Skin Cancer Classification

– Attention to The Detail –

Deep Learning Project

University of Klagenfurt

Markus Köfler

Course:

Deep Learning and Spiking Neural Networks
for Advanced Data Mining (700.390, 23S)

Instructor:

PD Dipl.-Inf. Dr.techn.habil. Fadi Al Machot

June 30, 2023

Contents

List of Figures

1 Introduction

Skin cancer is a severe global health concern, accounting for a substantial number of cancer cases and related deaths worldwide. A timely detection of skin cancer plays a crucial role in improving patient outcomes, as early intervention significantly increases the chances of successful treatment. Dermatologists are at the forefront of diagnosing skin cancer, relying on their expertise in visual inspection and analysis. Especially UV-light exposure contributes the growing incidence of skin cancer cases, posing a tough challenge for dermatologists, emphasizing the need for advanced technologies to aid in diagnosis.

Throughout the recent decade, deep learning, specifically Convolutional Neural Networks (CNNs), have emerged as a powerful tool for computer vision tasks, including image recognition and classification. CNNs have shown remarkable success in various domains, including medical image screening, and hold immense potential for assisting doctors in the early detection of cancer, among many other types, skin carcinomas.

The relevance of leveraging machine learning algorithms, such as CNNs, in skin cancer classification lies in their ability to process large volumes of medical imaging data with speed and accuracy. Dermatologists often face challenges in differentiating benign lesions from potentially malignant ones due to the subtle variations in visual appearance. By training CNN models on diverse datasets comprised of dermoscopic lesion images, it becomes possible to extract high-level features and learn complex patterns that aid in accurate classification.

The potential benefits of employing CNNs in skin cancer classification are manifold. For instance, they can help dermatologists overcome human subjectivity and variability in diagnoses, delivering an objective and consistent decision support system. Ideally, this lowers the chance of misdiagnosis, leading to more efficient and effective patient management. Another handy application to think of would be as a kind educational tool for dermatologists in training. Even embedded in telemedicine platforms, such systems can offer value to individuals who do not have access to good healthcare.

For this project, the paper *Soft-Attention Improves Skin Cancer Classification Perfor-*

mance published by Datta et al.[1] has been considered as a starting point. The paper investigates the effectiveness of soft-attention in deep neural architectures for skin cancer classification. The central idea of a soft-attention module is to emphasize the region of important features while suppressing the noise-inducing information like hair, superficial blood vessels or other non-related skin pigmentation. The authors compare the performance of the popular models VGG16, ResNet, Inception ResNet v2 and DenseNet architectures (some have been pre-trained) with and without the soft-attention layer. Accordingly, the original network when complemented by a soft-attention unit outperform the models without this additional feature in all instances, and is therefore be seen as a success. The aim of this report is do reproduce these results on a smaller scope basis and validating the findings of Datta et al. (2021). A brief and readable overview on the most essential concepts for understanding the mechanisms working inside of a CNN will be given, followed by a thorough description of the workflow, and finally, an explanation to the models' architectures.

2 The Data

The primary dataset used for this project is the HAM10000[2] dataset, which is publicly available on the ISIC homepage. The HAM10000 dataset provides a vast collection of dermatoscopic images from several sources of common pigmented skin lesions. It comes with 10015 labelled photographs that can be used for training of deep neural nets with the goal of classifying pigmented skin lesions. The people involved[3] collected dermatoscopic images from various populations. Dermatoscopy in that sense is a non-invasive diagnostic approach that uses a magnifying instrument and a light source to visualize beneath skin features. [4]

For further model evaluation, the dataset used in the 2019 ISIC challenge has been taken. As this dataset also includes two more classes, namely *squamous cell carcinoma* and *unknown*, the additional classes have been discarded because the models proposed

[1] Datta et al., 2021

[2] Abbreviation for *Human Against Machine*

[3] A team of researchers from the Medical University of Vienna and the University of Queensland, Australia. The team includes Philipp Tschandl, Cliff Rosendahl, Harald Kittler and Holger Haenssle.

[4] Tschandl, Rosendahl, and Kittler, 2018

in the upcoming sections have been trained only on the HAM10000 dataset. Hereby, it needs to be mentioned, that the HAM10000 images are already included in the ISIC 2019 challenge images.

Hereby follows a summary for the class distribution of the two datasets[5]:

Class	**2018**	**2019**
Melanoma	1113	4522
Melanocytic nevus	6705	12875
Basal cell carcinoma	514	3323
Actinic keratosis	327	867
Benign keratosis	1099	2624
Dermatofibroma	115	239
Vascular lesion	142	253
Squamous cell carcinoma	-	628
Other / Unknown	-	-
Total	10015	25331

Table 1: Data sets' class distributions

[5] Cassidya et al., 2021

3 A mathematical perspective on CNNs

This section aims to explain the basic mathematical concepts that are used in today's CNNs. But first, I want to address the relevance of CNNs. CNNs have revolutionized the sub-domain of computer vision by enabling highly effective analysis and recognition of visual data. Unlike regular Neural Networks (NNs), CNNs are specifically designed to extract spatial features from images, making them highly suitable for capturing intricate patterns and structures.[6]

CNNs are inspired by the human brain, in specific, the visual cortex. Internal hierarchical structures are capable of extract complex features, and the use of local receptive fields and weight sharing to capture spatial dependencies and achieve spatial invariance, mirroring the brain's ability to recognize patterns and tolerate variations in visual stimuli.[7] Implementing CNNs results in a drastically reduced number of parameters compared to an equivalent feed-forward NN having identical layer dimensions. This reduction arises due to the reuse of network parameters (also known as *parameter-sharing*) as the convolution kernel traverses the image. Therefore, convolution induces less memory intensive computation while maintaining statistically significant outputs. Intuitively, this phenomenon can be attributed to the ability to identify features within an image irrespective of their spatial location. This inherent adaptability of CNNs is referred to as *translation invariance*, which enables them to exhibit robustness in detecting features even when they undergo spatial transformations.[8] CNNs employ convolutional layers which, as their name suggests, convolve learnable filters with input images, allowing for automatic capturing of patterns. In conjunction with pooling layers for downsampling and spatial invariance allows to recognize objects and textures regardless of their position in the image. CNNs possess advantages over regular NNs in computer vision tasks, including efficient learning from large-scale image datasets, automatic feature learning, and exceptional performance in benchmark datasets and real-world applications.

[6] Campos, Jou, and Giró-i-Nieto, 2017

[7] Goodfellow, Bengio, and Courville, 2016

[8] Wood, n.d.

3.1 Convolution

In a mathematical application, a convolution is a operation that combines input data with a filter to produce a feature map. A convolution layer in a CNN model is capable of performing multiple convolutions to extract and capture spatial features from the input data.

