Vision and Perception Final Project

Breast Cancer Classification

Salvatore Cognetta 1874383
Daniele Appetito 1916560

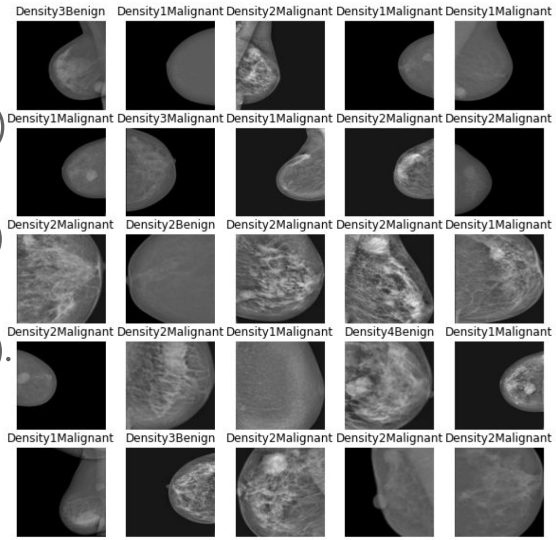
About the project

- Aim of the project was to use a known Neural Network on a dataset of our choice. Then create our own CNN and compare it to the previous networks, running different experiments.
- Different augmentation techniques were used.
- 2 separate networks, InceptionV3 and VGG19, were trained and tested on each type of augmentation.
- Resnet was considered but after producing substantially bad results, we decided to remove it.
- Code was written on a Colab python book using tensorflow framework and keras, sklearn libraries.

Dataset

The Daset (http://dx.doi.org/10.17632/x7bvzv6cvr.1) contains 213 different mammographies (taken from INbreast database) labeled with 8 different classes.
The eight categories are:

- breast density is 1 and breast mass is benign (Density1+Benign)
- breast density is 1 and breast mass is malignant (Density1+Malignant)
- breast density is 2 and breast mass is benign (Density2+Benign)
- breast density is 2 and breast mass is malignant (Density2+Malignant)
- breast density is 3 and breast mass is benign (Density3+Benign)
- breast density is 3 and breast mass is malignant (Density3+Malignant)
- breast density is 4 and breast mass is benign (Density4+Benign)
- breast density is 4 and breast mass is malignant (Density4+Malignant).



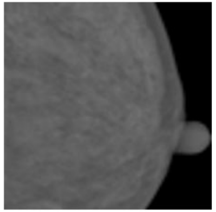
Each mammography of the Dataset is a 224x224 black and white image.

Splitting the dataset

The Dataset initially was not splitted in train/test, so we splitted it with ratio 80/20. Finally we have 169 images belonging to 8 classes for the train set and 44 images for the test set, divided like this:

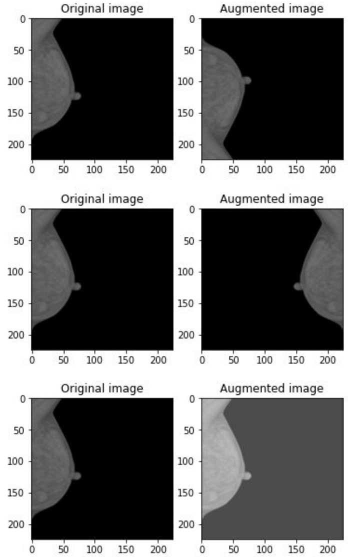
	Train	Test
Density 1 Malignant	48	12
Density 1 Benign	19	5
Density 2 Malignant	51	13
Density 2 Benign	6	2
Density 3 Malignant	13	3
Density 3 Benign	21	5
Density 4 Malignant	1	1
Density 4 Benign	10	2

Data Augmentations



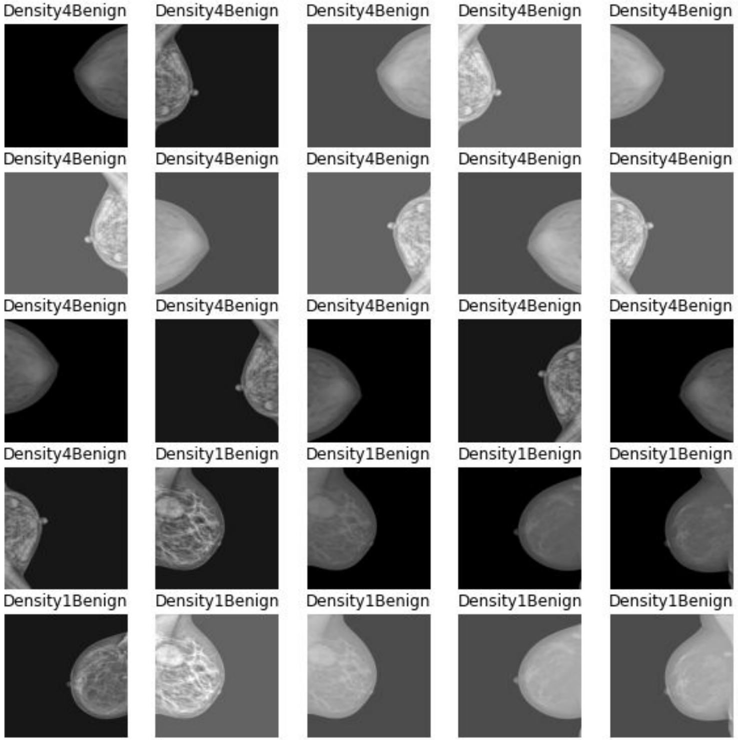
Considering the small Dataset, we applied different types of augmentation:

