Machine Learning and Content Analytics

Melanoma Classification

Ioanna Ioannou p2822008
Andriani Karpathaki p2822020
Nikolaos Panagiotopoulos p2822015

Business Analytics Master | Part Time 2020 – 2021

Content Table

Abstract 2

Introduction 2

Data Overview 3

Data Preprocessing 5

Dataset Division (Train – Validation – Test) 5

Overfitting 6

Data augmentation 6

Building the models 10

Convolutional Neural Network (CNN) 10

CNN model (without data augmentation) 14

CNN model (with data augmentation) 19

Pretrained model VGG16 22

Conclusions 26

Timeplan 28

Abstract

The goal of this project was to demonstrate the use of supervised machine learning techniques by developing predictive models for cancer diagnosis, particularly the classification of different types of skin cancer.

Skin cancer is one of the most common type of cancer. Although the majority of skin cancers can be treated adequately, this may not be the case if they are not discovered at an early stage. Having said that, knowing what to look for is of high importance in terms of recovery and treatment.

The learning methods developed by Machine Learning procedures offer tremendous potential to enhance medical research and clinical care, especially as providers increasingly employ electronic health records. The main idea is that systems can learn from data, identify patterns and make decisions with minimal human intervention and that is what we attempt to do in this report.

The data used was collected from Kaggle 1 and has been originated from the Department of Dermatology, Hospital Clinic de Barcelona and the Department of Dermatology, Medical University of Vienna. The dataset consists of 25.331 images belonging to 8 different diagnostic categories. Each image had a unique identification number and there was a csv file indicating the type of skin cancer for each image.

Introduction

The necessary steps followed to proceed with a Machine Learning analysis was first to import and prepare the dataset using python programming language. Data preprocessing was critical in order to ensure or enhance the performance of our models. During this process, the images were converted to RGB images, and were also normalized, preferred especially for neural network models.

The risk of over-fitting has been avoided by splitting the dataset into three segments: a training, a validation and a testing segment. The first one was used for training (fitting)

[1] https://www.kaggle.com/andrewmvd/isic-2019/code

the models, the second one was used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters and the last one was finally used for comparing the performance of the finalized models. Confusion matrix and classification reports were also used as evaluation measures to calculate the ratio of correct predictions over the total predictions made.

In this report we have built three models, one convolutional neural network (also known as CNN) based on the given images and one CNN model after conducting Data Augmentation techniques. For the third and last model, we used a pretrained VGG16 model, one of the most used image-recognition architectures.

Data Overview

As said before the data set consists of 25.331 images belonging to 8 different diagnostic categories. The types of skin cancer that are of our interest as well as their abbreviation names are represented in the following table (Table 1).

A/A	*Diagnostic Category*	*Abbreviation*
1	Melanoma	MEL
2	Melanocytic nevus	NV
3	Basal cell carcinoma	BCC
4	Actinic keratosis	AK
5	Benign keratosis	BKL
6	Dermatofibroma	DF
7	Vascular lesion	VASC
8	Squamous cell carcinoma	SCC

Table 1 - Data Overview - Diagnostic Categories

It is worth mentioning that there was a 9th category referring to images not belonging to any of the above-mentioned subclasses or for which information was missing. However, there were no images having this description, so we did not have to worry about excluding them from the dataset. A slight insight of how each type of cancer looks like, is presented below (Figure 1).

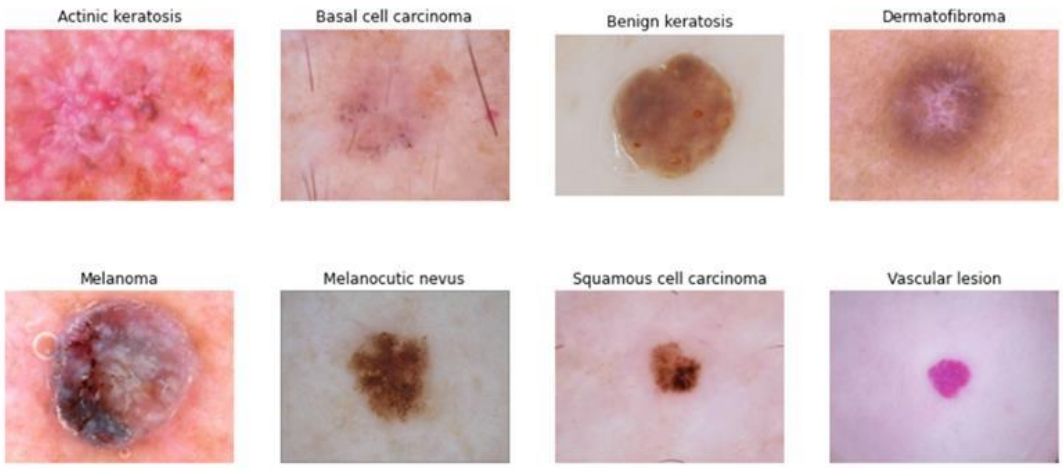


Figure 1 - Image example for each diagnostic category

The following bar plot shows the percentage distribution of disease type in our dataset (Figure 2). It is obvious that most units in the sample (almost 90%) come from four types of skin cancer, while one type of class represents more than half of the total population.

