Developing and Optimizing a Convolutional Neural Network for Skin Cancer Classification

Khagan Mammadov (2669723),
Wouter van den Broeke (2709623),
Adriana Gabriela Alexandru (2707285),
Petar Petrov (2696187)

Abstract

One of the most lethal forms of cancer is skin cancer. The likelihood of a full recovery from skin cancer depends heavily on early diagnosis and treatment. Given the current shortage of dermatologists around the globe, dealing with healthcare systems might lead to longer waiting times for patients, overworked specialists, and even untreated conditions. One way to make diagnosis easier is using the deep learning approach known as a Convolutional Neural Network (CNN). This paper aims to develop and optimize a CNN to predict and classify seven types of skin lesions. To achieve this, we train a deep learning model on the MNIST: HAM10000 dataset, which consists of 10015 labeled images of pigmented skin lesions. Consequently, our research may aid in the differentiation of various forms of skin cancer, thereby preventing unnecessary biopsies, saving lives, and lowering expenses for patients, dermatologists, and healthcare providers.

1 Introduction

In recent years, rapid advancements in artificial intelligence and deep learning have enabled technology to improve the quality of medical care. Some of the applications of deep learning in the biomedical field are disease diagnosis, patient risk identification, electronic health record, genomics, and drug development [1]. According to the World Health Organization [2], skin cancers are the most common group of cancers diagnosed worldwide, with more than 1.5 million new cases estimated in 2020. In 2020, an estimated 325 000 new cases of melanoma were diagnosed worldwide, and 57 000 people died from the disease.

Deep learning models have shown great promise in improving the accuracy and efficiency of skin cancer diagnosis by analyzing images of skin lesions and detecting patterns that may indicate the presence of cancer. As a contribution to the research in the healthcare industry, this paper aims to find an optimal model configuration to predict the skin cancer type based on images of pigmented skin lesions.

1.1 Research question

How can a CNN be optimized to accurately classify different types of skin cancer using dermatoscopic images?

1.2 Literature review

Deep learning has been widely used to assist diagnosticians in various patient care and intelligent health systems. CNNs are one of the best-performing models in object detection and classification tasks. By mastering highly discriminative features when being practiced end-to-end in a controlled manner, CNNs remove the need for manually handcrafting features.

Applications of CNNs include image classification, autonomous driving, biometric authentication, natural language processing, medical imaging, and many more. Due to their demonstrated capabilities in computer vision tasks, CNN architectures attract interest in many domains, and medical diagnosis is no exception. Medical research has used this type of deep learning architecture for diabetic retinopathy [3] and detecting lymph nodes in women with breast cancer [4].

1.3 Approach

A crucial element of deep learning is meaningful data. In this study, the publicly available Skin Cancer MNIST: HAM10000 dataset [5] is used to train and test the model for the automated diagnosis of pigmented skin lesions.

To develop the CNN, we utilize fundamental deep learning concepts such as convolutions, pooling, and activation functions. We then use trial and error to optimize the CNN.

2 Data analysis

2.1 Dataset

The CNN is trained on the MNIST: HAM10000 dataset [5]. HAM10000 is a collection of dermatoscopic images from different populations, acquired and stored by distinct modalities. The model had to classify the images by skin cancer type, accounting for seven classes.

Label	Description
akiec	actinic keratoses
bcc	basal cell carcinoma
bkl	benign keratosis-like lesions
df	dermatofibroma
mel	melanoma
nv	melanocytic nevi
vasc	vascular lesions

Table 1: Data labels.

More than 50% of lesions are confirmed through histopathology (`histo`), and the ground truth for the rest of the cases is either follow-up examination (`follow_up`), expert consensus (`consensus`), or confirmation by in-vivo confocal microscopy (`confocal`). The dataset includes lesions with multiple images, tracked by the `lesion_id` column within the `HAM10000_metadata.csv` file.

2.2 Data preprocessing

Before feeding the data to the model for training, we performed two main preprocessing steps: resizing and normalization.

First, we resized the images from their original size of 600x450 pixels down to 40x30 pixels. By resizing, we reduced the dimensionality of the images, making them easier and faster to process. Furthermore, the model showed worse performance when using the images in their original size, due to overfitting. Among other methods, reducing the data's dimensionality is a method commonly used to combat overfitting [6].