- Centered Crop and resize: created a bounding box around the breast and cropped the image
- Horizontal flip
- Vertical flip
- Brightness (+0.3)
- Crop + vertical flip
- Crop + vertical flip + horizontal flip
- All augmentation combined (after which the augmented dataset, which initially contained only 170 images, had 2720 images)



Data augmentation on validation set

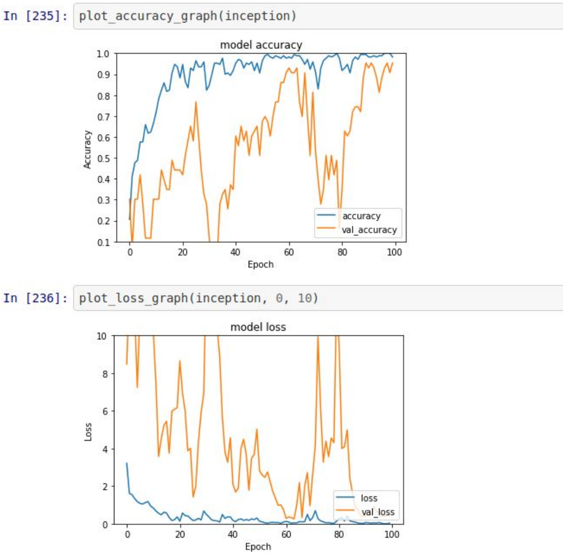
We also applied data augmentation on the really small validation set, but we didn’t consider the crop augmentation because it can possibly remove significant information of the image/breast. The final validation set contains 344 images and was augmented with vertical and horizontal flip and brightness modification, which doesn’t affected the contained informations of the images:



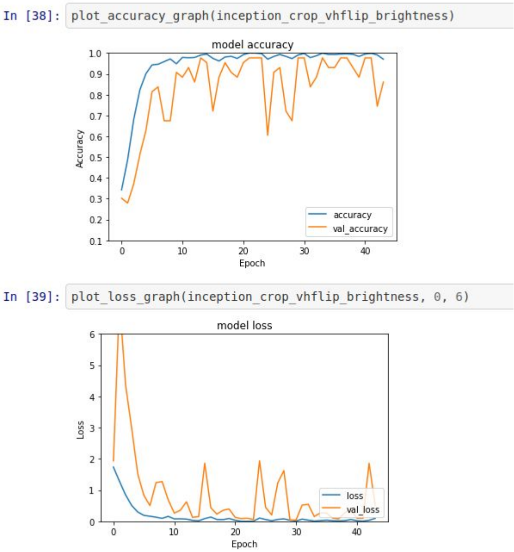
This validation set is used only for model prediction and creation of the classification report

Augmentation importance

As we can see on the following graphs the augmentation affected a lot the performance of the Neural Network, bringing to a more stable ANN in terms of accuracy and loss:



Accuracy and loss of Inception trained on standard Dataset



Accuracy and loss of Inception trained on fully augmented Dataset

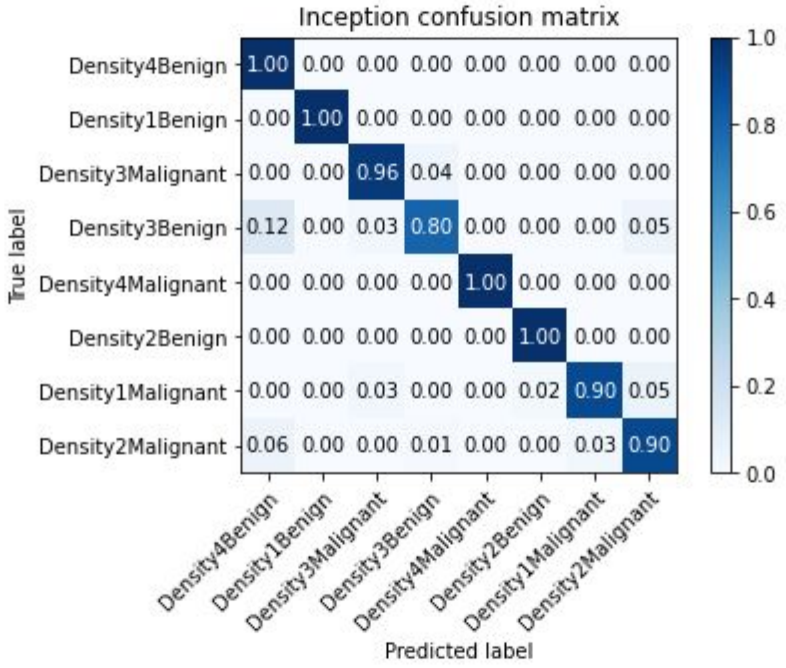
Training on Inception

- InceptionV3 is a widely-used image recognition model with 78.1% accuracy on ImageNet dataset. Instead of using transfer learning on our dataset, we trained all the Neural Network on our Dataset, unfreezing all the layers.
- Added two fully connected layers at the end of the model, with the last layer with 8 output nodes (our classes number).
- Using Adam optimizer.
- Using kullback leibler loss function.
- Saving best model as .h5 file to be able to use/ look at it later
- Training for 100 epochs using early stopping as regularization factor, to avoid overfitting (the metric used for early stopping is validation accuracy)

