Analysis of Dermatoscopic Images for Diagnosis of Pigmented Skin Lesions

Mastafa Foufa **Emin Bahadir Tülüce** **Musab Tayyib Gelisgen**

Abstract

Dermatoscopy is a commonly used diagnosis technique of pigmented skin lesions. With this technique, the accuracy of diagnostic tests increase compared to a test with an unaided eye. The scope of dermatoscopy can be increased with the help of modern deep learning applications.

A repetitive issue confronted when working with neural networks is that there is commonly insufficient information to boost the generalization capacity of the deep neural networks. There are numerous ways to solve this insufficiency in the dataset, and the methods in the paper are data augmentation, transfer learning, and dropout. We focused on different augmentation techniques from basic ones such as simple augmentation to more complex ones such as smart and strong augmentation.

Keywords: Dermatoscopy, Image generation with data augmentation, Binary classification, HAM10000 dataset, Unbalanced dataset

1 Introduction

In this project, we aimed to create an accurate diagnosis tool for pigmented skin lesions. We have a representative collection of all important diagnostic categories in the realm of pigmented lesions but we reduced the problem to a binary classification task of the two main classes. When dealing with deep neural networks, the most important task is to get access to enough good quality data. In our case, we do not have enough quality labeled data for a given class. In other words, we are dealing with the problem of an unbalanced dataset. As a result of that, classifiers tend to overfit and classify almost all images as the dominant class in the dataset. The network is not capable of generalizing the learned model to any other observations. To tackle this problem, mixing several datasets is often a good solution but it is not feasible in this case. Hence, for addressing this problem, we use "data augmentation". This technique describes the process of adding new observations that look like they were sampled from the true underlying distribution.

Augmentation is used as a form of regularization and as such reduces the risk of overfitting by extracting only the relevant information from the dataset.

The goal of this project is to compare different techniques of augmentation. We use for that different unsupervised augmentation methods: simple augmentations, strong augmentations, and augmentations with different architectures of autoencoders. These techniques are applied regardless of the label of the sample. A more challenging problem is to augment the data in a way that the classification task is optimized.

2 Related Work

In 1994, Binder effectively utilized dermatoscopic images successfully to train an artificial neural network to separate melanomas, the deadliest sort of skin malignancy, from melanocytic nevi [1].

We want to conduct the same classification between MEL and NV data, using new augmentation methods.

Augmentation is a technique to increase the amount of data by generating new data. There are different methods of augmentations one of which is manual augmentation. These include simple transformations, generative adversarial networks and learning with augmentation [2]. In this project, we have decided to apply and compare the results of simple transformations and different augmentation techniques.

The augmentation with autoencoders is done through an encoder and a decoder network. This method is described in [3]. The process briefly goes as follows: the images are fed into the encoder-decoder network for training the reconstruction process, then random noise is given as input to the decoder. The outputs are expected to be new random images with realistic characteristics.

Dropout is a technique introduced by Nitish Srivastava in 2014 [4]. The main idea in dropout is to randomly select units and drop them from the network as well as their connections while training the model. This prevents units from adapting to each other more than they need to, which in return reduces the overfitting.

Another way of expanding the generalization and decreasing the overfitting is transfer learning [5]. In this type of learning, an already existing knowledge from other domain is used to perform well in this current task. Transfer learning allows the tasks, domains, and distributions being used in training and testing to be different from each other. In the cases where the dataset is small just like in this project, the concept of transfer learning helps while training the network by starting from an already developed network in order to increase the success rate.

Just like the aforementioned regularization methods, there exists a method so-called Smart Augmentation which is usually used to enhance the regularization and reduce overfitting [6]. We don't want to deliver augmentations that seem characteristic. Rather, our system figures out how to consolidate pictures in manners that improve regularization. Smart augmentation can be utilized with other regularization systems, including the dropout and traditional augmentation.

3 Data

The dataset used in this project is called HAM10000 (Human Against Machine with 10000 training images), which is a collection of dermatoscopic images from different populations [7]. There are 10015 images in this dataset. The images in this dataset consist of seven different classes of skin cancer. We have proceeded to the usual data exploration techniques before diving into the problem. We can first transform the available images originally in .jpg format into *numpy* arrays and store them. When focusing on the distribution of each class, we notice that two classes in this dataset cover a huge proportion of the whole set which is approximately 78% and we have decided to focus on a binary classification of these two most occurring classes NV and MEL. The remaining 22% of the dataset is not used in this project.