Most of the time, we perform convolution on images, which can be translated into two-dimensional matrices, according to their pixel values. These two-dimensional arrays[9] are often treated as tensors of order one. Given an input image I with dimensions m rows and n columns, applying the convolution over both axis, an element-wise multiplication (indicated by $*$) a kernel K[10] runs over the image. The products are then summed up and placed into the corresponding location in the output, also known as feature map comprised of x rows and y columns.[11]
Mathematically, we can formulate this process as follows:

$$S(x,y) = (I * K)(x,y) = \sum_m \sum_n I(m,n) \cdot K(x-m, y-n) \tag{1}$$

$$= (K * I)(x,y) = \sum_m \sum_n I(x-m, y-n) \cdot K(m,n) \tag{2}$$

$$= (K * I)(x,y) = \sum_m \sum_n I(x+m, y+n) \cdot K(m,n) \tag{3}$$

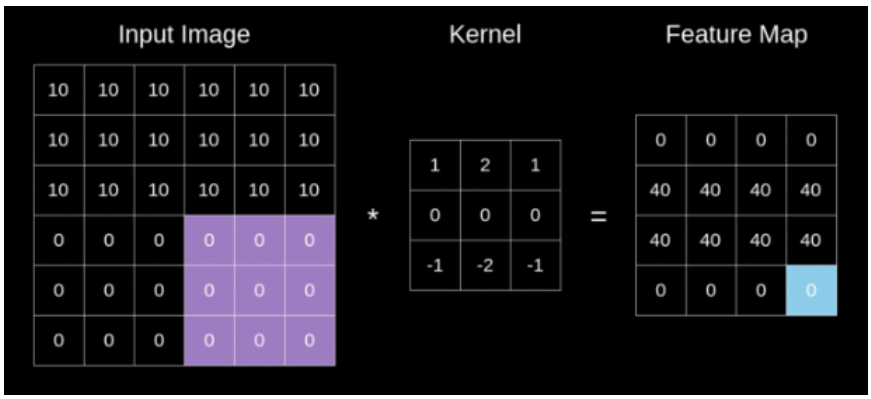
The three formulas provided showcase different arrangements of the input function and the kernel within the convolution operation. Formula (1) and Formula (2) are equivalent due to commutative properties of the convolution. In Formula (3), the commutative property is omitted, as it is not important in practical applications. To be more precise, (3) is actually the function for *cross-correlation*, which, unlike in convolutions, the kernel K is not flipped. When building a CNN model, the convolution would loose this commutative property regardless, as other functions are stacked on top of convolution layers.[12]

[9] Superset of matrices and vectors.

[10] The term kernel is analogous to filter, which describe a smaller matrix comprised of weights that transverses the input image.

[11] Skalski, 2019

[12] Goodfellow, Bengio, and Courville, 2016



Credit: *Piotr Skalski*

Figure 1: Convolution process

Figure 2 visualizes the result of running convolution over a sample skin lesion photograph:

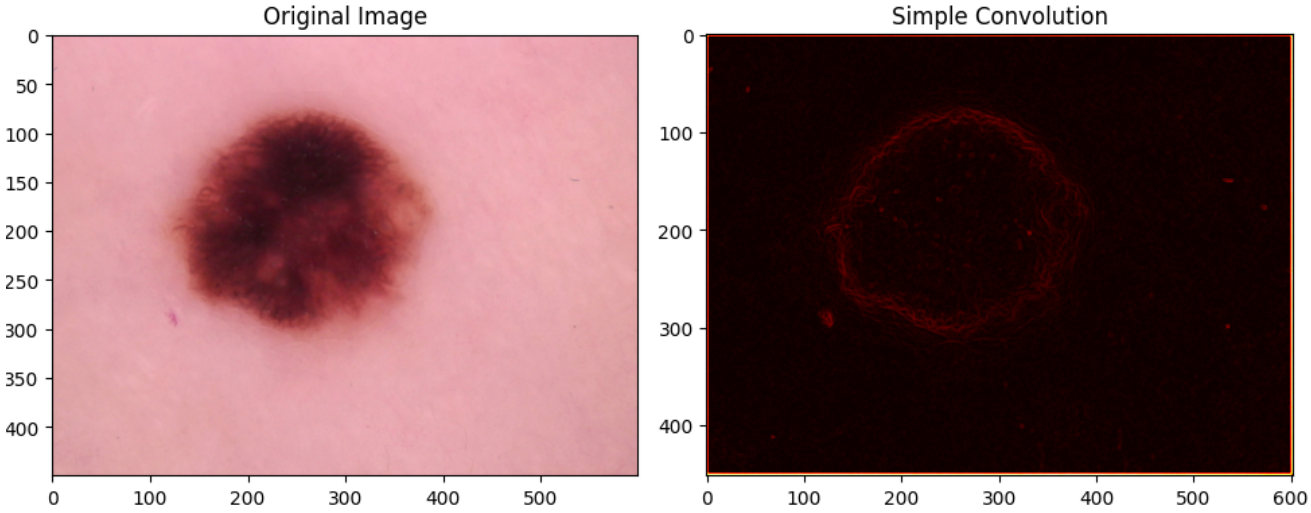


Figure 2: Simple convolution applied to skin lesion image

Covolutional layers are typically followed by an activation to introduce non-linearity. This way, a stronger contrast will be generated, enhancing interesting features.[13] Commonly, the ReLU function is preferred over other activation functions such as sigmoid as empirical investigations show that in classification tasks, ReLU leads to a better performance of the model.[14] The ReLU function can be written as follows:

$$ReLU(x) = max(0, x)$$

[13] Wood, n.d.