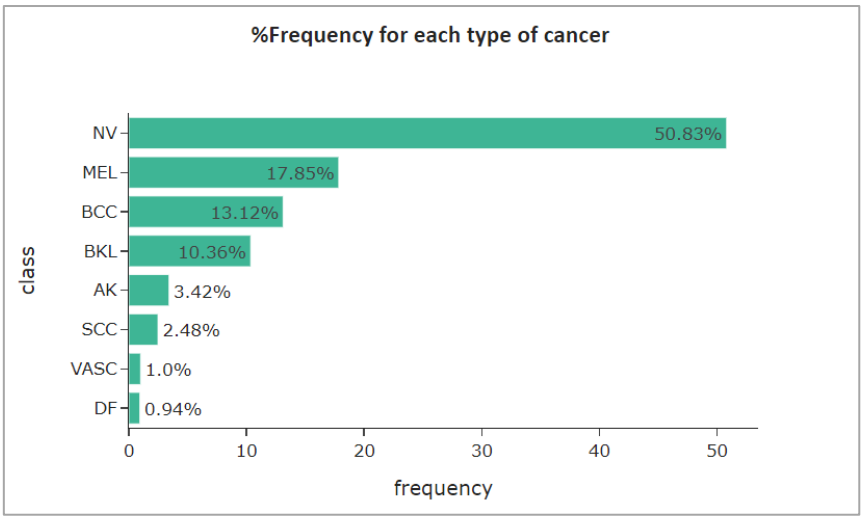


Figure 2 - % Frequency for each type of skin cancer

Data Preprocessing

First thing to do was to import into jupyter notebook all necessary libraries for developing machine learning and handling the data. Afterwards, we imported the csv file, indicating the type of subclass that each image belongs to. By checking the structure of the dataframe created, we noticed that the columns representing the categorical variable are in a dummy variable type of format. Thus, we reconstructed the categorical variable for cancer classes and added it as a new column into the existing dataframe.

Dataset Division (Train – Validation – Test)

To be noted that all images were automatically stored in a single file after downloading them from Kaggle. As already said, essential step was to divide the dataset into 3 segments a train, a validation, and a test set. For this purpose, we created new directories into the existing directory, one for each segment. Either for train, validation and test folder the images would be located into a subfolder, in accordance with their diagnostic category for reasons stated in the following section while using the ImageDataGenerator class.

It is worth describing the procedure for splitting the dataset. At first, the division referred to the train and the test set at 85% and 15% respectively. Afterwards, the train set was used as an input and then split into a train and a validation set (68% and 17% of the original dataset). The entire process can be depicted as per below (Figure 3).

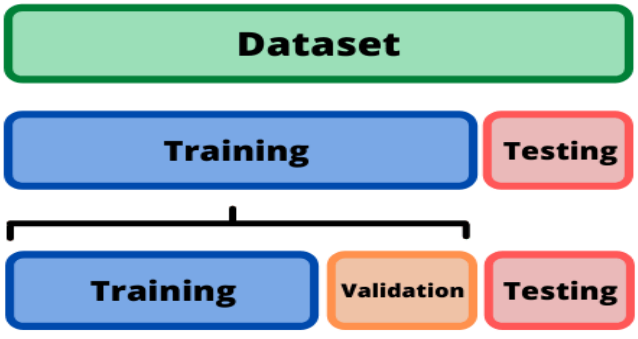


Figure 3 - Data division technique

The method used for splitting the images in parts was built in such a way to do it proportionately for subclasses. That is to say, by specifying the stratification attribute, the method returned training and test subsets that have the same proportions of class labels as the input dataset. Similarly, this happened for making the train and validation dataset. This process essentially contributes to overcoming the issue of overfit.

Overfitting

Overfit practically occurs when the model or the algorithm fits the data too well, but it negatively impacts the performance of the model on new data. In simple words, overfitting is a result of the model picking up and memorizing patterns of the training dataset which represent indeed that specific sample but may not apply to the entire population.

This may happen for a number of reasons, for example when the model is characterized by high variance or when the size of the training dataset used is not enough. The first cause was avoided by splitting the dataset in an appropriate way while the second one was handled with data augmentation techniques.

Data augmentation

A substantial way to expand the size of our dataset with the sole purpose to improve the performance and ability of the model to generalize, is the implementation of data augmentation technique. Briefly, and for ease of reference only, Image data augmentation is supported in the Keras library via the ImageDataGenerator class.

To use the ImageDataGenerator class the data on our machine must be in a specific directory structure while it accepts as input a path to the directory containing images sorted in sub directories. This is the reason why we constructed the folders in such a manner as mentioned previously. The ImageDataGenerator class takes also as input the image augmentation parameters.

In addition, the ImageDataGenerator class can be used to rescale pixel values from the range of 0-255 to the range 0-1 (also known as normalization) which is a good practice especially for neural network models.

Although, data augmentation includes a wide range of techniques used to generate "new" training samples, at the same time it ensures that the class labels of the data are not changed. For example, if an image deriving from the training dataset produces several other images, the last will maintain the class of the initial image.

As such, our model will train new, slightly modified versions of the original input data so it will be able to capture more evidence and learn strong features. For the validation and the test dataset, we proceeded only with rescaling of the data as other transformations are not required because we are not training the model on this data.

Data augmentation techniques used in this project are described as per below:

1. **Random Shifts**

A common issue is that the object we are looking for may happen to not be in the center of the training image. In this occasion, we are interested in the skin bumps, which are possibly located anywhere in the picture. To overcome this problem, we shifted the pixels of the image horizontally and vertically by adding a certain constant value to all the pixels, specifying this way the percentage of width and height of the image to shift. A relevant example is presented in the following figure (Figure 4)

Figure 4 - Random Shift example (width and height = 0,2)

2. Random Rotations

Image rotation, a widely used augmentation technique, is a geometric transform which allows the model to become unconquerable to the orientation of the object. More particularly, this method changes the angle that objects appear in the dataset during training. This is important because during the image collection process, some images may have been collected with an object being located horizontally, but in real life, the object could be skewed in any direction. The images can be rotated through any degree between 0 and 180 by providing an integer value in the rotation_range argument (Figure 5).

Figure 5 - Random Rotation (40)

3. Horizontal and Vertical flips

Flipping images vertically or horizontally (or both) is also a great augmentation technique for reasons as stated in the Random Rotations technique. An example of this application is presented below (Figure 6-7).

