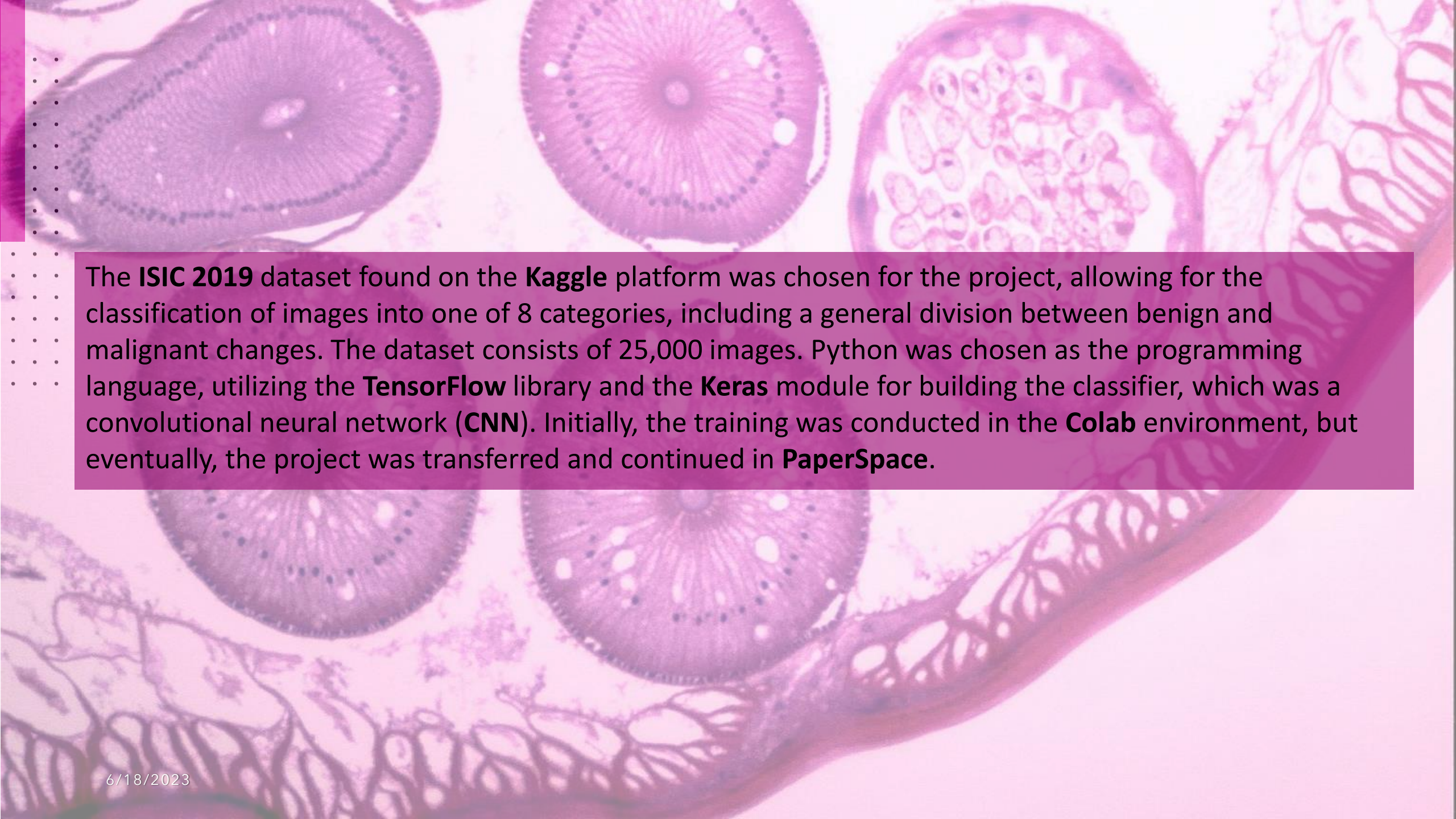


A microscopic image of skin tissue, showing various structures like hair follicles and sweat glands, overlaid with a semi-transparent purple rectangle containing text. On the left side of the purple rectangle, there is a vertical column of small white dots.

# SKIN LESION CLASSIFICATION WITH CONVOLUTION NEURAL NETWORK

AUTHOR: ROCH FEDOROWICZ





The **ISIC 2019** dataset found on the **Kaggle** platform was chosen for the project, allowing for the classification of images into one of 8 categories, including a general division between benign and malignant changes. The dataset consists of 25,000 images. Python was chosen as the programming language, utilizing the **TensorFlow** library and the **Keras** module for building the classifier, which was a convolutional neural network (**CNN**). Initially, the training was conducted in the **Colab** environment, but eventually, the project was transferred and continued in **PaperSpace**.

# IMAGE CATEGORIZATION

Due to the initial project's objective of categorizing images into malignant and benign, the images were divided into these two categories. Here are some sample images belonging to each category. The images were also resized to a resolution of 200 by 150.

Benign



Benign



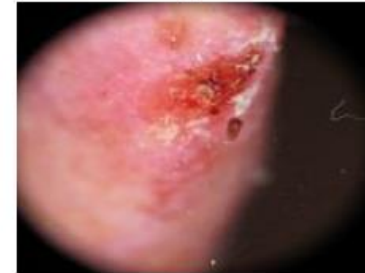
Malignant



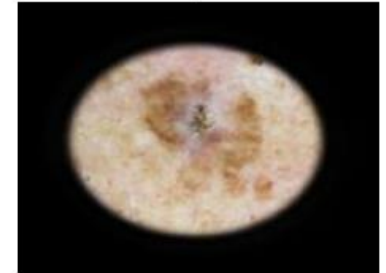
Malignant



Malignant



Malignant



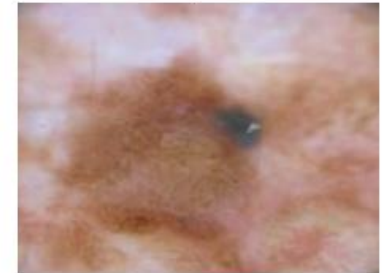
Benign



Benign

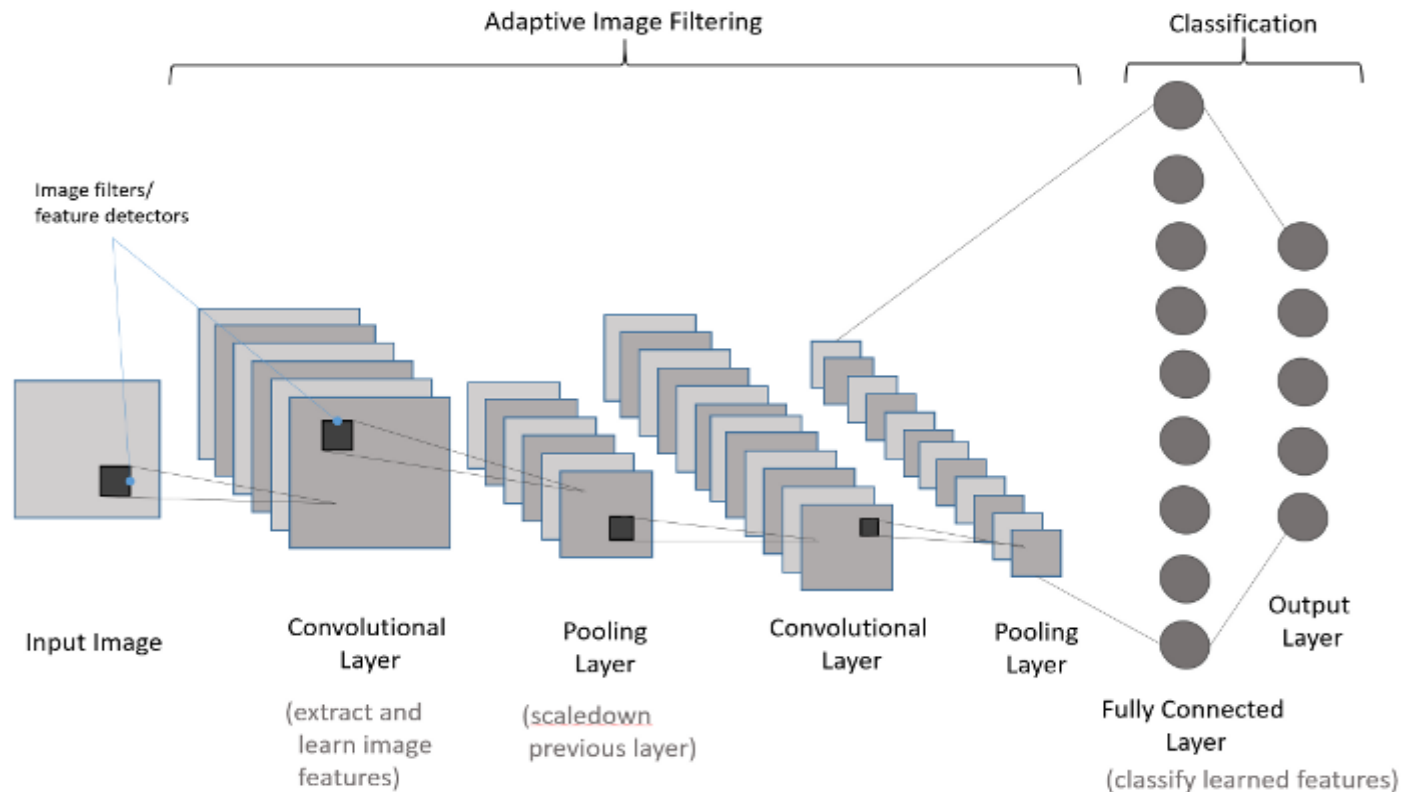


Malignant



# INITIAL MODEL CONCEPT

When creating a model for image classification, a typical approach was used, constructing a model based on convolutional layers interleaved with pooling layers. This approach effectively extracts information from the image and trains neurons to detect key features. Towards the end, everything is flattened to a single dimension, enabling the final layer to determine the image's belongingness to a specific class.