[14] Kuo, 2016

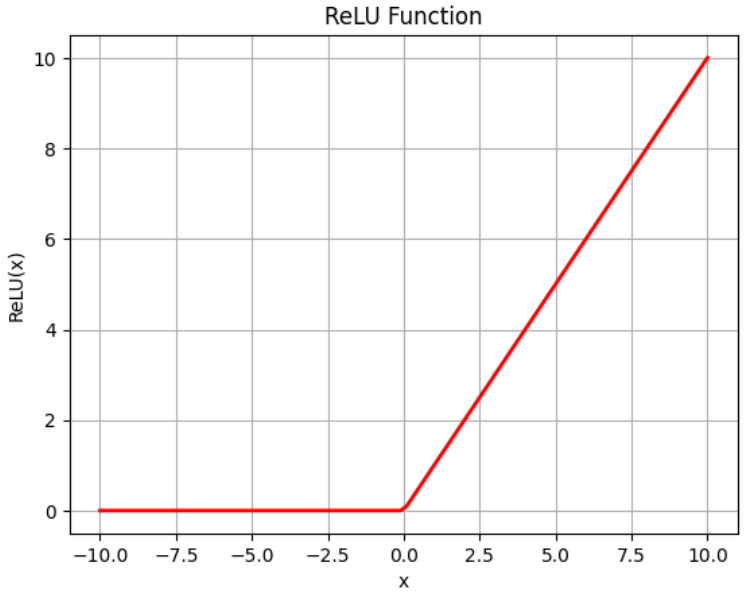


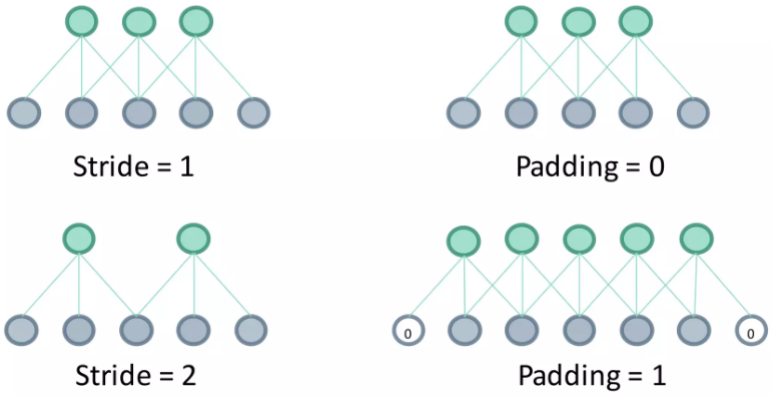
Figure 3: ReLU activation function

3.1.1 Strides and Padding

Stride and padding are both hyperparameters of a convolution layer, affect the output dimensions and the feature extraction pattern of the convolution. The stride in a convolution layer refer to the step size or spatial displacement applied to the filter during the sliding operation across the input, which can speed up reduce the computation intensity by downsampling when increased. Padding in a convolution layer involves adding additional border pixels around the input to maintain spatial dimensions, which would otherwise become smaller and smaller as we introduce more convolution layers. For example, setting stride equal to 1 (filter map moves to direct neighbor location)[15], we must deploy a padding of 1 (which adds a single pixel around the entire image) to prevent the output feature map from shrinking.[16]

[15] This is also known as *smae* padding. In contrast, with *full* padding, the maximum number of pixels given the filter size is added, for *valid* padding, no additional pixels are added.

[16] Pandey, 2020



Source: *https://www.slideshare.net/ckmarkohchang/applied-deep-learning-1103-convolutional-neural-networks*

Figure 4: Strides and padding

Now knowing the most crucial elements for a convolution layer, we can determine the output of a 2-D convolution of an input image. The formula here is given as follows:

$$o = \left[\frac{n + 2p - m}{s} + 1\right],$$

whereby n describes the input dimension (assuming equality of rows and columns), p the padding, m the filter size, and s the stride. The output dimensions of the convolution will be $o \times o \times K$, with K being the number of filters.

To demonstrate this, imagine we have an color image of dimensions $100 \times 100 \times 3$ (because of RGB channels), and run convolution with 64 filters of dimensions $6 \times 6 \times 3$, zero-padding and a stride of 3, we would get an output dimension of $32 \times 32 \times 64$:

$$o = \left[\frac{100 + 2 \times 0 - 6}{3} + 1\right] \tag{4}$$

$$= 32 \tag{5}$$

3.1.2 Pooling

Pooling layers are introduced for the sake of reducing tensor-sizes, which can severely speed up computation time. [17] Applying maximum (or just max) pooling, in the forward pass procedure, only the element with the highest numeric value within the filtered region is selected. This is achieved by applying a mask that memorizes the specific location of the maximum value. Accordingly, this channel-wise procedure maps the highest number of a specific sub-region of the input to the corresponding output feature

[17] Pandey, 2020

matrix.[18] Another widely used technique is average pooling, whereby the arithmetic mean is computed instead of selecting the maximum value. Figure 5 illustrates, how max pooling works:

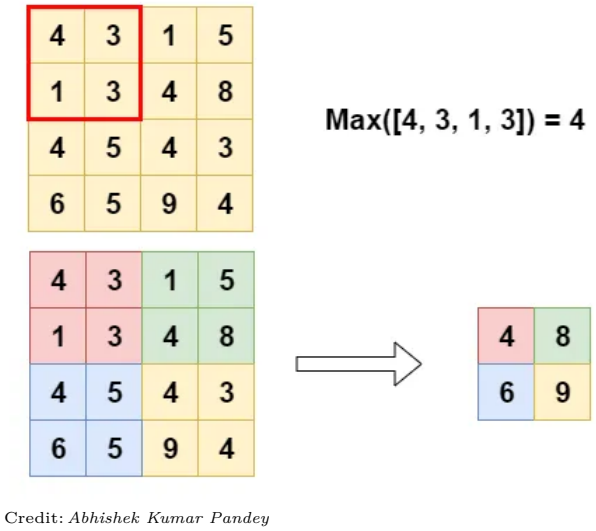


Figure 5: Max pooling

No learning take place within the pooling layer, as no parameters are initialized.

3.2 Batch-Normalization

Batch-Normalization is a concept widely used to accelerate the learning process by normalizing batches B similar to computing Z-scores and adding a small error term ϵ to increase the sample variance by a small degree. This is done to counteract the *internal covariance shift* of batches, and ϵ[19] to firstly avoid divisions by 0 and to introduce some variance, as sample typically show smaller variation than the actual population.[20] Thereby, we do not account for degrees of freedom when computing the standard deviation.

$$\mu_B = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\sigma_B = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu_B)^2}{n}}$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{(\sigma_B + \epsilon)^2}}$$

[18] A. Amidi and S. Amidi, 2018

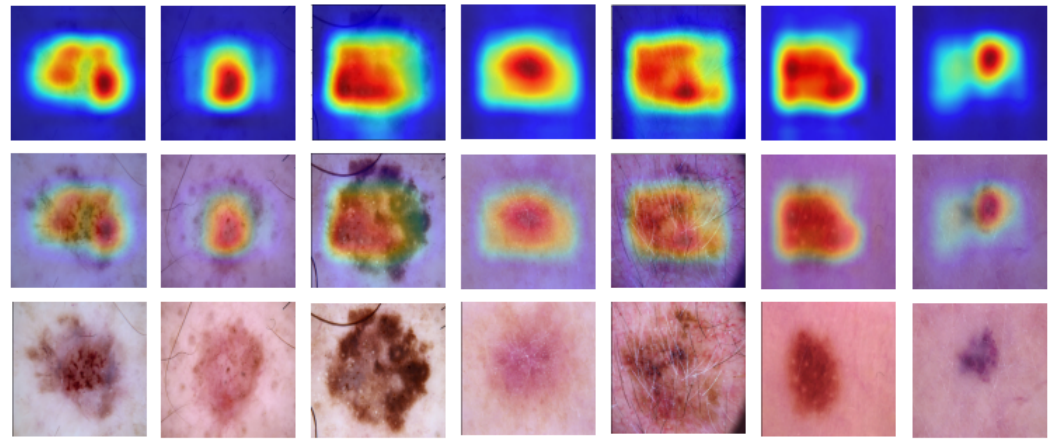
[19] Usually a rather small value between 0.1 down to 0.000001

[20] N., 2020

The learnable parameters are then given as $w_i\hat{x}_i + b_i \equiv BA_{w_i,b_i}(x_i)$.[21] Among the main benefits of introducing batch-normalization are that training happens faster, higher learning rates can be used, and noise is added to avoid overfitting.[22]

3.3 Soft-Attention Layer

Regarding the classification at hand, there are certain issues with the dermatoscopic images which can ultimately impact the outcomes. Many photographs of the lesion region also contain other features like hair, stains, superficial vascular structures and other pigmentation. These features ultimately represent noise, potentially distracting the model from capturing the relevant information, that is, the lesion region only. Datta et al. (2021) incorporated a *soft-attention* module to combat this issue and achieved better results that way.