Second, we normalized our images by subtracting the mean and dividing by the standard deviation. This step ensures that our data is standardized across the dataset, such that the images have similar brightness, contrast, and color.

After preprocessing, we split the data into training, testing, and validation sets, with a split ratio of 70/20/10, respectively. Splitting the data is essential to evaluate the performance of our model by using the validation data to fine-tune the hyperparameters and the test set for the final evaluation.

At this point our labels are still represented as strings, but our model requires numerical input. Therefore, we convert our classes into numerical ones using one-hot encoding. One-hot encoding ensures that the model makes no assumptions about ordering based on a numeric value, unlike integer encoding, where each class is represented by a single integer.

2.3 Class imbalance

Figure 1 illustrates the class imbalance in the dataset, where `nv` has the most entries by a wide margin.

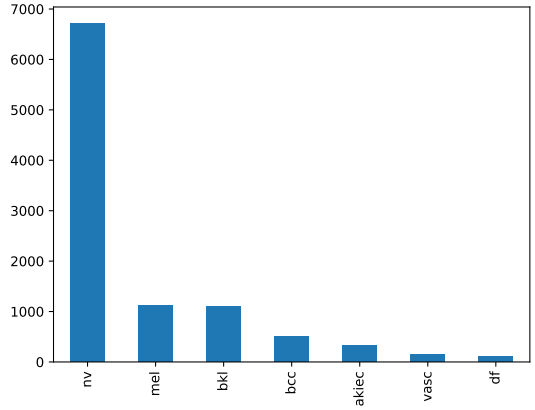

Figure 1: Data distribution.

We considered two approaches to combat the class imbalance: the Synthetic Minority Over-sampling Technique (SMOTE) and Keras' `ImageGeneratorClass`.

2.3.1 SMOTE

SMOTE generates synthetic data for the minority classes by creating new data points based on existing data. The algorithm selects one of the `k`-nearest neighbors of a randomly selected minority class sample and creates a new sample by interpolating between the two selected samples, repeating it until we reach the desired number of minority class samples.

SMOTE can improve the performance of machine learning models with class imbalance by balancing out the class distribution [7]. However, this approach introduces noise to the data and can be detrimental in cases where the minority class is well-represented in the dataset.

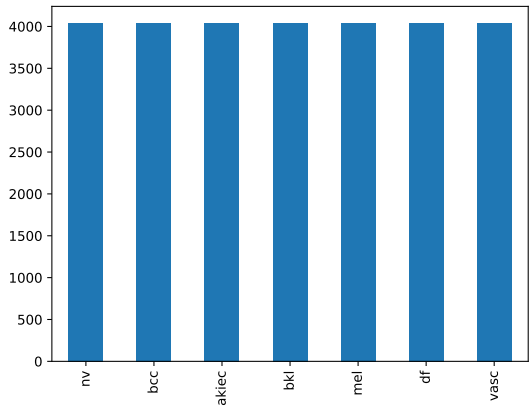

Figure 2: Training data distribution after applying SMOTE.

2.3.2 `ImageDataGenerator`

The `ImageDataGenerator` class is a tool provided by Keras, which allows for performing data augmentation on the images during training. The class does this by loading the images and applying a set of predefined image transformations, after which the samples are batched and fed to the model. This approach allows for on-the-fly augmentation and increases the size and diversity of the set.

However, `ImageDataGenerator` also comes with downsides; it uses more memory than SMOTE and significantly increases the training time, as the augmentation step is done during training. This approach can also lead to overfitting if the augmentations are too aggressive or do not fit the dataset.

3 Architecture

CNNs typically consist of several layers. These layers include an input layer, one or more convolutional layers that extract relevant features from the image, activation functions that introduce non-linearity into the model, pooling layers that reduce the spatial size of the feature maps, and fully connected layers that produce the final output of the model. Together, these layers form a powerful and flexible architecture that can learn complex representations of image data.

3.1 Input layer

The CNN receives the raw image data from the input layer. Images are a 3D array of pixels, with height, width, and number of color channels.

3.2 Convolution layer

In the convolutional layer, we apply filters to the input image, which detect features at different locations. The filters are small matrices of weights that convolve across the input image. Filters produce feature maps that highlight the important features present in the input image. Typically, multiple features are applied per layer, producing multiple feature maps representing different types of features. Convolutional layers are the core of CNNs as they enable the network to learn relevant features from images.