# FIRST ATTEMPTS - NETWORK

The first model that was created was a simple sequential model composed of the following layers:

- Input layer to rescale pixel values.
- First convolutional layer.
- First pooling layer.
- Second convolutional layer.
- Second pooling layer.
- Third convolutional layer.
- Third pooling layer.
- Flattening layer to reduce everything to a single dimension.
- Dense layer to detect the most abstract features.
- Output layer responsible for detecting the belongingness to a specific class.

```
model = Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(2)
])

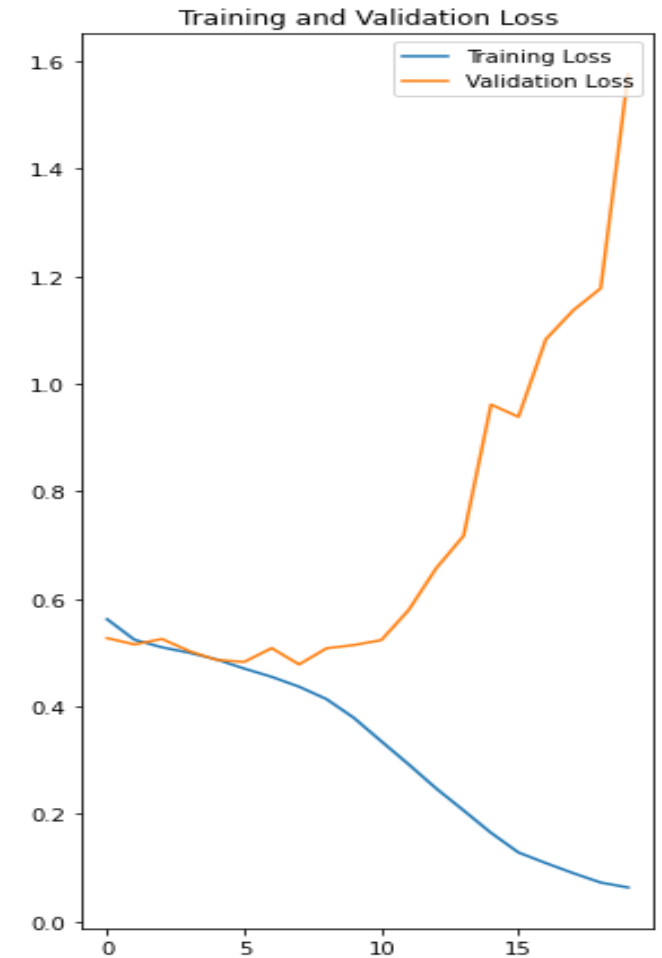
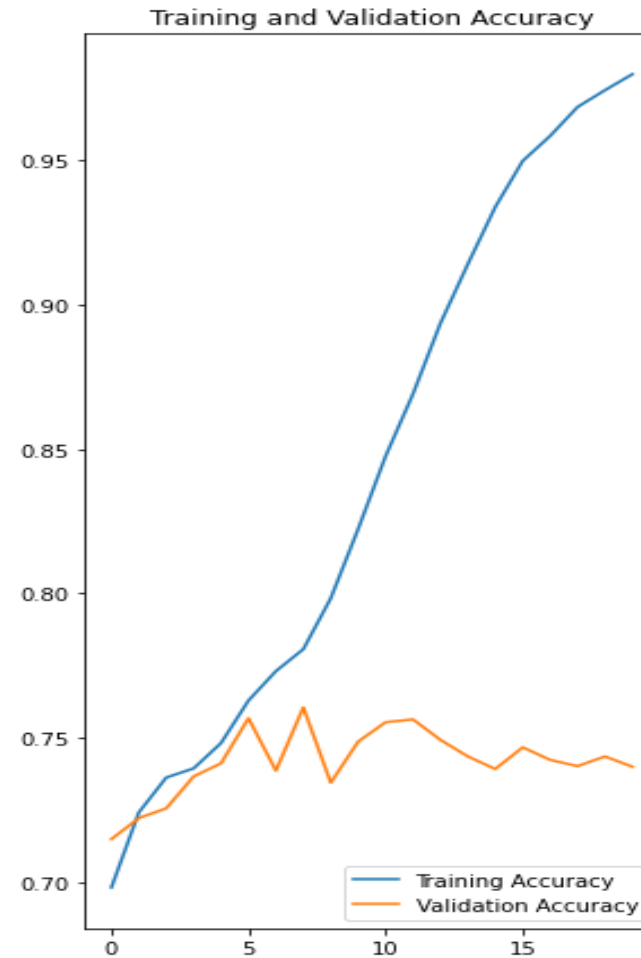
lossFn = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer = 'adam',
              loss = lossFn ,
              metrics = ['accuracy'])
```





# FIRST ATTEMPTS - RESULTS

The first encountered problem was excessive data fitting - overfitting. This phenomenon occurs when the model becomes overly adjusted to the training dataset. It is illustrated by the accuracy function, where the accuracy on the training dataset continues to increase steadily, while it reaches a plateau on the test dataset. This indicates that despite the network's improving recognition of the training data, it struggles to classify data outside of the training set effectively. In summary, the network does not generalize well.



# DATA AUGMENTATION

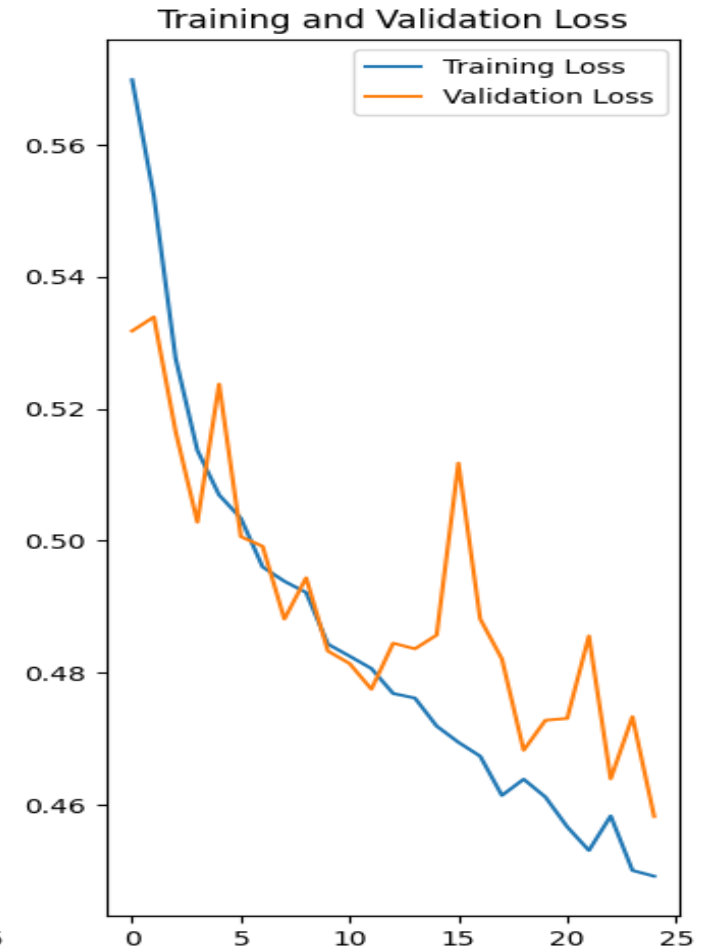
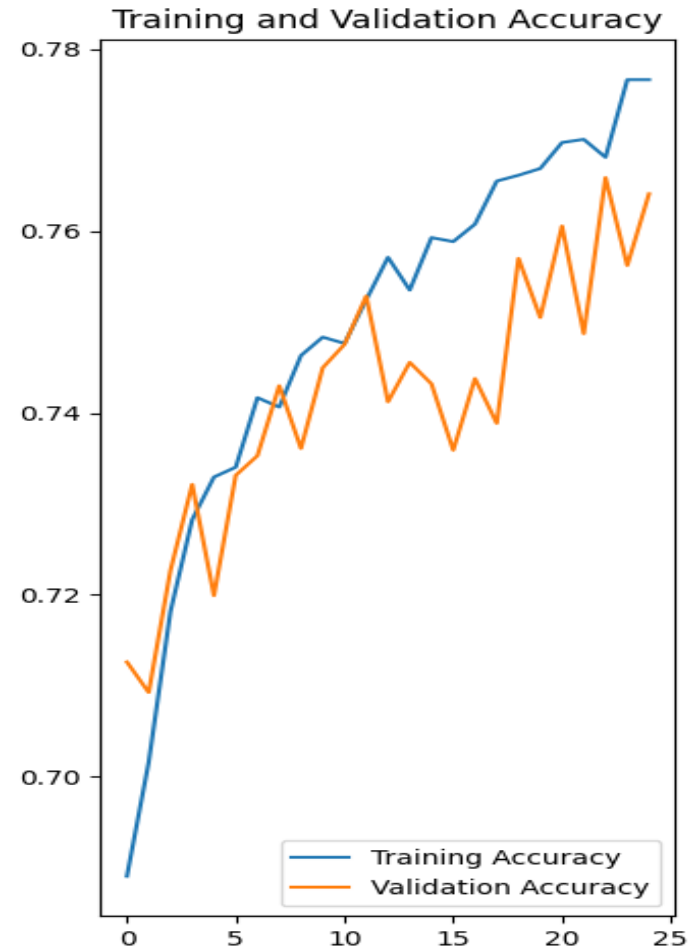
To improve the results, **data augmentation** was applied, which involves modifying the data to make their features independent of their position in the image. By adding three layers to the model - one for random horizontal flipping, another for random rotations, and a third one for random zooming - this augmentation technique was easily implemented using the Keras and TensorFlow libraries by directly adding these layers to the model. **Dropout** was also applied, which randomly deactivates neurons within the deep layers of the network responsible for feature detection. This allows for the independence of individual features and is expected to enhance the network's quality.

```
model = Sequential([
    layers.RandomFlip("horizontal",
        input_shape=(img_height,img_width, 3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(2)
])
```