Figure 6 - Horizontal flip

Figure 7 - Vertical flip

4. Random Zoom

The zoom augmentation method either zooms in or zooms out on the image. By specifying a negative number in zoom_range attribute then it will zoom in on the image, whereas a positive number will zoom out on the image. In our case we were interested in zooming in (Figure 8). The range for random zoom is defined as [lower, upper] = [1-zoom_range, 1+zoom_range].

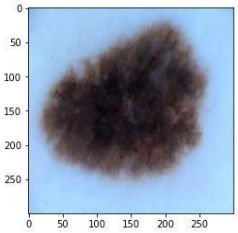
Figure 8 - Random Zoom (0,2)

5. Shearing

A map of a coordinate space in which one coordinate is held fixed and the other coordinates are shifted is called shearing transformation. Basically, it slants the shape of an object. The differences may not be noticeable as in the following example (Figure 9), because it depends on the value input (float number).

Figure 9 – Shearing (0,2)

Having implemented the above simultaneously, we can see practically what this application does in the following figures (Figure 10).



Original Image

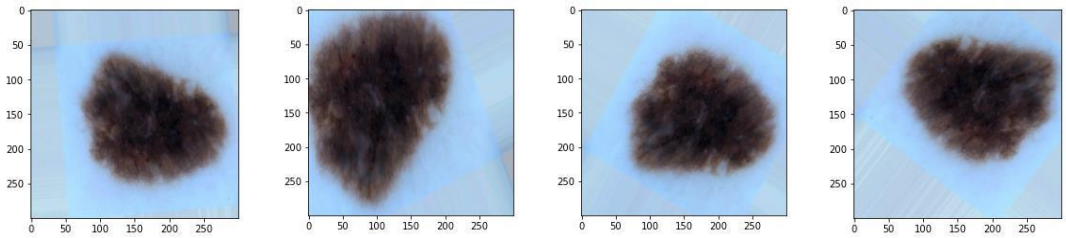


Figure 10 - Data Augmentation Results

The idea is to compare the performance of the model both, with and without augmentation to get an idea of how helpful augmentation is.

Building the models

In this report we have built three models, one convolutional neural network (CNN) and one CNN model after conducting data augmentation techniques. For the third and last model, we used a pretrained VGG16 model, one of the most used image-recognition architectures.

Convolutional Neural Network (CNN)

At the current state, the best way for solving image recognition problems is by using Neural networks and deep learning techniques. Neural networks, also known as artificial neural networks (ANNs) have a structure remarkably resembling the human brain and the way biological neurons signal to one another.

By looking at an overview of a deep neural network (Figure 11) we see that it is comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node of a layer connects to another node of the next layer (these connections are called edges), with the last having an associated weight and threshold.

If the output of any individual node is above the threshold value, then the node is activated and sends data to the next layer of the network. Otherwise, no data is passed along. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

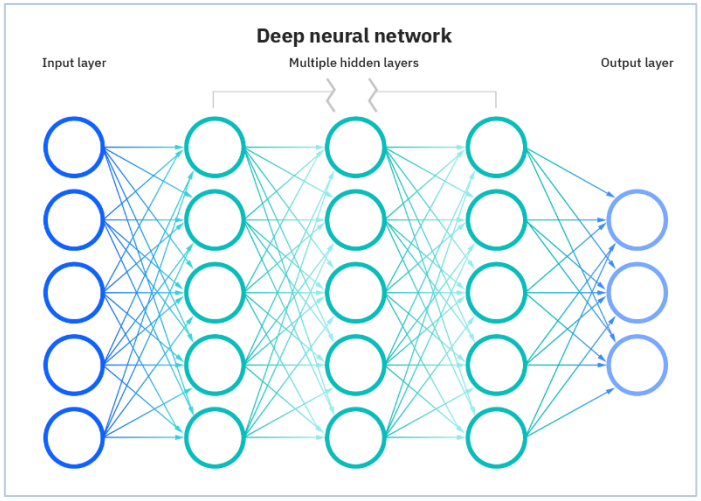


Figure 11 - Deep Neural Network Overview
Source: IBM, https://www.ibm.com/cloud/learn/neural-networks

One of the main parts of neural networks is Convolutional neural networks (CNN). CNNs primarily use image recognition in order to detect objects, in our case of study, skin lesion objects, and then classifies images based on the results. An overview of how CNNs work is stated below:

1. **Input images transformed into tensors is fed to CNN**:
   The tensor can be a vector or matrix depending on whether the image is in RGB format (colored images) or not. A colored image is formed as a (width) * (height) *3 dimension matrix.

2. **Convolution layers will extract features:**
   At this point images are convolved using filters or kernels. Filters are small units that we apply to the data via a sliding window producing a Feature map . This process involves taking the element product of filters in the image and then summing those specific values for every sliding action. At the same time and during every sliding action, the network passes the product through an

activation function such as ReLu or sigmoid and responds back with an output. The output deriving from any convolution layer and activation function passes through a Pooling layer which is responsible for reducing the spatial size of the convolved feature.

3. **Classification**

   Lastly, is the Fully Connected Layer, which involves Flattening. This process transforms the entire pooled feature map matrix into a single column which is then fed to the neural network for processing. With the fully connected layers, the features extracted previously are combined together and used to create the model. Finally, with an activation function such as softmax or sigmoid the outputs are classified. Basically, this activation function brings as output a probabilistic distribution of each class. The class that has the biggest probability value is the prediction of the model.

4. **Error Back propagation**

   This method helps calculate the gradient of a loss function with respect to all the weights in the network. In simple words, it will provide feedback to the network comparing predictions VS ground truth.

5. **Update weights**

   Network will update the weights of kernels based on the feedback of the back propagation.

6. **Steps 1 to 5 will repeat untill the network is trained (epochs).**

   The number of epochs, which is also specified, represents the number of times the model will repeat this process (steps 1-5).

In this project we have followed the above-described procedure which can also be viewed in the following figure (Figure 12).

