NOVA IMS Information Management School

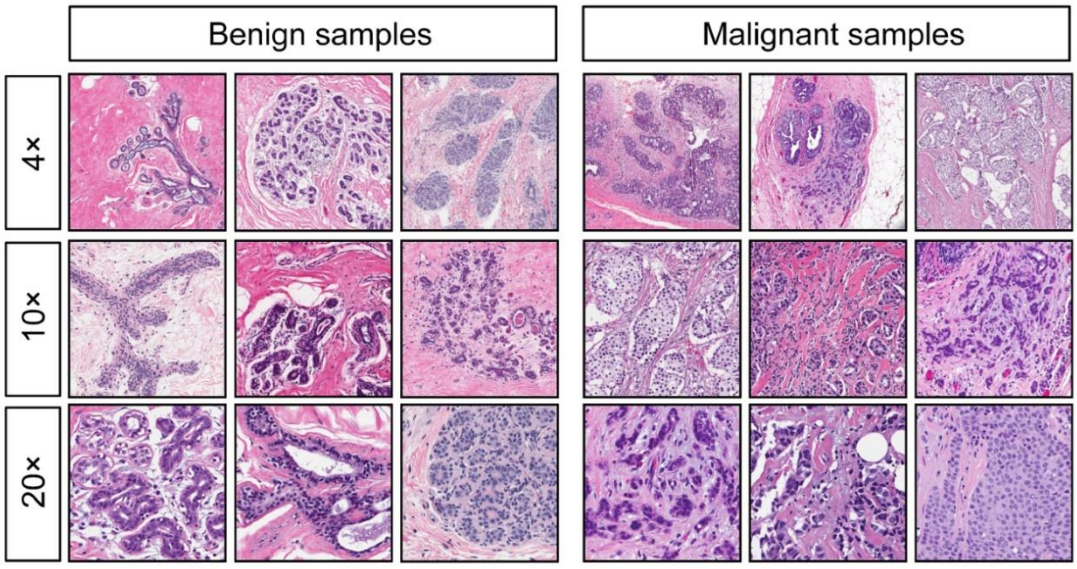
Data Science Bachelor

Academic year 2023 / 2024

15 December 2024

Deep Learning for Cancer Diagnosis:

Binary and Multiclass Classification of Cancer Images



Group 9

Miguel Mendes, No. 20221904

João Ferreira, No. 20221912

Rodrigo Maia, No. 20221934

Introduction

Cancer is one of the leading causes of death in the world[1], and early and accurate diagnosis plays a pivotal role in improving patient outcomes. Using neural networks to classify tumors correctly is a powerful tool that can help doctors obtain a more precise assessment. This report on cancer image classification is structured into two main stages: binary classification of tumors into benign or malignant categories (Stage 1) and multiclass classification into eight different cancer types (Stage 2). Additionally, using a new Python library called **imgaug**[2], we conducted several image transformation trials to assess the impact of various parameters, such as *blur, brightness, horizontal flipping*, and other augmentation techniques, on the model's performance.

Related work

In resemblance to our project, numerous studies have been carried out to achieve the goal of diagnosing breast cancer more accurately. Therefore, there are some scientific articles (e.g., [1], [2], [3]) that were used as a point of reference for our work since they helped us to understand how to approach the problem, how to deal with some challenges and, in general, how to have a better idea of the steps to adopt.

Data Pre-processing

As stated before, this report aims to develop a model that can classify microscopic images of breast tissue into binary and multiclass categories to assist in diagnosing breast cancer. To this end, a dataset[3] containing 7909 high-resolution images was used.

Here are the steps we followed to prepare the dataset for image classification: we started by examining the data (shape, type) and checking for missing or incorrect values. We found a few missing labels and manually imputed them based on the information derived from the file paths.