# DATA AUGMENTATION - RESULTS

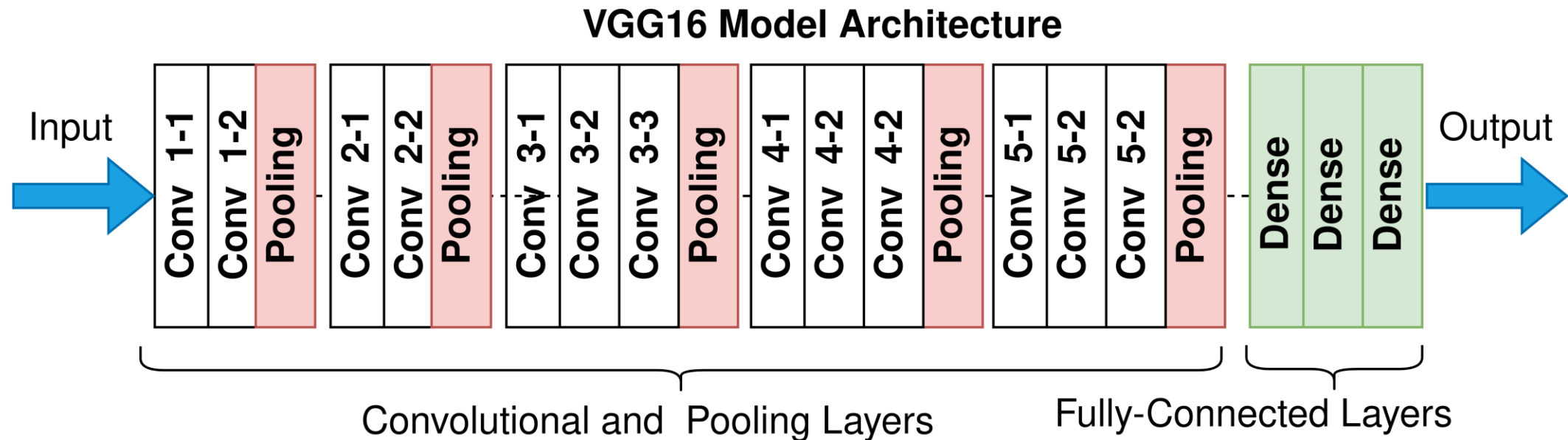
Despite mitigating overfitting (the accuracy on the training and test sets being similar), it was not possible to surpass an accuracy level of 80%.





# MODEL CHANGE

To improve the accuracy, a pre-trained model for image recognition and classification called VGG19 was utilized. VGG models are based on interleaved layers that perform information extraction (MaxPooling) and layers that process the data and detect dependencies between them. Additionally, the Keras library allows for the utilization of pre-trained weights with VGG19 models (as well as VGG16). This can be achieved through a process known as **fine-tuning**, where the already trained model is fine-tuned instead of training it completely from scratch.



# MODEL CHANGE - IMPLEMENTATION

During the creation of the VGG19 network, one can choose whether to use pre-trained weights in the neurons and whether to import the entire model or just the model without the last layers (fully connected layers). In this case, to customize the last layers for the specific problem at hand, they were omitted and manually written within the new sequential model. Similarly, data augmentation was added to the base VGG19 model using the same approach.

```
from keras.applications.vgg19 import VGG19

vgg19_base = VGG19(input_shape = (img_height, img_width, 3),
                   include_top = False,
                   weights = 'imagenet')

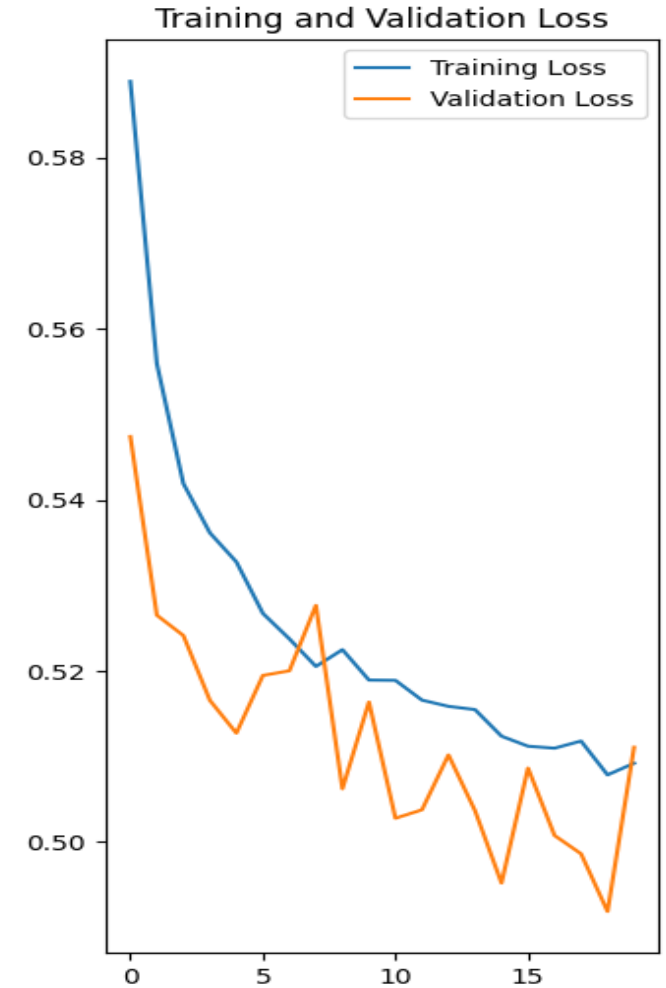
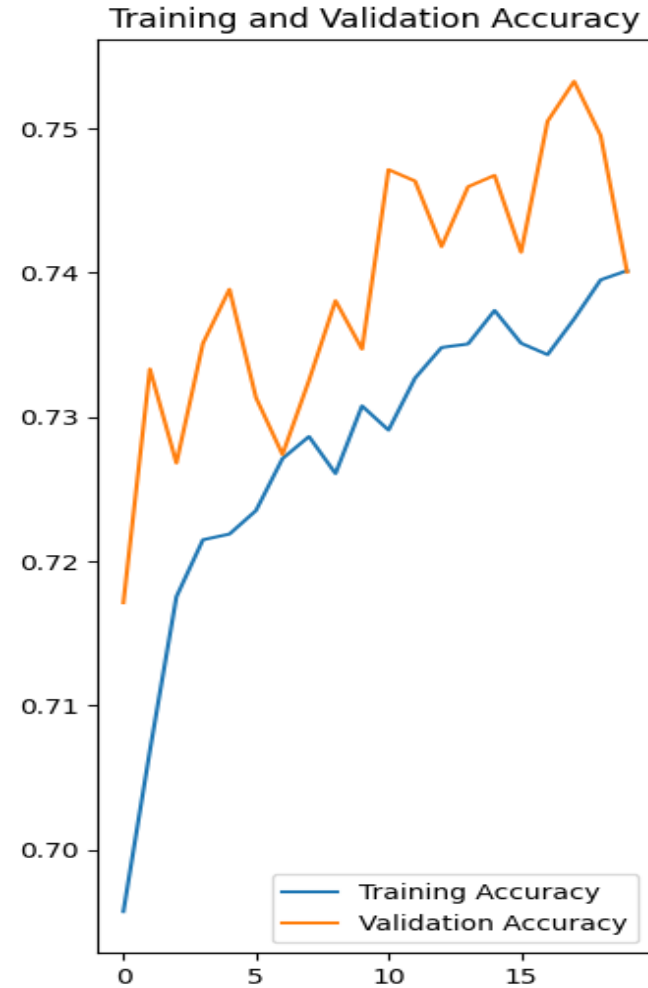
for layer in vgg19_base.layers:
    layer.trainable = False


model = Sequential([
    layers.RandomFlip("horizontal", input_shape=(img_height, img_width, 3)),
    layers.RandomRotation(0.5),
    layers.RandomZoom(0.1),
    layers.Rescaling(1./255),
    vgg19_base,
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(2, activation='softmax')
])
```



# MODEL CHANGE - RESULTS

After utilizing the VGG19 model, the accuracy on the test set turned out to be even better than on the training set. However, utilizing the pre-trained model did not have a positive impact on the network's accuracy, unfortunately.