3.3 Pooling layer

Pooling layers use downsampling to reduce the spatial size of the feature maps while retaining the most important features, reducing the number of parameters, and helping prevent overfitting.

To achieve this, pooling layers divide the input feature map into non-overlapping regions, and then compute a summary for each region. The summary is then used to create a smaller feature map with reduced dimensions.

CNNs use different pooling methods, such as L2, Max, and Average pooling, to combine values within each pooling window and reduce the spatial size of the feature map.

3.4 Batch normalization layer

The batch normalization layer normalizes the input using the mean and standard deviation of the current batch. The layer reduces the internal covariate shift. Internal covariate shift is a phenomenon that occurs when there is a change in activations due to a change in parameters during training [8].

Preventing covariate shift can improve a model's performance on new data by reducing its reliance on specific features of the training data and allowing it to better capture underlying patterns. This can result in a more accurate and computationally efficient model.

3.5 Dropout

The dropout layer is a regularization layer that randomly selects a predetermined fraction of the input to drop out. It is usually added to a model to prevent overfitting. By randomly dropping out input values during training, the layer forces the model to learn more general features in the data, since it has fewer data-points to work with. However, a large

dropout rate may cause the model to lose access to too much information, reducing performance.

3.6 Flatten layer

The flatten layer is a simple layer that takes the output of a previous layer and flattens it into a one-dimensional vector.

3.7 Dense layer

The dense layer, also known as the fully connected layer, connects every neuron in the current layer to every neuron in the previous layer and computes the output based on the result of a (non-linear) activation function. It requires a one-dimensional array of data. Therefore, the dense layer is often preceded by a flattened layer. The layer can be used to take a one-dimensional array of any length and output a one-dimensional array of another length.

Consequently, the dense layer is commonly used as the last layer of classification models, by specifying the output length to be the number of classes for the softmax activation function to choose from. The softmax function converts the input into a probability distribution, from which we can derive the target class the model is most confident in being the class of the input image. In our case, the last layer is always a dense layer with the output of a vector of length seven, one for each of the skin cancer types, using the softmax activation function.

4 Baseline model

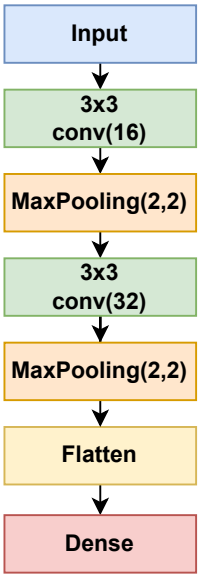

Figure 3: Baseline model architecture.

4.1 Motivation

The baseline model is a simple and straightforward model that serves as a benchmark for the optimized model. The baseline model was inspired by [9] and [10]. It provides us with a starting point for building and evaluating a more complex model.

4.2 Architecture

As illustrated in Figure 3, the baseline model is comprised of two pairs of convolutional and pooling layers followed by a flatten and dense layer. The activation function used for the convolutional layers is the Rectified Linear Unit function (ReLU), as it is computationally efficient and is able to solve the vanishing gradient problem, which arises when the gradient becomes excessively small during backpropagation and poses a challenge for the network to update its weights. The activation function used for the dense layer is the softmax function, as it is able to generate a probability distribution across the different classes, allowing the model to make more accurate predictions.

4.3 Optimization

An optimization present in both the baseline and optimized models is Keras' `ReduceLROnPlateau` function. `ReduceLROnPlateau` is a commonly used function in machine learning that reduces the learning rate of the optimizer when a monitored metric has stopped improving for a specified number of epochs. This can help the optimizer find a better minimum in the loss landscape and improve the convergence of the model.

`ReduceLROnPlateau` is used in both the baseline and optimized models to monitor the validation loss and validation accuracy and improve the performance of the model.

5 Optimized model

5.1 Motivation

The baseline model serves a starting point for further optimizations, so it is rarely the optimal model. The baseline model does not extract enough detail from the input images, resulting in poor performance on minority classes in the dataset. Thus, it is essential to explore various architectures and optimization techniques to enhance the model's performance and extract more complex features from the input data.