Credit: *Datta et al.*

Figure 6: Attention hat maps projected on HAM10000 classes

As proposed by Shaikh et al. (2020), the soft-attention layer utilizes 3D-convolution to extract only the useful patterns of an input tensor. Passing in an input tensor with dimensions $f^x \in \mathbb{R}^{m^x \times n^x \times d^x}$ in conjunction with K 3D kernels denoted as $g_1, \ldots, g_K$ of size $3 \times 3 \times d^x$ results in individual feature maps of $f_{3d} \in \mathbb{R}^{m^x \times n^x \times 1}$, and in total these amount to $f_{3d} \in \mathbb{R}^{m^x \times n^x \times K}$. The resulting feature maps are first normalized by assigning importance scores for the locations attention should be paid to in the feature map before

[21] Ioffe and Szegedy, 2015

[22] Therefore, drop-out rates should be chosen carefully, as batch-normalization already introduces noise.

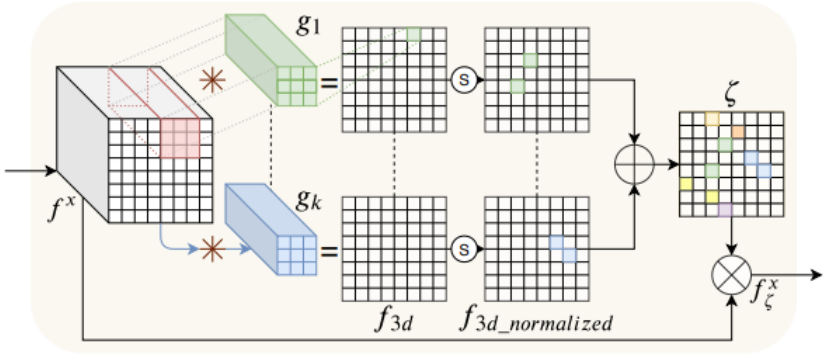
they are aggregated. The whole process leading to the attention scores ζ can be expressed as follows:

$$\zeta = \sum_{k=1}^{K} \frac{\exp(f_{3d_{i,j}})}{\sum_{i,j=1}^{m^x,n^x} \exp(f_{3d_{i,j}})}, \tag{6}$$

whereby $f_{3d} = g(f^x)$. In the final stage, the input f^x is multiplied with the soft attention scores ζ resulting in f^x_ζ. The regions with a corresponding low attention score will be closer to 0. The learnable parameters w are then multiplied by f^x_ζ, whereby the attention regions are applied to the parameters:[23]

$$o_\zeta = f^x + w f^x_\zeta$$

In Figure 7, S indicates the use of softmax for producing the attention maps, $\bigoplus$ is the aggregation procedure as demonstrated in equation (6), and $*$ is the convolution operation:



Credit: *Shaikh, Duan, Chauhan and Srihari (State University of NY at Buffalo)*

Figure 7: Internal architecture of soft-attention module

Such a soft-attention layer has also been incorporated in one of the models deployed in this project. The soft-attention module has been taken from Datta et al. Slight customizations have been made, though, the concept as discussed above is still applicable:

$$f^x_\zeta = \gamma f^x \left(\left(\sum_{k=1}^{K} softmax(w_k f^x) \right) \right)$$

The hyperparameters allow to specify the number of attention maps and whether to invoke aggregation (for weighting). $wf\zeta^x$ is then concatenated with the input feature tensor f^x resulting in o_ζ. γ poses an additional learnable parameter that adjusts the level of attention required by the model.[24]

[23] Shaikh et al., 2020

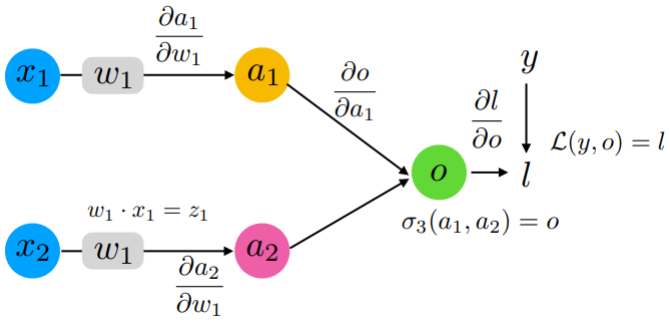
[24] Datta et al., 2021

3.4 Dropout-Rates

Another useful trick to avoid overfitting, hence, enhance generalization of deep CNNs is the introduction of dropout-rates. There are several versions of achieving dropout such as stochastic and proportional. In proportional dropout, a dropout rate is specified ($0 < p_{off} < 1$), at which nodes are set to 0 throughout the forward-pass with probability p_{off}. For stochastic dropout, the probabilistic dropout rate follows either a normal or uniform distribution, so $p_{off}\ \mathcal{N}(\mu, \sigma) \vee p_{off}\ \mathcal{U}(a, b)$.[25,26]

3.5 Loss-Backpropagation

The backpropagation procedure in convolution layers is very similar to standard feed-forward NNs, however, with an additional *weight sharing constraint*. Hereby, I illustrate the steps using a highly simplified example:
Given two different inputs x_1 and x_2, and using the same weight w_1, we still obtain two distinct activations α_1 and α_2 which ultimately lead to the output o.



Credit: *Sebastian Raschka*

Figure 8: Backpropagation with parameter-sharing

In order to backpropagate and update the parameters, we take the the partial derivative of the selected loss-function $\mathcal{L}$ w.r.t. to the weight parameter:

$$\frac{\partial \mathcal{L}(y, o)}{\partial w_1}$$

This is achieved through backtracking in the following manner:

[25] a and b are the lower and upper boundaries of the uniform distribution. If $p_{off} < 0$, it is set to 0
[26] Park and Kwak, 2017

1. partial derivative of the loss w.r.t. the output
2. partial derivative of the output w.r.t. to the activation
3. partial derivative of the activation w.r.t. to the weight

Combining this for the upper and lower channel by summation (due to the multivariable chain rule), the backpropagation is completed.

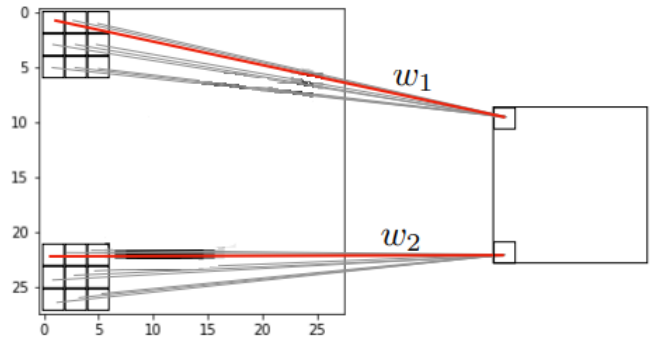
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}(y,o)}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} + \frac{\partial \mathcal{L}(y,o)}{\partial o} \cdot \frac{\partial o}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_2} \tag{7}$$

The left side of the addition in 7 corresponds to the upper channel, the right side to the lower channel.