Another approach that could have been tried to improve the classification quality of the network was the selection of appropriate parameters that influence the final classification. Since the selection of suitable **hyperparameters** can be challenging and often depends on the dataset, this was implemented using a simple genetic algorithm. It allowed for finding the best parameters for the final classification layers. This experiment was conducted based on the initial model that was created.



# ATTEMPT OF GENETIC ALGORITHM

The key functions of the class enabling the implementation of the genetic algorithm were as follows:

- **produceOffspring** - This function produces offspring by utilizing mutation and returns a new instance of the class.
- **train** - This function trains the model of the class (based on previously used functionality).
- **getAcc** - This function returns the highest achieved accuracy of the network (based on previously used functionality).

The genetic algorithm was created in a way that, after selecting the most effective networks, it reproduces them and repeats the process.

```
class MyModel:

    def __init__(self, img_height, img_width, features, dropout,
                 learning_rate, mutation_rate, base_model = None):
        ...

    def produceOffspring(self, external_mutation_rate = None):
        ...

    def train(self, train_ds, val_ds, epochs):
        ...

    def getAcc(self):
        ...
```

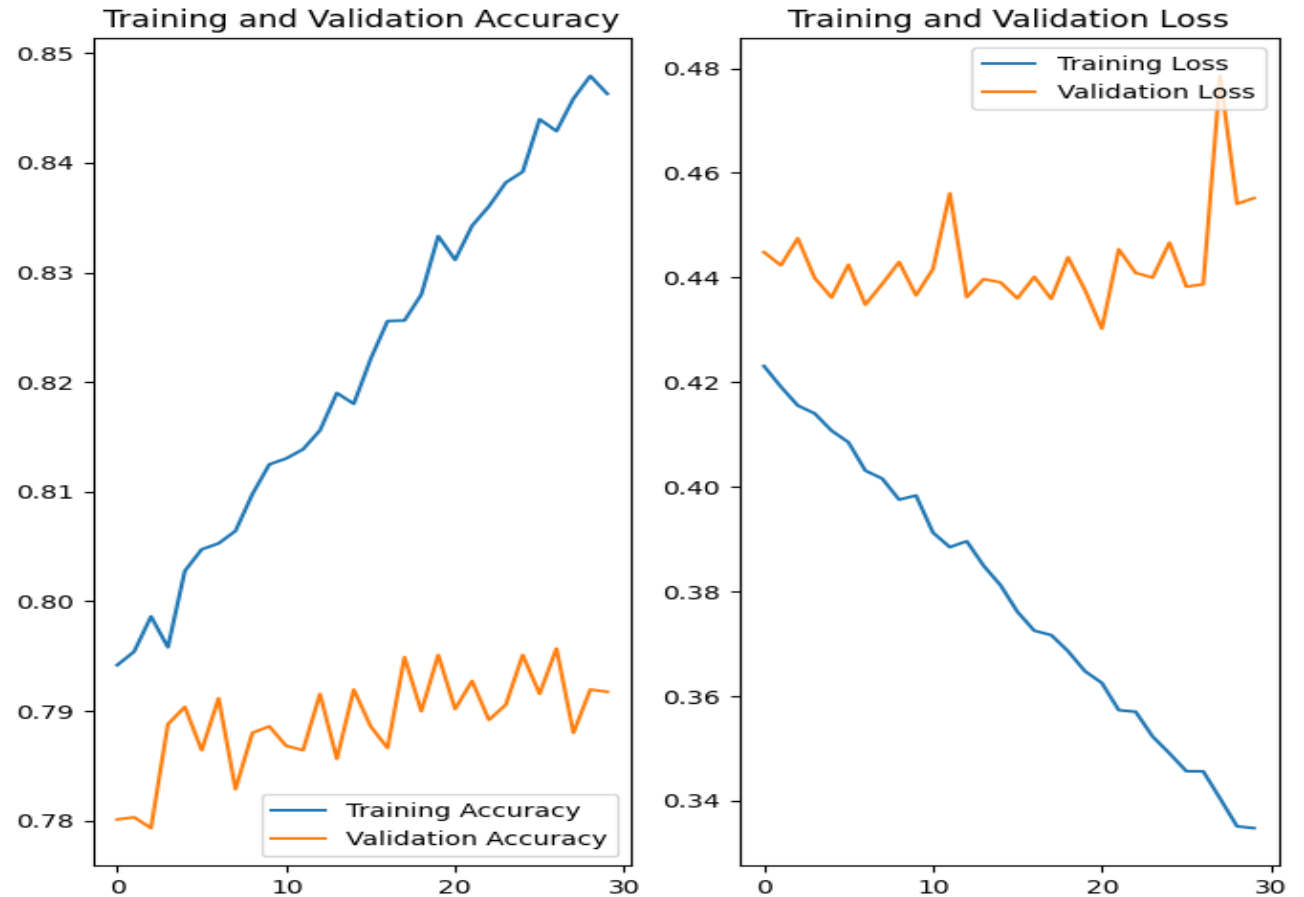


# ATTEMPT OF GENETIC ALGORITHM - RESULTS

In this way, a potentially well-adapted network with the following parameters was identified:

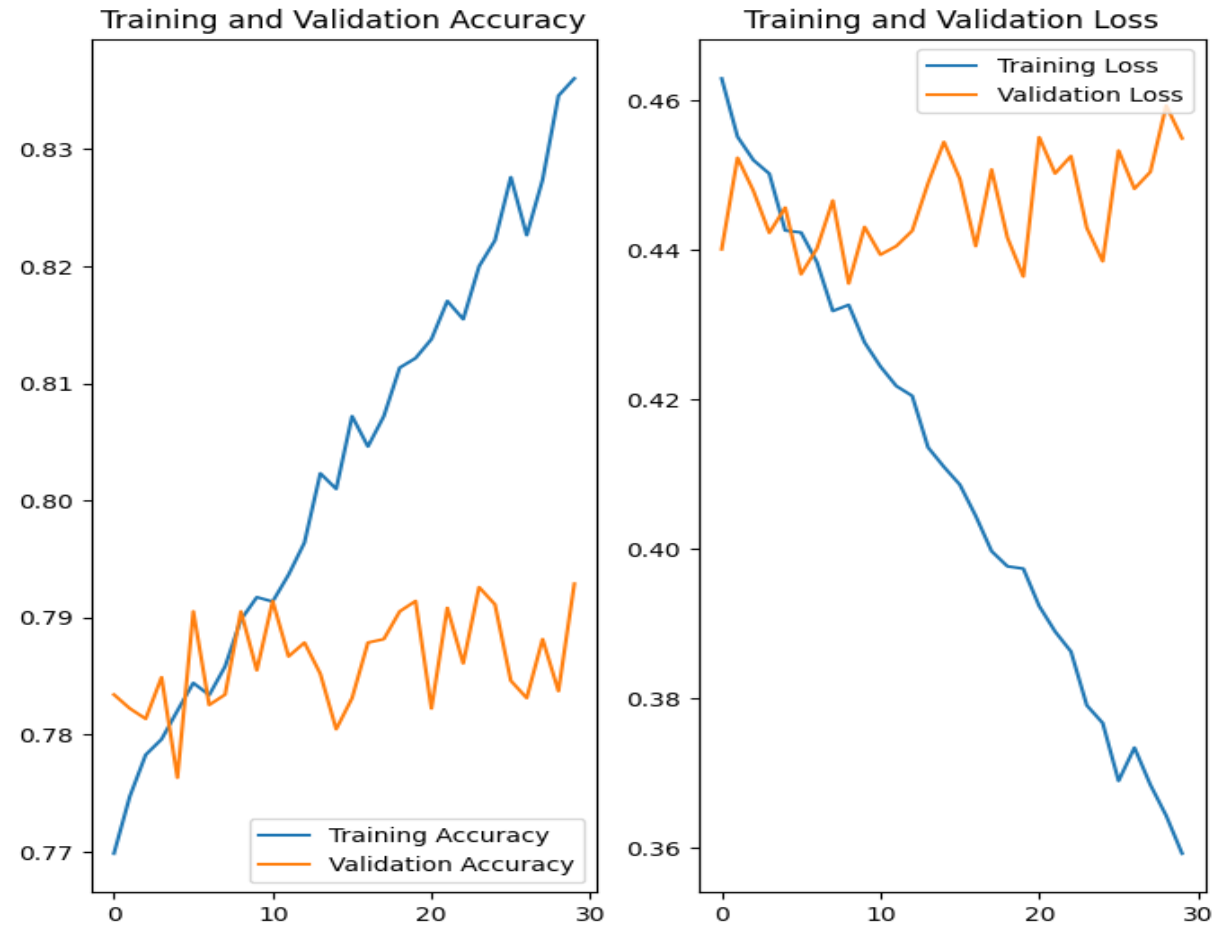
- Number of neurons in the fully connected classification layer: 112
- Dropout: 0.30
- Learning rate: 0.0001
- Mutation rate: 0.11
- Highest accuracy: 0.79

