Date of publication SEP-30, 2024, date of current version AUG-13, 2024.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.19.3.XX

Computer Modelling of Textures on Images with Human Skin Wound Areas

BOHDAN LUKASHCHUK
Ukrainian National Forestry University, Ukraine, Lviv, 103, Gen. Chuprynky St., 79057
Corresponding author: Bohdan Lukashchuk (e-mail: bohdan.lukashchuk@gmail.com).

ABSTRACT In this paper, we concentrate on the images of the wounds on the human skin and propose to consider each image as a set of smaller pieces – crops or patches containing different textures. We overview, develop and compare deep learning feature extraction methods to model image crops as 200-dimensional feature vectors using various artificial neural network architectures: convolutional autoencoders, variational convolutional autoencoders, and Siamese convolutional networks trained in the contrastive learning manner. Also, we develop a custom convolutional encoder and decoder, use them in the aforementioned architectures and compare them with the ResNet encoder and decoder alternatives. Finally, we train and evaluate k-nearest neighbors and Multi-Layer Perceptron classifiers on the features extracted with the model above options to discriminate skin, wound, and background image patches. Classification evaluation results on the features, extracted with the Siamese network, show the best test accuracy for all implementations without a significant shift between model versions (accuracy > 93%); variational autoencoders show random results for all options (accuracy around 33%), and convolutional autoencoders reached good results (accuracy > 77%) but with a noticeable difference between the custom and ResNet versions; the latter is better. Custom encoder and decoder implementations are faster and smaller than the ResNet alternatives but may be less stable on larger datasets, which still needs investigation. Possible applications of the feature vectors include an area of interest extraction during wound segmentation or classification and usage as patch embeddings while training vision transformer architectures.

KEYWORDS wound segmentation, wound classification, autoencoder, variational autoencoder, Siamese networks, contrastive learning, image feature extraction

I. INTRODUCTION

ARTIFICIAL neural networks (ANN) are commonly and widely used to solve classification, object detection, semantic segmentation, and other tasks in the different areas of the medical imaging domain [1-3]. In this research, we concentrate on the analysis of images of wounds on human skin.

Automatic wound analysis is essential in managing chronic wounds, such as diabetic ulcers, pressure sores, and venous leg ulcers. These conditions require regular monitoring to assess healing progress and adjust treatment plans accordingly. Automated systems can offer consistent and objective evaluations, reducing the errors, associated with manual assessments [4, 5]. Another practical condition for automatic wound detection is in post-surgical care. Automated systems can help monitor surgical sites for signs of infection or any other dangerous condition, providing early warnings that can lead to timely medical interventions. This is particularly beneficial when patients need to manage their recovery at home [5].

As a rule, wound images are quite complex - they contain many different types of objects, complex backgrounds, and skin on various body parts. Thus, researchers apply and evaluate their classification or segmentation methods on some human skin tissues, very commonly - chronic diabetic foot ulcer wounds [3, 6, 7]. This tissue domain is widely approached because the other kinds of wounds are not essential or classification/segmentation tasks are perfectly solved, but because the foot ulcer wounds dataset [3] is publicly available. Datasets for the other types of human skin wounds are primarily private, too small, or have poor quality.

The other important factor, except lack of data, is that most methods rely on supervised training. It brings the need for images to be annotated, either with class labels, object boundaries, or segmentation masks, depending on the task.

All these lead to the problem of lack of generalization of the trained models. For example, in our preliminary experiments conducted during the preparation of this study, the U-Net [2] segmentation model, trained on the foot ulcer images [3], reached a Dice score > 90% on the test dataset but was

ineffective during inference on the dataset used in "Implementation features of wounds visual comparison subsystem" [8] paper.

From the human perspective, wounds on the skin are recognizable. This research aims to discover ways to improve ANNs ability to generalize and detect wound and skin textures even on untypical data. To approach it, we suggest considering the large and complex image as a set of much smaller images, each representing some important texture information that can be extracted and used in semi-supervised classification tasks. A slightly similar approach, related to the burn tissues, is developed in [9]. In contrast with the abovementioned research, we perform feature extraction using deep-learning methods and not statistical approaches. We overview and develop different methods based on convolutional autoencoders, variational convolutional autoencoders, and Siamese networks and train them to convert each of these image crops to 200-dimensional vector-embedding that represent discriminative features of the particular crop and can be used in classification or clustering.

In this research, because of the limitation of the data available, we work with the three categories: skin, wound, and background textures, but the results can be extrapolated on the more significant number of classes, depending on the data available.

Training effective models that are able to extract feature vectors from the small parts of the image is the crucial step for further skin research, highlighting the contours of the wounds, analysis, and prediction of wound healing over time with or without different medications or other type of analysis.

Fig.1 below shows the skin/non-skin classification results with the simple multilayer perceptron based on the embedding vectors.

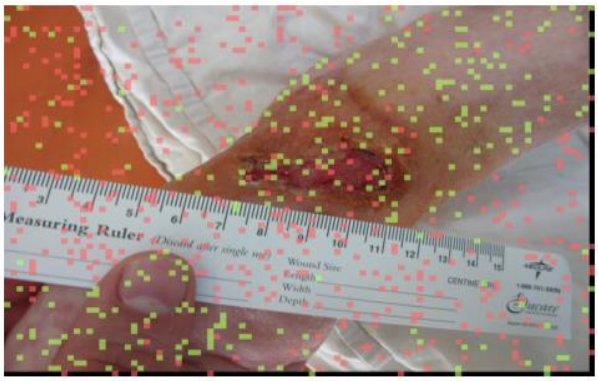

Figure 1. Classification of image areas as skin or not-skin. Green represents areas classified as skin, while red denotes areas classified as non-skin.

In this particular example, an image with a size of 3456 by 4608 pixels (in height and width, respectively) is submitted as input. After that, the image is divided by a uniform square grid with a step of 50 pixels. 6510 grid cells (segments) were obtained (70 in height and 93 in width). From the obtained data, 10 percent of the segments were randomly uniformly selected. After that, each pixel is classified as belonging to human skin or not. Those pixels classified by the method as skin are highlighted in green and vice versa in red. We can see that the classification is correct in most cases. It is also worth paying attention to the part of the image where the finger presses the ruler, it clearly shows how the pixels of the finger is classified as skin, and the pixels of the ruler are classified as not skin. Defects also occur where there is a transition between skin and non-skin, and the pixels of the segment is placed on the boundary of the separation or where the skin's texture is damaged or under atypical lighting; for example, these defects are visible in the wound area. A good result is also shown in the area in the upper left corner of the image, where the background color is exceptionally close to possible human skin color. Still, only a small number of segments are misclassified. This and other results are possible because of modeling not the specific and complex features on the images but rather more general-like textures. In this example, the convolutional neural network model, trained to extract embeddings from the 35x35 images, was used to extract semantic vectors and, later, with a dense neural network - discriminate vectors for the 50 by 50 images. While texture classification can show some excellent visual results, the main point is in the efficient semantic vector modeling of the skin/wound/background/mixed image crops, which can lead to the feature vectors, which are distributed in their high-dimensional space in an expected way, where skin vectors are close to each other, and others are further from them.

The overall structure of the paper is as follows:

- II Literature overview section reveals information about current approaches to image feature extraction and our motivation to use autoencoders, variational autoencoders, and Siamese networks for this task, as well as some basic descriptions of these architectures
- III Materials and methods contain information about data we use, calculations we performed to develop custom models' structure, and description of the experiments we conducted. Finally, it shows the methods we used for performance evaluation

The last sections contain information on the experiment results and conclusions on application prospects.