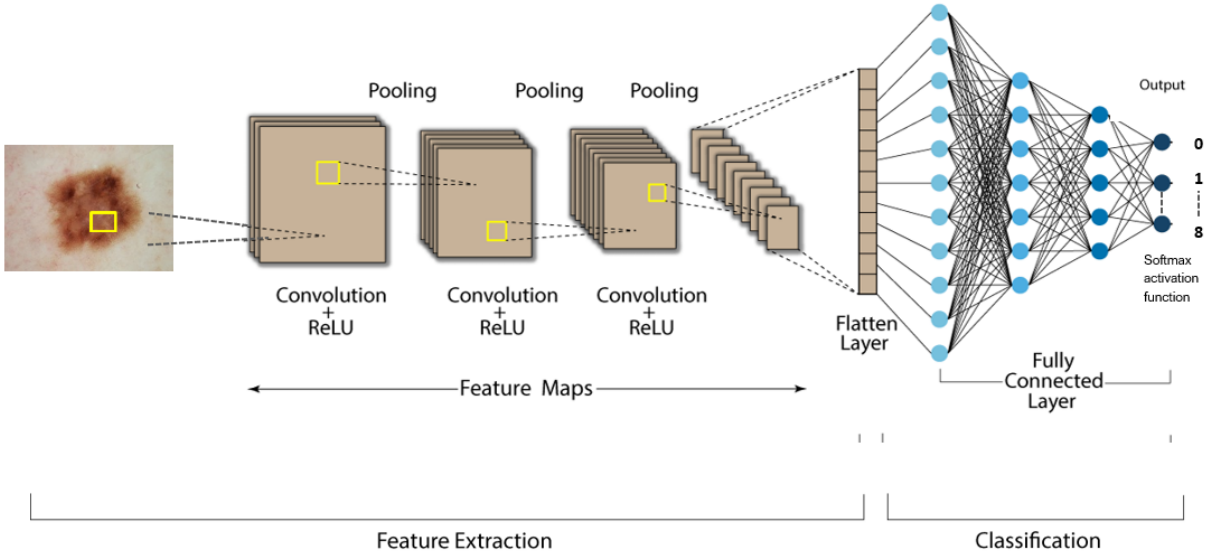


Figure 12 - CNN structure

When it comes to the activation function in the Feature Extraction Process, we have used **ReLU** function as it is the most recommended for CNN. Practically, Relu converts negative numbers to zeros and carry invariant the positive numbers. These means that it does not activate all the neurons at the same time as the neurons are only activated if the output of the linear transformation is more than 0. More precisely, the main advantage of using the ReLU function over any other activation, except of its simplicity, is that when a feature is not helping the network, then Relu does not carry it forward (makes it zero).

In terms of Pooling, we chose **Max Pooling**, which returns the maximum value from the portion of the image covered by the Convolutional Layer, over Average pooling. This is because, Max Pooling discards the noisy activations and also performs dimensionality reduction, while Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism.

Finally, the last activation function used in the neural network was the **Softmax Activation Function**. This function transformed the output of the network into probabilities. These probabilities are the model predictions for each of the 8 classes.

To summarize, convolution gives location to invariant feature detection and reduces overfitting via connections sparsity, the ReLu function forwards the important information and speeds up the training process, while max pooling comes to reduce the

dimensions and computation of the final product. After all that, we continue with a classic neural network model for the classification step which is accomplished via Softmax Activation Function.

CNN model (without data augmentation)

Having said the above we proceeded with building the CNN model. At this point we did not take into consideration the data augmentation techniques performed previously, so we used only the original dataset to train our model.

A summary of the process flow of our model comes as follows (Table 2):

1. Adding a two-dimensional convolutional layer with 32 neurons with kernel size of 3X3 and with ReLu activation.
2. Adding a two-dimensional convolutional layer with 64 neurons with kernel size of 3X3, with ReLu activation
3. Adding a Max Pooling layer with 2*2 filters and stride 2 to cut the size of the convolutional layers
4. Adding dropout to regularize the model and avoid the issue of overfitting
5. Flatten the MaxPooling Layer
6. Adding another Dense layer of 128 neurons with Relu activation
7. Adding Softmax Activation to make the probabilities

This procedure has been specified to repeat 50 times (no. of epochs)

Layer	Output Shape	Parameters
conv2d (Conv2D)	(None, 148, 148, 32)	896
conv2d_1 (Conv2D)	(None, 146, 146, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 73, 73, 64)	0
dropout (Dropout)	(None, 73, 73, 64)	0
flatten (Flatten)	(None, 341056)	0
dense (Dense)	(None, 128)	43655296
dropout_1 (Dropout)	(None, 128)	
dense_1 (Dense)	(None, 8)	1032
Total params: 43,675,720 **Trainable params: 43,675,720** **Non-trainable params: 0**		

Table 2 - Summary of CNN model

Interpretation of the CNN model

Assuming the size of each image is (150, 150, 3), keras appends an extra dimension for processing multiple batches. Since batch size can vary, its size is represented by None. Hence, the input shape becomes (None, 150, 150, 3).

Convolving a (150, 150) image with a (3, 3) kernel size filter, with strides and dilation rate of 1, and 'valid' padding, results in an output of size (150 - 3 + 1, 150 - 3 + 1) = (148, 148). Since we have 32 filters, the output shape becomes (148, 148, 32). Respectively by convolving a (148, 148) image with a (3, 3) kernel size filter, of 64 neurons, the output shape becomes (146, 146, 64).

The default MaxPooling kernel has a shape of (2, 2) and strides of (2, 2). Applying that to a (146, 146, 64) image results in an image of shape (((146 - 2)//2) + 1, ((146 - 2)//2) + 1)) = (73, 73), thus (73,73,64).