Evaluation of Inception

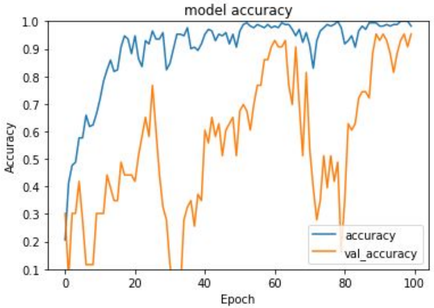
Results of Inception trained on fully augmented Dataset

	precision	recall	f1-score	support
Density4Benign	0.593	1.000	0.744	16
Density1Benign	1.000	1.000	1.000	40
Density3Malignant	0.852	0.958	0.902	24
Density3Benign	0.941	0.800	0.865	40
Density4Malignant	1.000	1.000	1.000	8
Density2Benign	0.889	1.000	0.941	16
Density1Malignant	0.966	0.896	0.930	96
Density2Malignant	0.931	0.904	0.917	104
accuracy			0.916	344
macro avg	0.896	0.945	0.912	344
weighted avg	0.928	0.916	0.918	344

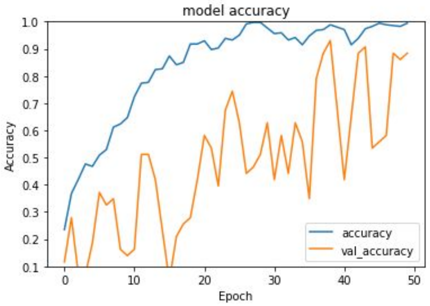


InceptionV3 Results

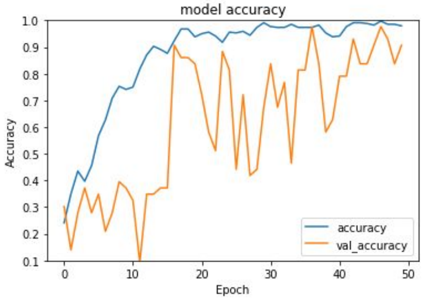
```
plot_accuracy_graph(inception)
```



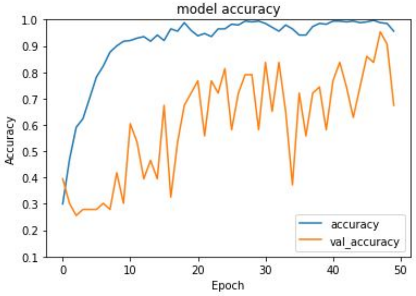

```
plot_accuracy_graph(inception_cropped)
```



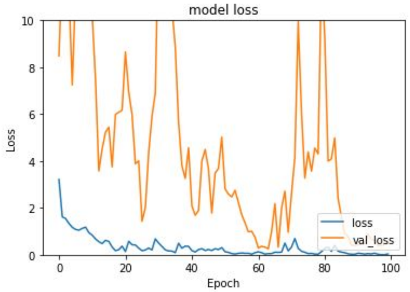

```
plot_accuracy_graph(inception_hflip)
```




```
plot_accuracy_graph(inception_vflip)
```

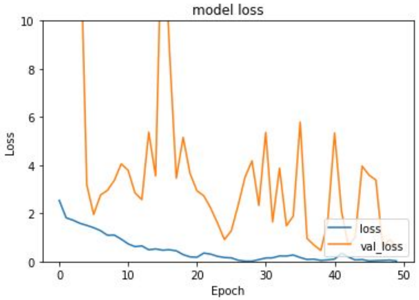



```
plot_loss_graph(inception, 0, 10)
```



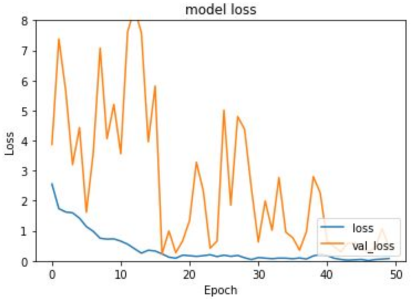
loss: 0.2546 - accuracy: 0.9302

```
plot_loss_graph(inception_cropped, 0, 10)
```



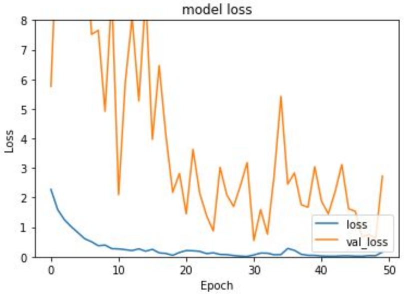
loss: 0.4595 - accuracy: 0.9302

```
plot_loss_graph(inception_hflip, 0, 8)
```



loss: 0.2274 - accuracy: 0.9070

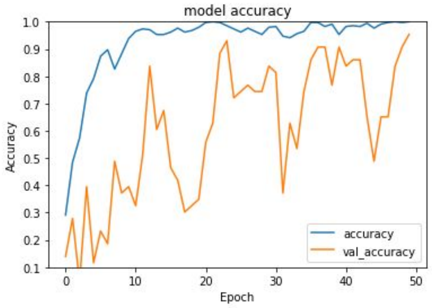
```
plot_loss_graph(inception_vflip, 0, 8)
```



