixup
Label [0.6, 0.4]

Figure 2: Illustration of a Mixup example

The labels are created with the size of the number of different original images and each number corresponds to the percentage of the original image in the final image. For example in Figure 2, the image 1 has the label [1.0,0.0] which means that it corresponds 100% to the first original image and 0% to the second one. Mixup encourages the model to behave linearly in-between training examples, thereby reducing the model's tendency to overfit and improving its generalization capabilities.

Advantages:

- **Memorization Prevention:** Mixup reduces the risk of neural networks memorizing incorrect labels by blending features and labels, fostering a more robust learning process.
- **Smooth Decision Boundaries:** It promotes smoother transitions between classes, leading to more refined uncertainty estimation and better generalization.

- **Robustness:** Enhances resilience against adversarial examples and stabilizes training in Generative Adversarial Networks (GANs), improving overall model reliability.
- **Domain Agnostic:** Suitable for diverse data modalities such as computer vision, natural language processing, and speech, making it widely applicable.

Limitations:

- **Inter-Class Mixing Only:** Mixup primarily supports mixing between different classes, limiting its effectiveness in scenarios where intra-class mixing is required.
- **Potential Unnatural Examples:** Generated examples may not accurately represent their respective classes, potentially confusing the model during training.
- **Compatibility Issues:** Less effective when combined with Supervised Contrastive Learning, which requires true labels during its pre-training phase.
- **Incompatibility with Label Smoothing:** Mixup and label smoothing may not work well together as both techniques modify labels, potentially compromising performance.

2.2.2.3 Cutmix

CutMix [2] enhances the training of convolutional neural network models by combining elements from two images through cut-and-paste operations. While Mixup blends entire images, CutMix creates new samples by cutting a random patch from one image and pasting it onto another. Corresponding labels are also mixed proportionally to the area of the patches involved.

Formally, given two input images (x_i, y_i) and (x_j, y_j), this method generates a new mixed image $\tilde{x}$ and a mixed label $\tilde{y}$:

$$\tilde{x} = x_i \odot M + x_j \odot (1 - M)$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

where M is a binary mask indicating the region to be replaced, and λ is the proportion of the area replaced, which determines how the labels are mixed[2].

Additionally, the parameters rx, rw, ry, and rh are used to define the region from where patches are cut and pasted:

- $rx \sim \text{Unif}(0, W)$ defines the horizontal position where the patch is cut from the image, with W being the image width.
- $rw = W\sqrt{1 - \lambda}$ determines the width of the cut patch, reduced proportionally by $\sqrt{1 - \lambda}$.
- $ry \sim \text{Unif}(0, H)$ defines the vertical position where the patch is cut from the image, with H being the image height.
- $rh = H\sqrt{1 - \lambda}$ determines the height of the cut patch, also reduced proportionally by $\sqrt{1 - \lambda}$.

These parameters ensure that this data augmentation technique preserves the structural coherence of the original images while blending local information from input images, encouraging

models to better learn object localization and focus on specific parts of objects rather than the entire object (See Figure 3).

Image 1
Label [1.0, 0.0]

Image 2
Label [1.0, 0.0]

Result of Cutmix
Label [0.45, 0.55]

Figure 3: Illustration of a Cutmix example

Advantages:

- **Improved Generalization:** CutMix helps models generalize better by introducing variety in training samples through the mixture of information from multiple images.
- **Enhanced Localization Learning:** By forcing the model to predict based on partial information due to patch overlapping, CutMix encourages precise object localization learning.
- **Robustness Against Overfitting:** Controlled mixing of samples reduces the risk of overfitting, allowing the model to focus on general object characteristics rather than memorizing specific examples.
- **Efficient Data Utilization:** By creating new synthetic samples from existing images, CutMix enables more efficient use of available dataset.

Limitations:

- **Impact of Mixing Mask:** The quality of mixing depends on the mask M used, and poorly chosen masks can introduce artifacts or conflicting information in generated samples.
- **Computational Complexity:** Implementing CutMix can increase computational complexity during training due to real-time operations involving cropping, mixing, and mask application.
- **Fine-tuning Requirements:** The effectiveness of CutMix may vary depending on the specific task and the model's sensitivity to artifacts introduced by mixing.

2.3 Tools used

To develop this project, we used Python as the programming language and some auxiliary libraries.

2.3.1 PyTorch

PyTorch is an open-source machine learning library developed by Meta AI Research. It is widely used for deep learning applications due to its flexibility, ease of use, and efficiency. PyTorch enables the creation and training of neural network models in a dynamic and interactive manner, which facilitates experimentation and the development of new ideas. One of its most important features is tensor computation with strong GPU support, which significantly accelerates calculations.

- **Torch:** is the core of PyTorch and contains the main functionalities of the library
- **Torchvision:** is a library that complements PyTorch with tools specific to computer vision. It includes popular datasets, image transformations, and pre-trained models.
- **Torchmetrics:** implements various metrics to evaluate the quality of the model.