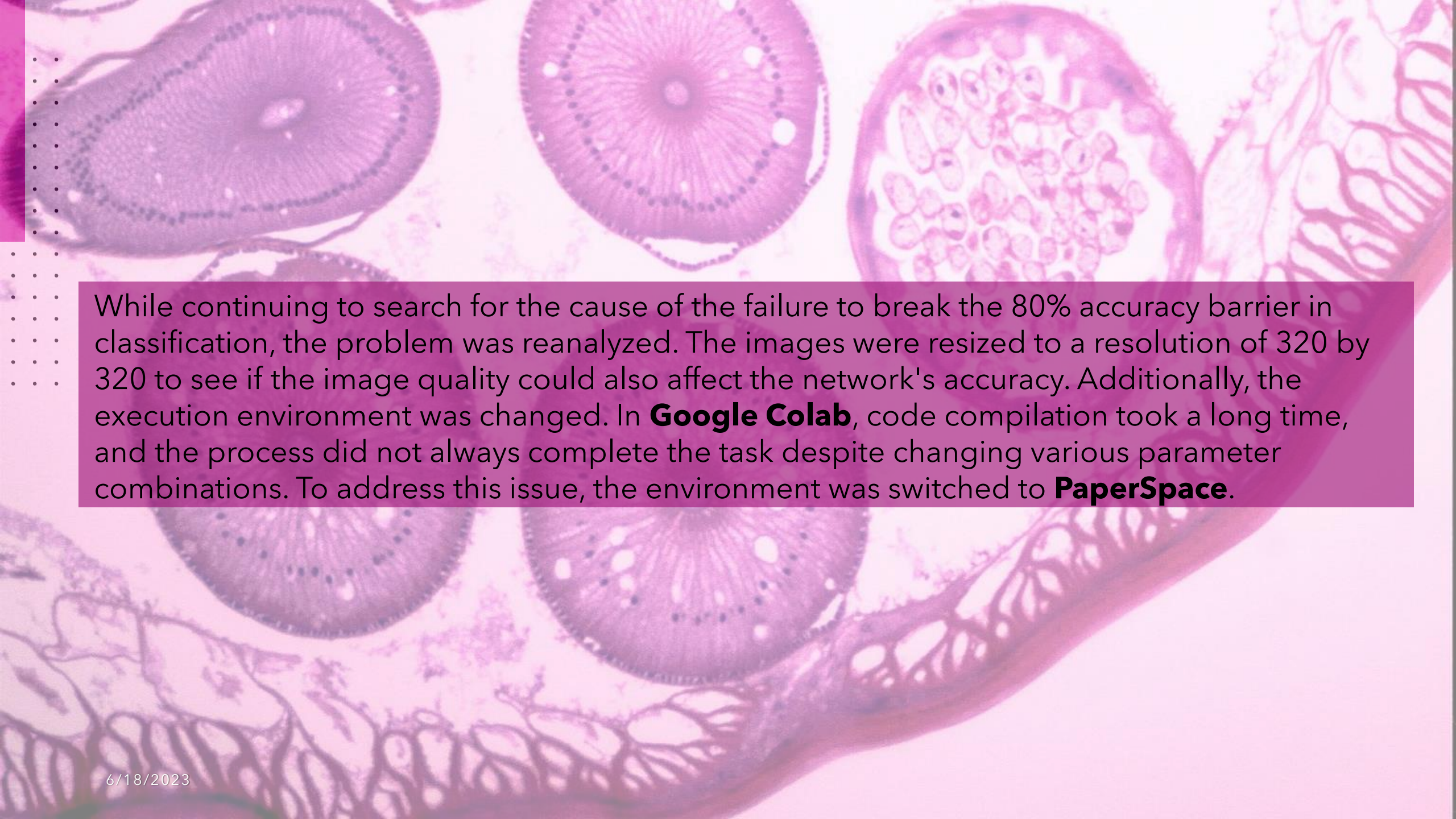
The selected parameters do not deviate significantly from the initially chosen ones, except for the number of neurons in the last layer. The next step was fine-tuning the most adapted model, but it did not yield the expected results. The model reached a plateau at 80% accuracy for the test data.



# CLASS IMBALANCE MITIGATION

Upon analyzing the dataset used for training these networks, it can be observed that unfortunately one of the classes is overrepresented. This can lead to difficulties in training, which is why the data was balanced and an additional 30 epochs were trained for the model obtained from the genetic algorithm to see if it would improve the results. Unfortunately, this also did not yield the intended effects.



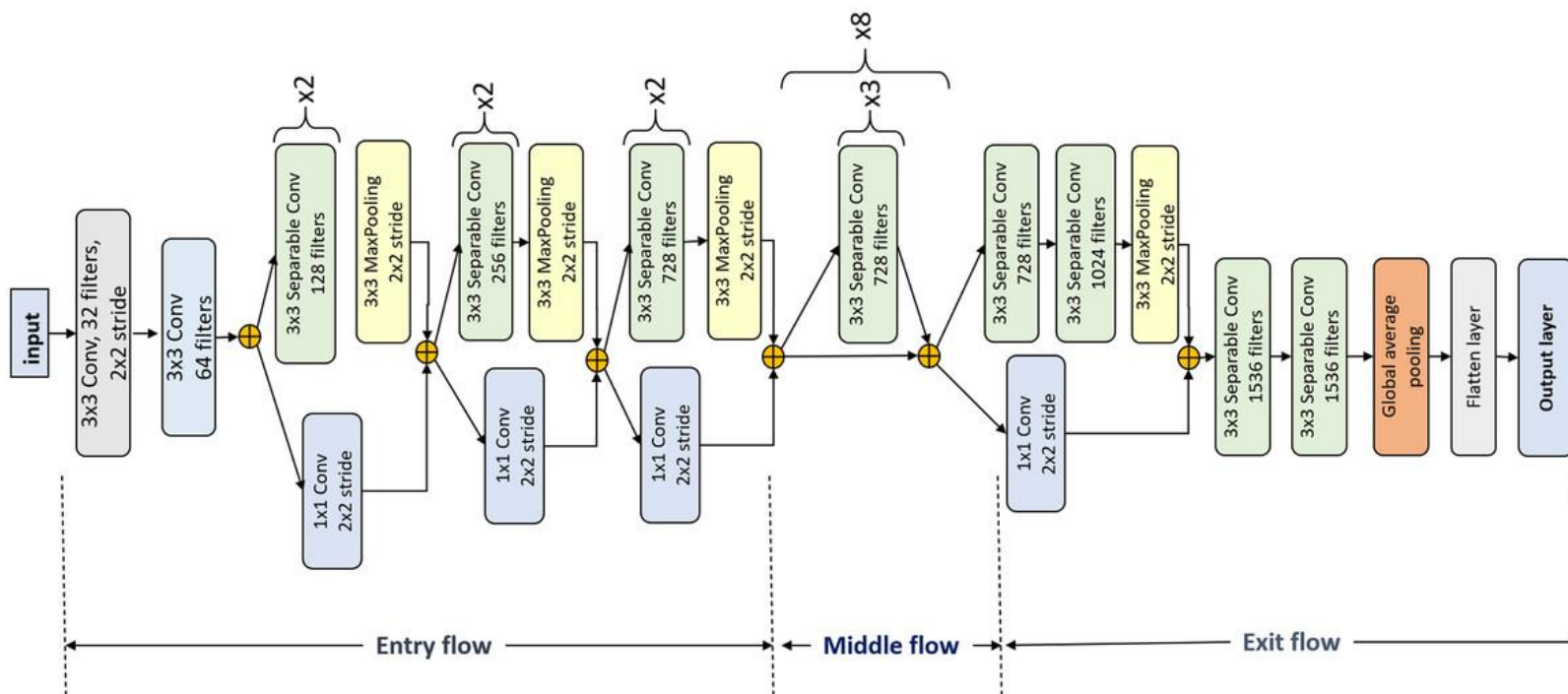
A microscopic image of plant tissue, likely a cross-section of a root or stem, showing various cellular structures. The image is overlaid with a semi-transparent purple rectangle containing text. On the left side of the purple rectangle, there is a vertical column of dots of varying sizes.

While continuing to search for the cause of the failure to break the 80% accuracy barrier in classification, the problem was reanalyzed. The images were resized to a resolution of 320 by 320 to see if the image quality could also affect the network's accuracy. Additionally, the execution environment was changed. In **Google Colab**, code compilation took a long time, and the process did not always complete the task despite changing various parameter combinations. To address this issue, the environment was switched to **PaperSpace**.



# NEW NETWORK MODEL - ARCHITECTURE

The network model has also been changed. Unlike the previously used sequential models (the initial model and models based on the VGG architecture), Xception is a model where each layer depends not only directly on the layer immediately following it but also on earlier layers. This allows the network to more comprehensively utilize the data and should facilitate the model in achieving high class discrimination.



