UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Image Classification with different CNN Architectures

Technical Report

Introduction

Skin cancer is one of the most prevalent forms of cancer worldwide, and early detection is essential to improving patient outcomes. Among its various types, melanoma poses a significant threat due to its aggressive nature and high mortality rate when not diagnosed early. In contrast, melanocytic nevus (nevus for short) is a common and benign skin lesion that typically requires no medical intervention. Given that these two conditions can often be visually distinguished by dermatologists, automated image classification techniques present a promising way for scalable and accessible early diagnosis.

The project aims to implement and evaluate several CNN-based models using PyTorch, incorporating techniques such as batch normalization, residual connections, data augmentation, and transfer learning. Performance is assessed on a validation set, with the final evaluation being performed on a separate test set. The report summarizes model architectures, training setup, experimental results, and key insights.

Technical solutions

We begin by introducing our starting point as well as the methods and techniques we used to iteratively improve upon this starting point. The results of these improvements are shown in the next section.

Base CNN

We define our baseline as a simple CNN without data augmentation, batch normalization, residual connections, or transfer learning. Below is a detailed description of this baseline. Every further improvement described within this report, with the exception of the transfer learning model, is an extension of this baseline model. A visual representation of this architecture is depicted in Figure 1.

The input images are saved in RGB format with dimensions $3 \times 128 \times 128$, so the first convolution layer ($conv1$) takes 3 input channels and applies 16 filters of size 3×3. In the second convolutional layer ($conv2$), the number of filters increases to 32.
Each convolutional layer is followed by a ReLU activation and a max pooling operation to reduce the spatial dimensions. After the convolutions and pooling, the resulting feature maps are flattened and passed through two fully connected layers ($fc1$, $fc2$) with 128 and 2 output units, respectively.
The final layer produces logits for binary classification, where the two output values represent the MEL and VN classes. Softmax is applied implicitly via the loss function.

Data Augmentation

We apply data augmentation to the training data using the `torchvision.transforms` module. This introduces greater variability in the input images, helping the model generalize better and reducing the risk of overfitting.

Batch Normalization Network

We use `nn.BatchNorm2d` to normalize the outputs of the convolutional layers before the activation functions. This stabilizes and accelerates the learning process by reducing internal covariate shift.

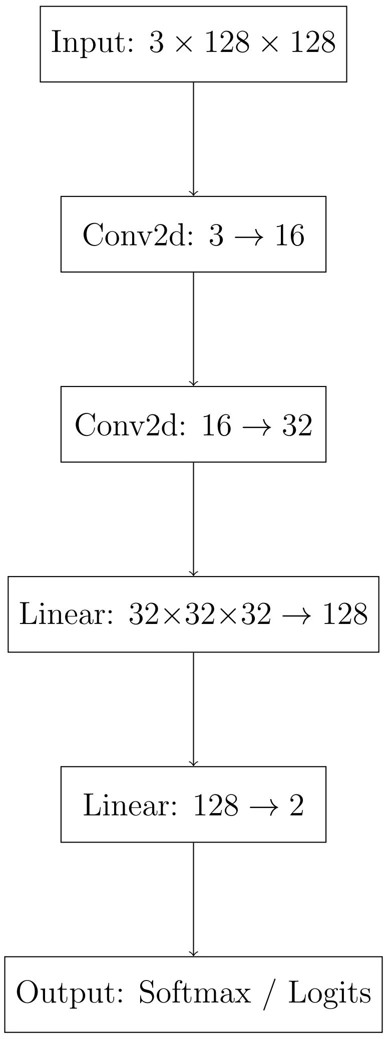
Input: $3 \times 128 \times 128$
Conv2d: $3 \rightarrow 16$
Conv2d: $16 \rightarrow 32$
Linear: $32{\times}32{\times}32 \rightarrow 128$
Linear: $128 \rightarrow 2$
Output: Softmax / Logits

Figure 1: Base Model Architecture

Residual Connections Network

Residual connections introduce shortcut paths that allow information to bypass one or more layers. We implement residual blocks after each convolutional layer to investigate whether this architecture improves gradient flow and overall model performance.

Transfer Learning with pre-trained model

In our transfer learning setup, we use the pre-trained VGG16 model to compare its performance against our custom CNN. We freeze all of the model's parameters and evaluate its classification performance without additional fine-tuning.

Experiments

For this section, we first want to quickly detail our experiment setup. First, we define a training function, which trains the neural network using *Adam* as the model optimizer and *CrossEntropyLoss* as the loss function. This function also computes the training accuracy and loss for the specific epoch. We also implement an evaluation function that computes the validation accuracy for the specific epoch. After some experimentation, we decided on 30 training epochs with early stopping after three epochs of no improvement in the validation accuracy and a standard learning rate of 0.001. Last important note for the training setup is that we used a learning rate scheduler to halve the learning rate every 3 epochs to improve training stability and achieve better convergence. We construct our training setup like this because we want to ensure that we start with a solid base so that we need to change as few variables as possible when testing different models.

We started our experiments by checking for class imbalance. This was not the case, so we proceeded with the analysis. Before detailing our findings, we want to note that the training time seemed to vary depending on the day, as we ran it on the Minerva cluster. All the values mentioned and a full summary of the results can be found in table 1.