2.3.2 Argparse

Argparse is a standard Python library for parsing command-line arguments, enabling to define which arguments your program accepts and parse those provided by the user. It enhances user interaction when running a program via the command line by clearly specifying expected arguments and how they should be interpreted. This includes mandatory and optional arguments, default values, and even flags that trigger specific behaviors.

2.3.3 Time

Time is a standard Python library that provides various time-related functions. It allows to measure time intervals, create delays in program execution, and retrieve the current time. We used this module to estimate the total time the algorithm would take to train.

2.3.4 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is widely used for generating plots, histograms, bar charts, scatter plots, and other types of graphs. The pyplot module in matplotlib provides a MATLAB-like interface, making it easier to create and customize plots. We used this module during the testing phase to see if the algorithms were working correctly.

3 Experimental Details

3.1 Proposed Methods

3.1.1 Ordinal Mixup

As mentioned before, Mixup is a popular data augmentation technique used to improve the robustness and generalization of models. In this method, two images X_1 and X_2 with their respective labels $Y_1 = (1, 0)$ and $Y_2 = (0, 1)$ are combined to produce a new image $X_3 = 0.25X_1 + 0.75X_2$ with a new label $Y_3 = (0.25, 0.75)$, meaning that image X_3 is composed by 0.25 of image X_1 and 0.75 of imageX_2 .

However, these approaches, in their original form, are not suitable for ordinal labels. For this reason, these data augmentation techniques can be adapted to be more suitable for ordinal data. For example, given three images X_1, X_2, and X_3 with their respective labels $Y_1 = (1, 0, 0)$, $Y_2 = (0, 1, 0)$, and $Y_3 = (0, 0, 1)$, the adapted result would be something like this: $X_4 = 0.25X_1 + 0.70X_2 + 0.05X_3$ with the label $Y_4 = (0.25, 0.70, 0.05)$, meaning that image X_4 is composed by 0.25 of image X_1, 0.70 of imageX_2 and 0.05 of image X_3.

Thus, Airton proposed [3] a strategy in which a unimodal exponential distribution (see Equation 1) is generated to impose unimodality and act as weights for each sample image in the mixing process [8].

$$\exp(j, k, \tau) = \text{sofmax}(-\frac{|j - k|}{\tau}) \tag{1}$$

Equation 1 shows how the exponential is calculated, receiving as parameters j, k and τ, where j represents the class for which we want to obtain the probabilities, k the class target and τ the standard deviation . The exp function performs a softmax of the negation of $(j - k)/\tau$, where τ can take values between 0 and 1, but Airton uses only the following values: [0.15, 0.45, 0.6, 1.0].

Contrary to Airton's approach, we employ a modified Poisson distribution that has already been proposed [9] (see Equation 2):

$$\text{log-poisson}(j, k, \tau) = \text{sofmax}(\frac{j \log(k) - k - \log(j!)}{\tau}) \tag{2}$$

Equation 2 shows how the probability is calculated using a modified Poisson distribution, receiving as parameters j and k, where j is the class for which we want to obtain the probabilities, k is a class target and τ is the standard deviation.

The modification consisted of dividing by τ so that we could control the variance of the distribution and applying a softmax to ensure the resulting function is a probability mass function. We started with $\tau \approx 1$, and at each epoch, a new τ is calculated using the following formula:

$$\tau = (1 - \frac{it - 1}{nits})1 + 0.00001$$

where *"it"*is a value between 0 and *nits - 1* and *nits* is the total number of iterations for training the

model.

The same modified Poisson distribution was also used in the following methods.

Here is the final result of the exponential mixup algorithm after modifying the probability function:

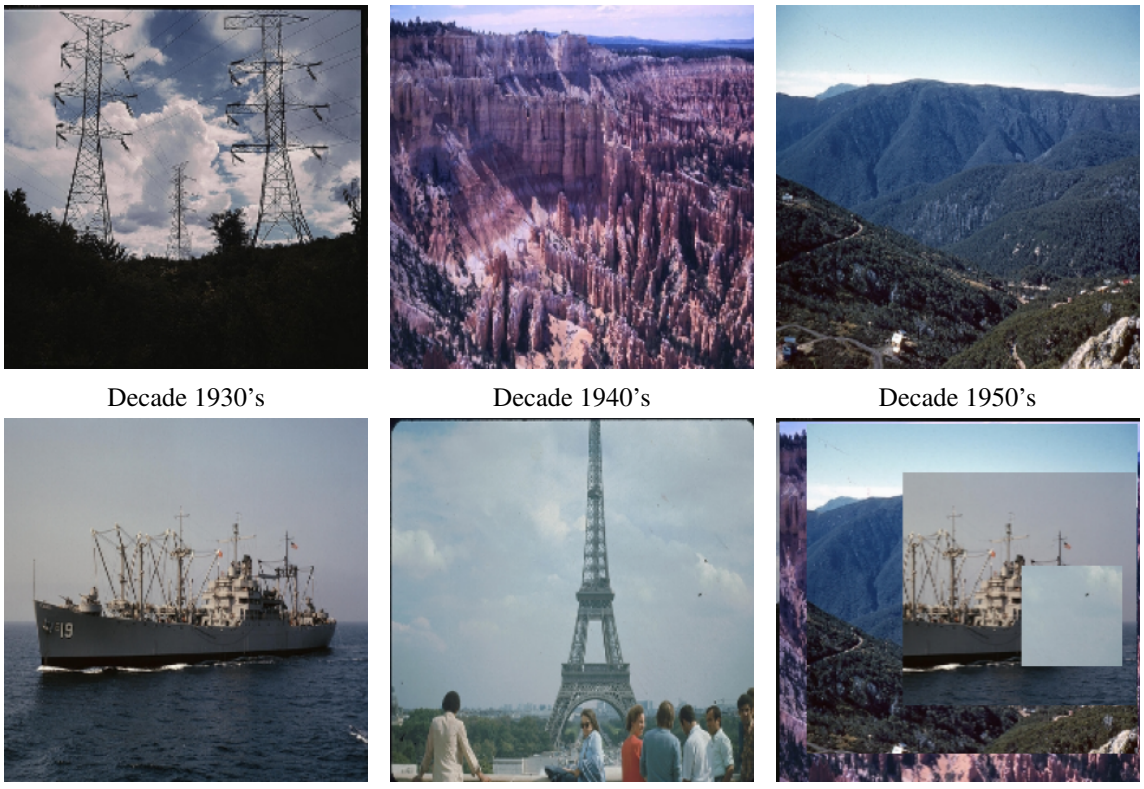
```
def ordinal_exponential_mixup(images, tau):
    device = images.device
    n = images.shape[0]
    num_classes = images.shape[1]

    labels = torch.randint(0, num_classes, [n], device=device) + 1
    labels = poisson_probs(num_classes, labels, tau, device)

    mixup_images = torch.sum(images * labels[:, :, None, None, None], 1)
    return mixup_images, labels
```

Listing 1: Ordinal Mixup

The following example shows the application of the ordinal mixup on the DHCI dataset.



Decade 1930's | Decade 1940's | Decade 1950's

Decade 1960's | Decade 1970's | Result

Figure 4: Illustration of a *Ordinal Mixup* exemple with $\tau = 0.5$

3.1.2 Ordinal Cutmix

As mentioned before, CutMix is a popular data augmentation technique used to improve the robustness and generalization of models. In the this method, two images X_1 and X_2 with their respective labels $Y_1 = (1, 0)$ and $Y_2 = (0, 1)$ generates a new image $X_3 = X_1 \odot M + X_2 \odot (1 - M)$ with a new label $Y_3 = (0.25, 0.75)$,meaning that image X_3 is composed of 0.25 of the area of image X_1 and 0.75 of the area of image X_2 .

However, these approaches, in their original form, are not suitable for ordinal labels. For this reason, these data augmentation techniques can be adapted to be more suitable for ordinal data, we employ two different strategies. For example, given three images X_1, X_2, and X_3 with their respective labels $Y_1 = (1, 0, 0)$, $Y_2 = (0, 1, 0)$, and $Y_3 = (0, 0, 1)$, the adapted result would be something like this: $X_4 = M_1 \odot X_1 + M_2 \odot X_2 + M_3 \odot X_3$ with the label $Y_4 = (0.25, 0.70, 0.05)$, meaning that image $X_4$is composed of 0.25 of the area of image X_1, 0.70 of the area of image X_2 and 0.05 of the area of image X_3.The values of M_1, M_2 and M_3 depend of the strategies used.

3.1.2.1 OSequential Ordinal Cutmix

The strategy in question aims to organize images sequentially, inserting them into one another at random positions according to their probabilities derived from a modified Poisson distribution.

The following code exemplifies this approach:

```
def each_nested(images, probabilities, x1, y1, x2, y2, output):
    H, W = images.shape[2:]
    for k in range(1, len(probabilities)):
        w = int(W*sum(probabilities[k:]))
        h = int(H*sum(probabilities[k:]))
        x1 = torch.randint(x1, x2-w+1, ())
        y1 = torch.randint(y1, y2-h+1, ())
        x2 = x1+w
        y2 = y1+h
        output[:, y1:y2, x1:x2] = images[k][:, y1:y2, x1:x2]
```

