Skin Cancer Classification
DAT341 Assignment 5 Group 66

Carl Lange
Chalmers University of Technology
carllang@chalmers.se

Pietro Rosso
Chalmers University of Technology
pietroro@chalmers.se

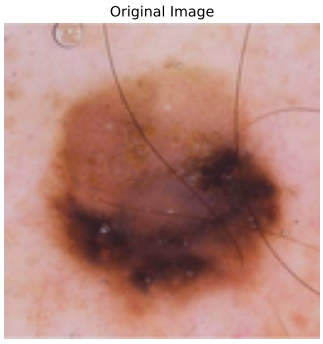
1 Introduction

Skin cancer is one of the most common types of cancer in caucasians, according to (Naldi and Diepgen, 2002). At the same time, skin cancer is one of the cancers that has a good prognosis when diagnosed and treated early. Therefore, there have been great efforts for automating skin cancer classification. Disseminating skin cancer (melanoma, MEL) from harmless birthmarks (melanocytic nevus, NEV) with fast, automated systems allows for population-scale early screening. While MEL images often have visual features deviating from NEV such as larger surface areas, more irregular boundaries, and a wider range of colors, these features don't lead to high classification accuracies in strict, rule-based approaches. Recent advances in the machine learning community have shown that Convolutional Neural Networks (CNNs) are a valuable tool for image analysis, as they are capable for identifying complex visual features by learning *what to look at* in an image (Sadoon, 2024). The ISIC challenge (Tschandl et al., 2018) provides a data set of RGB images showing nine different types of skin lesions. For this work, we focus on a subset of the data that contains images of melanoma and nevus only. We use this data set to assess the impact of different CNN architectures and data augmentation techniques on classification accuracy.

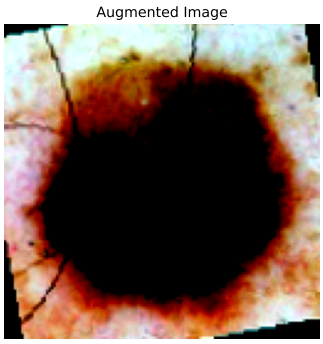
2 Methods

Data collection The data set consists of 9053 images, of which 6429 are train samples, 1255 are for validation, and 1369 for testing. The data set is balanced between the MEL and NEV classes. For details about the data collection process, please see (Tschandl et al., 2018). Our data set is a subset of the original data set, as ours contains just two classes. Each image in the data set contains three channels for RGB and has a dimension of 128×128 pixels. Given our batch size of 64, our models receive tensors of the shape $(64, 3, 128, 128)$.

Data preprocessing The approach applied in this assignment involves testing different combinations of learning techniques alongside various preprocessing strategies. Therefore, the training was performed using two distinct datasets, which allowed us to better evaluate the individual impact of each feature on the final accuracy. The only general preprocessing step applied consistently across both datasets was scaling the RGB values by normalizing pixel intensity values from an original range of 0 to 255 down to a range of 0 to 1. As already discussed, the first dataset does not present any change in the images, and we call it the *Default Dataset*. See Figure 1a for an example. Differently, the second one is subject to augmentation. It involves three main transformations: horizontal flip, rotation, and cropping of the image. Each of these types of augmentations is applied to an image with an independent probability of 50%. Furthermore, each image is normalized, where the pixel values are scaled by subtracting the mean and then divided by the standard deviation. An example of an augmented image can be seen in Figure 1b. These mean and standard deviation were computed on the train set images. It is important to note that the augmentations are only applied to the training set, whereas normalization is applied to both the training, validation, and test sets. This distinction arises because normalization alters the range of the values, while transformations (such as rotations or flips) preserve the original range of the data. Applying normalization consistently across both sets ensures that the data distributions remain aligned, which is crucial for model evaluation. The second dataset is called *Augmented Dataset*.



(a) Image from the default dataset



(b) Image from the augmented dataset showing rotation, horizontal flipping, and normalization

Figure 1: Comparison of before (a) and after (b) preprocessing

Model selection & Experiments We compare a total of eight different models. For this binary classification task, we use the accuracy on the validation set as the main metric of evaluation. All models were trained for a maximum of 30 epochs with a learning rate of 0.0001. Early stopping with a patience of 10 was used. The batch size for all models was 64. Batch size and learning rate were chosen by cross-validated grid search.