The Flatten layer takes all pixels along all channels and creates a one-dimensional vector (batch size is not considered). Therefore, an input of (73, 73, 64) is flattened to (73* 73 * 64) = 341.056 values.

Compiling the model

Compiling the model is an essential step that takes place after building the model . It takes three parameters: optimizer, loss and metrics.

Optimizer: As already stated, a neural network includes weights that are at first initialized randomly. Then they are updated in each epoch so as that they increase the overall accuracy of the neural network. After the completion of each epoch, the output of the training data is compared to true data (with the use of loss function), the error that took place is calculated and then the weight is updated accordingly.

The goal is to determine weights in a way that optimizes the accuracy of the model. This is exactly what optimizer does by controlling the learning rate. The learning rate determines how much it takes to calculate the optimal weights for the model. In this project we used ‘adam’ as an optimizer as it is widely used and needs little memory.

Loss Function: As part of the optimization process, loss function is necessary for estimating the loss of the model so that the weights can be updated to reduce the loss

on the next evaluation. Loss function measures the distance that is between the current output of the algorithm and the expected output. We used 'categorical_crossentropy' for our loss function which is the most common choice for classification. The lower the score the better the model performs and that is what the optimizer tempts to do.

Accuracy: Accuracy is a statistical measure which shows the proportion of correct predictions of a model, thus the closer the accuracy is to value one the better the model is. As such, we used the 'accuracy' metric to see the accuracy score on the validation set when we train the model.

Fitting the model

After fitting the model, we can see in the following diagram that the loss decreases rapidly for training, indicating that the network is learning fast. However, in terms of validation the loss curve flattens and then increases (Figure 13). This is commonly a result of overfitting. Although, we expect a model to perform better on the training set, since the model parameters are being shaped by the training set, the goal is to see both training and validation loss to decrease because in this way we know that our model is continuing to learn generalized things about our data.

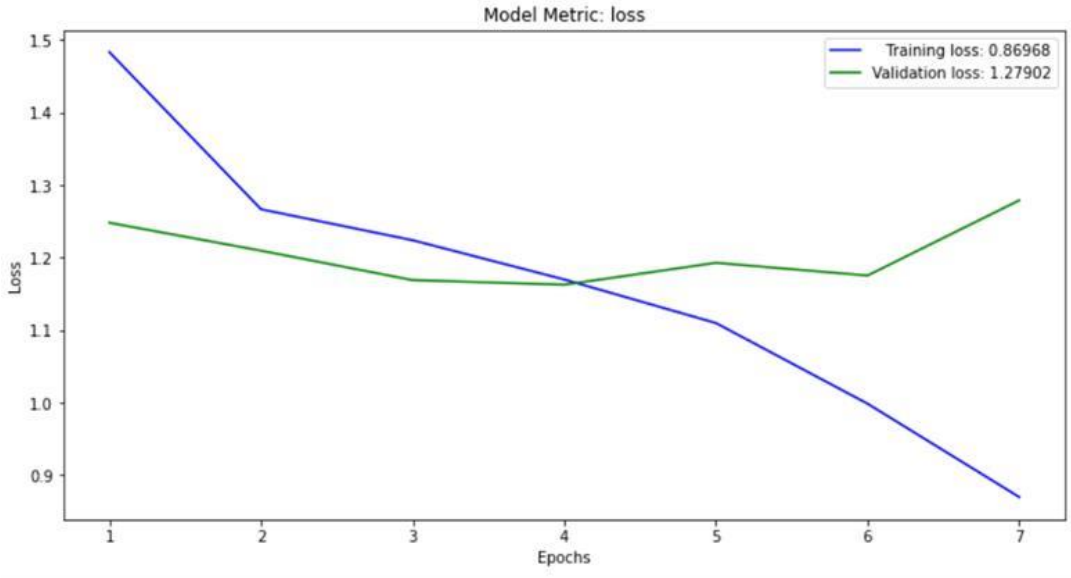


Figure 13 - Model metric: Loss Function

From the plot of accuracy, we observe that the model could probably be trained a little more as the trend for accuracy at least for the training dataset is still rising for the last few epochs (Figure 14). Unfortunately, this is not the case for the validation dataset. Generally, when the training data accuracy keeps improving while the validation data accuracy is getting worse, we are encountering overfitting, as already discovered by checking the evolution of the loss function.

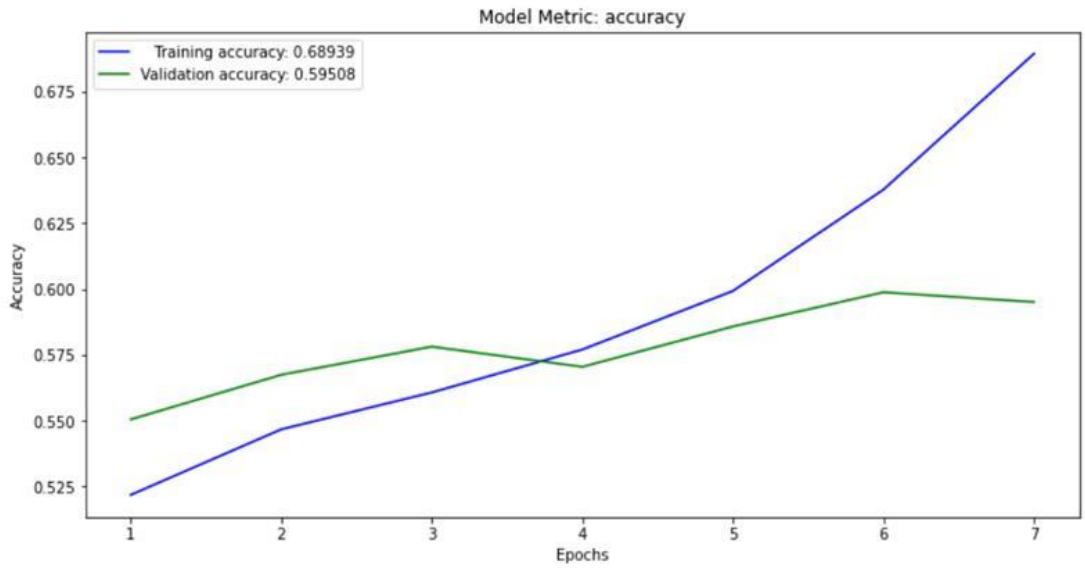


Figure 14 - Model Metric: Accuracy

Prediction

Next and final step is to figure the performance of the model in terms of prediction. At this point we use the test dataset for the first time. The classification results can be viewed by creating a classification report. The report shows the main classification metrics such as precision, recall and f1-score on a per-class basis. The metrics are calculated by using true and false positives (TP ans FP), true and false negatives (TN and FN).