Afterward, using the library **matplotlib**, we generated bar plots, which clearly show the class

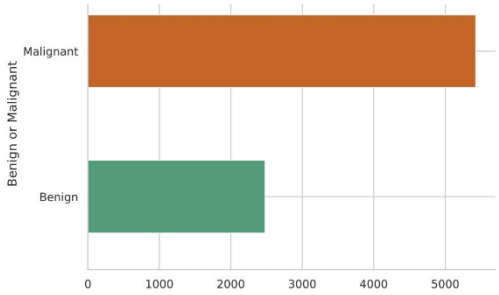


*Figure 1: Distribution of "Benign or Malignant" labels – the target from Stage 1 is imbalanced - the number of **Malignant** labels is more than double the number of **Benign** labels.*

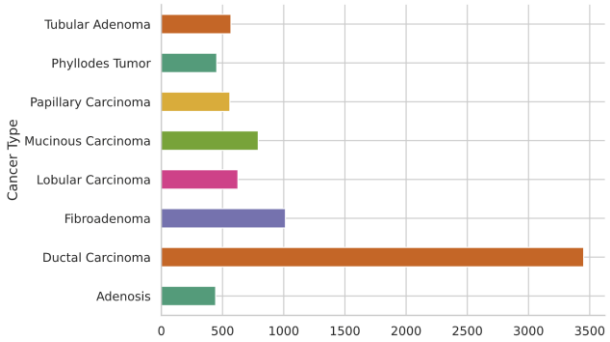


*Figure 2: Distribution of eight "Cancer Type" labels – there is classs imbalance in the target of Stage 2 since **Ductal Carcinoma** is overwhelmingly represented.*

imbalance for the target variable corresponding to each stage (see Figures 1 and 2).

In the final step, we iterated through all the image file paths and outputted the smallest and largest images. The maximum size of an image was (700, 460), and the minimum was (700, 456), which meant that there were no outliers in terms of image size and no image had to be discarded.

Stage 1: Binary Classification

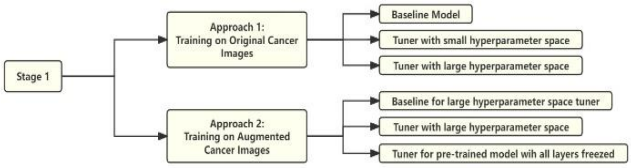


Figure 3: Flowchart presenting Stage 1 employed process

The flowchart presented in Figure 3 gives a scheme of the employed process in Stage 1 for classifying images as “Benign” or “Malignant” and the different baseline models and hyperparameter tuners used in each approach.

[1] https://www.who.int/news-room/fact-sheets/detail/cancer

[2] https://imgaug.readthedocs.io/en/latest/

[3] https://web.inf.ufpr.br/vri/databases/breast-cancer-histopathological-database-breakhis/

Data preparation and transformation

We started the **first approach** by loading the images (and their respective labels) and resizing them to a uniform dimension of (150x150) pixels using the library **cv2**[4]. Following this, we resized the pixels that initially took values between [0,255] and the range [0,1], allowing the model to converge faster and smoother. Using a label encoder from the **sklearn** library, we encoded the target labels "Benign" and "Malignant" to 0 and 1, respectively. We then used a 75% - 25% stratified split, dividing the dataset into a training and testing set and ensuring the same distribution of target classes in both sets. No image transformations were applied in this approach.

We followed a similar methodology for the **second approach** with two key differences: first, we split the data into training, validation, and test sets. The reason for doing this is that by splitting the data now (and not when fitting the model), it becomes easier to implement the data augmentation pipeline. Second, after data preparation, we performed image transformation only to the training set, by using a data augmentation pipeline from the **imgaug** library. This pipeline consisted of four different transformations: horizontal flip with 30% probability, slight scaling and translations (shifting images horizontally and vertically), brightness adjustment, and Gaussian blur (see Figure 4).

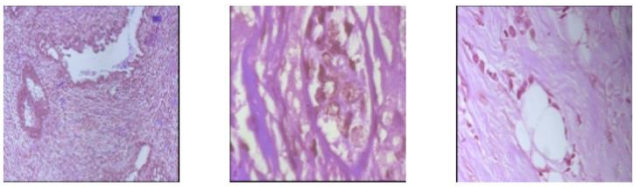


Figure 4: Sample of image transformations for the training set. The black borders (pixels = 0) are the result of slight scaling and translations.