The first model is a basic CNN and serves as the baseline. It features two blocks of convolutional layer + ReLU activation + MaxPooling, followed by two fully connected layers. The last fully connected layer contains one output neuron. The value of that neuron is passed through a sigmoid function, squashing it into the range $\hat{y} \in [0, 1]$. We classify an image as MEL if $\hat{y} \geq 0.5$ and as NEV otherwise. A detailed architectural overview of all models along with their hyperparameters can be found in the Appendix, section C.

We deploy three normalization techniques and test the effect of each one individually. The `BatchNorm` models adds batch normalization after each convolutional layer and follows the baseline architecture otherwise. Batch normalization (BN) was introduced in (Ioffe and Szegedy, 2015). In batch normalization, the mean and variance for each of our three channels is computed independently over the batch size, height, and width of the image so that each channel has a mean of zero and unit variance. Scaling and shifting is applied afterwards to reduce covariance shifts, and the scale and shift parameters are learned. The `LayerNorm` model has layer normalization (LN) layers after the convolutional layers. LN, introduced by (Ba et al., 2016), normalizes each image individually across the features (channels, height, width), so it does not mix information from images in the same batch. The last type of tested normalization is group normalization (GN), introduced by Wu and He (2018). Similar to LN, GN does not compute statistics across the batch. Instead, it divides the output channels of the preceding convolutional layers into n groups and performs normalization within each of these groups. We chose group sizes of $n = 8$, resulting in four groups given the 32 output channels of the convolutional layers.

One of the main challenges for deep learning was the vanishing gradient problem. He et al. (2016) introduced a deep CNN with residual connections that allow the model to skip layers during backpropagation. This enabled very deep networks. We test a shallow model with residual connections by constructing blocks consisting of convolutional and batch normalization layers, followed by "skip" connections that add the input to the block to the output of the last activation function. A more detailed overview is in section C.

A common benchmark in computer vision tasks is the VGG-16 model by Simonyan and Zisserman (2014). We test a custom version of VGG-16 that uses the original feature extractor, i.e. the convolutional layers, with the pretrained weights. We combine this with an MLP head that features four fully connected layers. For the training process, we use the pretrained feature extractor to precompute outputs for the train and validation data sets. During training, we use the precomputed outputs from the last layer of the feature extractor as inputs to our custom MLP head. The MLP head computes a prediction and the error is backpropa-

Model	Validation(%)	Training(%)
baseline	84.38	88.88
batch_norm	86.41	100.00
group_norm	85.29	99.65
layer_norm	83.96	99.47
augmented	86.90	88.63
resnet	85.62	99.00
vgg16	86.63	99.71
res+aug	87.32	90.36

Table 1: Validation and Training Accuracies

	Pred NEV	Pred MEL
NEV	564	119
MEL	62	621

Table 2: Test set confusion matrix

gated across the MLP head to update the weights. The feature extractor uses the `IMAGENET1K_V1` weights. To ensure input consistency, we use the original VGG-16 transforms to transform our default data set.

All of these models used the default data set with minimal preprocessing. According to Shorten and Khoshgoftaar (2019), image augmentation can improve model accuracy and robustness. We therefore test the `AugModel`, i.e. our baseline model on an augmented data set, which is described in section 2.

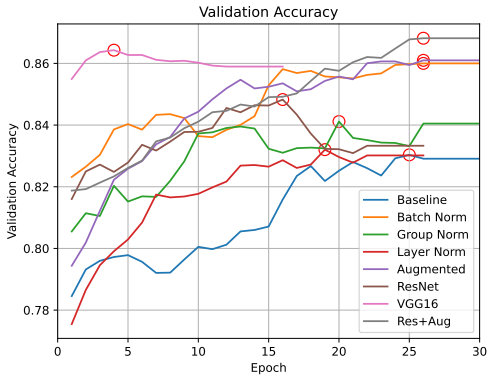
Lastly, we test a model that combines the residual connections with batch normalization and image augmentation. These architectural features were selected after evaluating the models.