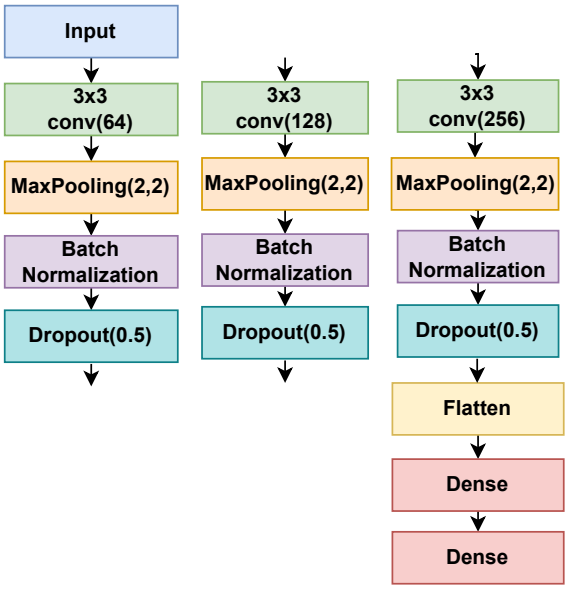

Figure 4: Optimized model architecture.

5.2 Architecture

As illustrated in Figure 4, the optimized model builds on top of the baseline model by introducing a number of optimizations.

5.3 Optimization

First, a larger number of convolutional layers is used, each with a larger number of filters, allowing the model to extract more features, particularly complex features, from the input image.

Second, batch normalization and dropout layers are used, which help combat overfitting and improve the model's accuracy when presented with new data.

Third, the leaky ReLU activation function is used. The Leaky ReLU activation function improves the performance of the model by addressing the dying ReLU problem, which can occur when ReLU activations become stuck at zero during training.

Finally, a larger number of dense layers is used, which allows the model to learn more complex relationships between the input and output data.

6 Results

6.1 Class imbalance

To decide on the optimal oversampling technique to use, we evaluated the performance of the baseline model under SMOTE, `ImageDataGenerator`, and no oversampling, and compared their results. It should be noted that accuracy was not used as a metric for model evaluation, as class imbalance was present in the case of no oversampling. The models were trained for 30 epochs with the Adam optimizer using categorical cross-entropy as the loss function.

Metric	Base	SMOTE	IDG
F1-score	0.759	0.701	0.741
ROC-AUC (OvR)	0.924	0.877	0.905
Sec. per epoch	2.033	7.466	4.733

Table 2: Baseline model oversampling performance metrics. The ROC-AUC is computed using the One vs Other (OvR) method, where each class is compared to the others individually.

As shown in Table 2, the baseline model performed best in the case of no oversampling. SMOTE performed the worst, producing the lowest weighted average F1-score, lowest average ROC-AUC score and the longest time to train. `ImageDataGenerator` (IDG) produced better results, but did not offer any advantages over using no oversampling.

Ultimately, applying either oversampling technique led to longer training times without any performance gain in return.

6.2 Model evaluation

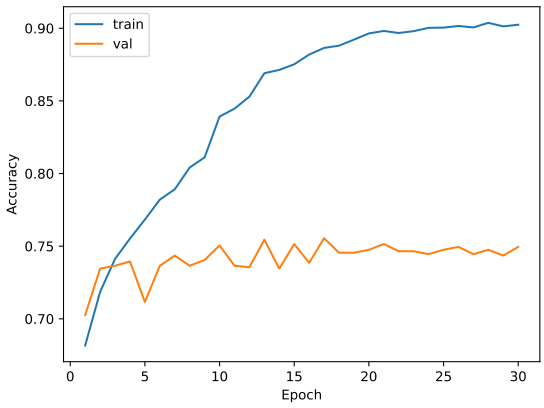

Figure 5: Baseline model history.

As illustrated in Figure 5, the validation accuracy of the baseline model plateaued past the first few epochs, while the training accuracy steadily increased. This indicates that the baseline model is prone to overfitting on the training data and is unable to generalize on the input data. It is important to note that training accuracy and validation accuracy are not part of the final evaluation, as the

models are ultimately evaluated on test data. However, they do provide a general overview of how the models behave during training.