II. LITERATURE OVERVIEW

An image can be easily treated as a flat vector by flattening its rows into one. However, this brings more disadvantages than advantages as the resulting vector is very long, so computationally expensive, and if we try to flatten images, we will face the problem of translation invariance; also, we will not get any semantics at all, at least any semantics generalization. A statistical approach can also be used, like in the previously mentioned research on burn images, which uses a combination of statistics of the grey-level histogram, Haralick texture features, and mean intensity values of color spaces [9]. This approach may struggle to capture features from various skin tones, particularly under different lighting conditions, and

may overlook small texture details and boundaries, such as the edges of wounds, that define different regions. Therefore, we propose using a trainable ANN approach. The outputs from the activation layers of convolutional neural networks can be treated as image semantic vectors. Depending on the network architecture, depth, and training data, these vectors may represent simple edges or more complex structures, such as wound boundaries or even entire object classes. By combining the outputs of different activations, the neural network learns a general representation of the data. We recommend extracting semantics from the simple yet sufficiently general parts of human skin in the image, which can provide information about color and texture without relying on highly complex or specific features. Thus, convolutional neural networks are an effective option for creating semantic vectors.

Recent studies in the deep learning field [10-12] show that feature vectors, or as we also call them in this paper - semantic vectors from the images or image parts, can be used in the unsupervised or semi-supervised manner to solve the wide variety of tasks.

The promising neural network architecture option emerges from anomaly detection, data compression, and noise reduction tasks. Different research [13-17] in this area uses neural network architecture called autoencoder. An autoencoder is an artificial neural network trained to copy input data to output data. Inside, a hidden layer h describes the hidden code used to represent the input data. The network consists of two parts: the encoder function and the decoder that creates the reconstruction [37]. In the classical autoencoder, the hidden layer has a dimension smaller than the input and output layers. This property is also used for data compression [14, 15].

Autoencoders can work with different types of data in different domains, but this study concentrates on medical images; thus, its main focus is on convolutional autoencoders, where dense layers are replaced with convolution layers.

When training an autoencoder, loss functions are used. They determine how much the reproduced object differs from the original x. In this case, we consider the task of working with images, so the convolutional autoencoder is used. Commonly used loss functions include Mean Square Error (1) and Binary Cross Entropy (2), which is a partial variant of Cross Entropy:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x}_i)^2, \quad (1)$$

$$BCE = \frac{1}{n}\sum_{i=1}^{n} x_i \log(\bar{x}_i) + (1 - x_i)\log(1 - \bar{x}_i), \quad (2)$$

where n is the number of images for which the error is calculated, x_i is the original image, and $\bar{x}_i$ is the reproduced one.

Even though some studies claim [18] that MSE gives better results on small-dimensional images and BSE on large-dimensional images, they are often used interchangeably.

The hidden layer output h of the encoder is the semantic vector we seek.

However, classic autoencoders have the disadvantage of generalizing poorly regarding things different from what they were trained on. The authors of [19] show this problem in more detail. They consider different types of autoencoders from the point of view of the properties of their hidden layers. They trained the autoencoder model on a large data set - a set of handwritten digits MNIST [20], saved the vectors obtained at the output of the h=f(x) encoder, and then reduced the dimensionality of the vectors to 2 using the method of principal components [21] and visualized the obtained points on a plane. Results show that the points are divided into clusters corresponding to different numbers, which is logical and desirable. However, the clusters are elongated and have a substantial spread of values. In addition, it is possible to trace a significant drop in the density — a 'gorge' at the boundaries of the cluster separation, which may indicate that when an atypical digit that was not in the training set appears, the reproduction by the classic autoencoder may be of poor quality. However, the authors also consider another neural network architecture — a variational autoencoder, in which these problems are minimized.

The variational autoencoder [22, 23], unlike the classical one, belongs to the class of generative neural networks. While discriminative modeling aims to train a predictor based on previously collected observations, generative modeling aims to solve the more general problem of learning the joint distribution of data [23]. Thus, generative models allow making assumptions about how data is formed. We can study the joint distribution of different skin textures and colors as 'skin texture modeling,' which also provides the ability to achieve a more general semantic vector representation.

Differences in the architecture are manifested in its center; if, in the classic autoencoder, the middle of the model is a vector of a small (smaller than the input vector) dimension, then in the variational one, it is somewhat different. The first part is the encoder, usually called the recognition model, and the decoder is called the generative model. The encoder can be written as q(z|x), where x is the input vector, z is the hidden middle layer vector, and the decoder is p(x|z)p(z). Since this model is generative, generation must be present somewhere. It happens precisely in the center of the model. Unlike a classical encoder that learns to find a reduced representation of the input data, in a variational autoencoder, the recognition model tries to approximate the input data distribution during training. Since it is impossible to approximate something for which no information is available, an assumption is traditionally made about the distribution, such as that it is normal. In the variational autoencoder, the hidden vector represents the parameters of the assumed normal distribution—specifically, the mean ("*mu*") and the standard deviation ("*std*"). These parameters are used to sample a new vector from this distribution, which is then used to reconstruct the image. Then a vector z is randomly selected from a normal distribution with

the aforementioned parameters. After that, the decoder reproduces the image from the selected vector z. Thus, considering the decoder as a product of probabilities p(x|z)p(z), we see that it reflects the probability of the simultaneous occurrence of two events p(zx) — the sampling of the vector z from the normal distribution with parameters mu and std, and the generation of the original image x by the decoder afterward.

The loss function ELBO (Evidence Lower Bound) [23] is essential, which, on the one hand, should minimize the error between the input and the generated image and, on the other hand, bring the distribution parameters mu and std closer to the parameters of the standard normal distribution.

$$ELBO = E_{q(Z|X)}[lnp(x|z)] - D_{KL}[q(x|z)||p(z)], \quad (3)$$

where the first part of $\ln[p(x|z)]$ is the likelihood function responsible for the correctness of the reconstruction and D_{KL} is the Kullback-Leibler divergence (KL-Divergence), which is also called the relative entropy, and this term of the loss function is responsible for ensuring that the distributions $q(x|z)$ (posterior probability) and $p(z)$ (prior probability) have the most similar parameters. This way, we get an artificial neural network trained to obtain the normal distribution parameters for the input image and then generate the output. Having the mu and std hidden parameters of the variational autoencoder network trained on the skin texture images, we can sample semantic vectors from the normal distribution with the aforementioned trained parameters that generalize the skin.

The third auspicious approach originates from the recent studies of deep metric learning. Metric learning aims to measure the similarity among samples while using an optimal distance metric for learning tasks [24]. The main goal of this approach is to bring similar objects closer and increase the distance between the different objects. In recent years ideas of deep metric learning have emerged in unsupervised learning for efficient semantic vector generation in computer vision and not only fields. [3, 10, 12, 25-27]. Contrastive learning is one of the most used and well-performing techniques in deep metric learning. The authors of SimCLR (A Simple Framework for Contrastive Learning of Visual Representations) [28] show how to use it to generate semantic vectors of images with a limited amount of data and in the way that similar embeddings will be and different will have large distances. SimCLR uses Siamese network architecture. It consists of twin networks that accept distinct inputs but are joined by an energy function at the top [29]. The networks share the same structure and hyperparameters, so in practice, one instance is used but fed with two different images during inference. The idea of training Siamese network in SimCLR is following:

Use two images from different classes as anchor images. If classes are unknown – sample two different images from the dataset.