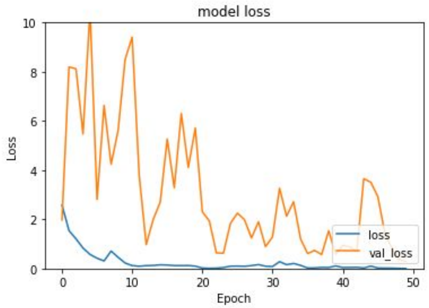
loss: 0.5519 - accuracy: 0.8372

InceptionV3 Results

```
plot_accuracy_graph(inception_brightness)
```

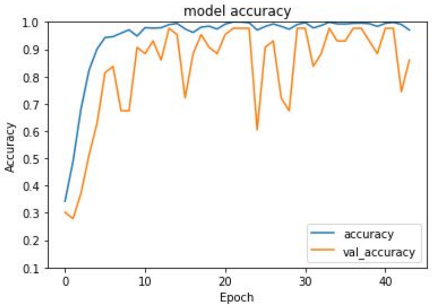



```
plot_loss_graph(inception_brightness, 0, 10)
```

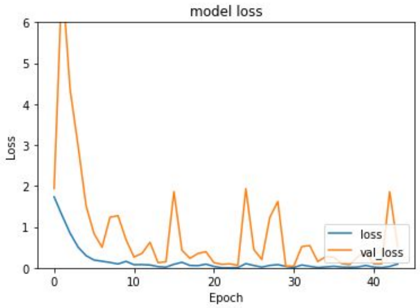


loss: 0.2834 - accuracy: 0.9535

```
plot_accuracy_graph(inception_crop_vhflip_brightness)
```




```
plot_loss_graph(inception_crop_vhflip_brightness, 0, 6)
```



loss: 0.0799 - accuracy: 0.9767

Considerations

As expected the validation accuracy and the validation loss is better on the fully augmented model.

We were surprised that the Dataset with plain images and cropped one wasn't good, while our thoughts at the beginning were the opposite. This can be due to the fact that the crop removed useful information about the cancerous mass, even if our crop was aimed to remove the least information possible.

The best overall augmentation is the brightness adjustment, our considerations was that adjust the brightness highlight breast cancer mass.

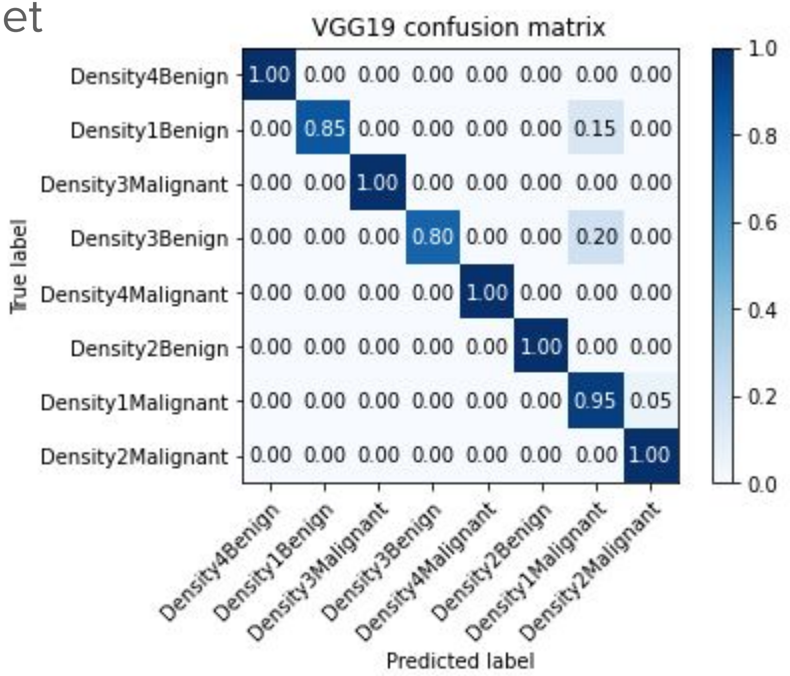
Training on VGG

- Also for VGG19, instead of using transfer learning on our dataset, we trained all the Neural Network on our Dataset, unfreezing all the layers.
- Added two fully connected layers at the end of the model, with the last layer with 8 output nodes (our classes number).
- In this case we used Stocastic Gradient Descent, with learning rate of 0.01, becuase with the Adam optimizer the accuracy didn’t changed, meaning that this optimizer was not suited for our Dataset using VGG
- Using Categorical Cross Entropy loss function.
- Saving best model as .h5 file to be able to use/ look at it later
- Training for 100 epochs using early stopping as regularization factor, to avoid overfitting (the metric used for early stopping is validation accuracy)

Evaluation of VGG

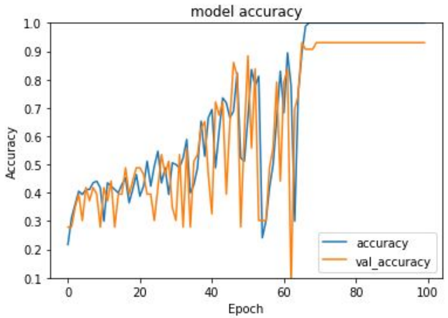
Results of VGG trained on fully augmented Dataset

```
11/11 [==============================] - 2s 153ms/step
                   precision    recall  f1-score   support

   Density4Benign      1.000     1.000     1.000        16
   Density1Benign      1.000     0.850     0.919        40
Density3Malignant      1.000     1.000     1.000        24
   Density3Benign      1.000     0.800     0.889        40
Density4Malignant      1.000     1.000     1.000         8
   Density2Benign      1.000     1.000     1.000        16
Density1Malignant      0.867     0.948     0.905        96
Density2Malignant      0.954     1.000     0.977       104

         accuracy                          0.945       344
        macro avg      0.978     0.950     0.961       344
     weighted avg      0.949     0.945     0.944       344
```