Model building and hyperparameter tuning

To correctly classify the image data, we used convolutional neural networks (CNNs). These networks are composed of three main operations:

1. **Convolution blocks:** sequence of convolutional layers and other operations that extract features from data. Each subsequent convolutional block has an equal or higher minimum filter value than the previous one. This allows the model to gradually explore and learn more complex image features.
2. **Pooling layers:** reduce the dimensions of the feature maps extracted by the convolutional layers.
3. **Fully connected (Dense) layers:** transform the previously extracted high-level feature maps into a final prediction or output. In this project, each dense layer has a ReLU activation function and is followed by a dropout layer to prevent overfitting.

After processing and transforming the data, we started to use the **Keras** library to build small baseline models (basic CNNs with standard parameters) and continuously add convolutional and dense layers until we obtained acceptable results. Consequently, we move on to build our main evaluation models, that is, the hyper-parameter tuners or fine-tuned models. These models were used to optimize parameters not learned during training (learning rate, filter size, number of layers, ...) and significantly improved the performance of our CNNs.

To further enhance the performance of the models, we integrated transfer learning into the binary classification task. Transfer learning consists of adapting pre-trained CNNs, such as ResNet50, DenseNet121, and EfficientNetB0, to a specific classification task. To prevent overfitting and reduce computational requirements, the convolutional base of the pre-trained models was frozen.

Since we are working with binary classification we used a sigmoid activation function for the output dense layer, giving us an output probability between 0 and 1. This output layer only has one neuron, as we are only required to predict a single probability value. Finally, we compiled the model using the Adam optimizer and binary cross-entropy loss.

Evaluation and Results

We chose weighted f1-score as our main metric since we are dealing with highly imbalanced classes, and metrics like accuracy are incapable of fully explaining the performance of our models.

[4] https://pypi.org/project/opencv-python/

From Table 1, we observe that we got better weighted average f1-score values in both training and testing sets for all models in **approach 2**.

This means that these models achieved better optimization (learning patterns in the training data) and generalization (learning patterns in unseen data) than the models in **approach 1**.

Table 1: Evaluation metrics for the different models in stage 1

		Metrics			
Approach	Model	Training Weighted avg f1-score	Testing Weighted avg f1-score	Validation loss	Train loss
1	Tuner with small hyperparameter space	0.91	0.89	0.2917	0.2406
	Tuner with large hyperparameter space	0.90	0.91	0.2574	0.2607
2	Tuner with large hyperparameter space	0.94	0.91	0.2595	0.1367
	Tuner for pre-trained model	0.96	0.90	0.2869	0.1202

On the other hand, we must also consider overfitting when evaluating the performance of a model. The gap between the validation and training loss is higher for the models in **approach 2**. While this gap isn't big enough to be a sign of clear overfitting, it suggests that at least the model should have stopped training a few epochs earlier.

Overall, applying data augmentation in the training set results in better model performance than using the original breast cancer images.

Strictly speaking on model performance, our best model was the "Tuner for pre-trained model" in the **second approach** since it recorded the smallest validation loss and highest weighted f1-score values. However, by considering overfitting, the best model for binary classification of cancer images, was the **Tuner with large hyperparameter space**[5] since it balances high performance and generalization while minimizing overfitting.

	Bening	Malignant
Bening	514	106
Malignant	62	1296

Table 2: Confusion matrix for the test set - best-performing model in the first stage (actual vs. predicted labels)

From Table 2 it can be revealed that the final model effectively distinguished between benign and malignant cases, with minimal misclassification. Considering our test set had 1978 images, we classified correctly 1810 images and misclassified 168. This gives us an overall classification accuracy of 91%, which aligns with the testing F1-score reported earlier.

Stage 2: Multiclass Classification

The flowchart presented in Figure 5 gives a scheme of the employed process in Stage 2 for the multi-class problem and the different baseline models and hyperparameter tuners used in each one of the **four approaches**.