- TN / True Negative: a case was negative and predicted negative
- TP / True Positive: a case was positive and predicted positive
- FN / False Negative: a case was positive but predicted negative
- FP / False Positive: a case was negative but predicted positive

Preferable results are those when TN and TP are high, and FN plus FP take small values. To be noted that those metrics takes values between 0 and 1.

Precision metric shoes the percentage of correct predictions versus the total predictions received for each class. It is defined as the ratio of true positives to the sum of true and false positives. (Precision = TP/(TP + FP)).

Recall metric shoes the percentage of correct predictions versus the times each class exists in the dataset. It is defined as the ratio of true positives to the sum of true and false positives. (Precision = TP/(TP + FN)).

The numbers show that on weighted average 53% of total predictions for each of the classes were actually of the predicted class, taking always into consideration the proportion of each class in the dataset (support). For classes that there was not enough information the results are obviously not good.

On the other hand, accuracy shows the percentage of true predictions in general. Specifically, in our model 58% of the total predictions were right.

Class	Precision	Recall	f1-score	support
0	0,00	0,00	0,00	130
1	0,44	0,40	0,42	499
2	0,50	0,01	0,02	394
3	0,00	0,00	0,00	36
4	0,62	0,17	0,27	678
5	0,59	0,96	0,73	1.931
6	0,00	0,00	0,00	94
7	0,62	0,34	0,44	38
Accuracy			0,58	3.800
macro avg	0,35	0,24	0,23	3.800
weighted avg	0,53	0,58	0,48	3.800

Table 3 - Classification Report

Another way to describe the performance of a classification model is by using a confusion matrix. In more detail, a confusion matrix is a symmetric table with dimensions (the number of classes) * (the number of classes), referring to all possible different combinations of predicted and actual values. The diagonal shows the number

of correct predictions, and the incorrect predictions are summarized with count values and broken down by each class.

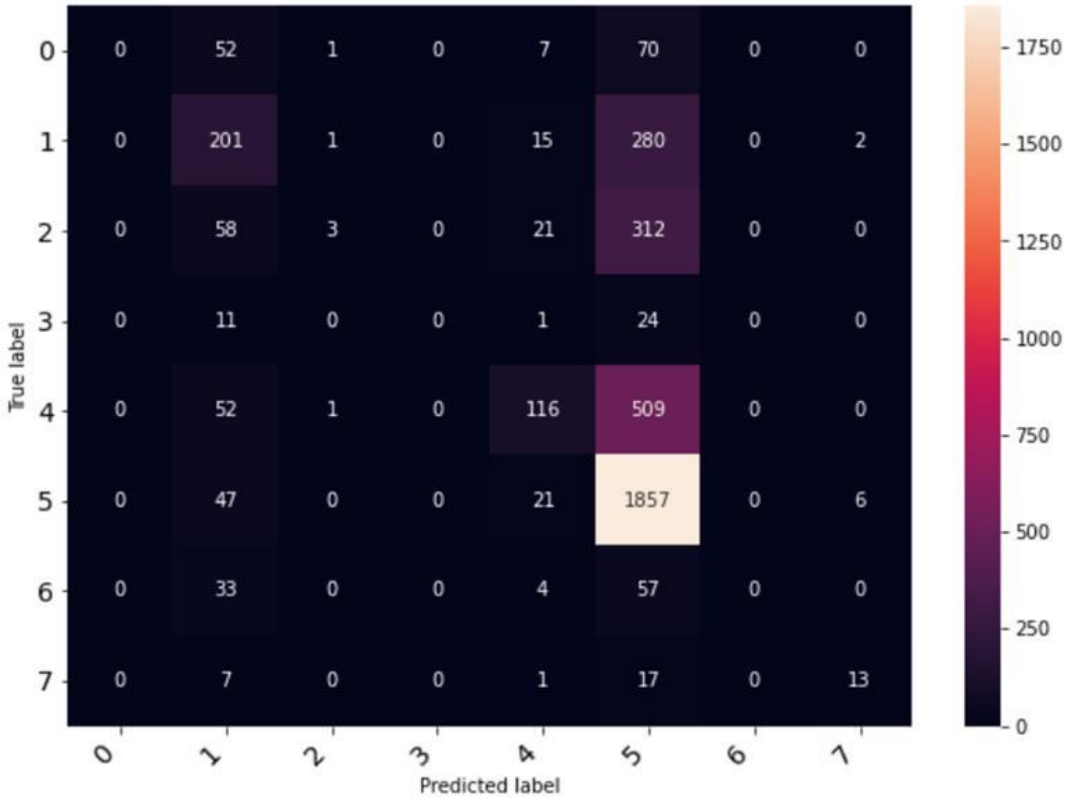


Figure 15 - Confusion Matrix

The results of our model based on the confusion matrix do not offer much more information in comparison to the classification report (Figure 15). However, it is clear that some specific categories are classified correctly (the ones prevailing in the dataset), while the rest have been mainly misclassified.