All models were implemented using PyTorch (Paszke et al., 2019).

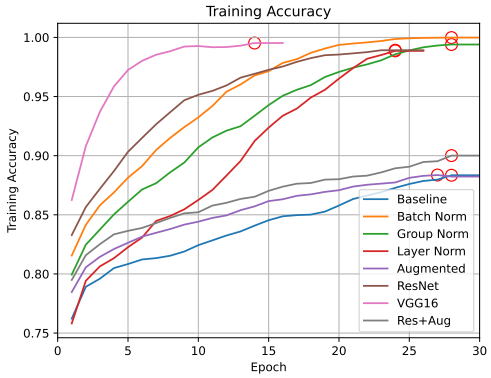
3 Results

We evaluated each model on the validation set and report the highest validation and training accuracy in Table 1. The data shows that the Res+Aug model performed the best with a validation accuracy of 87.32%, closely followed by the baseline model + augmented data. The Res+Aug model was subsequently evaluated on the test set and achieved an accuracy of 86.65%.

Table 2 shows a confusion matrix of the best model evaluated on the test set. The model reaches an F1 score of 0.8655, a sensitivity of 0.9092, and a specificity of 0.8258.



(a) Validation



(b) Training

Figure 2: Validation and Training accuracy over training epochs

Figure 2 shows the validation and training accuracies of all tested models during the training epochs. The curves are smoothened over a rolling five-epoch window. Red circles indicate the epoch where each model reached its highest validation accuracy. Some curves stop shy of 30 epochs due to early stopping.

4 Discussion

We tested eight different CNNs for binary classification of MEL vs NEV images. Our results show that the model with residual connections and augmented training data achieved the best accuracy. Out of the three normalization techniques, BN performed best. BN uses information from images across a training batch for normalization. If the images vary a lot in a batch, this effectively reduces intra-batch covariance. BN often performs well with larger batch sizes (Ioffe and Szegedy, 2015), which aligns with our results. All three normalization techniques permitted the model to overfit, as they reached training accuracies of 99-100%. Data augmentation, on the other hand, seems to counteract overfitting. The models trained on augmented data have the lowest and third-lowest training accuracies and the highest and second-highest validation accuracies. This is also evident in Figure 2b, where the purple and gray curves (augmented, Res+Aug models) and the blue curve of the baseline model remain much lower than the accuracy curves of the other models. The graphs demonstrate how complex models with several convolutional layers and nonlinear components can easily overfit and memorize training data if they are not well-regularized. In this context, regularization via image augmentation and distortion shows to be more effective than normalization techniques, which are sometimes used as implicit regularization methods.

Our results show that pretrained models like VGG16 can provide an initial advantage due to their learned general visual representations. However, the transferability of these features to specialized tasks like skin cancer classification seems limited. While VGG16 achieved good accuracy, further fine-tuning of the convolutional layers with domain-specific data could enhance performance.

The confusion matrix of our best model has a higher sensitivity than specificity. This indicates that it tends to overpredict MEL. In a high-stakes medical setting like cancer diagnosis, this is likely a desired trend. The costs of missing a cancer diagnosis (False negative) are much higher than the damage done by falsely diagnosing a healthy person with skin cancer (False positive). This is especially true since, in a real deployment setting, doctors could be instructed to have a second look or perform follow-up tests on patients that the model diagnosed with cancer.

To keep the comparison of models to the baseline as fair as possible, we changed only the distinctive parts of each model architecture and kept as many things constant as possible. For example, the normalization models only differ from the baseline model in that they have specific normalization layers. However, some of these architectural changes are often cited to enable further improvements in architecture. For example, models with BN can often use a higher learning rate to converge faster (Ioffe and Szegedy, 2015). When ResNet was released, some of its versions featured more than 150 layers because the residual connections enabled very deep networks. Yet our implementation is very shallow, for comparisons' sake as well as computational demands. These aspects have to be kept in mind when interpreting our results.