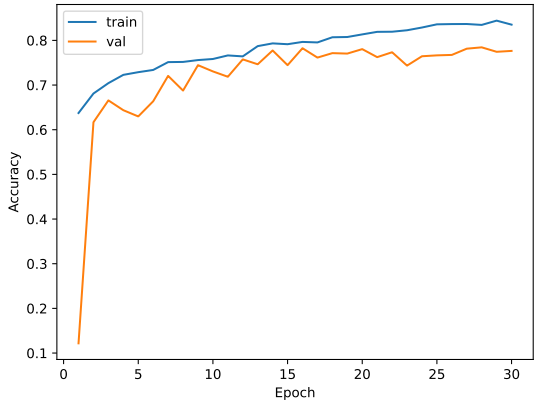

Figure 6: Optimized model history.

Figure 6 illustrates the improvement of the optimized model over the baseline model. The validation accuracy of the optimized model closely followed the training accuracy. This indicates that, unlike the baseline model, the optimized model is not prone to overfitting on the training data. This can primarily be attributed to the addition of dropout layers, as their primary function is to prevent overfitting.

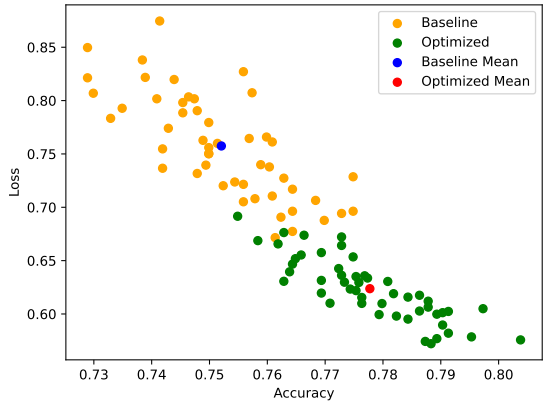

Figure 7: Baseline model and optimized model test loss and test accuracy scatter plot.

Figure 7 and Table 3 present a direct comparison of the test loss and test accuracy of the baseline model to that of the optimized model. The models were each trained 50 times for 30 epochs with the Adam optimizer using categorical cross-entropy as the loss function. The test loss and test accuracy were recorded for each run. As both models have been trained on a dataset with significant class imbalance, test loss and test accuracy were ultimately not used as an evaluation metric. However, they do provide general insight into how the models perform relative to one another.

Metric	Baseline	Optimized
Mean Loss	0.757	0.624
Median Loss	0.755	0.621
Minimum Loss	0.672	0.572
Mean Accuracy	0.752	0.778
Median Accuracy	0.751	0.776
Maximum Accuracy	0.775	0.804

Table 3: In-depth scatter plot statistics.

Table 4 illustrates the results of our model evaluation. The models were trained for 30 epochs with the Adam optimizer using categorical cross-entropy as the loss function. The metrics were collected after evaluating the model on the test data.

Metric	Baseline	Optimized
Precision	0.730	0.778 (+6.5%)
Recall	0.744	0.781 (+5.0%)
F1-score	0.759	0.775 (+2.1%)
ROC-AUC (OvR)	0.924	0.945 (+2.3%)
Sec. per epoch	2.033	3.300 (+62.3%)

Table 4: Performance metrics for the baseline model and the improved model. The ROC-AUC is computed using the One vs Other (OvR) method, where each class is compared to the others individually.

Figures 8 and 9 show that the baseline model performs worse than the optimized model when classifying images that belong to the minority class.

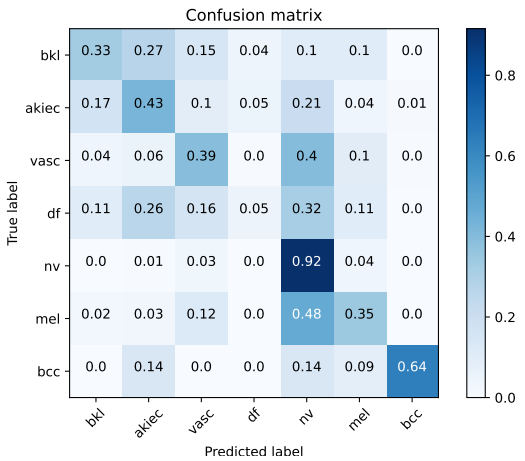

Figure 8: Baseline model confusion matrix.