Listing 2: Sequential Ordinal Cutmix

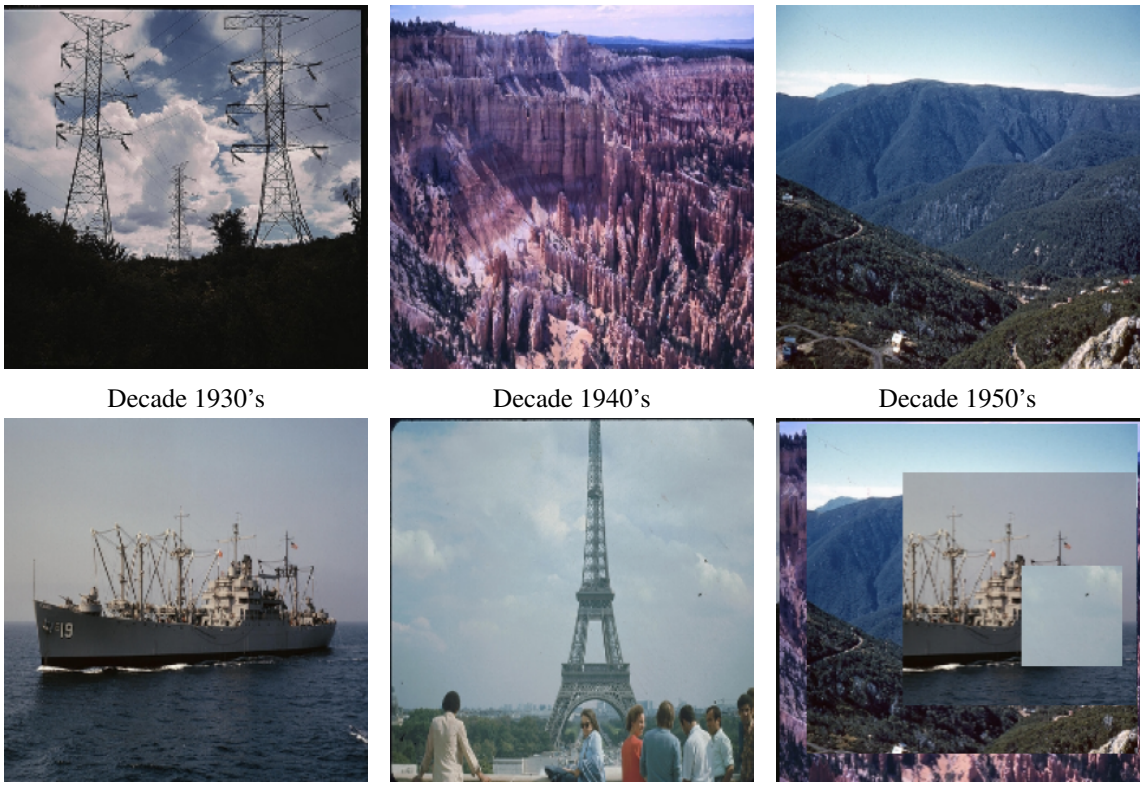
The function each_nested takes the following parameters:

- images: The images that the model will use.
- probabilities: Probabilities associated with each image.
- x1, y1, x2, y2: Dimensions of the final image.
- output: A clone of the first image (images[0].clone())

This function computes the maximum height and width (H, W) of the images. It then iterates through each image from images starting from index 1 (since index 0 is the base image), based

on the probabilities provided. For each image *images[k]*, is calculates the maximum width (w) and height (h) based on the sum of the probabilities from position k to the end of the *probabilities* list. It uses *torch.randint* to randomly select starting coordinates ($x1$ $y1$) within the allowed range, ensuring that the selected image images[k] fits entirely within the output image. Finally, it copies the selected portion of images[k] into the corresponding region of output.

The following example shows the application of the Sequential Ordinal Cutmix on the DHCI dataset.



Decade 1930's | Decade 1940's | Decade 1950's

Decade 1960's | Decade 1970's | Result

Figure 5: Illustration of a *Sequential Ordinal Cutmix* exemple with $\tau = 0.5$

3.1.2.2 Paired Ordinal Cutmix

For this strategy, instead of using the first image as the base, as in the previous proposal, we opted to use the image with the highest probability as the base image. Then, for images positioned on each side of the central class, we randomly select a position, ensuring that the two sides do not intersect, and perform the Sequential Ordinal Cutmix for each side.

The following code exemplifies this approach:

```
1 def paired_ordinal_cutmix(images, probabilities, center):
2     output = images[center].clone()
3     H, W = output.shape[1:]
4
```

```
    wl = int(W * sum(probabilities[:center]))
    hl = int(H * sum(probabilities[:center]))
    wr = int(W * sum(probabilities[center + 1:]))
    hr = int(H * sum(probabilities[center + 1:]))

    while True:
        x1_left = torch.randint(0, W - wl, ())
        y1_left = torch.randint(0, H - hl, ())
        x1_right = torch.randint(0, W - wr, ())
        y1_right = torch.randint(0, H - hr, ())

        if not intersects(x1_left, y1_left, wl, hl, x1_right, y1_right, wr,hr):
            break

    imag = images[:center + 1].tolist()
    imag = torch.tensor(imag[::-1])
    prob = probabilities[:center + 1].tolist()
    prob = torch.tensor(prob[::-1])

    each_nested(imag, prob, x1_left, y1_left, x1_left + wl, y1_left + hl,
        output)
    each_nested(images[center:], probabilities[center:], x1_right, y1_right,
        x1_right + wr, y1_right + hr, output)

    return output
```

Listing 3: Paired Ordinal Cutmix

The function paired_ordinal_cutmix takes the following parameters:

- images: The images that the model will use.
- probabilities: Probabilities associated with each image.
- center: The target class, that is, the class with the highest probability

This function begins by setting the output as a clone of the central image and calculating the height and width (H, W) of the images and the maximum width (*wl* and *wr*) and height (*hl* and *hr*) for the images based on the given probabilities. For the left side, these dimensions are computed from the sum of probabilities up to the center index. For the right side, they are computed from the sum of probabilities after the center index. Next, it enters a loop where it randomly selects starting coordinates (*x1_left*, *y1_left*, *x1_right*, *y1_right*) within the allowed range. The loop ensures that the selected regions do not overlap by checking for intersections.If the regions do not intersect, the loop breaks.Then, the code processes the images and probabilities. For the left side, it reverses the order of images and probabilities to maintain the correct sequence. The *each_nested* function is called for both the left and right regions, using the calculated coordinates. This function copies the selected portions of the images into the corresponding regions of the output. In essence, this process iterates through each image starting from the center, calculating the maximum dimensions

based on the probabilities, selecting random coordinates to ensure the image fits within the output without overlapping, and copying the relevant portions into the output.