For each anchor image, create two positive samples by applying different transformations on the anchor image like random cropping, random color distortion - a technique where the hue, saturation, brightness, and contrast of an image are randomly altered.

Use convolutional neural network (CNN) to calculate embeddings for each of the positive samples

Calculate the loss function between all of the four positive sample embeddings in the way that two positive sample embeddings from the one anchor image should be close to each other and far from the other anchor image positive samples in terms of the loss function

SimCLR uses a version of the contrastive loss that is known as Normalized Temperature-Scaled Cross-Entropy (NT-Xent) loss (4) as the error function:

$$l_{i,j} = -log \frac{\exp\left(\frac{sim(z_i,z_j)}{\tau}\right)}{\sum_{k=1}^{2N} \mathbb{I}_{k\neq i} \exp\left(\frac{sim(z_i,z_j)}{\tau}\right)}, \quad (4)$$

Where i and j denote positive pair of samples, z_i, z_j – embeddings of the corresponding vectors, $sim(z_i, z_j)$ – cosine similarity between the embeddings, $\mathbb{I}_{k\neq i} \in \{0,1\}$ is and indicator function, evaluating to 1 if $k \neq i$ and τ is a temperature parameter, N is a size of a randomly sampled minibatch of examples from the dataset. The same as for the one image example, two positive samples are created, and we have 2N embeddings.

With the trained network, one can create texture embeddings so that skin textures will be close to the other skin textures and other parts of the image will be far away in terms of the contrastive loss function and cosine similarity.

III. MATERIALS AND METHODS

The first part of this section describes the data and data extraction methods used for the experiments. The second describes the models we created and the main steps of setting up an experiment to compare the three aforementioned semantic vector generation methods applied and adapted to the skin wound and background texture images. Also, it contains an overview of the method to perform a visual comparison of the created vectors. The third, finally, describes the training and evaluation stages of k-nearest neighbors (KNN) and Multi-Layer Perceptron (MLP) classifiers to discriminate semantic vectors into skin, wound, and background classes. While we are not intended for simple texture classification, this is a straightforward way to evaluate the ability of models to create efficient semantic vectors.

A. THE DATA

In the previous sections, we repeatedly mentioned that data is crucial for good-performing machine learning models. If to be precise – data of good quality and in abundance. Also, we said there is a data quality and quantity problem in the medical image segmentation field. Thus, the decision was to study how to generalize some low-dimensional data.

The SFA dataset [30] contains skin images ranging from 1 to 35 pixels. We suggested using the largest 35px available images for the test. A total of 3354 skin images and 5590 non-skin samples are available in the set.

Wound images are not available in the aforementioned dataset, so to extract them, we use the ulcer wound segmentation dataset created by the authors of The Foot Ulcer Segmentation Challenge [3]. The overall wound data extraction is achieved in a few steps:

- apply segmentation mask on the image with foot ulcer tissue to retrieve wound image only
- find the center of the extracted wound image
- crop rectangle (35x35 px) from the center
- cut off black sides if present
- save the extracted rectangle as a wound texture image if not more than 50% black color.

We applied this method on 1210 images from the validation and train datasets of The FUSC [3] dataset. As a result of the extraction method described before, we received 530 wound texture images. Many wounds from the aforementioned FUSC dataset contain small and low-resolution images unsuitable for wound texture extraction.

Below, in Figure 2, there is an example of some images from the dataset. The first row contains skin images, the second - non-skin (possible backgrounds), third – extracted wound images.

Figure 2: Enlarged examples of skin, background and wound images

Although they have been enlarged for clarity, there are different skin textures, colors, and shades, so this set can competently represent a variety of skin. Moreover, background images also correspond to a great variety of non-skin textures. The problem with the wound texture images is that they sometimes contain large areas of black pixels. This issue arises from the low resolution or cropping process of the images in the FUSC dataset. These black regions can potentially mislead the neural network, causing it to mistakenly identify black or black-red edges as wound areas. Unfortunately, this is the only way to create the desired dataset for now. In the future, unsupervised segmentation, combined with this paper's semantic vector generation and classification method, may be used to create a better wound texture dataset from the unlabeled data.

Finally, we shuffled equal amounts of wound, skin, and background images and performed train/validation/test split of the data (preserving an equal amount of each class representative in the datasets):

- 1113 train images
- 318 validation images
- 159 test images

B. EMBEDDING GENERATION METHOD

The task is to define models to generate semantic vectors and the method of their comparison using the datasets described above.

Three different methods of image semantic vectors are used and compared:

Convolutional autoencoder [31] with the MSE (1) loss;

Variational convolutional autoencoder [23] with ELBO (3) loss;

Siamese network [29], inspired by the SimCLR framework [28] with the contrastive loss (4).

Each of the mentioned models shares the same convolutional encoder part. Different modifications of the ResNet [32] encoder are commonly used while experimenting with autoencoders or in SimCLR. Despite this, our initial guess is that ResNet is too complex for this type of task. For the experiment, we created custom architecture that is simple enough but also fulfills all needs of the experiment. Two critical factors are to be considered: the receptive field of the inner dense layer neurons and the hidden vector dimension.

The empirical suggestion is that the receptive field should be at least the size of the initial training data. While the pooling operation increases both theoretical and practical receptive field in a good way and the stacking of layers also increases it to some extent [33], they make the model more complex, moreover, decrease the dimensionality primarily of the already small input. Authors of [34] show that sequential usage of the dilated convolutions results in the exponential expansion of the receptive field size. According to [34] receptive field can be calculated recursively using (5) or in an iterative manner with (6):

$$r_{l-1} = s_l * r_l + (k_l - s_l) \qquad (5)$$

where l is a layer index, r – receptive field, s – stride, k - kernel size

$$r_0 = \sum_{l=1}^{L}((k_l - 1)\prod_{i=1}^{l-1} s_i) + 1 \qquad (6)$$

where L is the number of layers r_0 – receptive field of the final layer neuron.

If dilated convolutions are used in the layer, the value of kernel size k is calculated as follows (7):

$$k = \alpha(k - 1) + 1 \qquad (7)$$

where α is a dilation factor.

The other significant factor to consider is the area covered by each embedding vector neuron. (8), (9) [35] are used to calculate leftmost u_0 and rightmost v_0 indexes of the original image, which that are covered by the theoretical receptive field.

$$u_0 = u_L \prod_{i=1}^{L} s_i - \sum_{l=1}^{L} p_l \prod_{i=1}^{l-1} s_i \quad (8)$$

$$v_0 = v_L \prod_{i=1}^{L} s_i - \sum_{l=1}^{L} (1 + p_l - k_l) \prod_{i=1}^{l-1} s_i \quad (9)$$

The hidden layer dimension should be large enough to hold the semantic information of the texture and color but must not be just a flattened copy of the initial data. Also,

the size of the vector must be large enough for use in the statistical measures and not very computationally expensive for real-time evaluation using different metrics.

Based on these prerequisites and after calculating the theoretical receptive field value for the different architecture combinations, the suggested custom encoder and decoder architectures (Fig. 3, Fig. 4).

Each rectangle corresponds to the network layer. For example: ` 5x5 conv., 8, s=1, p=0, d=1 ` means 5 by 5 kernel size, 8 output channels, stride=1, padding=0, and dilation=1. The arrow after the layer shows the output dimensions of the transformed input.

This model takes a 35 by 35 by 3 image and produces 200-dimensional vector as a flattened output.

