A binary classification of skin cancer using Convolutional Neural Networks

Ghita Ait Ouhmane
Master student in Computer Science engineering
Université Libre de Bruxelles
ghita.ait.ouhmane@gmail.com

ABSTRACT

According to the WHO, The incidence of both non-melanoma and melanoma skin cancers has been increasing over the past decades. AI powered systems can help in the early detection and diagnosis of skin cancers. However, given the domain of application, these models have to provide an accurate enough procedure to avoid as much as possible giving false diagnoses. In this paper, we will be presenting a CNN built using transfer learning with the VGG16 architecture pre-trained on the ImageNet dataset, which is a very large dataset used for image classification and detection. The proposed model achieves a cross validation result of 85.21% (+/- 2.02%) and a testing accuracy of 84.54%.
The implementation of the model can be found in the following repository :
https://github.com/Ghita-Ait-Ouhmane/MedicalImageryProject

1 INTRODUCTION

In current medical practice for skin cancer diagnosis, dermatologists examine patients visually with assistance of polarized light magnification via dermoscopy. Medical diagnosis often depends on the patient's history, ethnicity, social habits and exposure to the sun. Lesions of concern are biopsied in an office setting, submitted to the laboratory, processed as permanent paraffin sections, and examined as representative glass slides by a pathologist to render a diagnosis.
AI-enabled computer-aided diagnostics (CAD) solutions are poised to revolutionize medicine and health care, especially in medical imaging. [10]. Among all the different type of models used in machine learning, Convolutional Neural Networks (CNNs) are by far the most popular ones for classification tasks, and the one that was used to construct our model.
In this paper, we will present a CNN model to classify images of skin texture into malignant or benign.

2 DATASET

The dataset used on the model is the *Skin Cancer: Malignant vs. Benign* [12] dataset from Kaggle. It is actually a processed version of the ISIC archive, an open source public access archive of skin images created for the development and testing of automated diagnostic systems to reduce melanoma mortality.[3]

The dataset is separated into two folders, for training/validation and for testing.

The figure 1 shows some random samples from the dataset.

	Training set	Testing set	Total
Benign	1440	360	1800
Malignant	1197	300	1497

Table 1: Image labels in dataset

There are approximately 3300 RGB images of size 224x224. The testing set contains 660 entries, so about 20% of the whole dataset. The images are organized in folders depending on their true classification labels.

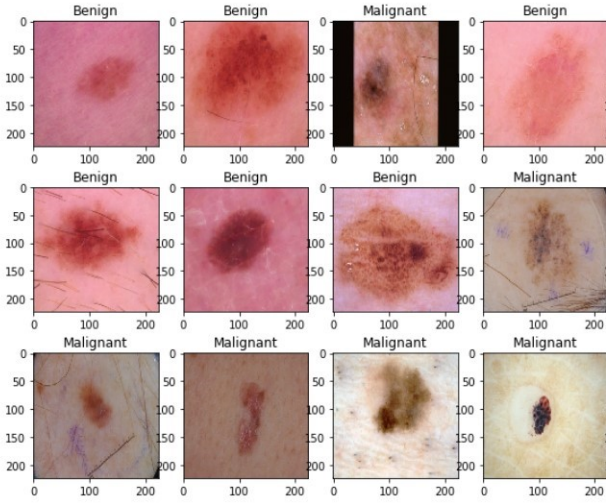

Figure 1: Samples of images from the Kaggle dataset [12]

A first question that can be asked is whether the dataset is balanced or not. Looking at the table 1, the number of moles classified as benign is close enough to the number of malignant ones, so we can consider that the dataset is balanced.
The images dimensions are all the same and don't need any form of preprocessing, so we can move on the next section which is the model's description.

3 MODEL

3.1 Presentation of CNNs

The model we have decided to use is a CNN : Convolutionnal Neural Networks.
Widely used for image recognition, CNNs are a particular type of feed-forward neural networks in AI. They represent the input data in the form of multidimensional arrays by extracting each and every portion of input images, which is known as the receptive field. It then assigns weights for each neuron based on the significant role of the receptive field, so that it can discriminate the importance of neurons from one another [14].