The following example shows the application of the Paired Ordinal Cutmix on the DHCI dataset.

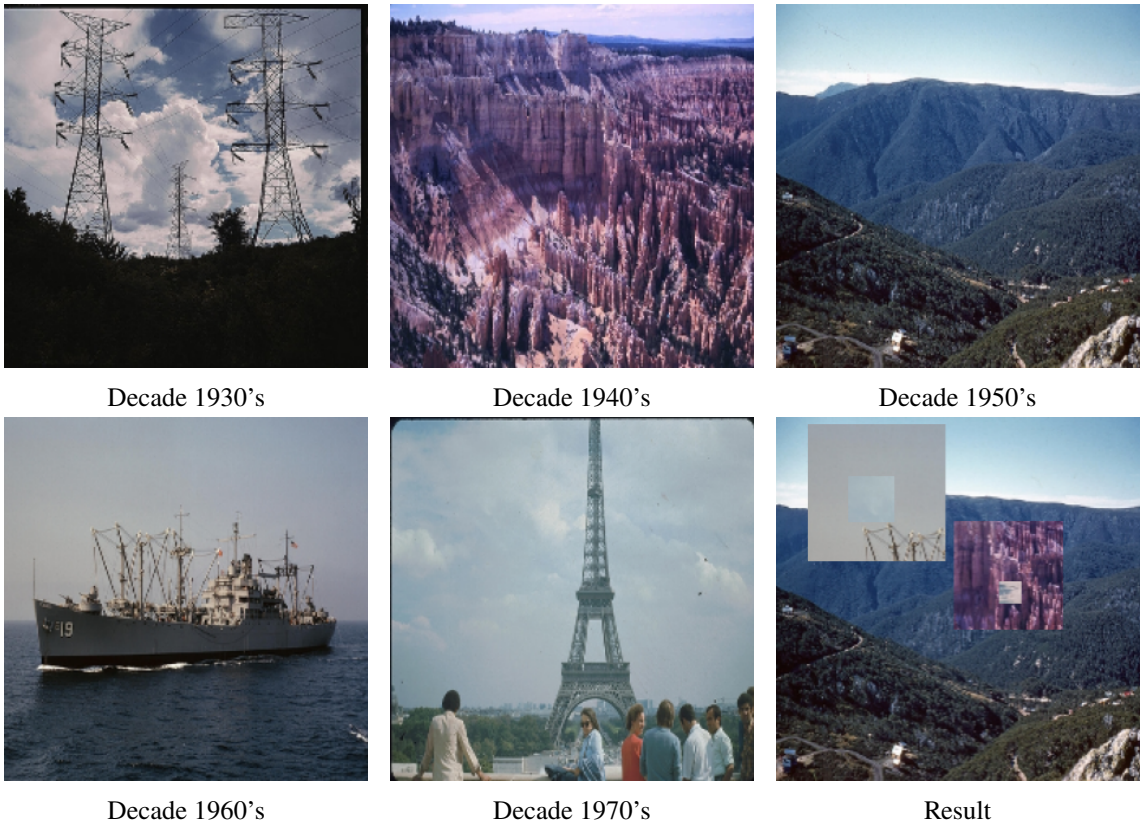


Figure 6: Illustration of a *Paired Ordinal Cutmix* example with $\tau = 0.7$

3.2 Datasets

A comprehensive collection of various ordinal datasets is available in this GitHub repository [10], encompassing both tabular and image datasets. In this work, our focus will be on image datasets, specifically the Pap Smear dataset (SMEAR2005)[11] and the Dating historical color images dataset (DHCI) [12].