For the pre-processing part, we had duplicate images in our dataset. For example, a given image can be taken under different angles and still refers to the same id. Hence, before separating our dataset into a training and a validation set, we needed to first identify which images have duplicates and only take into account the images with no duplicates in our validation set. That way, we reduce the bias due to an image already seen in the training process and hence correctly labeled in the validation set. We also separated our dataset in a way that the distribution of both classes is respected.

Finally, we assigned two different labels to both classes; class MEL has been assigned the label 1 which make the analysis easier and class NV the label 0. Those labels have been similarly stored in numpy arrays. Once that is done we can concatenate the arrays from class NV and class MEL and shuffle the data in a way that we have a traditional dataset with both classes. Then 80% is used for training and 20% is used for validation. Examples of images can be seen in Figure 1.

A problem with this dataset is that the distribution of the two classes NV and MEL differs a lot, meaning it is extremely unbalanced. Approximately, 85% of the images we have used is from class NV and 15% is from class MEL. Since this imbalance caused overfitting, we had to apply data augmentation on the data we use. Other issues have been faced while augmenting the data which are explained in the next sections.

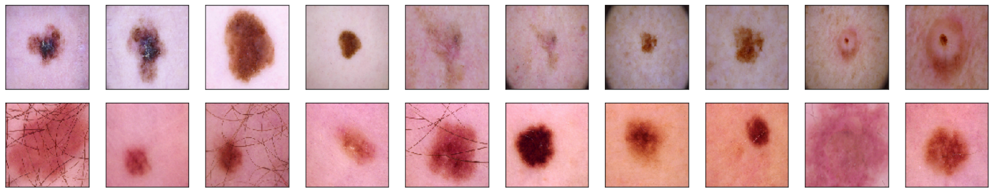
Figure 1: Sample images from the classes MEL (above) and NV (below)

4 Methods

To solve the imbalance problem in the dataset, we have used different data augmentation methods to create new images from the existing images. By combining the generated images and already existing images, we have created balanced datasets that should not cause overfitting problem while training. Then we fed these data to different classifiers and compared the results to see which augmentation methods and classifiers performed well.

To solve the unreliability problem of simple performance metrics, we have chosen some other metrics such as recall score of the classes and F1 score on classifiers.

These solutions and further problems related to these solutions will be discussed in this section.

4.1 Data Augmentation

The data augmentation is done through several autoencoders. The reason why we used several autoencoders is to see which one of them produce the best images and give the best results while classifying them.

4.1.1 Simple Transformations

The images are replicated using simple transformations chosen randomly for each image. These transformations include rotating, mirroring, flipping and zooming the image. We have included this method because it is a traditional way of data augmentation and we wanted to compare the performance of this with the autoencoders. Those transformations have been coded using the library *numpy*.

4.1.2 Strong Transformations

Going further with manual transformations, we decided to look at "stronger" transformations. Indeed, the previous transformations were not producing really different images. For this section, we used the third-party library *albumentations*.

Among the transformations we used, we randomly used one of the following:

- Additive Gaussian Noise: Add Gaussian noise to images.
- Motion Blur: blur images using Gaussian kernels.
- Random Brightness Contrast: a random brightness between 0 and 255 is used.
- Emboss: emboss images and overlays the result with the original image.
- Piecewise Affine: place a regular grid of points on an image and randomly move the neighbourhood of these points around via affine transformations.
- Hue Saturation: change the colorspace of images.

4.1.3 Variational Autoencoder (VAE)

AutoEncoder architecture is commonly used in deep learning. We have chosen to work with the Variational Autoencoder implemented with a Stochastic Gradient Variational Bayes estimator that is used as a stochastic objective function (Kingma and Welling, 2014). VAE is based on constraining the feature space to follow a simple distribution, here we use a Gaussian distribution. By resampling from this distribution, we have a reconstruction term that is added to the loss function that we want to minimize during training. We used convolutional layers to better deal with RGB images. Based on the paper from Kingma and Welling, we use a specific output "CustomVariationalLayer" that customized the traditional cross entropy loss and add a constraint on the distribution followed on the feature space via the KL-divergence term. The latent dimension is brought to a dimension of three.