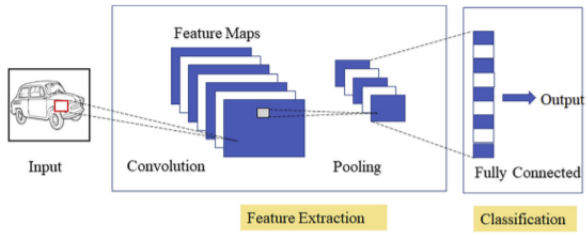

Figure 2: CNN architecture[14]

The architecture of CNNs is shown in Figure 2, and consists of three types of layers :

(1) Convolution : its goal is to obtain significant features from the input. The convolutional layer convolves the input image with set of kernels or filters and learnable parameters to generate a feature map.
It can do so by first performing the dot product between the filters and the receptive field of input image, and then shift the receptive fields step by step across the width and height of the input image.
In an effort to reduce the number of hyper parameters of the model, this layer shares the same parameters in every portion of the image.

(2) Pooling : It performs subsampling or downsampling of the images, thus reducing the dimensionality of the feature maps. It computes the mean or maximum value of the feature maps.

(3) Fully connected : In this layer, neighboring neurons are connected together to flatten the matrix. It classifies the features extracted in the previous layers to produce the final output.

This way, CNNs learn the relationship between the input objects and the class labels. Unlike regular neural networks, the hidden layers of a CNN have a specific architecture. In regular neural networks, each layer is formed by a set of neurons and one neuron of a layer is *connected* to each neuron of the preceding layer. As we have seen, the architecture of hidden layers in a CNN doesn't connect the neurons in a layer to those of the preceding layer; rather, they are connected to only a small number of neurons. This restriction to local connections and additional pooling layers summarizing local neuron outputs into one value results in translation-invariant features.
As a result we obtain simpler training procedures and a lower model complexity.[14][5]
This was a general introduction to CNNs. For our specific problem, we have decided to use the VGG16 CNN, which will be detailed in the following section.

3.2 VGG16

VGG16 is a convolutionnal neural network model that was submitted to the ILSVRC competition, which is a challenge that evaluates algorithms for object detection and image classification at large scale. It achieved 92.7% accuracy on the ImageNet dataset, which is a large database of over 14 million images. It was designed by academics intended for computer vision research.

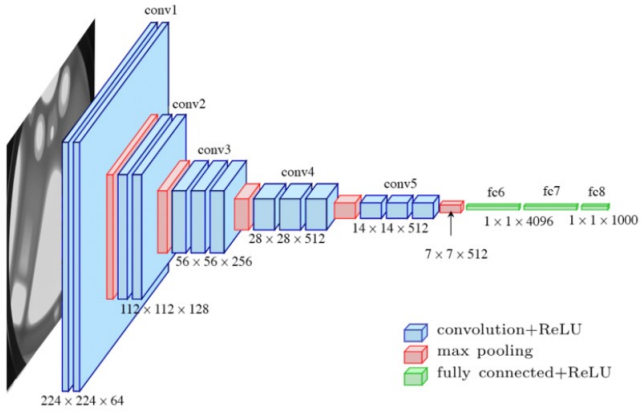

Figure 3: VGG16 architecture[8]

Layer	Convolution	Output dim	Pooling	Output
1 2	64 channels 3x3 kernel	224x224x64	maxpool stride 2 2x2	112x112x64
3 4	128 channels 3x3 kernel	112x112x128		56x56x128
5 6 7	256 channels 3x3 kernel	56x56x256		28x28x256
8 9 10	512 channels 3x3 kernel	28x28x512		14x14x512
11 12 13	512 channels 3x3 kernel	14x14x512		7x7x512

Table 2: Summary of layers characteristics in VGG16

3.2.1 *Model description*

. The architecture of the VGG16 is represented in figure 3.