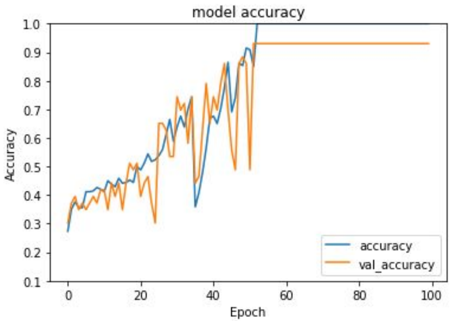


VGG19 Results

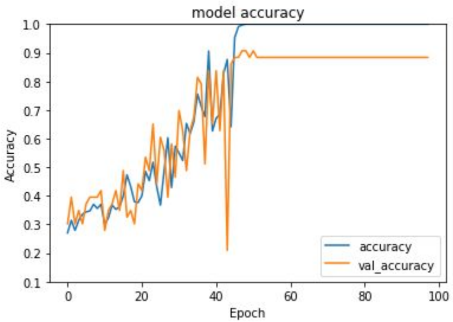
```
plot_accuracy_graph(VGG)
```



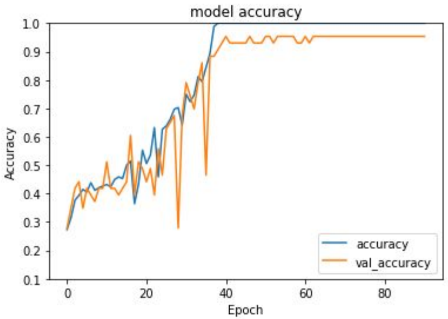

```
plot_accuracy_graph(vgg_cropped)
```



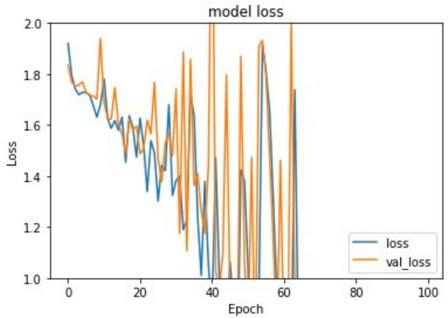

```
plot_accuracy_graph(vgg_hflip)
```




```
plot_accuracy_graph(vgg_vflip)
```

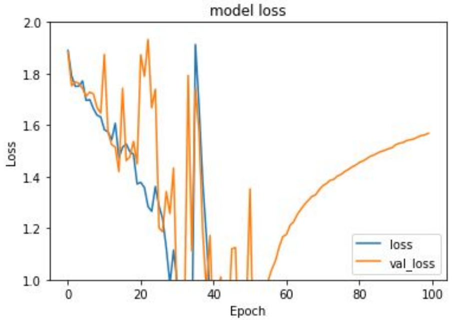



```
plot_loss_graph(VGG, 1, 2)
```



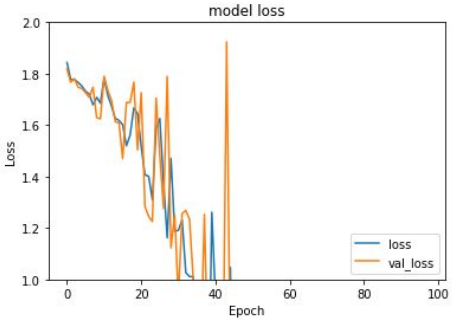
loss: 0.4396 - accuracy: 0.9070

```
plot_loss_graph(vgg_cropped, 1, 2)
```



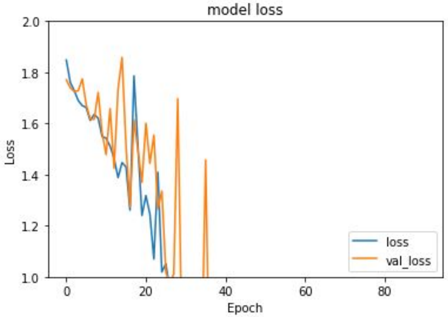
loss: 0.6414 - accuracy: 0.9302

```
plot_loss_graph(vgg_hflip, 1, 2)
```



loss: 0.3176 - accuracy: 0.9070

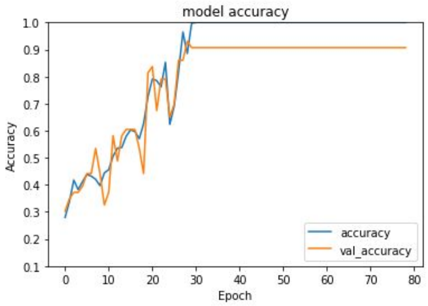
```
plot_loss_graph(vgg_vflip, 1, 2)
```