In a convolution, due to parameter sharing (same feature map):

$$w_1 := w_2 := w_1 - \eta \cdot \left(\frac{\partial \mathcal{L}(y,o)}{\partial w_1} + \frac{\mathcal{L}(y,o)}{w_2} \right), \tag{8}$$

whereby η denotes the learning rate.[27] Equation 8 represents the how the weights will be updated.



Credit: *Sebastian Raschka*

Figure 9: Backpropagation: multivariable chain rule

Making equation 8 more universally applying for each weight parameter in the feature map, it can be written:

$$w_i^* = w_i - \eta \cdot \frac{\partial \mathcal{L}(y,o)}{\partial w_i}$$

This concept of backpropagation can become extensively complex as more convolutional and other types of layers are stacked on top of each other, however, the concepts remain the same. A point here to mention is that the pooling layers (which do not provide any

[27] Raschka, 2021

trainable parameters) are ignored. During backpropagation, the gradients are passed only to the index locations where the maximum (or average) values have been selected during the forward pass. The gradients are then backpropagated through the non-maximum (non-averaged) values, hence, the modifications evoked by pooling are skipped during this procedure.[28]

3.5.1 Loss Function: Categorical-Cross-Entropy

A suited loss function for multi-class predictions is *categorical-cross-entropy* (CEE):

$$\mathcal{L}_{CCE} = -\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(p_{i,c}),$$

with n being the number of samples in the dataset, $C = 7$ representing the seven classes, $y_{i,c}$ are the labels corresponding to the ground truth for sample i and class $c \in [0, 1, \ldots, 6]$, which are binary indicators.[29] The predicted probability (output of softmax function) $p_{i,c}$[30] will be assigned to the corresponding sample i and class c by the model. In backpropagation, the CEE loss function penalizes the learning algorithm according to the log of the predicted probability for the class labels.

3.6 Prediction of Outputs

After receiving the flattened outputs of a dense layer with C output nodes (one node for each class), the array is passed onto the sigmoid activation function, which turns a vector of C numerical elements into a vector of C numerical probabilities which sum up to 1. Although similar, the key difference between softmax and sigmoid is that softmax takes in a vector, while sigmoid works with a scalar. Accordingly, softmax is the appropriate choice when working with more than two classes, given that these classes are *mutually exclusive*.[31] For example, when a lesion image is classified as *melanoma*, then it cannot be simultaneously be classified as *benign keratosis*. In mathematical terms, the softmax function is written as follows:

$$\text{softmax}(\mathbf{x}_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} = \hat{y}_i$$

[28] StakExchange, 2019

[29] 1 if the sample belongs to class c, and 0 otherwise.

[30] $p_{i,c}$ is the final output after the softmax function which is typically applied to the outputs of the last dense layer. It would correspond to o in equation 8.

[31] Wood, n.d.

But a vector of probabilities is not that useful yet. A best practice is to apply the argmax function on the output vector, which picks the maximum probability, and via its index, the corresponding class can be determined. Still, the probability vector may be complemented when interpreting the results, as it discloses the confidence at which the model predicts a given class.

In a nutshell, the training input x^1 is passed into the deep CNN with L layers, which has already learned the weight parameters $w^1, w^2, \ldots, w^{L-1}$. After being processed by the first layer, the output x^2 will be passed onto the next layer and so on. Eventually, the output of the last layer will be x^L which includes the posterior probabilities of the input x^1 for all C classes. Therefore it holds that $x^L \in R^C$. To determine the output in a categorical way (since we work with classes), we apply the argmax function $\underset{i}{argmax}(x_i^L)$[32] which tells the index of the highest *posterior probability*.

$$\text{prediction}_i = \underset{i}{arg\,max}(x_i^L)$$

[32] In Python, I implemented this using np.argmax() function.

4 Process

4.1 Data Engineering

Working with image data often require certain adjustments before used for training.[33] After loading the data into the run time, the images have been resized to 100×100 pixels by using the `resize` function.[34]. Such transformation has been done due to resource limitations. It shall be mentioned that certain detailed features can get lost during this step, and if accelerators like GPUs are available, the rescaling should be done such that minimal, or ideally, no loss of information occurs. In some instances, it might also be useful to crop the image if the target features to look out for are appear in the center of the image.

In the same token, the images data has been augmented by flipping as well as rotating in order to tackle the prevalent class imbalance. Except from the images corresponding to *melanocytic nevus*, every class has been augmented by adding five augmented versions of the image to the respective class. The augmentation procedure is documented in Table 2 below:

Lesion Type	Observations	Augmentation	Total
Melanocytic nevus	6705	None	6705
Melanoma	1113	×6	6678
Benign keratosis-like lesions	1099	×6	6594
Basal cell carcinoma	514	×6	3084
Actinic keratoses	327	×6	1962
Vascular lesions	142	×6	852
Dermatofibroma	115	×6	690
Σ	10015		26565

Table 2: Image augmentation

[33] Image classification is known to be a 2-D multichannel task.

[34] Downsampling is achieved through linear pixel-interpolation. For further details see OpenCV geometric transformations documentation.

Note that through data augmentation, the data set (pre-splitting) contains 26565 images in total. Despite the increased number of training images, this does not mean that all new images contribute to the learning process of a CNN. This is due to the fact that the convolution operation is not naturally *equivariant*, suggesting that transformation like rescaling, flipping and rotating does not necessarily induce weights to update, as only the location of a feature changes, but not the relative location to adjacent pixels.[35] In response to this, the number of epochs has been limited to 100 to avoid overfitting.

In the same token, the images have been labelled according to the class they belong to. The class labelling can be taken from the metadata accompanying the HAM10000 dataset. The images are stored in the input variable X and the class labels in the target variable y. In the final step of the data engineering phase, the variables have been split up into 67% training and 33% for testing.[36] Random sampling has been performed, whereby the classes have been stratified to maintain their class share, which might accidentally get lost if not done so. As we still face some class imbalances, class weights have been computed and used in the training process. During training, the model is deemed to adjust the loss according to these class weights.[37] The loss contribution of each sample is multiplied by the corresponding class weight. As a result, samples from the minority classes will have a higher impact on the training process, and vice versa for samples from the majority classes.

4.2 Construction of Models

Both models I created and deployed are inspired by the popular VGG-16 model which was first proposed by Simonyan and Zisserman (2014) from the Oxford University. With this model, the two researchers ranked first and second place in the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) on object detection and image classification tasks, respectively The VGG-16 model can be described as a deep convolutional neural network architecture widely known for its simplicity while providing outstanding performance in computer vision tasks. The model consists of a total of 16 layers, among

[35] Goodfellow, Bengio, and Courville, 2016

[36] We obtain 17798 training and 8767 test samples.