# NEW NETWORK MODEL - IMPLEMENTATION

In addition to data augmentation, the **ImageDataGenerator** class was utilized, which allows for better data augmentation and performs it before passing the data to the network. Two **callbacks** were also added - functions that influence the learning process of the network. One is responsible for adapting the learning rate, while the other interrupts the learning process if its accuracy doesn't improve under certain conditions. Similarly to the VGG19 model, the last layers were omitted to perform fine-tuning.

```
datagen = keras.preprocessing.image.ImageDataGenerator(...)

base_model = keras.applications.Xception(include_top = False,
                                         weights = 'imagenet',
                                         input_shape = training_shape)

for layer in base_model.layers:
    layer.trainable = False

model = Sequential([
    base_model,
    layers.Flatten(),
    layers.BatchNormalization(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Dense(2, activation='softmax')
])

learning_rate_reduction =
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',
    patience=5, verbose=1, factor=0.5, min_lr=0.00001)

early_stopping_monitor =
    tf.keras.callbacks.EarlyStopping(patience=20,
    monitor='val_accuracy', restore_best_weights=True)
```

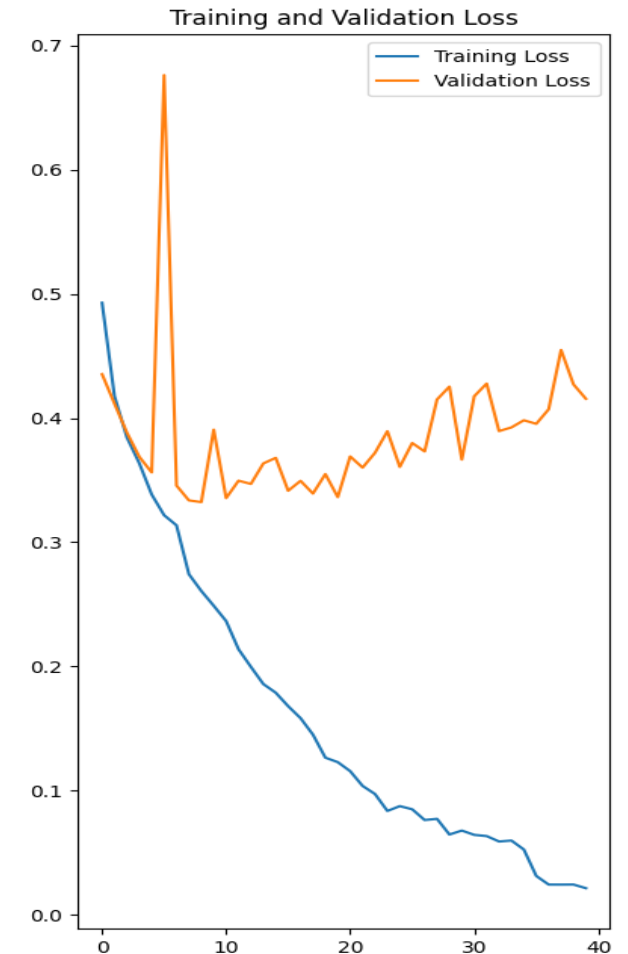
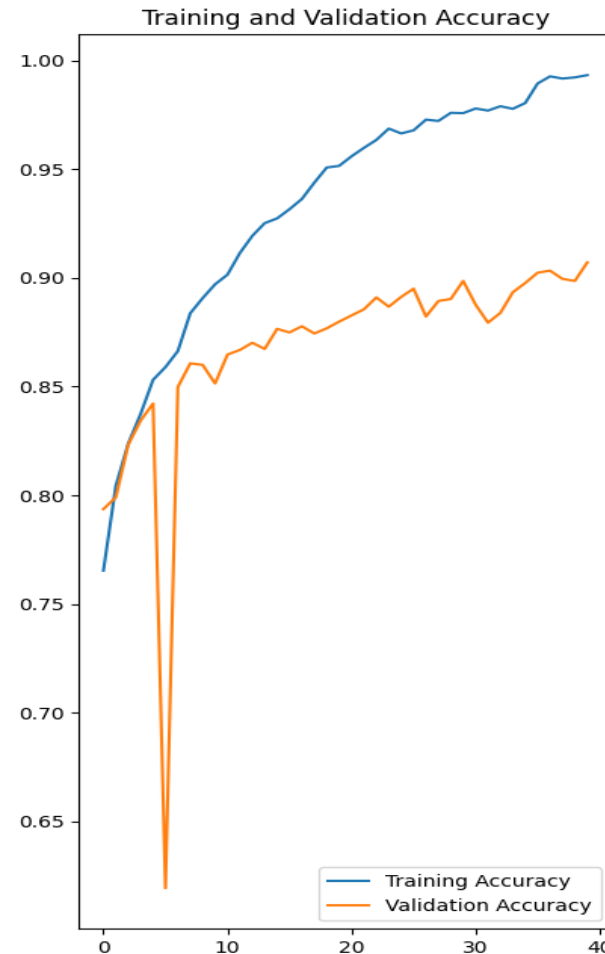


# NEW NETWORK MODEL - RESULTS

The change in environment, image quality, and network model finally brought benefits. Training the model using fine-tuning was conducted for **50 epochs**, which improved the network's accuracy to **81%**.

The next step was to unlock learning in all layers of the model and continue training for an additional **40 epochs**, resulting in an accuracy of **90%**, as seen in the graphs.

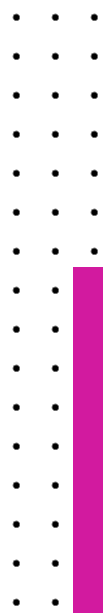
After achieving better accuracy results, the model was trained once again for **40 epochs** to see if further improvements were possible. However, the accuracy only increased to **91%**.



# NEW NETWORK MODEL - VALIDATION

Unfortunately, due to oversight, the model was trained on an imbalanced dataset, which may have led to a falsely high result. To verify the achieved accuracy, a validation was conducted by comparing the performance of the trained model on both balanced and imbalanced datasets. Despite concerns about the falsely high result, the model achieved even higher accuracy on the balanced dataset, surpassing **98%**. This suggests that the imbalanced dataset had no significant impact on the model's learning process in this case.

Model	Unbalanced data	Balanced data
After the first 50 epochs	0.8010	<b>0.8157</b>
After the next 40 epochs	0.9064	<b>0.9833</b>
After the last 40 epochs	0.9116	<b>0.9851</b>

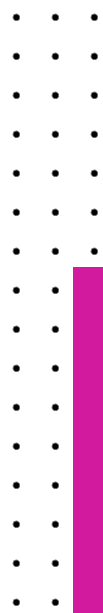


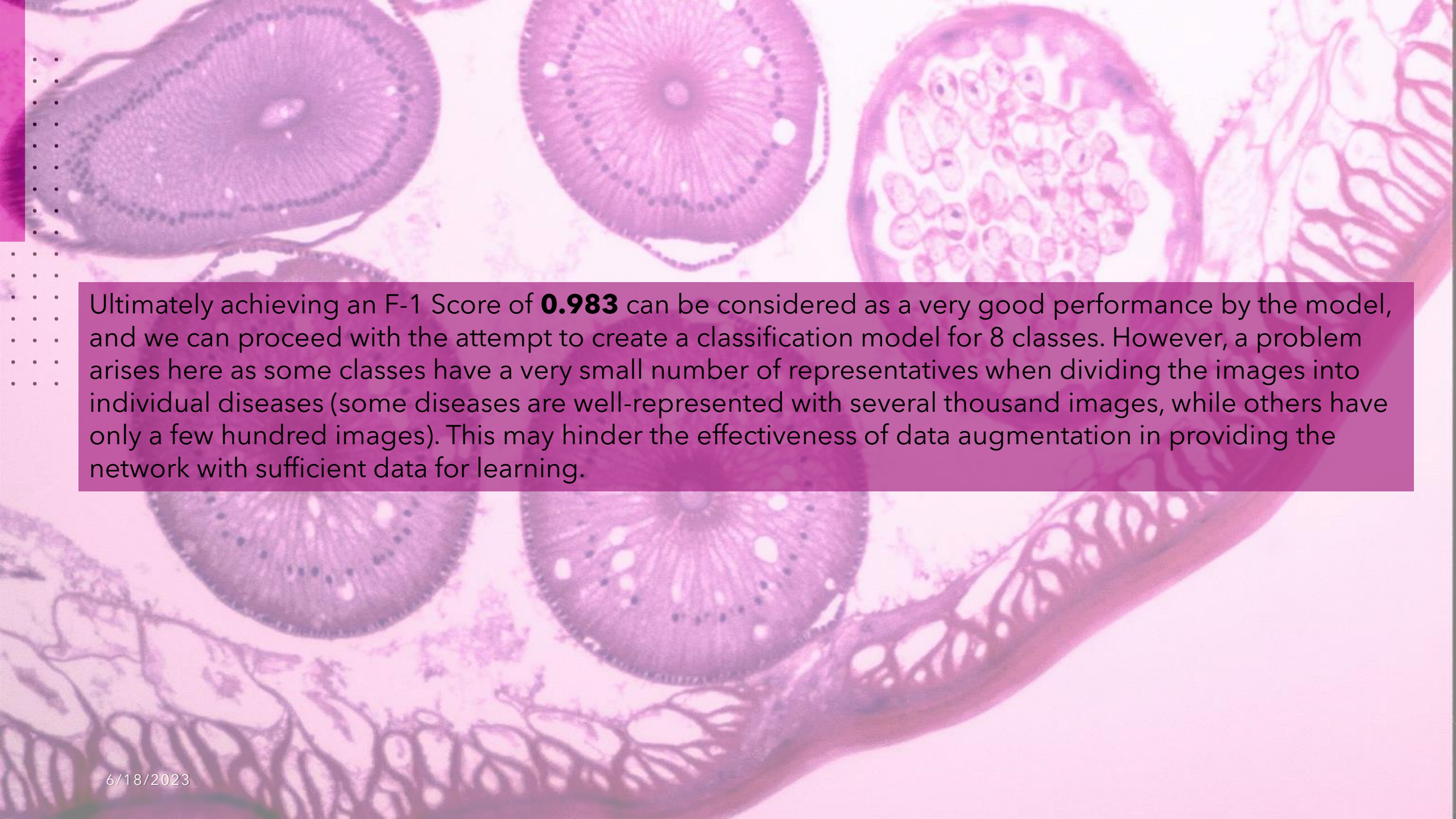


# NEW NETWORK MODEL – ADDITIONAL METRICS

Additionally, to evaluate the classification performance of individual classes, tests were conducted using additional metrics. Ultimately, the best model achieved a precision of nearly **99%**, recall of **98%**, and an F1-score of **98%**. It also demonstrated similar classification quality for benign and malignant changes.