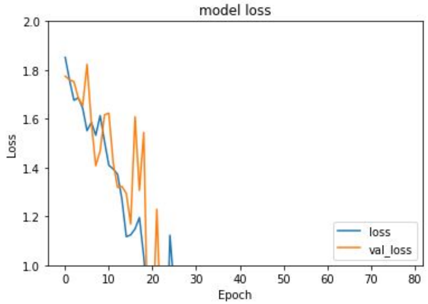
loss: 0.4109 - accuracy: 0.8837

VGG19 Results

```
plot_accuracy_graph(vgg_brightness)
```

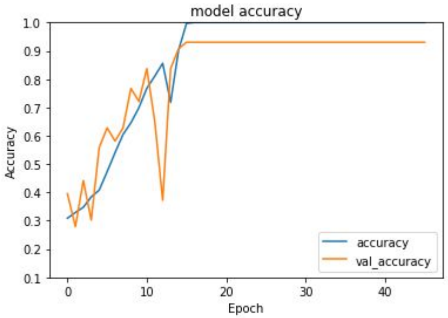



```
plot_loss_graph(vgg_brightness, 1, 2)
```

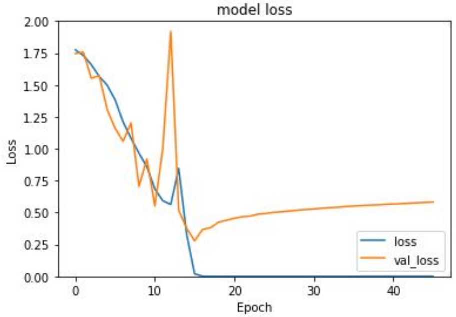


loss: 0.4140 - accuracy: 0.9302

```
plot_accuracy_graph(vgg_crop_vhflip_brightness)
```




```
plot_loss_graph(vgg_crop_vhflip_brightness, 0, 2)
```



loss: 0.2784 - accuracy: 0.9302

Considerations

Overall VGG19 performed better than Inception, reflecting the same considerations about data augmentation done before on Inception. In face as expected the validation accuracy and the validation loss is better on the fully augmented model.

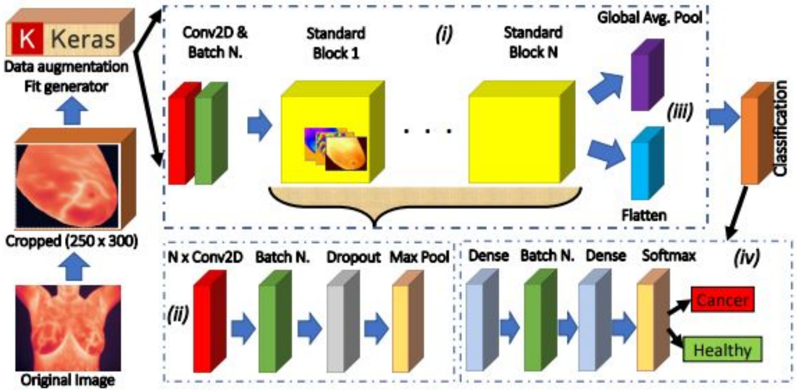
Also in this case the best overall augmentation is the brightness adjustment.

Custom CNN

For the custom CNN we take inspiration from the paper "*A CNN-BASED METHODOLOGY FOR BREAST CANCER DIAGNOSIS USING THERMAL IMAGES*".

The model consist of two part:

- One with a number n of 2D Convolutional layer, with small (3x3) kernels, standard padding (valid) and strides (1,1) and ReLU activation function. After it we added a batch normalization layer, because usually a training algorithm works better on normalized data. After this layer we added a Dropout layer, to reduce overfitting removing '*rate*' `(Fraction of the input units to drop)` neurons at training time. Finally there is a max pooling layer with default pool size (2x2)
- The second one is the classification part, with 2 fully connected layer and softmax activation. The last layer has obviously 8 output neurons.



CNN layout

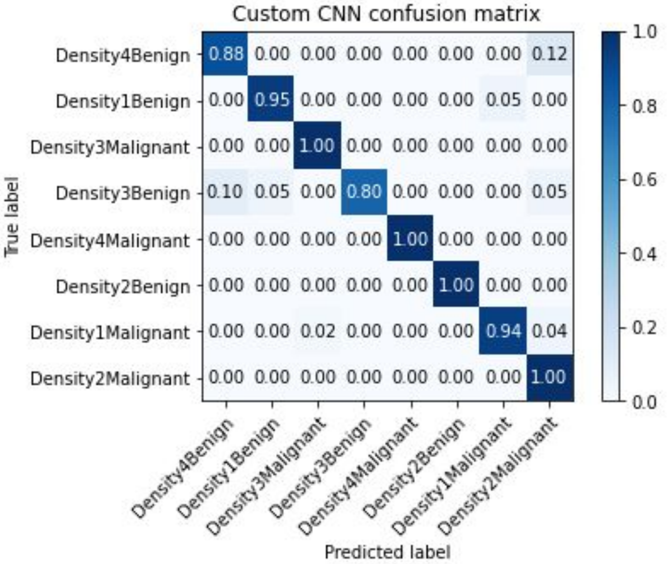
Sequential model with Convolutional layer, followed by a Relu activation layer, then a Batch Normalisation and Dropout layer. This block of layers can be repeated N times, but produced unsatisfying results for us as we increased the number of blocks.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 3)       84
_________________________________________________________________
re_lu (ReLU)                 (None, 222, 222, 3)       0
_________________________________________________________________
batch_normalization (BatchNo (None, 222, 222, 3)       12
_________________________________________________________________
dropout (Dropout)            (None, 222, 222, 3)       0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 3)       0
_________________________________________________________________
flatten (Flatten)            (None, 36963)             0
_________________________________________________________________
dense (Dense)                (None, 32)                1182848
_________________________________________________________________
batch_normalization_1 (Batch (None, 32)                128
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 264
_________________________________________________________________
softmax (Softmax)            (None, 8)                 0
=================================================================
Total params: 1,183,336
Trainable params: 1,183,266
Non-trainable params: 70
```

Evaluation of the custom CNN

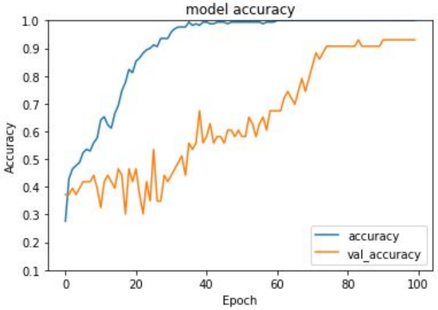
Results of VGG trained on fully augmented Dataset