4.1.4 Stacked Autoencoder (SAE)

Stacked autoencoder is a network consisting of a deep encoder and decoder. The structure of encoder consists of 2D convolutional max pooling layers where decoder has convolutional and upsampling layers. This autoencoder is also called deep convolutional autoencoder.

4.1.5 Denoising Autoencoder (DAE)

The thought behind denoising autoencoders is as follows. So as to force the hidden layer to find progressively robust features and keep it from learning the identity, we train the autoencoder to remake the input from a corrupted version of it. Denoising autoencoder's layer structure is similar to the stacked autoencoder's layer structure. In order to compensate some of the data-loss coming from added noise, we have chosen a larger bottleneck tensor between encoder and decoder.

We utilize denoising autoencoders by feeding it with the noisy HAM10000 images and therefore coming up with a model that is not prone to be affected by the noisy data.

4.2 Classification

We have designed and trained two different classifiers to reduce the effects of one classifier being biased to a specific set of images. Each of these classifiers will be trained separately for each of the augmented datasets. In figure 2, we present two classifiers that have been used for classification: a simple convolutional neural network and MobileNet V2.

4.2.1 Simple Convolutional Network

For the classification task on a small dataset, a deep architecture is not necessary. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. It is a powerful kind of neural network when dealing with images since it can catch the relevant patterns in a hierarchical manner over the layers.

4.2.2 MobileNet v2 (Transfer Learning)

Since we are dealing with image classification, we decided to make use of some state-of-art transfer learning networks. MobileNet v2 is a new one that looked promising so we have applied that with the following adjustments: we have frozen weights of all layers except the last one. One fully connected layer with 128 nodes is added to the network. To deal with overfitting, dropout with 0.5 percentage is applied to the last layer.

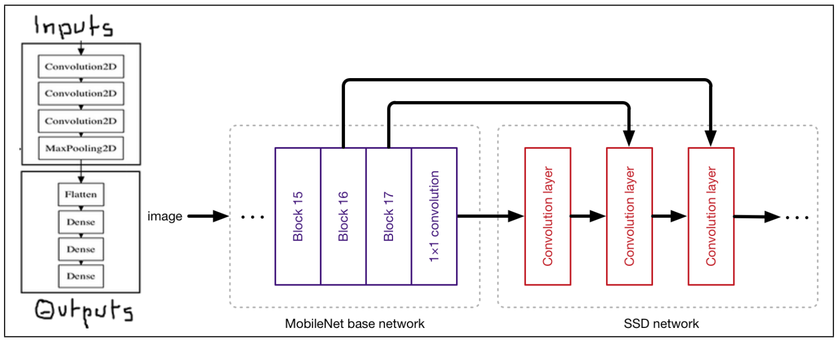

Figure 2: Classifiers used for the classification task. On the left, a simple CNN and on the right MobileNet Version2

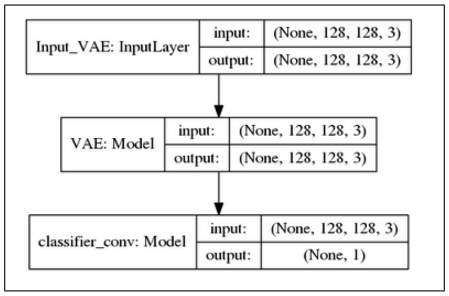
Input_VAE: InputLayer
input: (None, 128, 128, 3)
output: (None, 128, 128, 3)
VAE: Model
input: (None, 128, 128, 3)
output: (None, 128, 128, 3)
classifier_conv: Model
input: (None, 128, 128, 3)
output: (None, 1)

Figure 3: Smart augmentation diagram. The first network is VAE, the decoder part is used as a generator of new images. The second network is a classifier that determines the labels of the incoming images which are produced by the VAE.