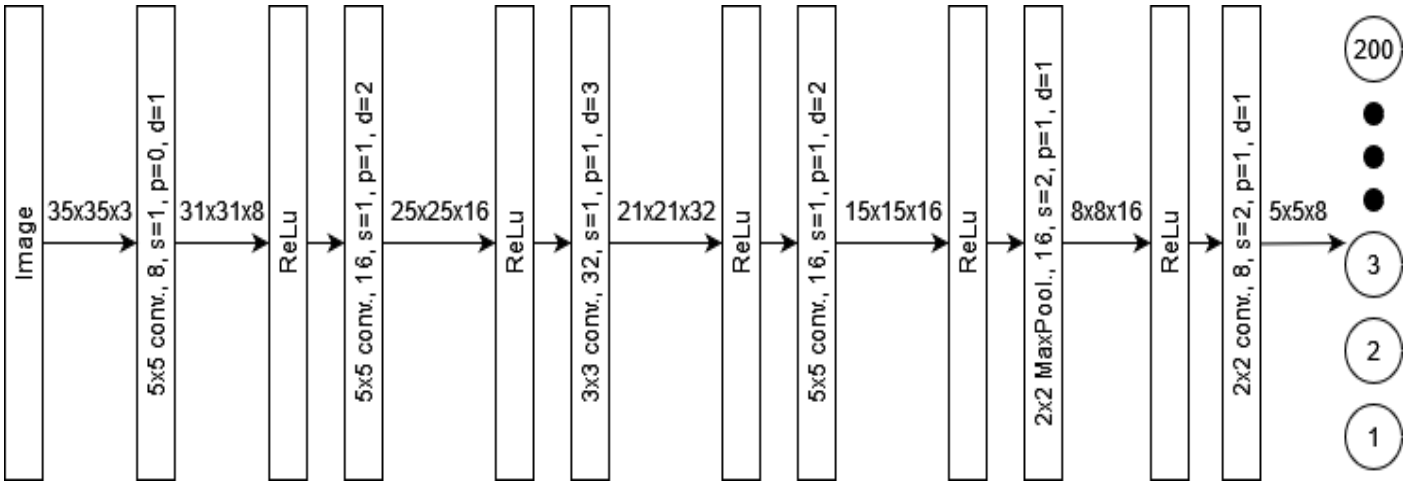

Figure 3 Custom encoder network architecture

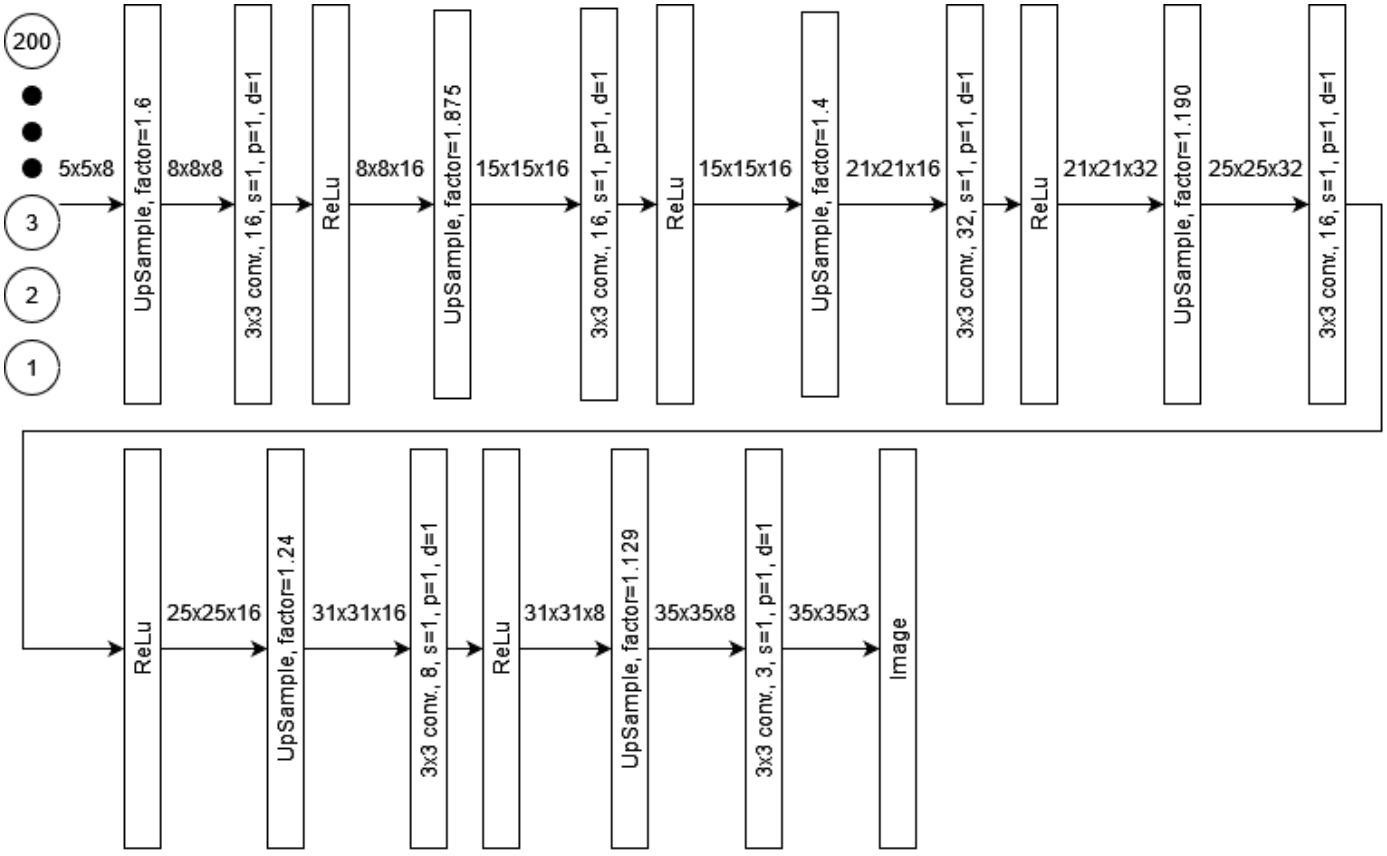

Figure 4 Custom decoder network architecture

Decoder architecture is opposite to the encoder. However, instead of transpose convolutions, which may cause a checkboard pattern effect [36], we use upsampling based on the nearest-neighbors approach with a learnable convolution layer afterward. Because of this, the decoder contains more layers than the encoder. Convolutions are not used to decrease input width and height, and they are configured in a way that does not change it; instead, they introduce learnable parameters – filters. This way, we try to go around the checkboard pattern effect but still have a learnable decoder network that reconstructs the image.

The receptive field of each value of the last convolutional layer output is 29 by 29 pixels, which is just slightly smaller than the original image size, and as shown in Fig. 5, it covers the whole input.

Axis represents the input image (brown skin rectangle in the middle) dimensions, red rectangle – area, which affects the value of the top-left output value of the final convolutional layer, blue – central, and green – bottom right. This shows that the whole image is covered by the final layer outputs and that every value from the output vector holds information about a large area of the input image.

Based on this encoder, three architectures are implemented for the experiment on the Fig. 6.