As we can see, there are five convolution steps, each followed by a pooling (max pool in this case).
The table 2 regroups the main characteristics of each convolution step. The size of kernels for all the convolution layers is always the same, 3x3. The maxpool is also the same. The only difference comes from the number of channels that changes in each stage of the convolution.
The model follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 3 FC (fully connected layers) : the first two have 4096 channels each, the third contains 1000 channels (one for each class). The final layer is the soft-max layer.[16][13]
This architecture results in a total of approximately 138 million parameters.

4 IMPLEMENTATION

In this section we will describe how we processed our data and used the VGG16 to build our binary classification model.

4.1 Pre-processing

The data was already split into a training folder and a testing folder, where each of the images have been labeled by being put in a subfolder "malignant" or "benign". So the data only needed to be loaded in lists, and the corresponding labels were set to 0 for benign and 1 for malignant moles.
The only pre-processing operation we apply on our images is normalization, to keep the values in a narrow specific range. We are dealing with 8-bit images, so we divide all inputs by 255 to normalize our data.

4.2 Model construction

To implement the VGG16 architecture, we used a predefined function in the `keras-application` library, called `VGG16` [15].
We explained earlier that the VGG16 architecture was initially built for an object detection and image classification competition, using the ImageNet dataset (150 GB), on which it performed with 92.7% accuracy.
The keras function offers a built in functionality which consists in loading these weights to build a new model : this is transfer learning.
Transfer learning consists of taking features learned on one problem, and leveraging them on a new, similar problem.[7] Since the number of parameters in the VGG16 model is quite large, this could be an interesting approach to reduce the training time on our dataset.
All the convolution and pooling layers weights were loaded using transfer learning, and "froze" so that they won't be modified during training, but the classification block was modified : for the fully connected layers, the number of channels was replaced to 32 for the first FC layer, 16 for the second one, and 1 for the last layer, which would according to the function documentation [15] correspond to the number of classes for our classification. The activation function was also modified from softmax to sigmoid. (Softmax is used for multi-classification problems, where as Sigmoid is used for binary classifications [4]).

The model summary is represented in the following :