4.3 Avoid overfitting on generated images

An unexpected problem occurred in the classification part of the project after creating the new images and training the classifiers with them. The classifiers seemed to be able to learn the augmentations. Hence, they will perform much better on the augmented class MEL than on class NV.

To overcome this issue, we also applied data augmentation on the class NV. So, the proportion of the fake looking images and the real images got close to each other in both classes. Then, training the models with these new set of images has overall yielded better results.

4.4 Merging augmentation and classification – Smart augmentation

An idea that is further used on GANs, for example, is to learn to produce images that actually help the classification task. Prior to that, we were augmenting the dataset regardless of the labels of the observations. But by merging the augmentation structure and the classifier, we expect to improve the generation process. This has only been realized with the VAE combined with the simple convolutional classifier. In figure 3, we can see simply how such structure looks like.

4.5 Performance Metric

Given that we have an unbalanced dataset, traditional metrics such as accuracy should be used with attention. Having a large accuracy can mean that we are overfitting the dataset. Therefore, we rather use recall on class MEL and the f1 score on the validation set to evaluate the performance of our classification task.

We have used F1 score as the main performance metric for evaluating our classifiers. The following formulas are given for a dataset with classes A (positive) and B (negative) where we focus on precision and recall on class A refering to class MEL here:

$$Precision = \frac{tp}{tp + fp}$$
$$Recall = \frac{tp}{tp + fn}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

We have also considered balancing validation data by cropping some of the dominant part, however this would lessen our information and reduce the reliability of the metric again.

5 Experiments

The experiments have been done in two steps: first, focus on data augmentation and then on the classification task. The results of the classification task are presented in Table 1.

5.1 Data Augmentation

The data augmentation has been realized manually using numpy and the library albumentations and with neural networks. In Figure 7, we can see the results obtained with a variational autoencoder and with strong augmentations. The VAE is trained on class MEL over 100 epochs. The learning rate is handled with RMSProp optimizer. Once training is done, we can only focus on the decoding part where we feed a Gaussian noise with mean 0 and variance 1.2. The very same is done with SAE and DAE. SAE and DAE surprisingly did not yield so different results and hence we just worked with SAE in the following. The learning process yield results shown in Figure 6.

As a quick note, we used several architectures for the autoencoders and working with convolutional layers and a latent dimension of size three seemed to bring the best results. We also tried with a latent space that has not been flattened and this led to bad results. Figure 4 shows a comparison of the original architecture used for the SAE and the final one. We observe that the VAE gives the best results overall. For each technique, not only do we fit the class MEL but also the class NV. This will prove to be useful for the classification task. We produce ten times the amount of images from clas MEL and twice the number of images from class NV. When only augmenting MEL, we produce 4913 additional images which allow us to have the exact same amount of images in both classes in the training set.

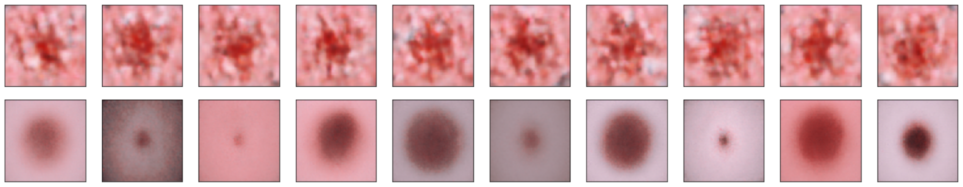
Figure 4: Bad (top) and good (bottom) autoencoders trained over 50 epochs show different results.

5.2 Classification

To proceed to the comparison of all the augmentation techniques, we use the two classifiers in figure 2. Based on the confusion matrix, we can check whether the model is actually overfitting the dataset or not.

The very first experiment was realized with no augmentation. We can notice that even with a categorical weighted cross entropy loss function, both our classifiers are overfitting the dataset since class MEL is not well represented. We present indeed a recall of 0.0 on the class MEL.

We focus on the recall and the f1 score to compare the different techniques. Hence, we customized the early stopping in *keras* that is usually based on accuracy to do early stopping based on the f1 score value by setting a threshold that is evaluated empirically for every augmentation method. Dropout (p = 0.5) has been used in the last layer of our CNN to avoid overfitting. Also, the learning rate is optimized using RMSprop on keras.