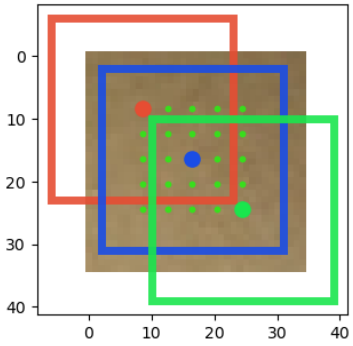

Figure 5. Receptive field of the output layer of the custom encoder network

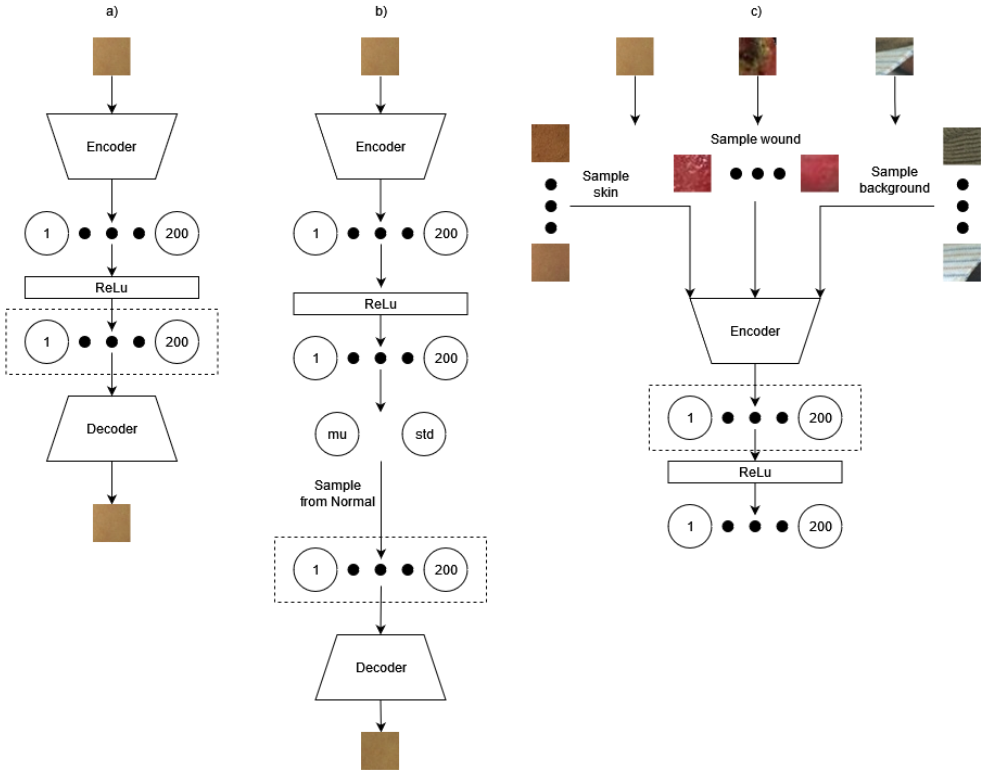

Figure 6. a) autoencoder; b) variational autoencoder; c) Siamese network; architectures

Outputs of the layers in dashed rectangles are used as semantic vectors. It is important to note that in the first two architecture options, we use a combination of linear and non-linear layers before the semantic embedding layer, this is because these options result in better embedding generations, according to my preliminary experiments, but We do not use the fully connected layer in the last CLR option. This is because in [28] the authors show that, for the SimCLR framework, the flattened layers before the non-linear projection head perform more than 10% better. That is why, we make this exception for the *c* network. Another critical difference is that while working with the ResNet model encoder (more on this in the later paragraphs), I use a linear reduction head, which decreases the encoder flattened output from the 512 dimensions to the 200.

The training of the autoencoder – model *a* is as follows:

- pass batches of training images through the encoder.
- reconstruct them back through the decoder.
- compare reconstructed with the original ones using MSE loss.

By doing this, we train the network to create hidden representations of the skin texture images and reconstruct the original images from these representations. Testing and validation work in a similar manner. After training, only the encoder is used for further experiments to produce vector

embeddings of the input image.

The training of the variational autoencoder – model *b* is slightly different. Image batches are passed in the same manner. However, in the middle of the network, after the encoder, 200-dimensional embedding that comes after one non-linear and one linear layer after flattening the image is reduced to the 2 nodes that represent parameters of the normal distribution. Then, the new 200-dimensional vector is sampled from the normal distribution that uses these two parameters, and the image is reconstructed from this vector in the decoder network. ELBO is used as a loss function. The model learns to generate semantic vectors of the skin texture. In opposite to the standard autoencoder, where embeddings from the input image are considered semantic vectors, in variational autoencoder, sampled from the Normal distribution, with the produced by the network parameters, vectors are the semantic vectors that are being sought. The motivation is that the generative model generalizes skin texture better than the discriminative. Important to note that two configurations of the ELBO loss function are tested: one with a KL-Divergence multiplier equal to 1 and one with a KL-Divergence multiplier being 0.1, which penalizes the distribution part of the loss function and, in theory, may result in better similarity of the sampled textures and training ones.

Model *c* is trained and evaluated differently. It has no decoder; only the encoder part is used.

For each training iteration, one image representing skin, background, and wound is selected from the batch. For each type of image, *n* skin (positive samples), *n* background (negative samples), and *n* wound (negative samples) images are randomly selected from the training data, with *n* being a fixed number. The original skin, background, and wound texture images are referred to as the anchor, while the additional selected images serve as positive and negative samples.

In contrast to the SimCLR framework, where image transformations are used to create negative and positive samples, this experiment employs a supervised approach.

The positive and negative images are passed through the network, and a contrastive loss with cosine similarity is used to calculate the difference between the pairs of produced embeddings. The loss function is designed to bring positive samples closer together and push negative samples further apart.

The temperature parameter, a crucial constant in the contrastive loss formula, plays a significant role. According to [37], a lower temperature parameter increases the penalization of complex negative samples, resulting in a more typical distribution. Based on this, we chose to train two CLR models: one with a temperature parameter of 0.07 and another with 0.2, as these values are referenced in the literature.

Training of two configurations of these three models is performed. The first is with the custom encoder and decoder, and the second is with the ResNet18 encoder and decoder. 10 models, considering different encoders, model types, and model parameters, are trained and evaluated in total, not considering some preliminary training aimed at defining architectures and some initial parameters. Comparison is performed in terms of test loss function values, training speed, evaluation speed, and model weight.

The next comparison stage is the visual evaluation of the created vector embeddings. Each model option is used to create embeddings from test dataset samples. 200-dimensional embeddings are reduced to the 2-dimensional vectors using Uniform Manifold Approximation and Projection (UMAP) [38] dimensionality reduction method and visualized on the 2D plane. UMAP is known for its efficiency on large amounts of data and its ability to preserve local connections between the data, which is very important. The goal is to determine the model that creates the best dense clusters from the data of one type that are also easily separable from each other. UMAP accepts hyperparameters, and after a set of experiments, we selected n_neighbors=15, min_dist=0.1. The other important hyperparameter is *metric*, which calculates the distance between the points during dimensionality reduction. We use two options: *cosine* similarity for the autoencoder and Siamese model-generated vectors and *KL Divergence* for the variational autoencoder results. More about metrics in the Results section.

The third stage includes applying two classification models on wound/skin/background embeddings generated by the embedding generation models. This stage quantitatively assesses how easy it is to discriminate the created vectors. We choose two different methods: k-nearest neighbors (KNN) and multi-layer perceptron (MLP). According to our intuition, KNN should work better for the models that generate vectors grouped into clusters with a considerable distance between them. MLP can model complex function that divides them. For the KNN, we decided to use *15 nearest neighbors* (as in the dimensionality reduction step) and apply weights on them, where the closest points have the largest weight. For the *metric* parameter, we use the same approach as for the dimensionality reduction. The MLP model consists of 4 hidden fully connected layers with 256, 128, 64, and 32 artificial neurons, respectively. Input – 200-dimensional vector, output – three neurons that correspond to one of the seeking classes. *Activation functions* for the hidden layers – *ReLu*, *L2 regularization* with *alpha=0.1* to prevent overfitting and *constant learning rate = 0.001*. *Optimizer – Adam.* As data is evenly distributed between classes', we use *accuracy* as an evaluation metric. We combine test and validation datasets into one and use a total of *477 images for evaluation and 1113 for training*.