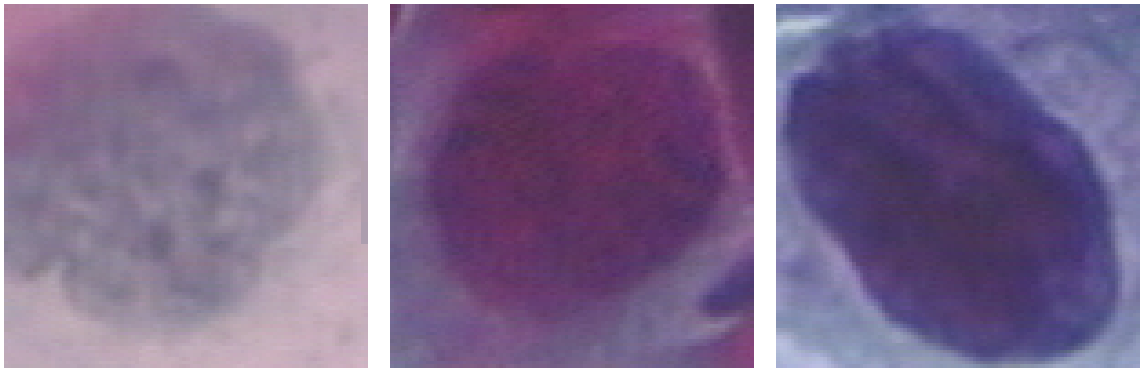
The datasets listed in Table I display considerable variation in terms of their size and number of classes. The SMEAR2005 dataset is a biomedical dataset with a relatively small number of images (917) but includes a higher number of classes (7), making it well-suited for detailed classification tasks in medical research. In contrast, the DHCI dataset is focused on historical color image reconstruction and contains a larger number of images (1325) but has fewer classes (5).

Datasets	Number of images	Classes
SMEAR2005	917	7
DHCI	1325	5

Table 1: Information of the datasets used for this experimentation

3.2.1 Pap Smear (SMEAR2005)

The SMEAR2005 dataset [11] comprises cytology images categorized into seven classes representing various cell types, used for training machine learning models to predict cervical malignancy.



Class 3: Light Dysplastic Class 4: Moderate Dysplastic Class 5: Severe Dysplastic

Figure 7: Illustration of a Example of SMEAR2005 classes

The figure 7 illustrates three classes from the dataset: class 3 represent ligth dysplastic cells, followed by class 4 indicating moderate dysplastic cells, and class 5 representing severe dysplastic cells.These classes are encoded as integers ranging from 0 to 6. Images are loaded from specified directories, with labels assigned based on subfolder names corresponding to different cell types.

Here's a brief explanation of the dataset's classes:

- **Normal superficial:** Normal cells found in the superficial layers of the epithelium.
- **Normal intermediate:** Normal cells located in the intermediate layers of the epithelium.
- **Normal columnar:** Normal columnar cells, typically found in glandular structures or inner linings of organs.
- **Light dysplastic:** Cells showing mild dysplasia, indicating early changes potentially leading to cancer.
- **Moderate dysplastic:** Cells exhibiting moderate dysplasia, a more advanced stage of abnormal growth with increased cancer risk.
- **Severe dysplastic:** Cells showing severe dysplasia, where abnormality is pronounced and cancer risk is imminent.
- **Carcinoma in situ:** Cancer cells present but confined within the epithelium, representing a pre-invasive stage of cancer.

The classes are ordered by severity, emphasizing the importance of using an ordinal data approach to develop accurate machine learning models capable of distinguishing between these ordered targets. Misclassification could have significant implications for patient treatment decisions.

3.2.2 Dating historical color images (DHCI)

The DHCI [12] dataset is a collection of color photographs spanning several decades, specifically from the 1930s to the 1970s. This dataset can be used to train machine learning models to predict the decade in which a photograph was taken.

Class 0: Decade 1930s | Class 2: Decade 1950s | Class 3: Decade 1960s

Figure 8: Illustration of a Example of DHCI classes

The figure 8 illustrates three classes from the dataset: class 0 represent the decade 1930s, followed by class 2 indicating the decade 1950s, and class 3 representing the decade 1960s. These classes are encoded as integers ranging from 0 to 4. Images are loaded from specified directories, with labels assigned based on subfolder names corresponding to different decades.

Below is a brief explanation of the dataset composition:

- **1930s:** This category includes color photographs taken in the 1930s, characterized by early experiments with color processes, often showing technical limitations and faded colors.
- **1940s:** This category includes color photographs taken in the 1940s, characterized by early color processes such as Kodachrome.
- **1950s:** Photographs from the 1950s, often displaying the vibrant colors and unique hues of mid-century color film processes.
- **1960s:** This category includes images taken in the 1960s, a decade known for advancements in color film technology and the introduction of new color processes.
- **1970s:** Photographs from the 1970s, showcasing the evolution of color photography with more refined color processes and better film quality.

The order of the decades represents the chronological progression of color photography technology, making it important to use a chronological data approach to create a machine learning model capable of distinguishing between the different time periods. Incorrect classification could lead

to misinterpretation of the historical context and reduce the model's effectiveness in applications such as historical research or archival organization.

3.3 Experimental Protocol

For training the methods discussed earlier, we utilized a pre-trained ResNet-18 model with ImageNet datasets, leveraging its prior training across a wide range of images to enhance feature extraction capabilities in diverse domains.

We decided to modify the final layer by replacing it with a linear one. This new layer is designed to perform a linear transformation with 512 inputs, corresponding to the features extracted by ResNet-18, and an output space that aligns with the number of classes in the dataset of interest. This adjustment tailors the model specifically for the task of classification.

Each method underwent 200 epochs of training. Additionally, we employed the Adam optimizer, which dynamically adjusts the learning rate based on exponential decay averages of past gradients and squared gradients, effectively adapting to the training needs of deep neural networks. We specifically selected a learning rate of 10^{-4} for layers excluding the final classifier (model.fc),for which 10^{-3} was used, aiming to balance efficiency and precision in gradient descent.