[37] The computed weights for the seven classes are represented in the following dictionary:

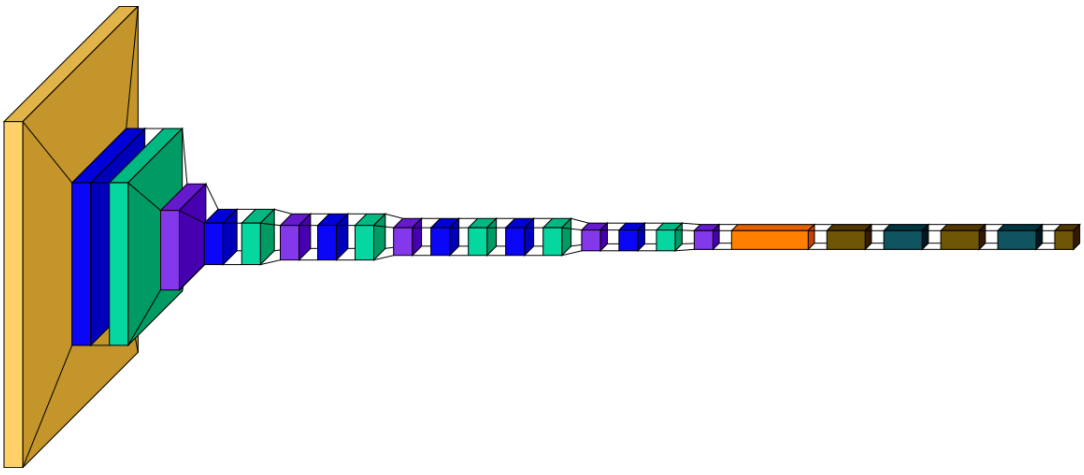
```
{0:  0.57, 1:  0.57, 2:  0.58, 3:  1.23, 4:  1.93, 5:  4.45, 6:  5.5}
```

which 13 are convolutional layers and 3 fully connected (dense) layers. Relatively small 3×3 kernels are employed throughout the network, making the network deeper while limiting the amount of trainable parameters. Sequences of several convolutional layers[38] are followed by max pooling layers for the sake of spatial downsampling.[39] The model's construction puts emphasis on altering the depth while maintaining a small receptive field. This shall result in improved feature representation and discrimmination capabilities, that is, the model's capacity to learn and extract meaningful information from the input images, allowing for enhanced differentiating between the represented classes.

The created models relate to the VGG-16 model in that sense that the number of kernels grow by incrementing the power of 2 by 1 with increasing depth. The first layer has $2^5 = 64$ filters, the second sequence has $2^6 = 128$ filters and so on.

4.3 Model Architecture

Each convolution layer[40] is followed by batch normalization before max pooling layer, except from the fourth sequence. The reason for this is to prevent the dimensions from shrinking too much. After the last max pooling layer, the output has dimensions 4×4×512, and has been flattened out to a one-dimensional array (a vector with dimension 4×4×512= 8192) to be passed onward to the two fully connected dense layers with 4096 neurons each. These dense layers are followed by a dropout regularization with a dropout rate of 50%.



[38] The number of filters progressively increases after each sequence of convolutional layers.

[39] Simonyan and Zisserman, 2015

[40] Same-padding has been used in all layers, strides vary from layer to layer, the number of filter maps increases progressively. For the activation function, ReLU has been used.

Figure 10: Architecture of CNN

Figure 11: Architecture of CNN with Soft-Attention layer

The entire flow chart of the process and the model construction is depicted in Figure 12 below. On the visualization, the model incorporating the soft-attention layer is displayed, the same applies also to the model without soft-attention, but with a minor difference, as the max pooling has been left out after the fifth convolutional layer as it is added after the soft-attention layer to maintain the same dimensions and parameter sizes.

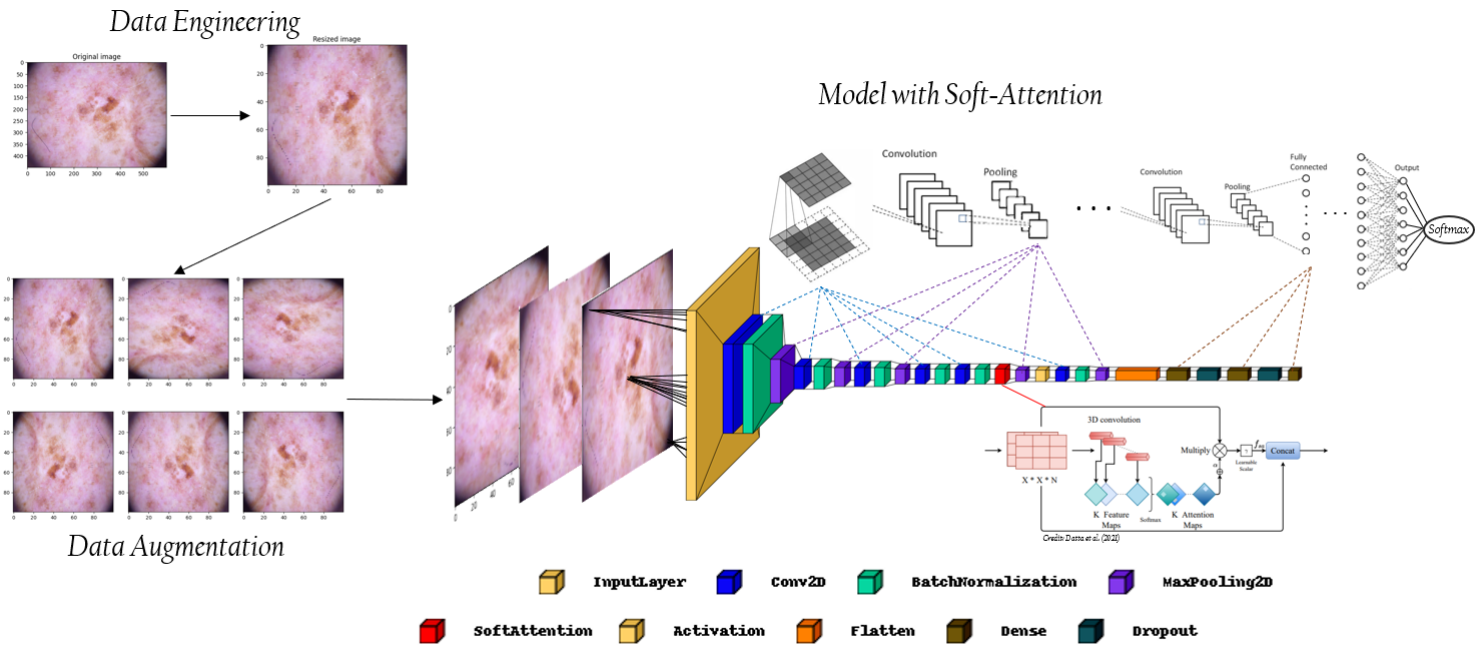


Figure 12: Process and model flow chart

4.4 Training and Hyperparameter Tuning

Specifying the batch size at 32 and the number of epochs at 100 represents a standard, but well established setup when training the model.[41]

4.4.1 Adam-Optimizer

A popular optimizer is provided with Adam (abbreviation for adaptive moments). Adam yields several advantages over other optimizer algorithms:

1. Adaptive learning rates
2. Efficiency
3. Momentum
4. Robustness to hyperparameters

Adaptive learning rates are computed for each parameter which helps with convergence. The individualized learning rates are then maintained over and stored over the iterations, saving computational resources. Bias correction happens by debiasing the decay rates ensuring that they are biased towards zero, especially if the decay rates are close to 1.