CNN model (with data augmentation)

Up until now we have built a CNN model based on the initial dataset. The model performance was not adequate, and this is probable due to overfitting. This is exactly the reason why we performed data augmentation techniques at an early stage. Let us now see to what extent this will fix this issue.

To be noted that the model was built similarly to the previous model, where the steps have been explained in a detailed way.

Fitting the model

In this case the loss function decreases rapidly in the first four epochs, and from that point and on the curve starts to flatten in terms of training. As expected, the results are significantly better to those derived from the previous model as the validation loss, even though decreasing slightly, it ends up approaching the training loss value (Figure 16).

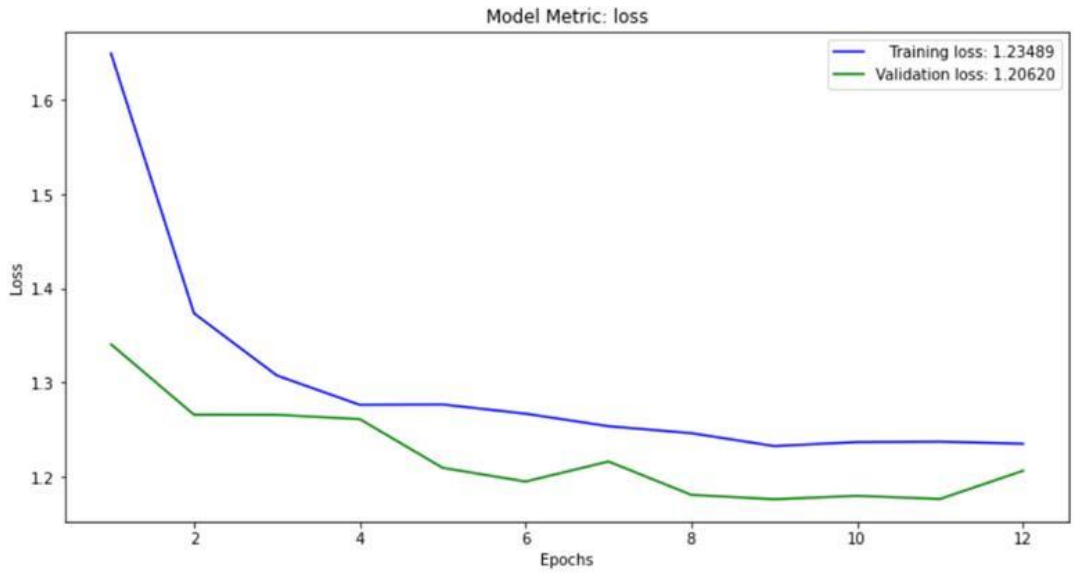


Figure 16 - Model metric: loss

In terms of accuracy, the results are also satisfactory as both metrics are increasing through epochs (Figure 17). However, we observe better accuracy values for validation than for training dataset. This may be attributed to the fact that there might be difficult examples in the training dataset, but only simple examples in the validation datasets. Thus, our model can be more accurate with the easy examples.

Although we may have encountered the problem of overfit, it is worth mentioning that the accuracy has not changed significantly as it remains approximately to 53%.

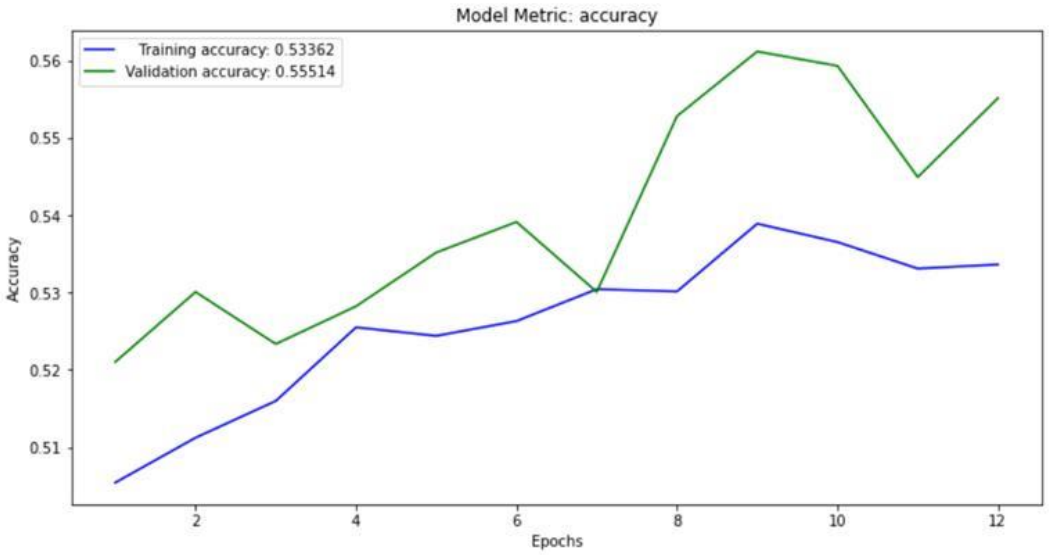


Figure 17 - Model metric: accuracy

Prediction

Even though we expect better results in comparison to the previous model, we can only be sure by checking its prediction performance. As such, we proceed again with the classification report.

Class	Precision	Recall	f1-score	Support
0	0,00	0,00	0,00	130
1	0,38	0,49	0,43	499
2	0,00	0,00	0,00	394
3	0,00	0,00	0,00	36
4	0,48	0,22	0,30	678
5	0,60	0,89	0,72	1.931
6	0,00	0,00	0,00	94
7	0,00	0,00	0,00	38
Accuracy			0,56	3.800
macro avg	0,18	0,20	0,18	3.800
weighted avg	0,44	0,56	0,48	3.800

Table 4 - Classification Report

Based on the weighted average, 44% of predictions for each of the classes were actually of the predicted class, taking again into consideration the proportion of each class in the dataset (support). For classes that we did not have many image samples, the results are even worse than those of the previous model (Table 4). The accuracy metric, though,

has remained at the same level. By looking at the confusion matrix (Figure 18), we come to the same conclusion.