The batch size which determines the number of data samples processed in each forward and backward pass during training was set to 32.

All methods were experimented with in both the original image space and the neural network's latent space (features extracted from ResNet-18 before the classifier). This dual approach allowed us to comprehensively evaluate the methodologies in different representation spaces.

3.4 Evaluation Metrics

To evaluate the performance of the models we use the following metrics:

- **Accuracy:** is the measure of how often a classification model makes correct predictions.It can be calculated as:

$$\text{Accuracy}(y_i, \hat{y}_i) = \frac{1}{N} \sum_{i}^{N} 1(y_i = \hat{y}_i)$$

  where y_i is a tensor of target values, and $\hat{y}_i$ is a tensor of predictions.
- **Mean Absolute Error:** is the average of the absolute differences between actual values and predictions made by a model.It can be calculated as:

$$\text{MAE}(y_i, \hat{y}_i) = \frac{1}{N} \sum_{i}^{N} |y_i - \hat{y}_i|$$

  where y_i is a tensor of target values, and $\hat{y}_i$ is a tensor of predictions.

- **Cohen Kappa:** is a statistical measure that assesses the agreement between two raters by accounting for the agreement occurring by chance. It can be calculated as:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

where p_o is the empirical probability of agreement and p_e is the expected agreement when both annotators assign labels randomly.

3.5 Results

The results are systematically presented in Tables 2 and 3. Table 2 provides a comparative analysis of different methods on the DHCI dataset, while Table 3 offers a comparative analysis of various methods on the SMEAR2005 dataset.

- **Analysis of DHCI**

1. **Baseline Methods:** The baseline methods show moderate performance, with the latent space version slightly outperforming the original space in terms of accuracy (0.5434 vs. 0.5245) and Cohen's Kappa (0.4277 vs. 0.4083). Both have the same MAE of 0.7698.
2. **Mixup:** Mixup in both the original and latent spaces provides a notable improvement over the baseline, particularly in the latent space where accuracy reaches 0.5547 and Cohen's Kappa is 0.4419.
3. **Adjacent Mixup:** Adjacent Mixup does not perform as well as standard Mixup. In the original space, it shows a lower accuracy (0.5132) and Cohen's Kappa (0.3917). In the latent space, it slightly improves in terms of Cohen's Kappa but has higher MAE (0.7472 vs. 0.7208).
4. **Exponential Mixup:** Exponential Mixup shows poor performance in both original and latent spaces, with accuracy dropping to 0.3396 and 0.3623, respectively, and Cohen's Kappa similarly low. The high MAE values indicate significant prediction errors.
5. **Cutmix:** Cutmix in the original space yields the highest accuracy (0.5774) and Cohen's Kappa (0.4719) among all methods, showing that this technique is particularly effective. In the latent space, Cutmix shows slightly lower performance in accuracy (0.5283) and Cohen's Kappa (0.4121).
6. **Sequential and Paired Sequential Cutmix:** Sequential Cutmix in the original space achieves good results with an accuracy of 0.5547 and Cohen's Kappa of 0.4444. In the latent space, the performance is slightly lower. Paired Sequential Cutmix shows moderate performance, with a slight improvement in the latent space for both accuracy and Cohen's Kappa compared to the original space.

Overall, the DHCI dataset experiments highlight the effectiveness of Cutmix, particularly in the original space, and the generally good performance of Mixup methods.

Dataset DHCI			
Methods	**Accuracy**	**MAE**	**CohenKappa**
Baseline (Original Space)	0.5245	0.7698	0.4083
Baseline (Latent Space	0.5434	0.7698	0.4277
Mixup (Original Space)	0.5472	0.7547	0.4343
Mixup (Latent Space)	0.5547	0.7434	0.4419
Adjacent Mixup (Original Space)	0.5132	0.7208	0.3917
Adjacent Mixup (Latent Space)	0.5132	0.7472	0.3919
Exponential Mixup (Original Space)	0.3396	1.1849	0.1657
Exponential Mixup (Latent Space)	0.3623	1.0981	0.1915
Cutmix (Original Space)	0.5774	0.7019	0.4719
Cutmix (Latent Space)	0.5283	0.8075	0.4121
Cutmix: Sequential (Original Space)	0.5547	0.7321	0.4444
Cutmix: Sequential (Latent Space)	0.5509	0.7849	0.4354
Cutmix: Paired Sequentials (Original Space)	0.5208	0.7736	0.4008
Cutmix: Paired Sequentials (Latent Space)	0.5434	0.7434	0.4286