While classification accuracy is the primary focus, the computational demands of different models significantly impact their practical usability. The Res+Aug model, resulted in the best accuracy, but this comes at the expense of increased complexity and computational resources during training and inference. In contrast, simpler models like the baseline CNN offer faster training but achieve slightly lower accuracy. Therefore, selecting an appropriate model should consider both the accuracy requirements and available computational resources, especially when deploying models in resource-limited environments such as remote clinical settings.

5 Conclusion

We compared eight CNNs for their ability to classify images into cancerous vs. healthy skin marks. Our results demonstrate that CNNs with residual connections trained on augmented images perform well and appear robust against overfitting, reaching an accuracy of 86.65% on the test set.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr.

L. Naldi and T. Diepgen. 2002. The epidemiology of skin cancer. *British Journal of Dermatology*, 146.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Toqa A Sadoon. 2024. An overview of skin cancer classification based on deep learning. *Iraqi Journal for Computers and Informatics*, 50(2):173–185.

Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. 2018. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5(1):1–9.

Yuxin Wu and Kaiming He. 2018. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19.

A Limitations

While this study provides valuable insights into automated skin cancer classification, it is important to acknowledge its limitations. Firstly, the accuracy of the model holds only with a certain format of the picture. Each picture in the data set has about the same distance and angulation relative to the camera. If this model is deloed in a real-life scenario, this can lead to potential issues if images have vastly different camera angles. The reported accuracy reflects these strict conditions, and we do not have the possibility to test the model with different perspectives. Therefore the model might behave unpredictably under different settings. Secondly, the dataset consists exclusively of images featuring individuals with lighter skin tones. This lack of diversity may introduce bias into the model, potentially resulting in better diagnostic performance for lighter skin tones compared to other skin types. Even if skin cancer is more prevalent in Caucasians than other populations (Naldi and Diepgen, 2002), this still poses a large risk and limitation for population-wide model deployment. Lastly, the model does not present interpretable results. Due to the complexity of the architecture, it is not possible to derive a reasonable explanation of resulting classifications. This limitation can be more pronounced if the model is supposed be used as an aid by a medical professional who does not have an understating of machine learning. Doctors usually expect explanations for any diagnosis, and this type of machine learning model can't provide them.

B Ethical considerations

In addition to the methodological and practical aspects of this study, it is essential to address the ethical considerations and aspects that are relevant for this type of project. As previously cited, the data collection might lead to potential bias towards subjects with lighter skin tones. One way to mitigate such a bias in the future would be to collect an inclusive dataset with a more representative distribution of skin tones. Additionally, the use of real images of individuals' skin for training the model raises important ethical considerations, particularly regarding informed consent. Even though the subjects in the data collection study gave their consent (Tschandl et al., 2018), this remains an important aspect for future studies. Furthermore, releasing a system that produces a significant number of false negatives without requiring a secondary review by a medical professional could raise serious ethical concerns. Such a system may lead to misdiagnoses, delayed treatment, or a false sense of security for patients, potentially compromising their health. To lower these risks, it is essential to implement mandatory verification by qualified professionals, to ensure the system's outputs are accurate. Lastly, we advocate for ML-based classification systems to be used only in combination with human professionals. In our opinion, the classification results, even if the classification is

done by the system alone, should be communicated to the patient by a human doctor. We believe that, whatever the classification outcome is, humans prefer human-to-human interaction when it comes to high-stakes or very private matters.

C Methods: Model architectures

C.1 Baseline Model

```
==========================================================================
Layer (type)                                Output Shape     Param #
==========================================================================
Sequential                                  [64, 1]          --
+ Conv2d                                    [64, 32, 128, 128] 896
+ ReLU                                      [64, 32, 128, 128] --
+ MaxPool2d                                 [64, 32, 64, 64] --
+ Conv2d                                    [64, 64, 64, 64] 18,496
+ ReLU                                      [64, 64, 64, 64] --
+ MaxPool2d                                 [64, 64, 32, 32] --
+ Flatten                                   [64, 65536]      --
+ Linear                                    [64, 128]        8,388,736
+ ReLU                                      [64, 128]        --
+ Linear                                    [64, 1]          129
+ Sigmoid                                   [64, 1]          --
==========================================================================
Total params: 8,408,257
Trainable params: 8,408,257
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 6.33
==========================================================================
Input size (MB): 12.58
Forward/backward pass size (MB): 402.72
Params size (MB): 33.63
Estimated Total Size (MB): 448.94
==========================================================================
```