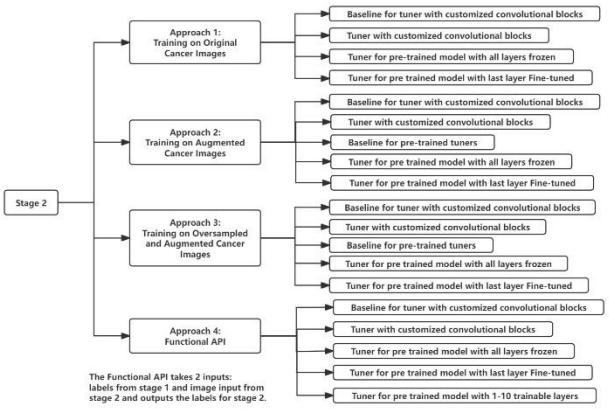


Figure 5: Flowchart presenting the Stage 2 employed process

Data preparation and transformation

The data preparation methodology in the **first approach** of Stage 2 was identical to the corresponding approach in Stage 1. The key differences were that when loading the images labels, we now did it for the target "Cancer Type" and not for "Benign or Malignant". Consequently, when using label encoder on the eight target classes, we encoded the labels into integers from 0 to 7. Finaly, we performed a standard 75% - 25% stratified split and refrained from applying any transformation to the original image data. The last significant change was the calculation of the class weights for the labels in the training set after splitting the data. In Stage 1, while there was class imbalance, we could still classify both classes quite well. On the contrary, for Stage 2, as we have one class that is significantly more represented than the other classes, we need to apply some balancing technique. Otherwise, many classes will have close to zero classifying capability.

For the **second approach**, we followed a similar data preparation methodology as the one used in the second approach of Stage 1, where we performed two initial splits and applied

[5] See Table A1 in Annex

transformations to the images in the training set. The target "Cancer Type was labelled encoded by integers between 0 and 7. The image transformations (see Figure 6) made in this approach were slightly different from the ones in Stage 1 since we excluded the slight scaling and translation parameters in the data augmentation pipeline.

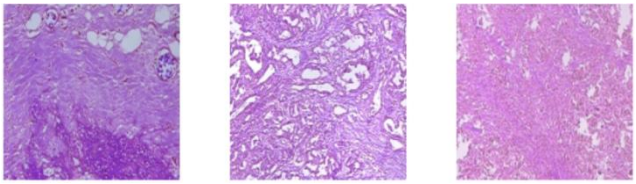


Figure 6: Sample of image transformations for the training set in stage 2 approach 2. There are no black borders (pixels = 0) since we removed the parameter that caused slight scaling and translations.

In the **third approach,** we tried experimenting with something new. We started by manually dividing the original data into X (path to the images) and Y (Cancer type) lists. Afterwards, we encoded the eight cancer type labels and performed two splits. These splits were made considering that we would perform oversampling in the training data later, so we split the data to have a bigger validation and testing set. The outputted subsets had the following size percentages: training (18.0%), validation (42.0%), and testing (40.0%). After splitting the data, we calculated the class weights. To load the images and their labels for the training data, we used a function that called an augmentation pipeline to oversample the underrepresented classes in the training set by multiplying the original count of each class by three and adding those augmented images to the original images. For instance, if a class had 100 images, after oversampling, it would have 300 augmented images in addition to the 100 original images. Finally, we used two new simple functions to load the images for the validation and testing sets without applying any transformations to the data. The new subsets after oversampling had the following size percentages: training (47.0%), validation (27.0%), and testing (26.0%). It is important to note that we kept the same proportions for each class in the training data after oversampling. The image transformations applied in this approach were similar to the ones applied in Stage 1, but now we increased the range for the scaling and translation parameters (see Figure 7).

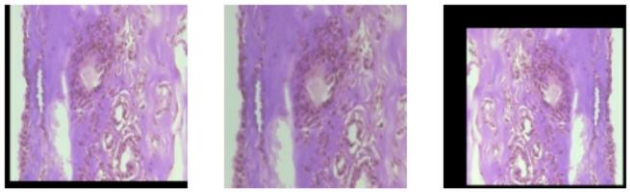


Figure 7: Sample of image transformations for the training set. In stage 2 approach 3.