Model	Precision	Recall	F1-Score	Confusion matrix	
After training on the imbalanced dataset	0.892	0.838	0.864	2667	142
				228	1184
After training on the balanced dataset	<b>0.988</b>	<b>0.978</b>	<b>0.983</b>	<b>1388</b>	<b>16</b>
				<b>31</b>	<b>1381</b>



A microscopic image of plant tissue, likely a cross-section of a root or stem, showing various cellular structures. The image is overlaid with a semi-transparent purple rectangle containing text. On the left side of the purple rectangle, there is a vertical column of dots of varying sizes.

Ultimately achieving an F-1 Score of **0.983** can be considered as a very good performance by the model, and we can proceed with the attempt to create a classification model for 8 classes. However, a problem arises here as some classes have a very small number of representatives when dividing the images into individual diseases (some diseases are well-represented with several thousand images, while others have only a few hundred images). This may hinder the effectiveness of data augmentation in providing the network with sufficient data for learning.

# MULTICLASS CLASSIFICATION

During the initial attempt to create a model that classifies 8 classes of diseases, the approach was similar to the one used for creating a well-performing model for benign and malignant classification. The use of ImageDataGenerator class, callbacks, the pre-trained Xception model, and the stepwise training approach starting from fine-tuning remained unchanged. In order to train the model under as similar conditions as possible, the dataset initially remained **unbalanced**. The only difference was the number of neurons in the last layer.

```
model = Sequential([
    base_model,
    layers.Flatten(),
    layers.BatchNormalization(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Dense(8, activation='softmax')
])
```

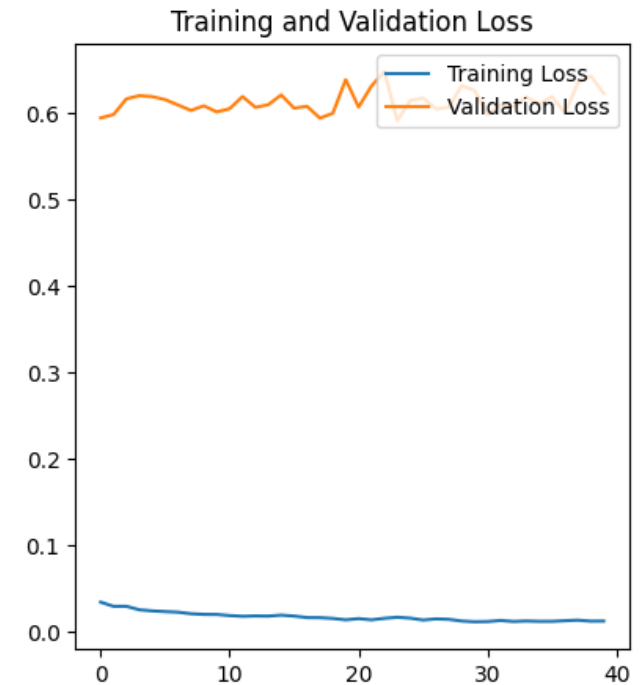
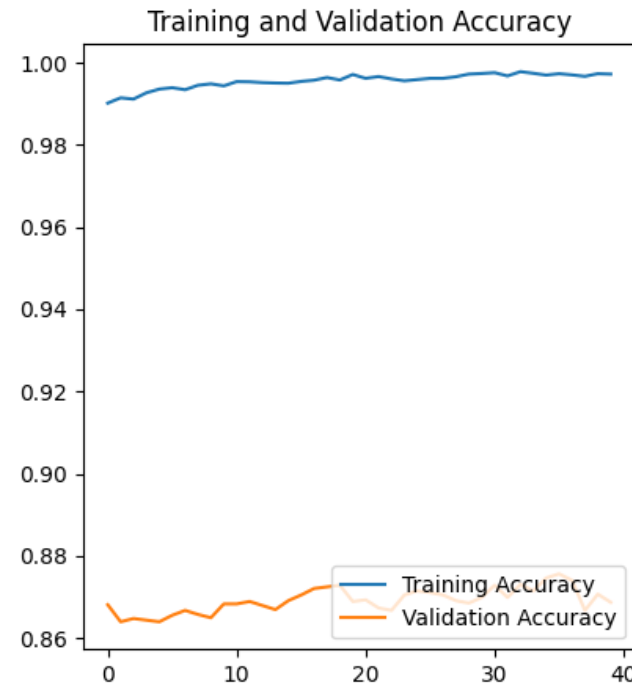


# MULTICLASS CLASSIFICATION - RESULTS

This time, the model underwent 4 stages, consisting of:

- 30 epochs of fine-tuning
- 40 epochs of training the entire network
- 40 epochs of training the entire network
- 40 epochs of training the entire network (the training progress chart is visible alongside)

During the last stage, a plateau in the learning process could be observed, as the accuracy of the network on the training set was already very high.

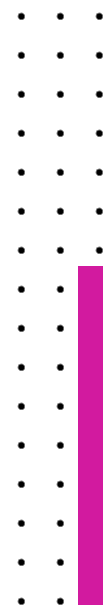




# MULTICLASS CLASSIFICATION - VALIDATION

In this case, the fact that the network was trained on an unbalanced dataset had a significant impact and led to a falsely high result for the unbalanced set. The likely factor that differentiates the unbalanced 8-class dataset from the 2-class dataset is the ratio between the most represented and the least represented class. For the 2-class dataset, this ratio is 2-fold, while for the 8-class dataset, the ratio is already over **10-fold**.

Model	Unbalanced Data	Balanced Data
After the first 30 epochs	0.7053	<b>0.4506</b>
After the next 40 epochs	0.7302	<b>0.4798</b>
After the next 40 epochs	0.8577	<b>0.7143</b>
After the last 40 epochs	0.8695	<b>0.7123</b>

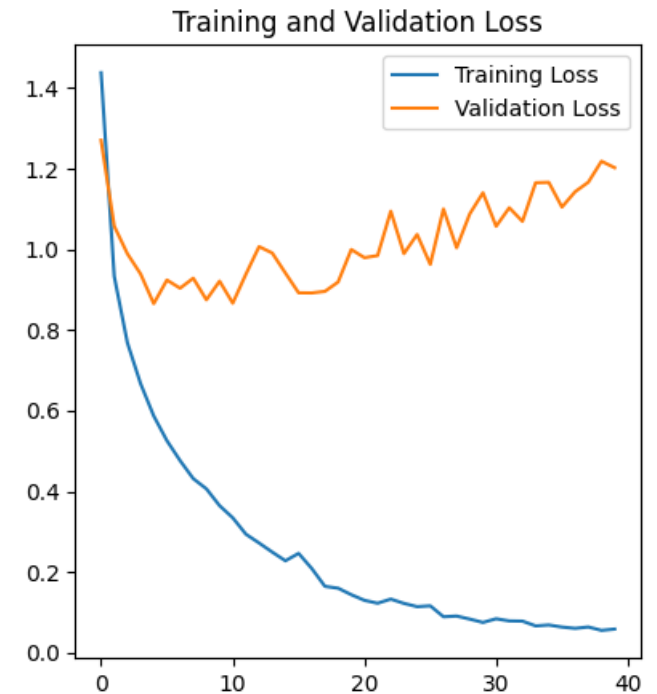
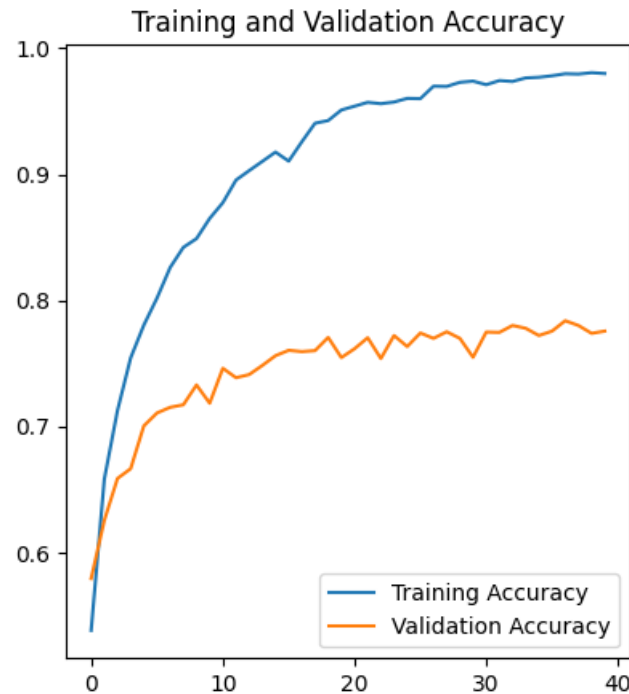


# BALANCING THE DATASET - RESULT

The dataset was balanced by selecting an intermediate number between the quantity of images in the most and least numerous classes, and then equalizing the number of samples for all classes to that value.

The model underwent the same training processes, with the difference being that the training consisted of the following stages:

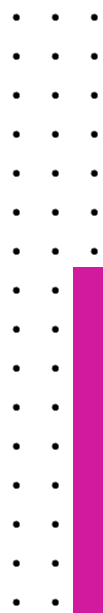
- 40 epochs of fine-tuning
- 40 epochs of fine-tuning
- 40 epochs of training the entire network



# BALANCING THE DATASET - VALIDATION

This time, to extract the most from the fine-tuning stage, an additional 40 epochs were conducted. The network achieved better results, but it is likely that the larger impact was due to training the model on the balanced dataset rather than the additional fine-tuning stage. This is indicated by the stagnation observed in the second stage of the model's training from the previous slide.