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 32)                802848
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                528
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 15,518,081
Trainable params: 803,393
Non-trainable params: 14,714,688
```

Figure 4: Model summary

As we can see, the majority of the parameters don't need to be trained since we are doing transfer learning, only the one added for the FC layers need to be.

5 TRAINING

Once the model has been built, the training phase can start. Training consists in finding a set of weights which will fit best to our model and be able to , given input data and the expected true values, predict the correct labels.
But before starting the training, we have to define which parameters will be used to define how "far" we are from the true values, and how many times our predictions are right.

5.1 Metrics

The first metric we will introduce is the *loss* : it specifies how training penalizes the deviation between the predicted output of the network, and the true data labels [2].It can be seen as the distance between the true values of the problem and the values predicted by the model.
There exists numeral functions to evaluate the loss during the training. We will be using *binary cross entropy* since we are dealing with a binary classification problem.

$$\text{Loss} = -\frac{1}{\underset{\text{size}}{\text{output}}} \sum_{i=1}^{\underset{\text{size}}{\text{output}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

Figure 5: Binary cross entropy [1]

Another parameter that will be used is the *accuracy* : it is the fraction of predictions the model got right, compared to the total number of predictions.[9]

The last metric we will define is the *optimizers* : these are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses [6]. One of the most popular for its effectiveness and speed is Adam, so we will use it.

5.2 Parameters

Now that the metrics are all defined, we can define the training parameters, which are summarized in the following table .

validation split	nb epochs	batch size
0.2	20	64

Table 3: Training parameters

The validation split represents which percentage of the dataset will be used to evaluate the training parameters. The number of epochs is the number of times the whole dataset will be fed to the model. As for the batch size, it represents the number of inputs to work through before updating the model parameters.

5.3 Results

Since the number of trainable parameters is rather small, the training goes fast.
The evolution of the model accuracy and loss during both training and validation have been plotted as a function of the number of epochs the model has gone through.

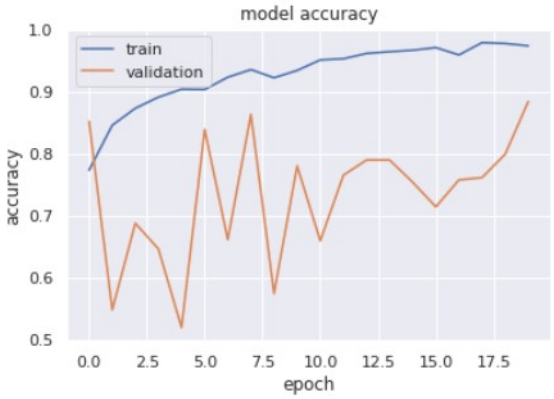

Figure 6: Accuracy training plot

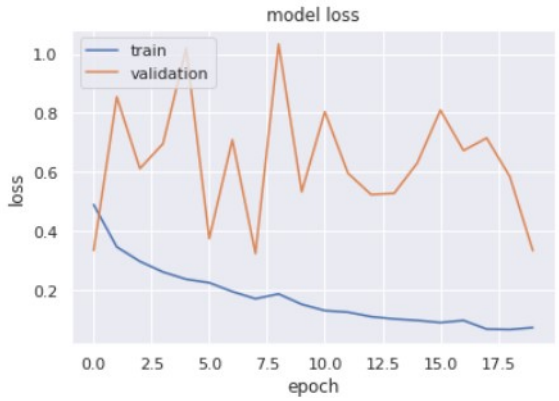

Figure 7: Loss training plot

5.4 Discussion

The training accuracy and loss plots both follow a steady evolution : the accuracy grows which each epoch, meaning we predict more and more samples correctly, where as the loss decreases, meaning that we succeed in minimizing the loss function between the predicted labels and the true values.

The validation, however, doesn't give as promising results; the values oscillate a lot, only starting to monotonically increase/decrease on the last 5 epochs.
This could suggest some overfitting. However, since the model evolves steadily at the end, we will see what kind of performance we obtain with it.

To be able to assess the estimated accuracy of our model without using a testing set, we will perform cross validation.

6 CROSS VALIDATION

Crossed validation is a method used to get an estimate of a model's generalization performance. If a model performs well on the training data but generalizes poorly according to the cross-validation metrics, then it is overfitting. If it performs poorly on both, then it is underfitting. This is one way to tell when a model is too simple or too complex [11].

We will be using 3-fold cross validation, which consists in dividing the dataset into 3 folds and iteratively training it on 2 folds and make a prediction on the last one, 3 times. This way, we verify if our model is not overfitting with the given dataset.
We obtain the following accuracies :

86.92% - 82.37% and 86.35% so 85.21% (+/- 2.02%) as an average.

We can see that the accuracies values are not too far from each other, suggesting that the model is not overfitting.

7 TESTING

Now that we trained and cross validated our model, we can finally predict the output of the testing set.