Table 2: Results for the Dataset DHCI

- **Analysis of SMEAR2005**

1. **Baseline Methods:** The baseline methods in both the original and latent spaces show decent performance, with the latent space version outperforming the original space in terms of accuracy (0.6831 vs. 0.6557) and Cohen's Kappa (0.6220 vs. 0.5879). The MAE is significantly lower in the latent space (0.4317 vs. 0.6175).
2. **Mixup:** Mixup significantly improves performance, especially in the original space, where it achieves the highest accuracy (0.7268) and Cohen's Kappa (0.6755). The latent space Mixup also performs well but slightly less effectively than the original space Mixup.
3. **Adjacent Mixup:** Adjacent Mixup shows moderate improvement over the baseline. The latent space version performs slightly better in terms of MAE and Cohen's Kappa compared to the original space.
4. **Exponential Mixup:** Exponential Mixup performs poorly, with the lowest accuracy (0.5738 and 0.6230) and Cohen's Kappa (0.4920 and 0.5534). The high MAE values indicate significant prediction errors.
5. **Cutmix:** Cutmix in the original space yields high accuracy (0.7213) and Cohen's Kappa (0.6682), comparable to Mixup. In the latent space, Cutmix still performs well but shows slightly lower accuracy (0.7104) and Cohen's Kappa (0.6567).
6. **Sequential and Paired Sequential Cutmix:** Sequential Cutmix shows good results, with original space performance being slightly better than in the latent space. Paired Sequential Cutmix performs poorly in the original space but shows significant improvement in the latent space.

Overall, the SMEAR2005 dataset experiments reveal that Mixup and Cutmix methods generally outperform the baseline methods, with Mixup in the original space providing the best results.

Dataset SMEAR2005			
Methods	**Accuracy**	**MAE**	**CohenKappa**
Baseline (Original Space)	0.6557	0.6175	0.5879
Baseline (Latent Space)	0.6831	0.4317	0.6220
Mixup (Original Space)	0.7268	0.4590	0.6755
Mixup (Latent Space)	0.6940	0.5027	0.6379
Adjacent Mixup (Original Space)	0.6667	0.5301	0.6029
Adjacent Mixup (Latent Space)	0.6831	0.4590	0.6228
Exponential Mixup (Original Space)	0.5738	0.6120	0.4920
Exponential Mixup (Latent Space)	0.6230	0.5956	0.5534
Cutmix (Original Space)	0.7213	0.4645	0.6682
Cutmix (Latent Space)	0.7104	0.5683	0.6567
Cutmix: Sequential (Original Space)	0.6667	0.6284	0.6074
Cutmix: Sequential (Latent Space)	0.6612	0.5355	0.5979
Cutmix: Paired Sequentials (Original Space)	0.5027	1.0055	0.4226
Cutmix: Paired Sequentials (Latent Space)	0.6393	0.5683	0.5745

Table 3: Results for the Dataset SMEAR2005

4 Conclusion

The results obtained indicate that the new probability function performed worse in exponential mixup, showing lower precision and Cohen's Kappa values compared to traditional approaches.

In contrast, applying the new probability function to ordinal cutmix methods showed significant improvements. The Ordinal Cutmix methods, especially in Sequential and Paired variations, demonstrated superior performance in terms of both precision and Cohen's Kappa, in both the original and latent spaces.

Furthermore, ensuring that at least one pixel from each image is present, even when the selection probability is low, can enhance the representativeness of the generated images, ensuring a minimal presence of both original images.

These results suggest that while the new probability function may not be ideal for all augmentation methods, it has significant potential when applied appropriately.

References

[1] Hongyi Zhang et al. "mixup: Beyond Empirical Risk Minimization". In: *International Conference on Learning Representations*. 2018.

[2] Sangdoo Yun et al. "CutMix: Regularization strategy to train strong classifiers with localizable features". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6023–6032.

[3] Airton M. A. F. Tiago. "Data Augmentation for Ordinal Data". Master's thesis. University of Porto, 2024. URL: https://repositorio-aberto.up.pt/handle/10216/157714.

[4] Beatriz Sá. *Ordinal Cutmix*. 2024. URL: https://github.com/beatrizmsa/ordinal_augmentation.

[5] Haseeb Hassan et al. "Review and classification of AI-enabled COVID-19 CT imaging models based on computer vision tasks". In: *Computers in biology and medicine* 141 (2022), p. 105123.

[6] Mingle Xu et al. "A comprehensive survey of image augmentation techniques for deep learning". In: *Pattern Recognition* 137 (2023), p. 109347.

[7] Terrance DeVries and Graham W Taylor. "Improved regularization of convolutional neural networks with cutout". In: *arXiv preprint arXiv:1708.04552* (2017).

[8] Xiaofeng Liu et al. "Unimodal regularized neuron stick-breaking for ordinal classification". In: *Neurocomputing* 388 (2020), pp. 34–44.

[9] Christopher Beckham and Christopher Pal. "Unimodal probability distributions for deep ordinal classification". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 411–419.

[10] Tome Albuquerque. *Ordinal Datasets*. Acessado: March-June 2024. 2024. URL: https://github.com/tomealbuquerque/ordinal-datasets.

[11] George Dounias et al. "Automated identification of cancerous smears using various competitive intelligent techniques". In: *Oncology reports* 15.4 (2006), pp. 1001–1006.

[12] Frank Palermo, James Hays, and Alexei A. Efros. "Dating Historical Color Images". In: *ECCV (6)*. 2012, pp. 499–512.