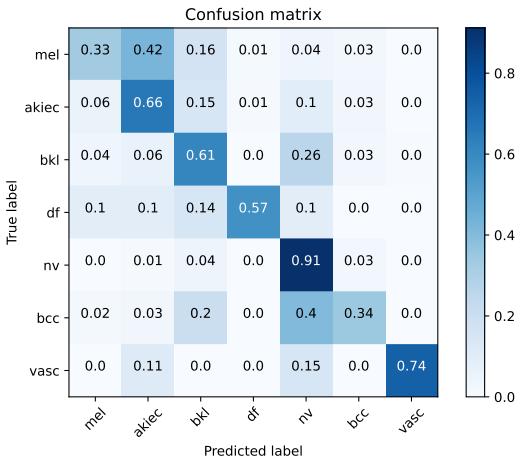

Figure 9: Optimized model confusion matrix.

Based on these results, we can conclude that the optimized model performs better across the board, but takes longer to train. However, depending on the use case and the environment the model is deployed in, it may be more fitting to use the baseline model if training time is an important factor.

7 Discussion

Developing and optimizing the models presented us with several obstacles that we were unable to fully overcome.

First, we were unable to adapt our models to efficiently work with SMOTE and `ImageDataGenerator`. As a result, oversampling the minority classes had a detrimental effect on the performance of our models. To further tackle this issue, we need to explore more oversampling techniques, such as SMOTE-NC [7], ROS, KDE-SMOTE, and ADASYN [11]. Additionally, collecting more data will also help improve the performance of the model.

Another area for future research is utilizing Keras' `KerasTuner` to further optimize our models. We obtained the hyperparameter values for the optimized model through trial and error. As this is a time-consuming process, we were unable to further fine-tune the hyperparameters of the optimized model. We believe that, given enough time, `KerasTuner`'s random search can select more optimal hyperparameters.

The primary focus of our research was developing and optimizing a single model for skin cancer classification. However, the authors of [4] also directly compared the performance of different pre-trained models (e.g. VGG16, ResNet, and Inception [12]) on the HAM10000 dataset. While their initial attempts achieved worse performance than our baseline model, we could explore the use of transfer learning to leverage pre-trained CNN models by combining them with different layers.

Finally, incorporating clinical information such as age, sex, and medical history could improve the accuracy of the CNN. These areas of research can help improve the performance of our models and make them more useful in clinical practice.

8 Conclusion

To conclude, we were able to develop a CNN and apply several optimizations to improve its performance on the HAM10000 dataset. However, we also recognize that there are additional oversampling techniques and optimization methods we were unable to explore in this study due to time constraints and our limited experience working with Keras. Ultimately, we believe that our efforts have contributed to our understanding of model construction and optimization, and we are eager to continue exploring this field in the future.

References

[1] S. Yang, F. Zhu, X. Ling, Q. Liu, and P. Zhao, "Intelligent Health Care: Applications of Deep Learning in Computational Medicine," *Frontiers in Genetics*, vol. 12, p. 444, 2021.

[2] World Health Organization, "Skin cancer." [Online]. Available: https://www.iarc.who.int/cancer-type/skin-cancer/

[3] A. S. Shete, A. S. Rane, P. S. Gaikwad, and M. H. Patil, "Detection of Skin Cancer using CNN Algorithm," *International Journal of Advance Scientific Research and Engineering Trends*, vol. 6, 2021.

[4] T. M. Alam, K. Shaukat, W. A. Khan, I. A. Hameed, L. A. Almuqren, M. A. Raza, M. Aslam, and S. Luo, "An Efficient Deep Learning-Based Skin Cancer Classifier for an Imbalanced Dataset," *Diagnostics*, vol. 12, 2022.

[5] P. Tschandl, C. Rosendahl, and H. Kittler, "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," *Scientific Data*, vol. 5, 2018.

[6] L. van der Maaten, E. Postma, and J. van den Herik, "Dimensionality Reduction: A Comparative Review," *Journal of Machine Learning Research - JMLR*, vol. 10, 2007.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority oversampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[8] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015.

[9] K. Pai and A. Giridharan, "Convolutional Neural Networks for classifying skin lesions," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, 2019, pp. 1794–1796.

[10] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, 2021.

[11] H. He, Y. Bai, E. Garcia, and S. Li, "ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning," in *Proceedings of the International Joint Conference on Neural Networks*, 07 2008, pp. 1322–1328.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

A Appendix

Source code: https://www.kaggle.com/code/khaganmv/skin-cancer-mnist/notebook