Therefore, we started to compare different augmentation techniques. We start with simple transformations with numpy and that already led to good results. What we notice during certain training phases is that when only augmenting MEL data, the classifier starts then to perform very well on MEL but nearly takes a chance when it sees an observation from class MEL. To tackle this problem, we augment both class MEL and class NV and still bring the same number of observations in the training set. This, in most cases, has proven to lead to better results.

Smart augmentation has only been realized by merging the VAE (A) and the simple convolutional network (B). Their loss is combined using a combined loss such that the final loss is f(La, Lb; alpha, beta). We work with a simple affine function:

$$f(La, Lb; \alpha, \beta) = \alpha * La + \beta * Lb \tag{1}$$

We set $\alpha = 0.5$, $\beta = 1.0$

Once trained with the classifier, we can generate images of the class MEL. It has proven to produce a better performance than the simple VAE, with a recall of 0.85 and an f1 score slightly lower of 0.53.

Such architecture takes time to train and tuning of the parameters of function f can help us improve such results.

Overall, the VAE is the structure that best generates augmented images in comparison with the other autoencoders. We reach for the simple classifier a recall of 0.59 and an f1 score of 0.61. For MobileNet version2, we reached the best f1 score of 0.542.

However, surprisingly, the strong manual augmentations yield very good results and even outperform deep learning techniques for the first classifier. On Figure 5 are plotted the confusion matrices obtained for strong augmentation and smart augmentation on the first classifier.

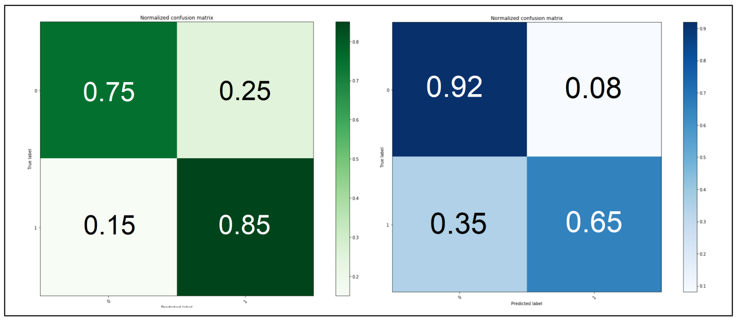

Figure 5: Confusion matrix for the smart augmentation (left) and the strong augmentation (right) trained on Simple CNN.

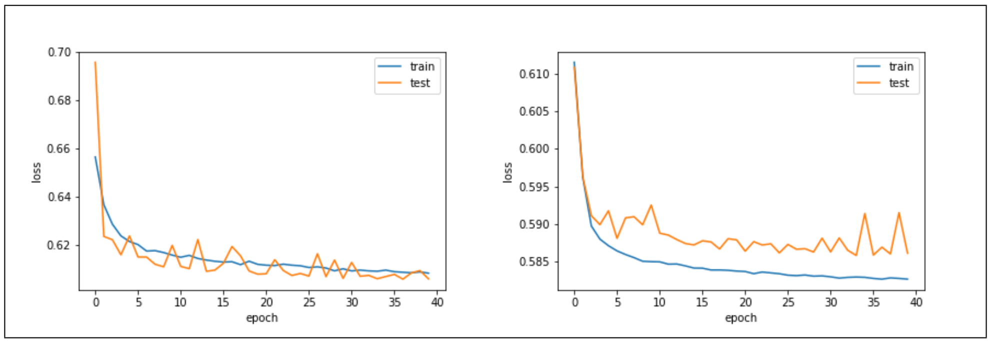

Figure 6: Training process for the stacked autoencoder for MEL (left) and NV(right)

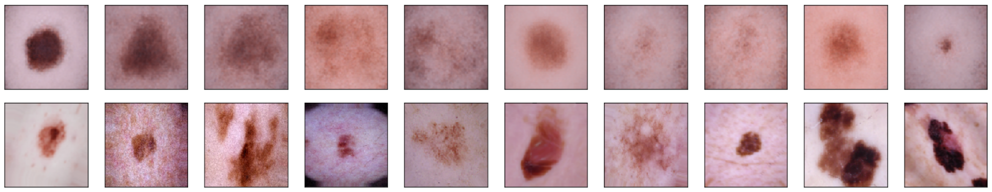

Figure 7: Generated images using VAE (above) and Strong Augmentation (below) for the class MEL

6 Conclusion

Data Augmentation has shown the potential to increase the performance of different classifiers by avoiding overfitting.