```
11/11 [==============================] - 1s 87ms/step
                   precision    recall  f1-score   support

   Density4Benign      0.778     0.875     0.824        16
   Density1Benign      0.950     0.950     0.950        40
Density3Malignant      0.923     1.000     0.960        24
   Density3Benign      1.000     0.800     0.889        40
Density4Malignant      1.000     1.000     1.000         8
   Density2Benign      1.000     1.000     1.000        16
Density1Malignant      0.978     0.938     0.957        96
Density2Malignant      0.929     1.000     0.963       104

         accuracy                          0.948       344
        macro avg      0.945     0.945     0.943       344
     weighted avg      0.951     0.948     0.947       344
```

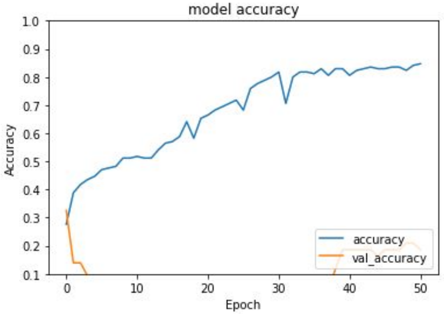


Custom CNN Results

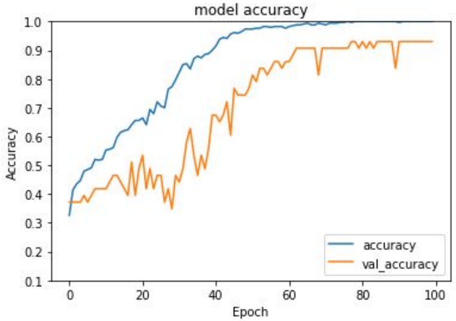
```
plot_accuracy_graph(custom_cnn)
```



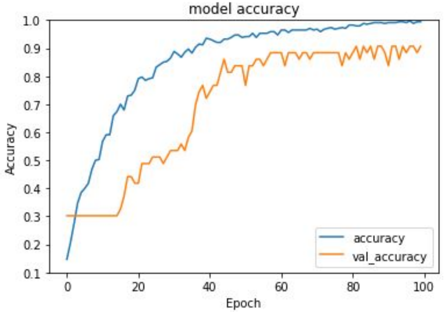

```
plot_accuracy_graph(custom_cnn2)
```



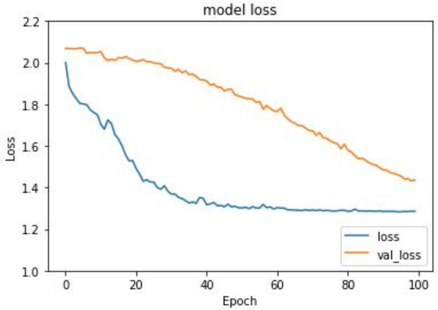

```
plot_accuracy_graph(custom_cnn_crop)
```




```
plot_accuracy_graph(custom_cnn_crop2)
```

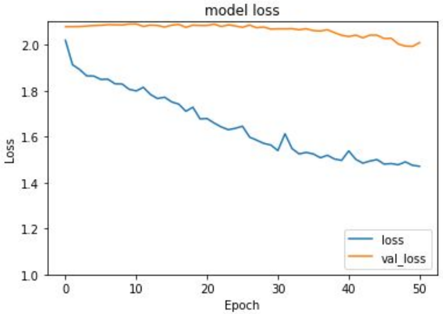



```
plot_loss_graph(custom_cnn, 1, 2.2)
```



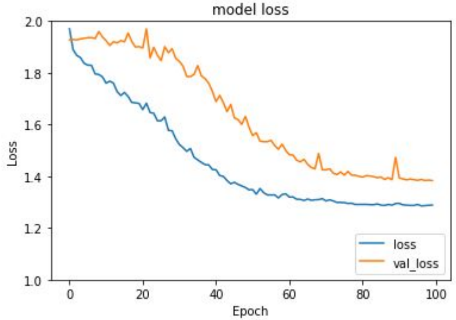
loss: 1.4318 - accuracy: 0.9302

```
plot_loss_graph(custom_cnn2, 1, 2.1)
```



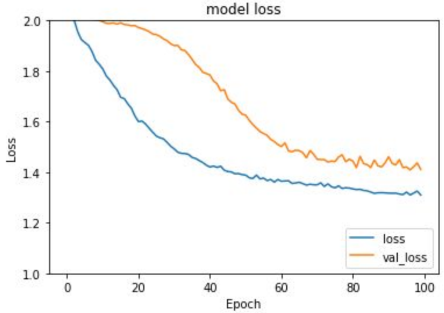
loss:1.9924 - accuracy: 0.2093