C.2 `Batch_Norm model`

```
==========================================================================
Layer (type)                                Output Shape     Param #
==========================================================================
Sequential                                  [64, 1]          --
+ Conv2d                                    [64, 32, 128, 128] 896
+ BatchNorm2d                               [64, 32, 128, 128] 64
+ ReLU                                      [64, 32, 128, 128] --
+ MaxPool2d                                 [64, 32, 64, 64] --
+ Conv2d                                    [64, 64, 64, 64] 18,496
+ BatchNorm2d                               [64, 64, 64, 64] 128
+ ReLU                                      [64, 64, 64, 64] --
+ MaxPool2d                                 [64, 64, 32, 32] --
+ Flatten                                   [64, 65536]      --
+ Linear                                    [64, 128]        8,388,736
+ ReLU                                      [64, 128]        --
+ Linear                                    [64, 1]          129
+ Sigmoid                                   [64, 1]          --
==========================================================================
Total params: 8,408,449
```

```
Trainable params: 8,408,449
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 6.33
==========================================================================
Input size (MB): 12.58
Forward/backward pass size (MB): 805.37
Params size (MB): 33.63
Estimated Total Size (MB): 851.59
==========================================================================
```

C.3 Group_Norm model

```
==========================================================================
Layer (type)                                Output Shape     Param #
==========================================================================
Sequential                                  [64, 1]          --
+ Conv2d                                    [64, 32, 128, 128] 896
+ GroupNorm                                 [64, 32, 128, 128] 64
+ ReLU                                      [64, 32, 128, 128] --
+ MaxPool2d                                 [64, 32, 64, 64] --
+ Conv2d                                    [64, 64, 64, 64] 18,496
+ GroupNorm                                 [64, 64, 64, 64] 128
+ ReLU                                      [64, 64, 64, 64] --
+ MaxPool2d                                 [64, 64, 32, 32] --
+ Flatten                                   [64, 65536]      --
+ Linear                                    [64, 128]        8,388,736
+ ReLU                                      [64, 128]        --
+ Linear                                    [64, 1]          129
+ Sigmoid                                   [64, 1]          --
==========================================================================
Total params: 8,408,449
Trainable params: 8,408,449
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 6.33
==========================================================================
Input size (MB): 12.58
Forward/backward pass size (MB): 805.37
Params size (MB): 33.63
Estimated Total Size (MB): 851.59
==========================================================================
```

C.4 Layer_Norm model

```
==========================================================================
Layer (type)                                Output Shape     Param #
==========================================================================
Sequential                                  [64, 1]          --
+ Conv2d                                    [64, 32, 128, 128] 896
+ LayerNorm                                 [64, 32, 128, 128] 1,048,576
+ ReLU                                      [64, 32, 128, 128] --
+ MaxPool2d                                 [64, 32, 64, 64] --
+ Conv2d                                    [64, 64, 64, 64] 18,496
```

```
+ LayerNorm                                [64, 64, 64, 64] 524,288
+ ReLU                                     [64, 64, 64, 64] --
+ MaxPool2d                                [64, 64, 32, 32] --
+ Flatten                                  [64, 65536]      --
+ Linear                                   [64, 128]        8,388,736
+ ReLU                                     [64, 128]        --
+ Linear                                   [64, 1]          129
+ Sigmoid                                  [64, 1]          --
==========================================================================
Total params: 9,981,121
Trainable params: 9,981,121
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 6.43
==========================================================================
Input size (MB): 12.58
Forward/backward pass size (MB): 805.37
Params size (MB): 39.92
Estimated Total Size (MB): 857.88
==========================================================================
```