After training and evaluating these classification models, we could use a simple method shown in Figure 1 in the introduction section to see how effective our classification is on the completely new and untypical wound image. As a reminder, the method is shown in Fig. 7.

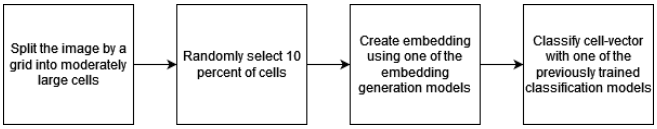

Figure 7. Simple embedding application

This method, for sure, does not provide a way of effective segmentation; here, we mainly use it to visually determine the effectiveness of vector generation from the image crops that are larger than 35x35 pixels. It gives a basic intuition if a specific vector generation model produces semantic vectors that can be used in more complex skin and wound analysis methods.

IV. RESULTS AND DISCUSSION

We start with the results of the vector generation models' parameters, training, and evaluation. Next, we show how skin, wound, and background embeddings created with the models above are distributed on the plane and make initial conclusions.

And finally, we analyze the classification results to rely on some metrics while making the conclusions.

Table 1 contains the results of the model training.

Table 1. Model comparison

Model param. / Model name	Total parameters	Model size (MB)	Average inference time (ms)	Test MSE loss	Test NT-Xent loss	Test ELBO loss
Custom CAE	102200	0.66	1.12±0.3	0.0119	-	-
ResNet CAE	11311632	53.71	5.72±1.06	0.01504	-	-
Custom VAE 1	182600	0.98	2.12±0.69	0.05409	-	0.09106
ResNet VAE 1	11392032	54.03	6.66±1.84	0.05424	-	0.06788
Custom VAE 0.1	182600	0.98	2.23±0.9	0.05438	-	0.05513
ResNet VAE 0.1	11392032	54.03	6.63±1.44	0.05449	-	0.05484
Custom CLR 0.07	62000	0.51	1.21±0.63	-	1.492	-
ResNet CLR 0.07	11271432	53.56	6.18±1.33	-	1.159	-
Custom CLR 0.2	62000	0.51	1.12±0.5	-	0.89	-
ResNet CLR 0.2	11271432	53.56	6.03±1.17	-	0.8369	-

The experiments were conducted using an Nvidia GeForce RTX 2060 GPU, equipped with 6GB of GDDR6 memory, to ensure efficient training and inference of the models. We use Adam optimizer with constant learning rate=0.0001 and batch size 128 for 1000 epochs.

The first word in the model's name denotes the model type: Custom or ResNet 18. Second model architecture: convolutional autoencoder (CAE), variational autoencoder (VAE), or contrastive learning encoder (CLR). Third word (number) – the value of some specific for the model loss function multiplier: KL divergence weight for VAE architectures and temperature parameter for the CLR models. A detailed description of each model is the in the previous section.

We can see from Table 1 above that Custom models are much smaller in terms of the number of parameters. Custom models have roughly 100 times fewer parameters than their ResNet equivalents for CAE architecture, with 60 times fewer parameters for VAE and 180 for CLR models. Custom models are less than 1 MB large, and ResNet options all have about 54 MB – roughly 50 larger. In terms of inference time, all models are high-speed (range 1-8 ms), and the difference is not so significant as Custom models are 2-3 times faster. Important to notice that models' inference time is affected by the hardware but is still relevant to analyze their performance in terms of time efficiency relative to each other. We used an average of 10000 evaluations on one sample to calculate inference time for every configuration. One may say that the small model size's drawback is its performance, but the evaluation metrics from the test data show that this statement is not precisely correct. For the CLR models, the smallest loss function value is achieved by ResNet 0.2, but it is much smaller than the Custom 0.2 model result (0.8369 and 0.89, respectively).

Regarding mean squared error (MSE) loss (reconstruction loss), the smallest result is achieved by Custom CAE, and the ResNet CAE result is slightly larger (0.0119 and 0.01504, respectively). Regarding ELBO loss, the smallest result is 0.05484 with ResNet VAE 0.1, which is slightly better than Custom VAE 0.1 with 0.05513. Unfortunately, these results do not allow us to filter models for the next experiment, but the vital conclusion lies in different model loss function configurations. For both VAE models, 0.1 multipliers for the KL-Divergence results in a much better final test loss function value, the same as for the 0.2 temperature parameter for the CLR models.

Fig. 8 shows the 2-dimensional visualization of the test and validation embeddings. As described in the previous section, we used UMAP to reduce 200-dimensional vectors into 2 dimensions. Our preliminary experiments showed no use in *cosine* distance measures while reducing semantic vectors created by the VAE models. All points are uniform without any separation. Thus, we used *cosine* distance for all the configurations except VAE. We used *KL-Divergence* as it is part of the ELBO loss function. While KL-Divergence is not a distance measure, it can be used as a measure of similarity or dissimilarity between vectors.

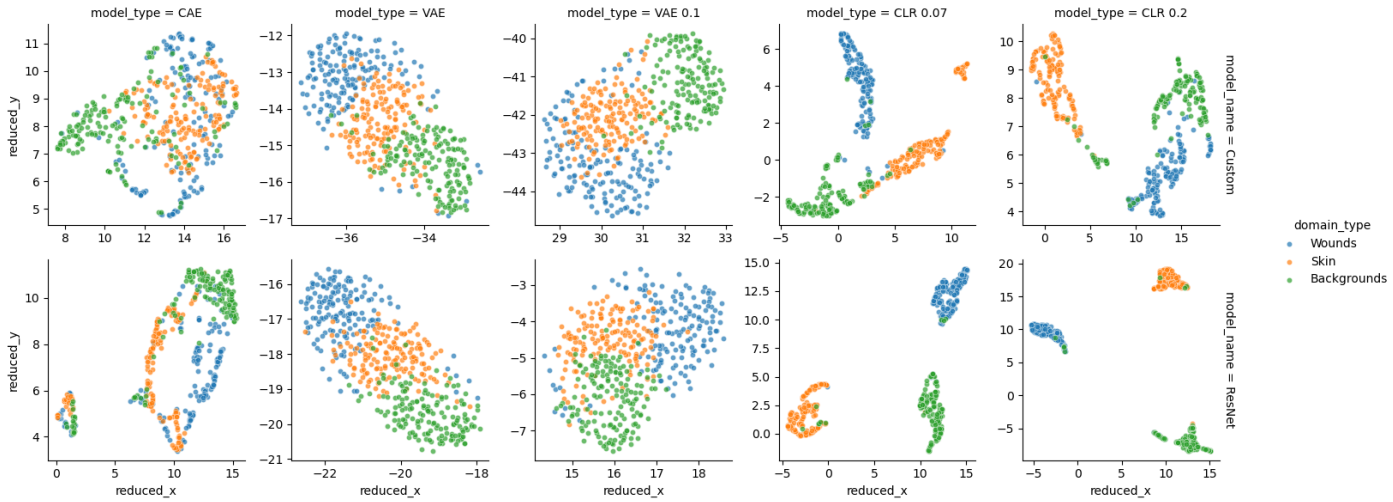

Figure 8. 2-dimensional embedding visualization

CAE models separate different textures but with lots of spatial mixes - some groups that are separated from the others are filled with different types of textures, like the bottom-left group from the first row, second column graph (CAE ResNet). This may be a sign that the neural network is learned to extract features that are not discriminative in terms of the wound/skin/background domains.