```
plot_loss_graph(custom_cnn_crop, 1, 2)
```



loss: 1.3833 - accuracy: 0.9302

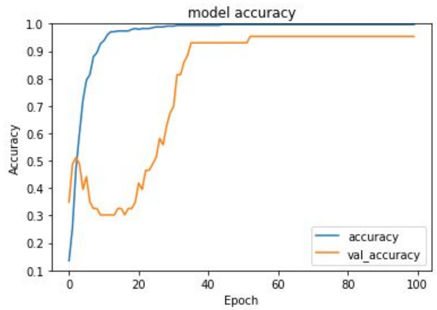
```
plot_loss_graph(custom_cnn_crop2, 1, 2)
```



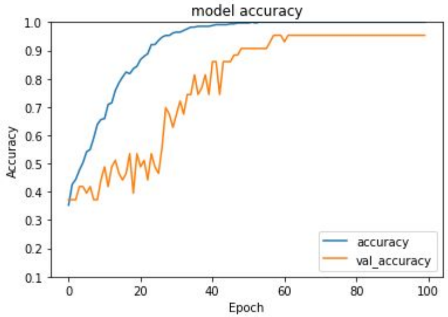
loss: 1.4088 - accuracy: 0.9070

Custom CNN Results

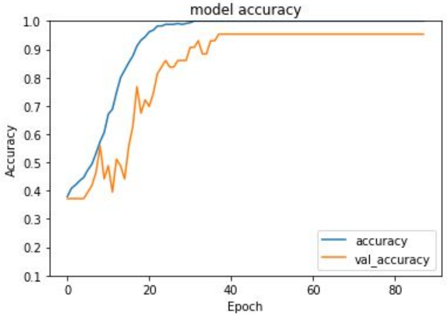
`plot_accuracy_graph(custom_cnn_hflip)`



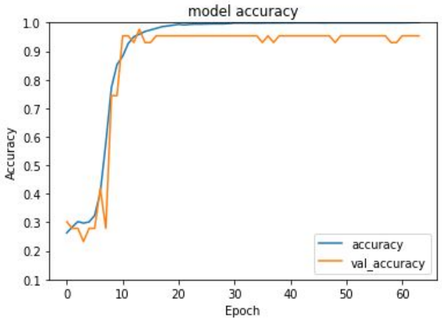
`plot_accuracy_graph(custom_cnn_vflip)`



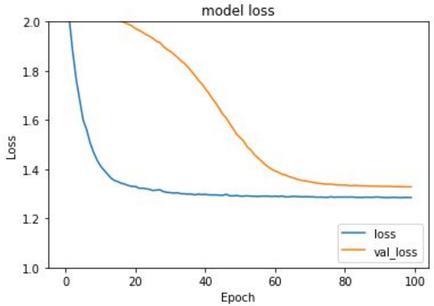
`plot_accuracy_graph(custom_cnn_brightness)`



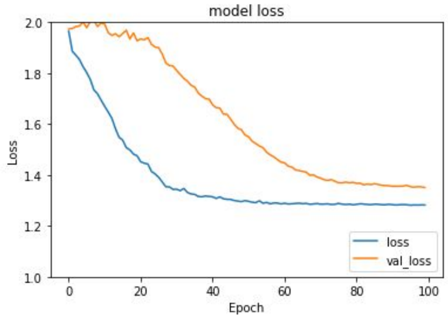
`plot_accuracy_graph(custom_cnn_crop_vhflip_brightness)`



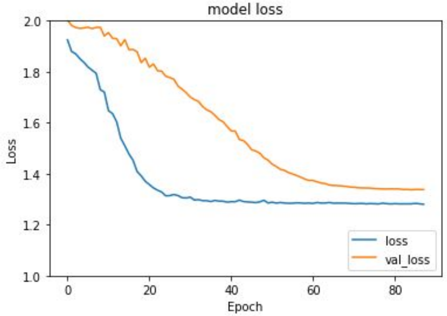
`plot_loss_graph(custom_cnn_hflip, 1, 2)`



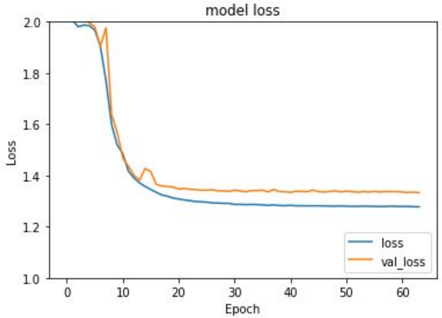
`plot_loss_graph(custom_cnn_vflip, 1, 2)`



`plot_loss_graph(custom_cnn_brightness, 1, 2)`



`plot_loss_graph(custom_cnn_crop_vhflip_brightness, 1,`



loss: 1.3280 - accuracy: 0.9535

loss: 1.3507 - accuracy: 0.9535

loss: 1.3370 - accuracy: 0.9535

loss: 1.3324 - accuracy: 0.9535

Considerations

Overall our CNN performed surprisingly well, although the loss is very high compared to the other two Networks. This could be due to the unbalanced nature of the data, as in the plain dataset there is only 1 image in “Density4Malignant”. Such dataset could have resulted in overfitting.

In this case brightness and horizontal flip were equally good, with a higher accuracy than the other augmentations.

CNN2 was a model created with multiple “standard” blocks instead of just the one. This can be seen to perform worse overall, especially on the plain dataset where accuracy plummets.

References

- https://arxiv.org/pdf/1910.13757.pdf