C.5 Augmented model

```
==========================================================================
Layer (type)                              Output Shape     Param #
==========================================================================
Sequential                                [64, 1]          --
+ Conv2d                                  [64, 32, 128, 128] 896
+ ReLU                                    [64, 32, 128, 128] --
+ MaxPool2d                               [64, 32, 64, 64] --
+ Conv2d                                  [64, 64, 64, 64] 18,496
+ ReLU                                    [64, 64, 64, 64] --
+ MaxPool2d                               [64, 64, 32, 32] --
+ Flatten                                 [64, 65536]      --
+ Linear                                  [64, 128]        8,388,736
+ ReLU                                    [64, 128]        --
+ Linear                                  [64, 1]          129
+ Sigmoid                                 [64, 1]          --
==========================================================================
Total params: 8,408,257
Trainable params: 8,408,257
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 6.33
==========================================================================
Input size (MB): 12.58
Forward/backward pass size (MB): 402.72
Params size (MB): 33.63
Estimated Total Size (MB): 448.94
==========================================================================
```

C.6 ResNet model

```
==========================================================================
Layer (type)                                Output Shape     Param #
==========================================================================
ResModel                                    [64, 1]          --
+ Conv2d                                    [64, 16, 128, 128] 448
+ ReLU                                      [64, 16, 128, 128] --
+ MaxPool2d                                 [64, 16, 64, 64] --
+ ResBlock                                  [64, 32, 32, 32] --
|    + Conv2d                               [64, 32, 32, 32] 4,640
|    + BatchNorm2d                          [64, 32, 32, 32] 64
|    + Conv2d                               [64, 32, 32, 32] 9,248
|    + BatchNorm2d                          [64, 32, 32, 32] 64
|    + Sequential                           [64, 32, 32, 32] --
|    |    + Conv2d                          [64, 32, 32, 32] 544
|    |    + BatchNorm2d                     [64, 32, 32, 32] 64
+ ResBlock                                  [64, 64, 16, 16] --
|    + Conv2d                               [64, 64, 16, 16] 18,496
|    + BatchNorm2d                          [64, 64, 16, 16] 128
|    + Conv2d                               [64, 64, 16, 16] 36,928
|    + BatchNorm2d                          [64, 64, 16, 16] 128
|    + Sequential                           [64, 64, 16, 16] --
|    |    + Conv2d                          [64, 64, 16, 16] 2,112
|    |    + BatchNorm2d                     [64, 64, 16, 16] 128
+ Dropout                                   [64, 64, 16, 16] --
+ Flatten                                   [64, 16384]      --
+ Linear                                    [64, 128]        2,097,280
+ ReLU                                      [64, 128]        --
+ Linear                                    [64, 1]          129
+ Sigmoid                                   [64, 1]          --
==========================================================================
Total params: 2,170,401
Trainable params: 2,170,401
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 2.49
==========================================================================
Input size (MB): 12.58
Forward/backward pass size (MB): 285.28
Params size (MB): 8.68
Estimated Total Size (MB): 306.54
==========================================================================
```