For our **fourth approach**, we implemented a functional API with two inputs: original labels from Stage 1 and images from Stage 2. To load the images for this approach we used two identical functions. The first function returned the images and corresponding labels for the first stage and the second function returned the images and labels for the second stage. Since we were dealing with two inputs, we made sure that the images in the training set for both inputs were the same. Otherwise, we would be incurring data leakage. To further mitigate the risk of data leakage from combining two inputs, we decided to equally split the data (for both inputs) into train, validation and test sets at the beginning and immediately isolate the last two sets.

Model building and hyperparameter tuning

The baseline models and tuners we implemented in this stage were highly similar to the ones implemented in the previous stage. The main differences would be the addition of Batch Normalization layers after each convolutional and dense layer (for the second approach). We also applied higher class weights to the least represented categories to reduce the impact of these imbalances during training.

To further enhance the performance of the pre-trained models, we created a new tuner for each approach where we unfroze the last layer of the pre-trained model and replaced that same layer with a fine-tuned convolutional block. This allowed us to further adapt the pre-trained models to our task. For the functional API we also tried unfreezing more than 1 layer.

Since we are working with multiclass classify-cation, we used a SoftMax activation function that outputs a probability distribution that sums to 1. The output layer has 8 units as we are predicting eight different probabilities. Finally, we compiled the model using the Adam optimizer and sparse categorical cross-entropy loss function.

Evaluation and Results

In the **first approach**, the models show moderate generalization (Table 3). While the first tuner has the smallest overfit out of all the models, it also has the lowest weighted f1-score for both training and testing sets. The model wasn't great at fitting the training data or learning on new unseen data. While the remaining two models in this approach generalized better than the first, they also had a bigger overfit.

Table 3: Evaluation metrics for the different models in stage 1

Approach	Model	Metrics			
		Training Weighted avg f1-score	Testing Weighted avg f1-score	Validation loss	Train loss
1	Tuner with customized convolutional blocks	0.47	0.46	1.4165	1.2791
	Tuner for pre-trained model with all layers frozen	0.64	0.58	1.2199	0.9252
	Tuner for pre trained model with last layer Fine-tuned	0.64	0.54	1.3536	0.9923
2	Tuner with customized convolutional blocks	0.83	0.78	0.6652	0.5381
	Tuner for pre-trained model with all layers frozen	0.65	0.58	1.2641	0.9919
	Tuner for pre trained model with last layer Fine-tuned	0.82	0.62	1.1922	0.8647
3	Tuner with customized convolutional blocks	0.52	0.49	1.4547	1.4042
	Tuner for pre-trained model with all layers frozen	0.59	0.49	1.4914	1.1986
	Tuner for pre trained model with last layer Fine-tuned	0.60	0.46	1.5432	1.2909
4	Tuner with customized convolutional blocks (A)	0.66	0.64	0.9631	0.8906
	Tuner with customized convolutional blocks (B)	0.73	0.66	0.9625	0.7617
	Tuner for pre-trained model with all layers frozen	0.63	0.57	1.1383	1.0283
	Tuner for pre trained model with last layer Fine-tuned	0.68	0.60	1.1720	0.9253
	Tuner for pre trained model with 1-10 trainable layers	0.95	0.85	0.5461	0.0577

We obtained our best model in the **second approach**: **Tuner with customized convolutional blocks**[6]. Besides the last tuner discarded due to clear overfitting, this model obtained the best f1-score values out of all the models and the lowest train and validation losses, suggesting that augmentation helped the model learn richer image features (Table 3).

The **third approach** had quite underwhelming results, but this is most likely due to the lousy image transformations applied in this approach. Nonetheless, it still performed better compared to the first approach.

Finally, we achieved the best overall results in the **fourth approach**. While the best model doesn't belong to this approach, we were able to achieve the best balance between training and testing performance.

After observing the heatmap in Figure 8, we can conclude that our best model is good at classifying all classes since the diagonal values representing the correctly classified classes for each cancer type have high values. The only cancer type that had significant misclassification is "Tubular Adenoma," which was misclassified 96 times as "Phyllodes Tumor" and 72 times as "Papillary Carcinoma."

Figure 8: Heatmap of the confusion matrix for the testing set of the best performing model in the second stage.