VAE models provide vectors that are not easily separable into clusters, which is expected, as ELBO loss penalizes vectors to be normally distributed. However, points are grouped according to the domain, with some mix at the borders of the groups. It is seen that skin texture vectors are placed between the wound and background ones. This may mean that VAE models can be used to discriminate wounds and backgrounds. We expect that the KNN classifier will suffer with the correct classification as groups of different textures are close to each other and do not have any free buffer space, so it is easy to misclassify the sample when checking the nearest neighbors. Still, MLP can learn how to model function that separates these close groups efficiently.

CLR provides the best separation of textures, especially the ResNet version. Still, there are some mixes, this we can conclude that possible issues with the CAE model may also affect CLR.

Table 2 shows classification results using KNN and MLP on the vectors generated by each type of embedding generation model. Details on classifier training are described in the previous section.

Table 2. Embedding classification results

Evaluation parameter / Model name	KNN train accuracy	KNN test accuracy	MLP train accuracy	MLP test accuracy
Custom CAE	1	0.777778	0.880503	0.813417
ResNet CAE	1	0.796646	0.988320	0.905660
Custom VAE 1	0.351303	0.339623	1	0.339623
ResNet VAE 1	0.349506	0.331237	1	0.345912
Custom VAE 0.1	0.350404	0.329140	1	0.333333
ResNet VAE 0.1	0.361186	0.341719	1	0.358491
Custom CLR 0.07	1	0.953878	0.986523	0.945493
ResNet CLR 0.07	1	0.960168	1	0.953878
Custom CLR 0.2	1	0.945493	0.991914	0.941300
ResNet CLR 0.2	1	0.960168	1	0.955975

These results show that embeddings generated by VAE models are utterly useless in terms of texture discrimination. MLP can learn to train VAE vector distribution as expected, but test accuracy shows that model overfits. KNN classifier shows a random result - 0.33 on all VAE model embeddings. Thus, variational autoencoders are unsuitable for generating embeddings intended to be used in classification tasks. Nevertheless, they can be used to generate artificial skin and wound textures. Using KL-Divergence instead of cosine similarity as a metric for KNN classification of VAE-generated vectors does not result in better accuracy, which is an unexpected result and may be caused by some non-obvious errors in the experiment setup.

In general, it is shown that both KNN and MLP classifiers are easily overfitted in almost all configurations. Thus, more robust regularization during training and other techniques to prevent overfitting must be introduced.

CLR model embeddings result in the best classification results for both MLP and KNN classifiers (more than 0.93), with no significant difference between them. This shows that Custom CLR models are the best option for semantic vector generation when we need to use them in the classification task. These modes are smaller and faster than ResNets' and slightly less performant on the classification task.

CAE models show moderate accuracy, better than VAE, but not so good as CLR models. The only advantage of CAE over CLR is that CLR models were trained using all three data classes, and CAE – used only skin textures. CAE models can be used when only one data class and no others are present during embedding model training time.

While CLR models show the best potential to be used for semantic vector generation for classification tasks, important to highlight that the way CLR models are trained implies that

similar vectors are to be grouped into clusters that are spatially far from each other, and this gives a significant advantage over the autoencoder way of training.

Some other sidenote conclusions from the training and evaluation process are:

- feature vector standardization does not affect MLP and KNN classifiers' training and evaluation results
- image patch standardization to the mean (0.458) and standard deviation (0.17) of a skin/wound/background dataset improves Custom model variants test loss values, decreases pre-trained ResNet loss values, and improves not-pre-trained ResNet loss values. However, standardization does not affect classification accuracy.

Even with low-dimensional data, it is important to emphasize that the quality of these data is crucial. As mentioned in the Data section, poor-quality images (due to noise, low resolution, lossy compression or other reasons) can negatively impact classification results by damaging important features and as a result tricking models. This highlights the need for careful preprocessing (utilizing denoising autoencoders or other techniques) in case of working with such data.

V. CONSLUSIONS

In conclusion, we highlight the main outputs of the conducted research. While these results may be suitable for different areas, we only state them here for the human skin wound texture domain.

- variational autoencoders are not suitable for generating feature texture vectors intended to be used in classification tasks. However, they are efficient in generating synthetic skin or wound textures and are to be used for this.
- ResNet variational autoencoders generate much better synthetic skin than Custom models.
- Custom models (in CLR architecture) are smaller, faster, and almost as effective as ResNets. In contrast, ResNets provide more stable results and should be considered after the accumulation of a large amount of training data to prevent embedding model overfit.
- CLR architecture is the smallest, fastest, and most efficient in generating semantic vectors for classification. It should be used when we have at least two classes during model training. For example, skin/non-skin, wound/non-wound. CAE architecture should be used when this is impossible and only one data class is available, so convolutional autoencoder may be trained on it.
- 0.1 multiplier for KL-Divergence calculation in ELBO loss function during VAE training results in faster convergence than the usage of the unit multiplier. Other multipliers are to be considered.
- Usage of the 0.2 temperature parameter for NT-Xent loss during CLR model training results in faster convergence and provide less uniform distribution of the vectors than the usage of the 0.07 temperature value.

The application of the semantic feature vector generation models and the semantic feature vectors of textures in the human skin wound domain may vary. We could generate synthetic textures for data augmentation during classification or segmentation model training or to find and classify skin/wound areas on large, complex, and non-typical images. There may be applications in vision transformer models or others still to be found. In this paper, we analyzed this topic, evaluated a few deep-learning-based feature vector generation methods, and generated options for further research.

VI. ACKNOWLEDGEMENTS

We express our respect and gratitude to all the brave ones defending Ukraine in the war with the russian federation.

VI. REFERENCES

[1] Z. Sanchez, A. Alva, M. Zimic, and C. del Carpio, “An algorithm for characterizing skin moles using image processing and machine learning,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 4, p. 3539, Aug. 2021, doi: https://doi.org/10.11591/ijece.v11i4.pp3539-3550.

[2] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *Lecture Notes in Computer Science*, vol. 9351, pp. 234–241, 2015, doi: https://doi.org/10.1007/978-3-319-24574-4_28.

[3] C. Wang *et al.*, “FUSeg: The Foot Ulcer Segmentation Challenge,” *arXiv:2201.00414 [cs, eess]*, Jan. 2022, Available: https://arxiv.org/abs/2201.00414

[4] Héctor Carrión, M. Jafari, M. D. Bagood, H.-L. Yang, Roslyn Rivkah Isseroff, and M. M. Gomez, “Automatic wound detection and size estimation using deep learning algorithms,” vol. 18, no. 3, pp. e1009852–e1009852, Mar. 2022, doi: https://doi.org/10.1371/journal.pcbi.1009852.

[5] T. Deng, S. Gulati, A. Kumar, W. Rodriguez, Benoit Dawant, and A. Langerman, “Automated detection of surgical wounds in videos of open neck procedures using a mask R-CNN,” Feb. 2021, doi: https://doi.org/10.1117/12.2580908.

[6] G. Scebba *et al.*, “Detect-and-segment: A deep learning approach to automate wound image segmentation,” *Informatics in Medicine Unlocked*, vol. 29, p. 100884, 2022, doi: https://doi.org/10.1016/j.imu.2022.100884.