C.7 VGG-16 model

```
==========================================================================
Layer (type)                                Output Shape     Param #
==========================================================================
VGG                                         [64, 1]          --
+ Sequential                                [64, 512, 4, 4]  --
|    + Conv2d                               [64, 64, 128, 128] 1,792
|    + ReLU                                 [64, 64, 128, 128] --
```

```
|    + Conv2d                                [64, 64, 128, 128] 36,928
|    + ReLU                                  [64, 64, 128, 128] --
|    + MaxPool2d                             [64, 64, 64, 64] --
|    + Conv2d                                [64, 128, 64, 64] 73,856
|    + ReLU                                  [64, 128, 64, 64] --
|    + Conv2d                                [64, 128, 64, 64] 147,584
|    + ReLU                                  [64, 128, 64, 64] --
|    + MaxPool2d                             [64, 128, 32, 32] --
|    + Conv2d                                [64, 256, 32, 32] 295,168
|    + ReLU                                  [64, 256, 32, 32] --
|    + Conv2d                                [64, 256, 32, 32] 590,080
|    + ReLU                                  [64, 256, 32, 32] --
|    + Conv2d                                [64, 256, 32, 32] 590,080
|    + ReLU                                  [64, 256, 32, 32] --
|    + MaxPool2d                             [64, 256, 16, 16] --
|    + Conv2d                                [64, 512, 16, 16] 1,180,160
|    + ReLU                                  [64, 512, 16, 16] --
|    + Conv2d                                [64, 512, 16, 16] 2,359,808
|    + ReLU                                  [64, 512, 16, 16] --
|    + Conv2d                                [64, 512, 16, 16] 2,359,808
|    + ReLU                                  [64, 512, 16, 16] --
|    + MaxPool2d                             [64, 512, 8, 8]  --
|    + Conv2d                                [64, 512, 8, 8]  2,359,808
|    + ReLU                                  [64, 512, 8, 8]  --
|    + Conv2d                                [64, 512, 8, 8]  2,359,808
|    + ReLU                                  [64, 512, 8, 8]  --
|    + Conv2d                                [64, 512, 8, 8]  2,359,808
|    + ReLU                                  [64, 512, 8, 8]  --
|    + MaxPool2d                             [64, 512, 4, 4]  --
+ AdaptiveAvgPool2d                          [64, 512, 7, 7]  --
+ VGG_MLP_Head                               [64, 1]          --
|    + Sequential                            [64, 1]          --
|    |    + Linear                           [64, 4096]       102,764,544
|    |    + ReLU                             [64, 4096]       --
|    |    + Dropout                          [64, 4096]       --
|    |    + Linear                           [64, 512]        2,097,664
|    |    + ReLU                             [64, 512]        --
|    |    + Dropout                          [64, 512]        --
|    |    + Linear                           [64, 128]        65,664
|    |    + ReLU                             [64, 128]        --
|    |    + Dropout                          [64, 128]        --
|    |    + Linear                           [64, 1]          129
|    |    + Sigmoid                          [64, 1]          --
==========================================================================
Total params: 119,642,689
Trainable params: 119,642,689
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 327.71
==========================================================================
Input size (MB): 12.58
Forward/backward pass size (MB): 2267.35
```

```
Params size (MB): 478.57
Estimated Total Size (MB): 2758.50
==========================================================================
```

C.8 Res+Aug model

```
==========================================================================
Layer (type)                                Output Shape     Param #
==========================================================================
ResModel                                    [64, 1]          --
+ Conv2d                                    [64, 16, 128, 128] 448
+ ReLU                                      [64, 16, 128, 128] --
+ MaxPool2d                                 [64, 16, 64, 64] --
+ ResBlock                                  [64, 32, 32, 32] --
|    + Conv2d                               [64, 32, 32, 32] 4,640
|    + BatchNorm2d                          [64, 32, 32, 32] 64
|    + Conv2d                               [64, 32, 32, 32] 9,248
|    + BatchNorm2d                          [64, 32, 32, 32] 64
|    + Sequential                           [64, 32, 32, 32] --
|    |    + Conv2d                          [64, 32, 32, 32] 544
|    |    + BatchNorm2d                     [64, 32, 32, 32] 64
+ ResBlock                                  [64, 64, 16, 16] --
|    + Conv2d                               [64, 64, 16, 16] 18,496
|    + BatchNorm2d                          [64, 64, 16, 16] 128
|    + Conv2d                               [64, 64, 16, 16] 36,928
|    + BatchNorm2d                          [64, 64, 16, 16] 128
|    + Sequential                           [64, 64, 16, 16] --
|    |    + Conv2d                          [64, 64, 16, 16] 2,112
|    |    + BatchNorm2d                     [64, 64, 16, 16] 128
+ Dropout                                   [64, 64, 16, 16] --
+ Flatten                                   [64, 16384]      --
+ Linear                                    [64, 128]        2,097,280
+ ReLU                                      [64, 128]        --
+ Linear                                    [64, 1]          129
+ Sigmoid                                   [64, 1]          --
==========================================================================
Total params: 2,170,401
Trainable params: 2,170,401
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 2.49
==========================================================================
Input size (MB): 12.58
Forward/backward pass size (MB): 285.28
Params size (MB): 8.68
Estimated Total Size (MB): 306.54
==========================================================================
```