Conclusion and future work

We developed several approaches to find the best models for binary and multiclass classification. After comparing the fine-tuned models, we concluded that for both stages, models performed better in the approaches where data augmentation was applied. An exception to this rule is the Functional API, which performed well even without data augmentation.

With this in mind, for future work, we would focus on developing the Functional API, especially applying image transformation in order to get even better results. Additionally, if given more time, we would have experimented with using different image transformations for the third approach in the second stage.

Overall, we are satisfied with the results for both stages, even though the results in Stage 2 were worse than those in Stage 1. This happens because the model in Stage 1 only needs to decide between two classes, while in Stage 2, the model needs to classify the best class (highest probability) out of the eight classes.

[6] See Table A2 in Annex

References

[1]. Kumar, Y., Shrivastav, S., Garg, K., Modi, N., Wiltos, K., Woźniak, M., Ijaz, M., Automating cancer diagnosis using advanced deep learning techniques for multi-cancer image classification. *Sci Rep.,* 2024, Oct 23;14(1), doi: 10.1038/s41598-024-75876-2.

[2]. Urabinahatti, S. and Jayadevappa. D., Breast Cancer Detection Using Deep Learning Technique, *2023 International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*, India, 2023, pp. 1-5, doi: 10.1109/ICDCECE57866.2023.10150859.

[3]. Yu, Y., Favour, E. and Mazumder, P., Convolutional Neural Network Design for Breast Cancer Medical Image Classification, *2020 IEEE 20th International Conference on Communication Technology (ICCT), Nanning, China,* 2020, pp. 1325-1332, doi: 10.1109/ICCT50939.2020.9295909.

Annex

Image transformation trials:

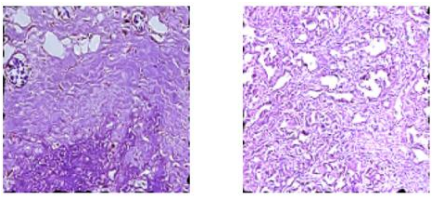


Figure A1: image transformations for trial 1: new parameters added such as sharpen and Elastic transformation.

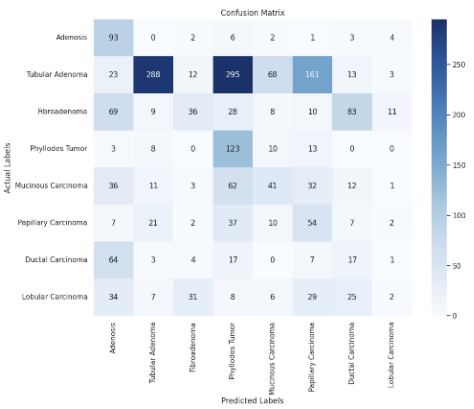


Figure A2: heatmap for image transformation trial 1: good at classifying classes Adenosis and Phyllodes Tumor.

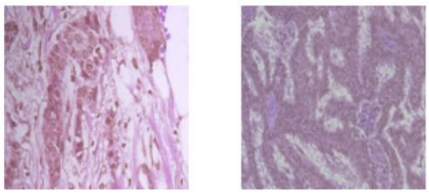


Figure A3: image transformations for trial 2: new parameters added such as sharpen and Elastic transformation.

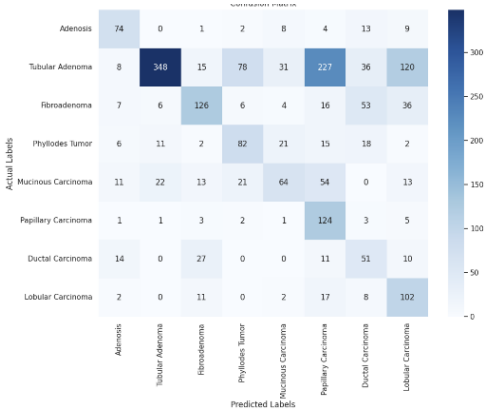


Figure A4: heatmap for image transformation trial 2: good at classifying classes Papillary Carcinoma and Lobular Carcinoma.