[7] C. Wang *et al.*, “Fully automatic wound segmentation with deep convolutional neural networks,” *Scientific Reports*, vol. 10, no. 1, Dec. 2020, doi: https://doi.org/10.1038/s41598-020-78799-w.

[8] N. Jaworski, Ihor Farmaha, Uliana Marikutsa, Taras Farmaha, and Vasyl Savchyn, “Implementation features of wounds visual comparison subsystem,” *International Conference on Perspective Technologies and Methods in MEMS Design*, Apr. 2018, doi: https://doi.org/10.1109/memstech.2018.8365714.

[9] U. Şevik, E. Karakullukçu, T. Berber, Y. Akbaş, and S. Türkyılmaz, “Automatic classification of skin burn colour images using texture-based feature extraction,” *IET Image Processing*, vol. 13, no. 11, pp. 2018–2028, Sep. 2019, doi: https://doi.org/10.1049/iet-ipr.2018.5899.

[10] Z. Wu, Y. Xiong, S. Yu, and D. Lin, “Unsupervised Feature Learning via Non-Parametric Instance-level Discrimination,” *arXiv:1805.01978 [cs]*, May 2018, Available: https://arxiv.org/abs/1805.01978

[11] R. D. Hjelm *et al.*, “Learning deep representations by mutual information estimation and maximization,” *arXiv:1808.06670 [cs, stat]*, Feb. 2019, Accessed: Apr. 23, 2023. [Online]. Available: https://arxiv.org/abs/1808.06670

[12] P. Bachman, R. D. Hjelm, and W. Buchwalter, “Learning Representations by Maximizing Mutual Information Across Views,” *arXiv:1906.00910 [cs, stat]*, Jul. 2019, Accessed: Mar. 27, 2021. [Online]. Available: https://arxiv.org/abs/1906.00910

[13] N. Pal and T. T. Johnson, “Work In Progress: Safety and Robustness Verification of Autoencoder-Based Regression Models using the NNV Tool,” *Electronic Proceedings in Theoretical Computer Science*, vol. 361, pp. 79–88, Jul. 2022, doi: https://doi.org/10.4204/eptcs.361.8.

[14] D.-M. Tsai and P.-H. Jen, “Autoencoder-based anomaly detection for surface defect inspection,” *Advanced Engineering Informatics*, vol. 48, p. 101272, Apr. 2021, doi: https://doi.org/10.1016/j.aei.2021.101272.

[15] J. K. Chow, Z. Su, J. Wu, P. S. Tan, X. Mao, and Y. H. Wang, “Anomaly detection of defects on concrete structures with the convolutional autoencoder,” *Advanced Engineering Informatics*, vol. 45, p. 101105, Aug. 2020, doi: https://doi.org/10.1016/j.aei.2020.101105.

[16] L. Theis, W. Shi, A. Cunningham, and F. Huszár, “Lossy Image Compression with Compressive Autoencoders,” *arXiv:1703.00395 [cs, stat]*, Mar. 2017, Accessed: Apr. 23, 2023. [Online]. Available: https://arxiv.org/abs/1703.00395

[17] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, “Deep Convolutional AutoEncoder-based Lossy Image Compression,” *2018 Picture Coding*

Symposium (PCS), Jun. 2018, doi: https://doi.org/10.1109/pcs.2018.8456308.
[18] N. Khare, Poornima Singh Thakur, P. Khanna, and A. Ojha, "Analysis of Loss Functions for Image Reconstruction Using Convolutional Autoencoder," *Communications in computer and information science*, pp. 338–349, Jan. 2022, doi: https://doi.org/10.1007/978-3-031-11349-9_30.
[19] T. Spinner, J. Körner, J. Görtler, and O. Deussen, "Towards an Interpretable Latent Space – An Intuitive Comparison of Autoencoders with Variational Autoencoders," *thilospinner.com*, Oct. 22, 2018. https://thilospinner.com/towards-an-interpretable-latent-space/ (accessed Apr. 23, 2023).
[20] Li Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, Nov. 2012, doi: https://doi.org/10.1109/msp.2012.2211477.
[21] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, Nov. 1901, doi: https://doi.org/10.1080/14786440109462720.
[22] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv.org*, 2013. https://arxiv.org/abs/1312.6114
[23] D. P. Kingma and M. Welling, "An Introduction to Variational Autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019, doi: https://doi.org/10.1561/2200000056.
[24] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum Contrast for Unsupervised Visual Representation Learning," *arXiv:1911.05722 [cs]*, Mar. 2020, Available: https://arxiv.org/abs/1911.05722
[25] A. Dosovitskiy, P. Fischer, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative Unsupervised Feature Learning with Exemplar Convolutional Neural Networks," *arXiv:1406.6909 [cs]*, Jun. 2015, Accessed: Apr. 23, 2023. [Online]. Available: https://arxiv.org/abs/1406.6909
[26] A. van den Oord, Y. Li, and O. Vinyals, "Representation Learning with Contrastive Predictive Coding," *arXiv:1807.03748 [cs, stat]*, Jan. 2019, Available: https://arxiv.org/abs/1807.03748
[27] Y. Sun, K. Fu, Z. Wang, C. Zhang, and J. Ye, "Road Network Metric Learning for Estimated Time of Arrival," *arXiv:2006.13477 [cs, stat]*, Jun. 2020, Accessed: Apr. 23, 2023. [Online]. Available: https://arxiv.org/abs/2006.13477
[28] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A Simple Framework for Contrastive Learning of Visual Representations," *arXiv:2002.05709 [cs, stat]*, Jun. 2020, Available: https://arxiv.org/abs/2002.05709
[29] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition." Available: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf
[30] "SFA - Human Skin Image Database - SEL/EESC/USP," *www1.sel.eesc.usp.br*. http://www1.sel.eesc.usp.br/sfa/
[31] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning," *Deeplearningbook.org*, 2016. https://www.deeplearningbook.org/
[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Jun. 2016, doi: https://doi.org/10.1109/cvpr.2016.90.
[33] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks," *arXiv:1701.04128 [cs]*, Jan. 2017, Available: https://arxiv.org/abs/1701.04128
[34] F. Yu and V. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," *arXiv:1511.07122 [cs]*, Apr. 2016, Available: https://arxiv.org/abs/1511.07122
[35] A. Araujo, W. Norris, and J. Sim, "Computing Receptive Fields of Convolutional Neural Networks," *Distill*, vol. 4, no. 11, Nov. 2019, doi: https://doi.org/10.23915/distill.00021.
[36] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and Checkerboard Artifacts," *Distill*, vol. 1, no. 10, Oct. 2016, doi: https://doi.org/10.23915/distill.00003.
[37] F. Wang and H. Liu, "Understanding the Behaviour of Contrastive Loss." Accessed: Apr. 23, 2023. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021/papers/Wang_Understanding_the_Behaviour_of_Contrastive_Loss_CVPR_2021_paper.pdf
[38] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," *arXiv.org*, 2018. https://arxiv.org/abs/1802.03426

BOHDAN LUKASHCHUK is a machine learning engineer at Binariks, a teaching assistant, and a Postgraduate student in the Ukrainian National Forestry University CS department. He earned Bachelors's and Masters's degrees in computer science at Lviv National Polytechnic University, CAD department.
Main research interests: natural language and image processing using deep learning.
Email: bohdan.lukashchuk@gmail.com
Linkedin: https://www.linkedin.com/in/blukash/