Model	Unbalanced Data	Balanced Data
After the first 40 epochs	0.6076	<b>0.5951</b>
After the next 40 epochs	0.6236	<b>0.6105</b>
After the last 40 epochs	0.7444	<b>0.7827</b>



# MODEL MODIFICATION

The last attempt to improve the model involved modifying the final layers responsible for classifying the most abstract features of the images. In addition to increasing the number of neurons in the penultimate dense layer to 256 neurons, an additional dense layer with 1024 neurons was added.

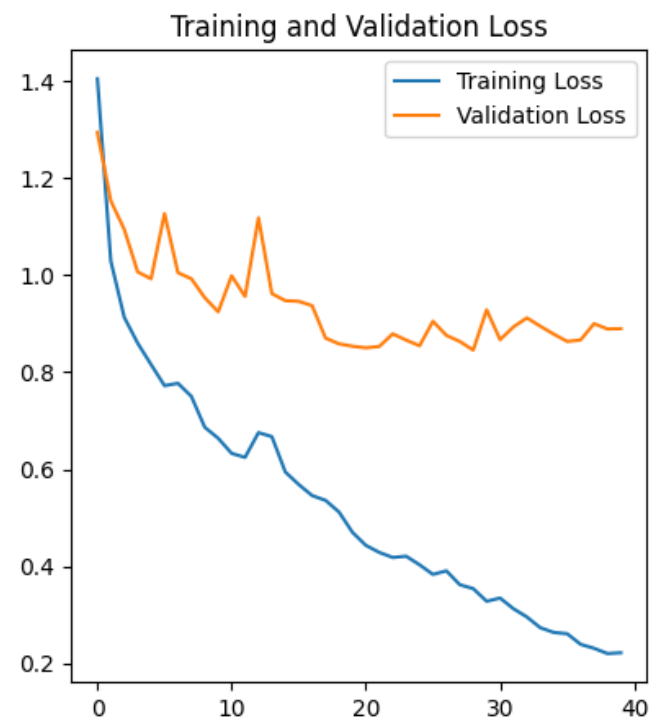
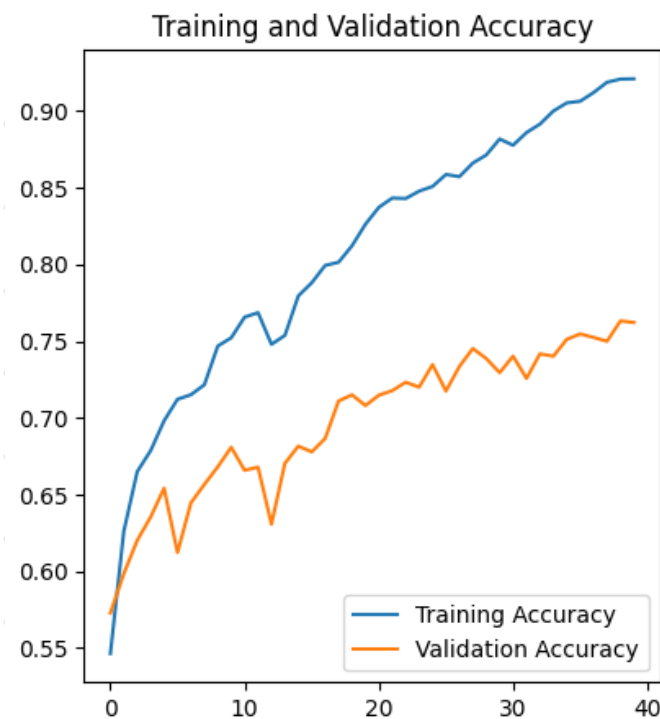
```
model = Sequential([
    base_model,
    layers.Flatten(),
    layers.BatchNormalization(),
    layers.Dense(1024, activation='relu'),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Dense(8, activation='softmax')
])
```



# MODEL MODIFICATION-RESULTS

This time, considering that additional fine-tuning stages do not bring any additional benefits, the model underwent 3 stages consisting of:

- 40 epochs of fine-tuning
- 40 epochs of training the entire network
- 40 epochs of training the entire network (no saved chart)



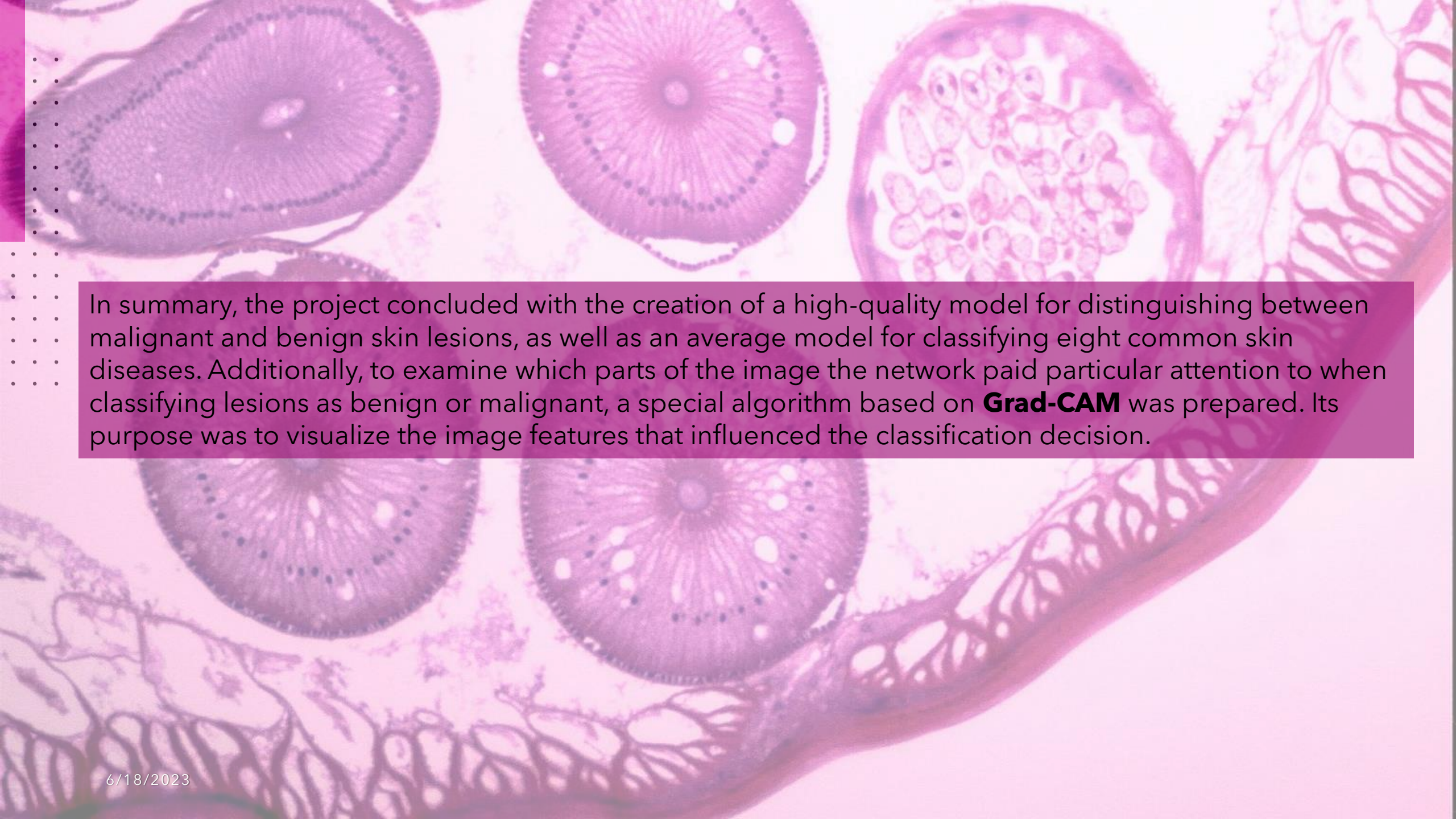


# MODEL MODIFICATION - VALIDATION

Looking at the validation of the model, it cannot be said that changing the last layers had a significant impact on the quality of the network's output in this case. In fact, it seems to be even lower by one percentage point. Unfortunately, this leaves the final model in a state where, unlike binary classification, it was not possible to exceed 80% accuracy.

Model	Unbalanced Data	Balanced Data
After the first 40 epochs	0.6281	<b>0.5913</b>
After the next 40 epochs	0.7317	<b>0.7550</b>
After the last 40 epochs	0.7499	<b>0.7740</b>



A microscopic image of skin tissue, likely a histological section, showing various cellular structures. The image is overlaid with a semi-transparent purple rectangle containing text. On the left side of the purple rectangle, there is a vertical column of dots. The text describes the project's conclusion, mentioning the creation of a high-quality model for distinguishing between malignant and benign skin lesions, an average model for classifying eight common skin diseases, and the use of Grad-CAM to visualize image features influencing classification decisions.

In summary, the project concluded with the creation of a high-quality model for distinguishing between malignant and benign skin lesions, as well as an average model for classifying eight common skin diseases. Additionally, to examine which parts of the image the network paid particular attention to when classifying lesions as benign or malignant, a special algorithm based on **Grad-CAM** was prepared. Its purpose was to visualize the image features that influenced the classification decision.

# GRAD-CAM FOR BINARY CLASSIFICATION