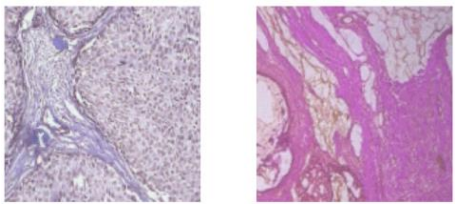


Figure A5: image transformations for trial 3: new parameters added such as Linear Contrast, Hue and Saturation and Additive Gaussian Noise

Figure A6: heatmap for image transformation trial 3: good at classifying classes Adenosis, Papillary Carcinoma, Lobular Carcinoma and Phyllodes Tumor.

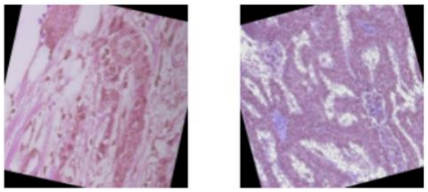


Figure A7: image transformations for trial 6: new parameters added such as Flips, Rotations and Hue and Saturation.

Figure A8: heatmap for image transformation trial 6: good at classifying Lobular Carcinoma.

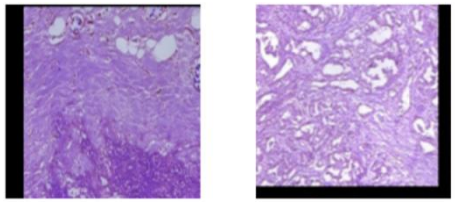


Figure A9: image transformations for trial 7: new parameter added such Gamma Contrast.

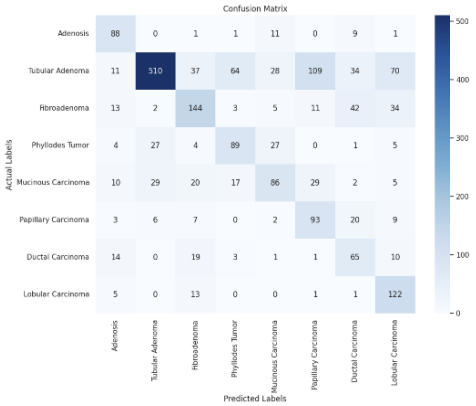


Figure A10: heatmap for image transformation trial 7: good at classifying classes Fibroadenoma, Phyllodes tumor, papillary carcinoma and lobular carcinoma.

Table A1: best combination of parameters for the best model in Stage 1.

Parameter	Value
filters_block_A	45
padding_block_A	same
num_conv_block_A	3
pool_size_block_A	3
filters_block_B	60
padding_block_B	same
num_conv_block_B	3
pool_size_block_B	3
filters_block_C	120
padding_block_C	valid
num_conv_block_C	3
pool_size_block_C	3
filters_block_D	155
padding_block_D	same
num_conv_block_D	3
pool_size_block_D	2
Initial_units	250
num_dense_layers	4
l2_strength	0
dropout_0	0
dropout_1	0
dropout_2	0
dropout_3	0.1
number_of_units_1	64
dense_l2_strength	1e-05
dense_dropout_1	0.2
learning_rate	0.0001

Table A2: best combination of parameters for the best model in Stage 2.

Parameter	Value
filters_block_A	70
num_conv_block_A	2
filters_block_B	120
num_conv_block_B	3
filters_block_C	190
num_conv_block_C	3
filters_block_D	200
num_conv_block_D	2
filters_block_E	380
num_conv_block_E	1
conv_l2_strength	1e-05
num_layers	1
number_of_units_1	64
dense_l2_strength	1e-05
dense_dropout_1	0.2
learning_rate	0.0001

More detailed explanation of libraries used in this project:

To implement and refine the classification models, several libraries were leveraged: **Keras** served as the primary library and was used to train the deep learning models; **sklearn** was used for pre-processing tasks and model evaluation; **cv2** was used for image preprocessing, namely resizing images and normalizing arrays of pixel values of images; **numpy** was used for array manipulation and **matplotlib** to visualize the results and dataset characteristics. Finally, a new library called **imgaug** was not introduced in prior coursework and was used for image augmentation.