In this project, we discuss different approaches to perform data augmentation. We focus on simple learning augmentation and more advanced ones with autoencoders. We further focus on learning

Table 1: Final results

Classifier1 / Classifier2	**MEL Data**	**NV Data**	**Recall**	**F1 Score**
Simple Convolutional / MobileNet V2	Original + Weighted Cross Entropy	Original	0.0 / 0.0	0.0 / 0.0
Simple Convolutional / MobileNet V2	Original + Augmented[Simple]	Original	0.54 / 0.487	0.40 / 0.331
Simple Convolutional / MobileNet V2	Original + Augmented[Strong]	Original	0.65 / 0.575	0.62 / 0.516
Simple Convolutional / MobileNet V2	Original + Augmented[Strong]	Original + Augmented[Strong]	0.66 / 0.567	**0.62** / 0.520
Simple Convolutional / MobileNet V2	Original + Augmented[SAE]	Original	0.557 / 0.662	0.424 / 0.443
Simple Convolutional / MobileNet V2	Original + Augmented[VAE]	Original	0.840 / 0.711	0.49 / 0.536
Simple Convolutional / MobileNet V2	Original + Augmented[VAE]	Original + Augmented[VAE]	0.59 / 0.668	0.61 / **0.542**
Simple Convolutional / MobileNet V2	Original + Augmented[Smart]	Original	**0.85** / NA	0.53 / NA

augmentations that take advantage of the information of the class. The proposed solutions were tested on different classification networks to reduce the bias of the network selection. The different experiments presented in this work demonstrate that autoencoders are powerful architectures for data augmentation. Also, combining the classification task and the augmentation task can yield interesting results but demand more computational time.

In conclusion, we have proved that data augmentation can be used to reduce overfitting during training and augment the performance of the classification task. It is worthwhile noticing that manual augmentations already yield very good results but need advanced programming.

Future work may include other datasets, more tuning of the function used for combining losses in the smart augmentation technique and training our networks on more epochs.

7 References

1. M. Binder. "Application of an artificial neural network in epiluminescence microscopy pattern analysis of pigmented skin lesions: a pilot study". Br J Dermatol 130, 460–465 (1994).

2. L. Perez and J. Wang. “The Effectiveness of Data Augmentationin Image Classification using Deep Learning”. In:CoRRabs/1712.04621(2017). arXiv:1712.04621.url:http://arxiv.org/abs/1712.04621.

3. U. G. Maestre. "Effective Techniques of the Use of Data Augmentation in Classification Tasks".url:https://rua.ua.es/dspace/bitstream/10045/76988/1/Tecnicas_efectivas_del_uso_del_aumentado_de_datos_en_tare_Garay_Maestre_Unai.pdf.

4. N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting.” Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, 2014.

5. S. J. Pan and Q. Yang, “A survey on transfer learning,” IEEE Transactions on knowledge and data engineering, vol. 22, no. 10, pp. 1345–1359, 2010.

6. I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” arXiv preprint arXiv:1412.6572, 2014.

7. P. Tschandl, C. Rosendahl, and H. Kittler. “The HAM10000Dataset: A Large Collection of Multi-Source Dermatoscopic Images of Com-mon Pigmented Skin Lesions”. In:CoRRabs/1803.10417 (2018). arXiv:1803.10417. url:http://arxiv.org/abs/1803.10417

Learning Outcomes

Mastafa Foufa:

1. Supervising and integrating a DL project
2. Processing and exploring the data
3. Designing and training neural networks: VAE and CNN
4. Implementing a supervised version of data augmentation

—

Emin Bahadir Tülüce:

1. Designing and training stacked autoencoders
2. Data augmentation using autoencoders
3. Transfer learning with MobileNet v2

—

Musab Tayyib Gelisgen:

1. Designing and training denoising autoencoders
2. Data augmentation using autoencoders
3. Some methods to solve overfitting problems