We obtain a value of **84.54%**, which isn't so far from the validation accuracies ! To obtain a more visual vision of which labels have been (un)correctly predicted, we can plot the confusion matrix of the predictions, compared to the expected results.

8 CONFUSION MATRIX

To evaluate the classification quality, the class assigned by the model is compared with the true class. This allows the objects to be divided into four categories [5] :

- True positive (TP): the classifier correctly predicts the positive class.
- True negative (TN): the classifier correctly predicts the negative class.(here 0)
- False positive (FP): the classifier incorrectly predicts the positive class.
- False negative (FN): the classifier incorrectly predicts the negative class.

Knowing the true labels, these values can easily be computed. That is what the confusion matrix represents.

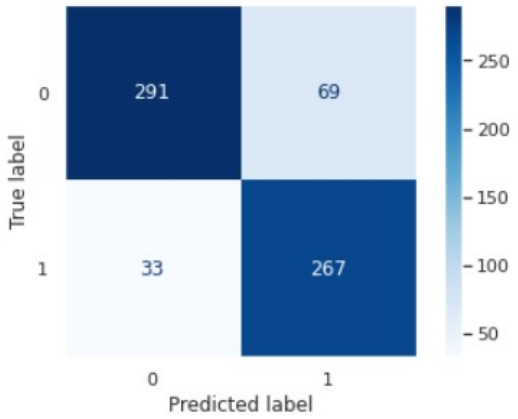

Figure 8: Confusion matrix

We can see in figure 8, as expected that around 80% of the labels are correctly predicted. As for the mislabelled ones, 11% of the malignant are classified as benign, and slightly more for the benign moles (18%).

9 CONCLUSION

We have presented a Convolutionnal Neural Network that used transfer learning to realize binary classification. The model achieved an accuracy of 84.54%, which is a good value given the small time needed to train the model.
However, for an application in the medical field, where we would use for example our model to make earlier predictions, or to avoid invasive procedures, these accuracies are still not good enough; indeed, out of each 100 individuals, around 15 would obtain a false diagnosis. If the case of a false positive isn't dangerous since it can be verified with an biopsy, a false negative can't be accepted.

ACKNOWLEDGMENTS

I would like to thank Professor Olivier Debeir and Adrien Foucart for their continuous help and advices throughout the whole duration of this project.
The code implemented was first inspired by the following :
https://www.kaggle.com/fanconic/cnn-for-skin-cancer-detection/notebook

REFERENCES

[1] [n.d.]. *Binary crossentropy.* https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy
[2] [n.d.]. *Convolutional neural network.* https://en.wikipedia.org/wiki/Convolutional_neural_network
[3] [n.d.]. *ISIC Archive.* https://www.isic-archive.com/#!/topWithHeader/tightContentTop/about/isicArchive
[4] Nikola Basta. 2020. *The Differences between Sigmoid and Softmax Activation Functions.* https://medium.com/arteos-ai/the-differences-between-sigmoid-and-softmax-activation-function-12adee8cf322#:~:text=Softmax%20is%20used%20for%20multi,similar%20to%20the%20Sigmoid%20function.&text=This%20is%20main%20reason%20why%20the%20Softmax%20is%20cool.
[5] Titus Josef Brinker, Achim Hekler, Jochen Sven Utikal, Niels Grabe, Dirk Schadendorf, Joachim Klode, Carola Berking, Theresa Steeb, Alexander H Enk, and Christof von Kalle. 2018. *Skin Cancer Classification Using Convolutional Neural Networks: Systematic Review.* https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6231861/
[6] Sanket Doshi. 2019. *Various Optimization Algorithms For Training Neural Network.* https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
[7] fchollet. 2020. *Transfer learning fine-tuning.* https://keras.io/guides/transfer_learning/
[8] GeeksforGeeks. [n.d.]. *VGG-16 | CNN model.* https://www.researchgate.net/figure/Fig-A1-The-standard-VGG-16-network-architecture-as-proposed-in-32-Note-that-only_fig3_322512435
[9] Google. [n.d.]. *Classification : accuracy.* https://developers.google.com/machine-learning/crash-course/classification/accuracy
[10] Manu Goyal, Thomas Knackstedt, Shaofeng Yan, and Saeed Hassanpour. 2020. *Artificial intelligence-based image classification methods for diagnosis of skin cancer: Challenges and opportunities.* https://www.sciencedirect.com/science/article/pii/S0010482520303966
[11] Aurélien Géron. 2017. *Hands-On Machine Learning with Scikit-Learn and TensorFlow* (2. ed.). O'Reilly Media.
[12] Kaggle. [n.d.]. *Skin Cancer : Malignant vs Benign.* https://www.sciencedirect.com/science/article/pii/S0010482520303966
[13] Neurohive. 2018. *VGG16 – Convolutional Network for Classification and Detection.* https://neurohive.io/en/popular-networks/vgg16/
[14] S.Shajun Nisha and M.Nagoor Meeral. 2020. *Handbook of Deep Learning in Biomedical Engineering* (1. ed.). Academic Press.
[15] Github : Keras Team. [n.d.]. *vgg16.py.* https://github.com/keras-team/keras-applications/blob/master/keras_applications/vgg16.py
[16] Rohit Thakur. 2019. *Step by step VGG16 implementation in Keras for beginners.* https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c