[41] The `batch_size` parameter in Keras is used for defining the number of samples (images) processed before the parameters are updated. The `epochs` parameter is used for telling the algorithm, how often the entire training data is feed into the network.

This debasing step helps to counteract the initial bias in the estimates of the first and second moments.

In a nutshell, we can say that Adam combines the advantages of fellow optimizers Momentum, AdaGrad, and RMSProp by incorporating the momentum concept to accelerate convergence through accumulated gradients, the adaptive learning rates of AdaGrad to individually adapt the learning rates for each parameter, and the root mean square (RMS) gradients of RMSProp for normalizing the gradient updates.[42]

Adam behaves like a very heavy ball that rolls downward fast, but with high friction towards diminishing slopes. This behavior allows it to converge efficiently and reliably towards the optimal solution while navigating through different regions of the loss landscape.

In most scenarios, Adam is the go-to optimizer algorithm due to its robustness and versatility. It performs well with default parameter values and requires less manual tuning compared to other optimization algorithms,[43] explaining its popularity amongst deep learning applications.

I use a learning rate $\eta = 0.0001$, beta values of $\beta_1 = 0.9$ for the first and $\beta_2 = 0.999$ for the second moment decay rate, an epsilon value of of $\epsilon = 0.001$. When η (determining the step size of parameter updates) is set to high, the model might overshoot and miss the optimum, when set to low, the model fails to converge. β_1 determines the weight given to the recent gradient in the current update stage. Meanwhile, β_2 defines the exponential decay rate of the second moment, allocating the emphasis towards recent squared gradients, the closer set to 1.

[42] Kingma and Ba, 2017

[43] Goodfellow, Bengio, and Courville, 2016

5 Evaluation

The authors of the original paper use, among others, precision, recall and accuracy scores. Also, the F1-score has been taken into consideration for evaluating the models' performances. These metrics build upon the confusion matrix, which simplifies the concept of possible outcomes:

		Predicted	
		True	**False**
Actual	**True**	True Positive	False Negative
	False	False Positive	True Negative

Table 3: Confusion Matrix

Precision measures the proportion of correctly predicted positives, minimizing false positives. Recall, on the other hand, calculates the proportion of correctly predicted positives out of actual positives, minimizing false negatives. The F1-score thereby merges precision and recall into a single measure by taking their harmonic mean.[44] Precision is advantageous for identifying low false positives, while recall is better in minimizing false negatives. The F1-score finds an equilibrium among these two metrics, thereby offering special utility for imbalanced datasets. However, it doesn't account for true negatives and may not optimize the threshold. A formal representation of evaluation metrics is provided below:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{False Positives} + \text{True Negatives} + \text{False Negatives}}$$

$$= \frac{\text{True Predictions}}{\text{Total Predictions}}$$

[44] Wikipedia contributors, 2023

The performance measures of both models on the HAM10000 dataset is provided in Table 4. For the sake of comparison, the statistics of best preforming model from Datta et al., a VGG16 model with integrated Soft-Attention mechanism, have been added as well. Note that Datta et al. used 85% for training, while I used only 67%, hence, the approximately ten times larger number of available testing samples.

Class	Precision	Recall	F1-Score	Support
nv	0.92	0.88	0.90	2213
mel	0.84	0.96	0.89	2204
bkl	0.92	0.81	0.86	2176
bcc	0.95	0.80	0.87	1018
akiec	0.86	0.87	0.86	647
vasc	0.97	0.99	0.98	281
df	0.56	1.00	0.72	228
Accuracy	-	-	0.88	8767
Macro Avg	0.86	0.90	0.87	8767
Weighted Avg	0.89	0.88	0.88	8767

(a) CNN

Class	Precision	Recall	F1-Score	Support
nv	0.97	0.79	0.87	2213
mel	0.86	0.94	0.90	2204
bkl	0.89	0.93	0.91	2176
bcc	0.93	0.97	0.95	1018
akiec	0.89	0.95	0.92	647
vasc	0.95	1.00	0.97	281
df	0.93	0.97	0.95	228
Accuracy	-	-	0.91	8767
Macro Avg	0.92	0.94	0.92	8767
Weighted Avg	0.91	0.91	0.91	8767

(b) CNN-SA

Class	Precision	Recall	F1-Score	Support
nv	0.95	0.95	0.95	663
mel	0.43	0.68	0.53	34
bkl	0.63	0.44	0.52	66
bcc	0.62	0.81	0.70	26
akiec	0.70	0.61	0.65	23
vasc	1.00	0.90	0.95	10
df	0.50	0.33	0.40	6
Accuracy	-	-	0.88	828
Macro Avg	0.69	0.67	0.67	828
Weighted Avg	0.88	0.88	0.88	828

(c) VGG16+SA from Datta et al. (2021)

Table 4: Classification reports for CNN, CNN-SA and VGG16+SA on HAM10000

5.1 Evaluation on ISIC 2019 Dataset

For further validation, the models were deployed in predicting partially new images from the ISIC 2019 dataset.[45] The outcomes are again provided in Table 5 below:

[45] For further information on the adjustments performed on the data, see 2.

Class	Precision	Recall	F1-Score	Support
nv	0.85	0.88	0.87	280
mel	0.73	0.61	0.67	87
bkl	0.47	0.38	0.42	39
bcc	0.77	0.52	0.31	67
ak	0.35	0.34	0.35	7
vasc	1.00	0.86	0.92	7
df	0.35	0.75	0.48	4
Accuracy	-	-	0.70	491
Macro Avg	0.58	0.52	0.55	491
Weighted Avg	0.62	0.60	0.61	491

(a) CNN

Class	Precision	Recall	F1-Score	Support
nv	0.82	0.97	0.89	280
mel	0.76	0.57	0.65	87
bkl	0.62	0.75	0.68	39
bcc	0.94	0.45	0.61	67
ak	0.50	0.42	0.45	7
vasc	1.00	0.95	0.98	7
df	0.50	0.50	0.50	4
Accuracy	-	-	0.72	491
Macro Avg	0.64	0.60	0.62	491
Weighted Avg	0.63	0.63	0.63	491

(b) CNN-SA

Table 5: Classification reports for CNN and CNN-SA on ISIC 2019 dataset

6 Discussion

In medical settings, it sounds more appropriate to assign higher meaning to the recall or F1-score, as rather accepting that a few too many screenings are diagnosed as being cancerous is intuitively easier to justify than knowing that the model would miss out on an actual case where treatment is due as early as feasible.