As described in the technical solutions section of this report, we will start with the base architecture depicted in Figure 1 and build upon this. All the models we used can be found in the python file attached to this report. After training the base CNN we got our first results. They were: average training time per epoch = 6.32 seconds, best validation accuracy = 0.8427. This was not bad for a start, but we also saw at this stage that training accuracy increased way faster than validation accuracy, and the final training loss was very small (0.0867) leading us to believe that the model was overfitting.

Next, we added data augmentation like rotation and color jitter. This resulted in more stable training, meaning train and validation accuracy grew more similarly, but there was no improvement to accuracy.
After data augmentation, we added batch normalization. This made the training even more stable and increased the best validation accuracy a bit from 0.8427 to 0.8578.

Now, we integrated residual connections which increase training time and worsened accuracy. Instead of the expected improvement, we got a best validation accuracy of 0.8307. This is probably because the CNN we currently use has too few layers. This means that the advantages of the residuals cannot be leveraged and the additional overhead only makes the model worse. Because of this we decided to disregard the residual blocks for now.

At this point, we felt like we had approached the limits of this very simple CNN and decided to add more layers, while keeping the data augmentation and batch normalization. With a third convolutional layer, we got a best validation accuracy of 0.8642; with a fourth, we got 0.8674; and with a fifth, we got 0.8666. Therefore, we chose four layers as our best model for now.

We believe the model to be very stable and optimized at this point, so next, we started to tune the hyperparameter, namely the learning rate. We changed the learning rate from

0.001 to 0.0005. This made the validation accuracy less stable and showed no improvement. So we changed it back to 0.001.

The last improvement we tried for our own model was to try and see if adding residuals could improve our now slightly more complex model, four-layered batch-normalized CNN. Again, we could not see any improvement, but training time increased significantly. For this reason, we disregarded the residuals again.

Lastly, we tried to see if a pre-trained model, namely the VGG16 model, could improve upon our at this stage best validation accuracy of 0.8674. We downloaded the model, froze the parameters, and evaluated the VGG16 model. This resulted in a best validation accuracy of 0.8474, meaning our model seems better tuned for this particular task.

Model	**Last Ep Loss**	**Last Ep**	**Avg Time (s)**	**Best Val Accuracy**
Base	0.0867	10	6.32	0.8427
Base+DataAug	0.3059	17	5.98	0.8427
Every custom model from here on has data augmentation.				
Base+BatchNorm	0.3575	19	7.17	0.8578
Every custom model from here on has batch normalization.				
Base+Residuals	0.3716	12	9.29	0.8307
3 conv layers	0.3481	15	8.88	0.8642
4 conv layers	0.3366	20	8.40	0.8674
5 conv layers	0.3385	19	8.73	0.8666
Every custom model from here on has 4 conv layers.				
LR(0.0005)	0.3770	8	8.66	0.8435
4-Layer+Residuals	0.3755	11	35.12	0.8474
Transf Learning	0.3637	9	34.98	0.8422

Table 1: Comparison of CNN models.

Evaluating Model

As discussed in the previous section, our best model was the CNN that included batch normalization and four convolutional layers. We also saw that the learning rate was optimal at 0.001. Evaluating this model on the test set gave an accuracy of 0.8551. This is slightly worse than our best validation accuracy, which is not unnatural when testing the model on new data and still an acceptable result in our eyes.

Conclusion

This project explored convolutional neural networks for classifying skin lesion images as either melanoma or nevus. Starting from a simple baseline, we applied techniques like data augmentation, batch normalization, and transfer learning to improve model performance. Our best models demonstrated strong accuracy and generalization, showing the effectiveness of deep learning in skin lesion classification.

Limitations

During testing, we gradually expanded the convolution network by adding new layers. We measured that the best accuracy was achieved by 4 layers (0.8674), with a network with 5 layers already having worse accuracy (0.8666). This limitation might have come due to the fact that there is not enough data for the model to learn complex relations or that the number of epochs was too small.

We also need to acknowledge the fact that we ignored the common practice of fine-tuning the pre-trained model. This was not performed and could be part of future work.

Another limitation is that we only use accuracy as a metric for evaluating the tested models. In a more comprehensive study, other metrics like precision, recall, and F1-score could be interesting to look at, especially when dealing with more imbalanced data sets.

With our solution having mainly a didactic purpose, the model recognizes only 2 simplified classes: melanoma and nevus. The original task involved classifying 9 different types of skin marks. Having that in mind, our solution would be of limited use in a real-world environment.

Ethical considerations

Using deep learning for medical diagnosis raises several ethical concerns. First, patient privacy must be protected, especially when handling sensitive medical images. All data used should be anonymized and comply with data protection regulations.

Second, model bias is a critical issue. If the training dataset lacks diversity in skin tones, ages, or lesion types, the model may perform poorly on underrepresented groups, leading to unequal healthcare outcomes.

Finally, it is important to ask as to who bears the responsibility for the model's decisions. In a situation where the model diagnoses a patient with a skin cancer and he/she undergoes a surgery that ends with complications. Who should be held responsible for the decision? These models should assist, not replace, medical professionals. Overreliance on automated systems without human oversight can lead to misdiagnosis and harm. It's essential that such tools are used responsibly, with transparency and explainability in their decisions.