At the first glance, when looking at the scores of each individual class, no clear winner can be determined. What stands out, however, is that the model incorporating the soft-attention unit outperforms the regular CNN for rather underrepresented classes *akiec*, *vasc* and *df*. Except from the class *nv*, CNN-SA yields better or at least equal scores than CNN with improvement of 3% in terms of weighted average w.r.t. F1 score. Now accounting the accuracy, the macro and weighted averages, CNN-SA overall gives more accurate predictions across classes. Surprisingly, both models perform better than VGG16+SA, which may be due to several factors.[46]

Regarding the ISIC 2019 dataset, the models did not as well as on HAM10000. Nonetheless, the results again appear to be in favor of CNN-SA. This time, precision, recall and F1-score are marginally better for CNN. To quantify the improvement achieved by adding the soft-attention unit to the network, an 2% increase has been achieved in the F1-score.

Considering these scoring-criteria, the implementation of a soft-attention layer can thus be seen as a success, ultimately verifying the results of Datta et al. (2021). Surprisingly, both models perform considerably well on the underrepresented classes, suggesting that augmentation and weighting worked.

[46] Most likely, the data engineering process has severely affected the outcome. While Datta et al. scale the images to 224×224, I performed downsampling to 100×100 pixels, causing the image to loose resolution.

7 Conclusion

By running this project, the results of increased performance by incorporating the soft-attention mechanism can be validated. Overall, identifying the winner solutions of the classification challenges on Kaggle shows that there are various suitable approaches in terms of model architecture to achieve impressive outcomes. Concerning the future of medical healthcare, especially for diagnosis and early disease detection, implementing Deep Learning models project a promising innovation. This does not necessarily mean that doctors will be totally replaced by any means, but rather that these models serve as assisting or complementary tools for medical staff, potentially lowering human error while altering productivity or even offsetting personnel bottlenecks.

For future works, it might be necessary to preserve as high of an image resolution as possible. Furthermore, once appropriate computing resources are available, it can be experimented around with incorporating several soft-attention layers into the CNN architecture. Eventually, ISIC will release further, more comprehensive skin lesion data sets, containing even more classes and samples. Paired with the right equipment, knowledge and growing amount of data (and patient outcomes), with computer vision, we face a promising technology with the potential to tremendously improve healthcare in the upcoming years and decades.

References

Amidi, Afshine and Shervine Amidi (2018). *Convolutional neural networks cheatsheet.* URL: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks (visited on 06/22/2023).

Campos, Víctor, Brendan Jou, and Xavier Giró-i-Nieto (2017). *From pixels to sentiment: fine-tuning cnns for visual sentiment prediction.* In: *Image and vision computing* 65, pp. 15–22. ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2017.01.011. URL: https://www.sciencedirect.com/science/article/pii/S0262885617300355.

Cassidya, Bill et al. (2021). *Analysis of the isic image datasets: usage, benchmarks and challenges.* In: *Computerized medical imaging and graphics* 93, p. 101966. ISSN: 0895-6111. DOI: https://doi.org/10.1016/j.compmedimag.2021.101966. URL: https://www.sciencedirect.com/science/article/pii/S1361841521003509.

Datta, Soumyya Kanti et al. (2021). *Soft-attention improves skin cancer classification performance.* arXiv: 2105.03358 [eess.IV].

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning.* MIT Press, pp. 326–366. ISBN: 9780262035613.

Ioffe, Sergey and Christian Szegedy (2015). *Batch normalization: accelerating deep network training by reducing internal covariate shift.* In: *Arxiv preprint arxiv:1502.03167.*

Kingma, Diederik P. and Jimmy Ba (2017). *Adam: a method for stochastic optimization.* arXiv: 1412.6980 [cs.LG].

Kuo, C-C Jay (2016). *Understanding convolutional neural networks with a mathematical model.* In: URL: https://arxiv.org/pdf/1609.04112.pdf.

N., Yashwanth (Sept. 2020). *What is batch normalization?* In: URL: https://towardsdatascience.com/what-is-batch-normalization-46058b4f583 (visited on 06/22/2023).

Pandey, Abhishek Kumar (2020). *Convolution, padding, stride, and pooling in cnn.* In: URL: https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26 (visited on 06/22/2023).

Park, Sungheon and Nojun Kwak (2017). *Analysis on the dropout effect in convolutional neural networks.* In: *Computer vision – accv 2016.* Ed. by Shang-Hong Lai et al. Cham: Springer International Publishing, pp. 189–204. ISBN: 978-3-319-54184-6.

Raschka, Sebastian (2021). *Introduction to convolutional neural networks*. URL: https://sebastianraschka.com/pdf/lecture-notes/stat453ss21/L13_intro-cnn__slides.pdf (visited on 06/26/2023).

Shaikh, Mohammad Abuzar et al. (Oct. 2020). *Attention based writer independent verification*. In: *2020 17th international conference on frontiers in handwriting recognition (ICFHR)*. IEEE. DOI: 10.1109/icfhr2020.2020.00074.

Simonyan, Karen and Andrew Zisserman (2015). *Very deep convolutional networks for large-scale image recognition*. arXiv: 1409.1556 [cs.CV].

Skalski, Piotr (Apr. 2019). *Gentle dive into math behind convolutional neural networks*. URL: https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9 (visited on 06/19/2023).

StakExchange (2019). *Max pooling has no parameters and therefore doesn't affect the backpropagation*. URL: https://datascience.stackexchange.com/questions/55352/max-pooling-has-no-parameters-and-therefore-doesnt-affect-the-backpropagation (visited on 06/26/2023).

Tschandl, Philipp, Cliff Rosendahl, and Harald Kittler (2018). *The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions*. In: *Scientific data* 5.1, pp. 1–8. DOI: https://doi.org/10.1038/sdata.2018.161.

Wikipedia contributors (2023). *F-score — Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/w/index.php?title=F-score&oldid=1148225663 (visited on 06/24/2023).

Wood, Thomas (n.d.[a]). *Convolutional neural network*. URL: https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network (visited on 06/15/2023).

– (n.d.[b]). *What is the softmax function?* URL: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer (visited on 06/23/2023).

A Resources

A.1 Online Resources

- Kaggle
- Google Colaboratory
- Jupyter
- Anaconda

A.2 Libraries & Code Documentation

- python 3.8.10
- pandas
- numpy
- matplotlib
- opendatasets
- tensorflow/keras
- sklearn
- cv2
- SoftAttention [Datta et al. (2021)]

A.3 Code

Artifacts and supplementary material are made available on GitHub (click link), and may be used used for the intend of replication. The data can be found at the ISIC-challenge archive, and needs to be loaded into the runtime. Apart from the setup (including installation of packages), all code cells can be run hierarchically by one click and will thereafter execute automatically, until all cells have been finished. Training of a model (similar to the ones proposed) can be expected to take up around three to four hours, when using the free cloud GPU in a Kaggle or Colab Notebook.

```
@misc{CNNKoefler2023,
      title={Skin Cancer Classification - Attention to The Detail},
      author={K{ö}fler, Markus},
      month={7},
      year={2023},
      url={https://github.com/MarkusStefan/DeepCNN_SoftAttention}
     }
```