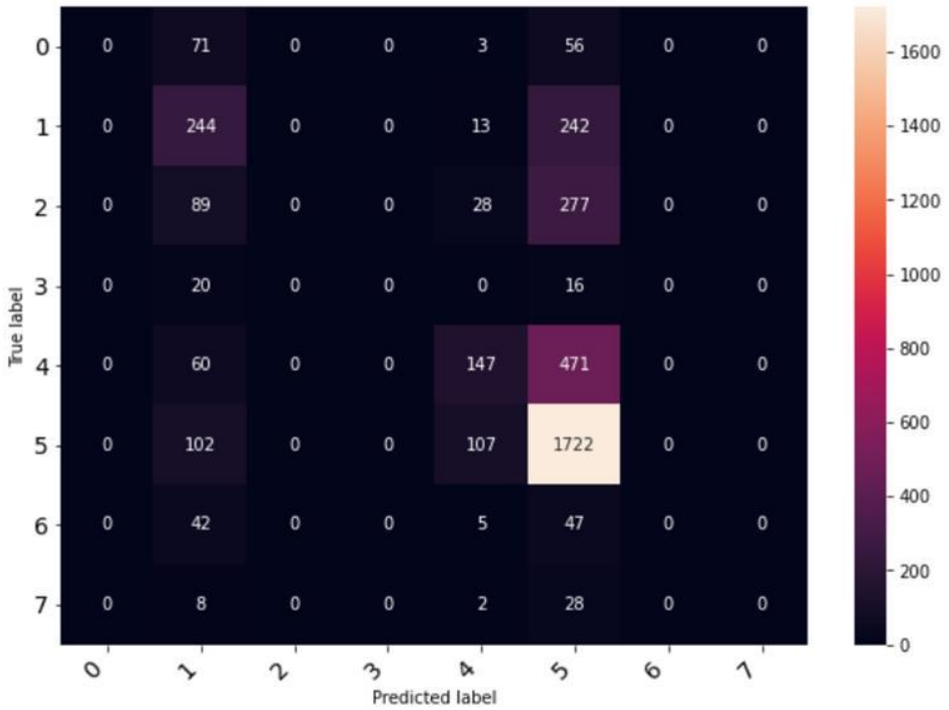


Figure 18 - Confusion Matrix

Pretrained model VGG16

So far, we have built two CNN models. One for the original images and one having implemented data augmentation techniques. However, neither of them brought sufficient results. Taking these into consideration, tried a different approach.

Transfer learning is a well-known and widely used technique by which we can use the model weights trained on standard datasets (such as ImageNet) in order to enhance the efficiency of a given task. This application has many advantages. Primarily, it gives us the chance to use a model which has been trained on millions of images, while this would take days or even weeks for a custom-made convolutional neural network. In addition, it does not need a vast training dataset as the model can already detect easily patterns and it also brings better accuracy results.

Having said the above, we decided to use a pretrained model, VGG16.

VGG generally addresses a very important aspect of CNNs, which is depth. More precisely, VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR (Imagenet) competition in 2014 and up until now it is considered as one of the best practices in terms of deep learning . The 16 in VGG16 refers to the16 layers that have weights. The architecture of the model is depicted below (Figure 19).

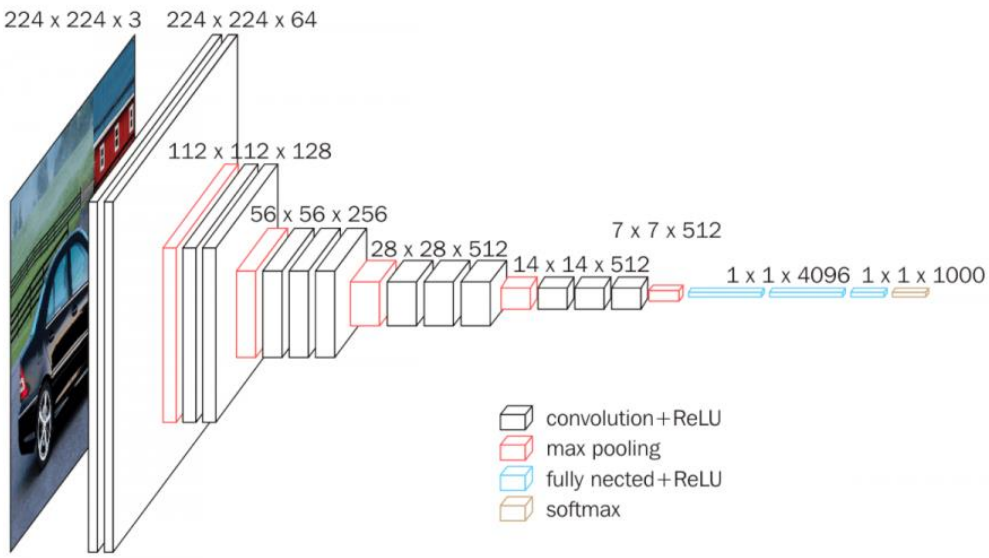


Figure 19 - Architecture of VGG16 model

Source: Neurohive, VGG16 – Convolutional Network for Classification and Detection, https://neurohive.io/en/popular-networks/vgg16/

A special feature about VGG16 is that it focusses on having invariant convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. This sequence of convolution and max pool layers is preserved throughout the whole architecture. In the end it has 2 Fully connected layers, followed by a Softmax Activation function for output.

Fitting the model

Again, we examine the loss and accuracy function for the training and test dataset. For the pretrained model the loss function for both validation and training dataset decreases significantly for the first 10 epochs (Figure 20). After that point the two curves start to stabilize and take values upside-down between a range of numbers. Either way, the

results are promising as loss is optimized for both datasets in comparison to the other models.

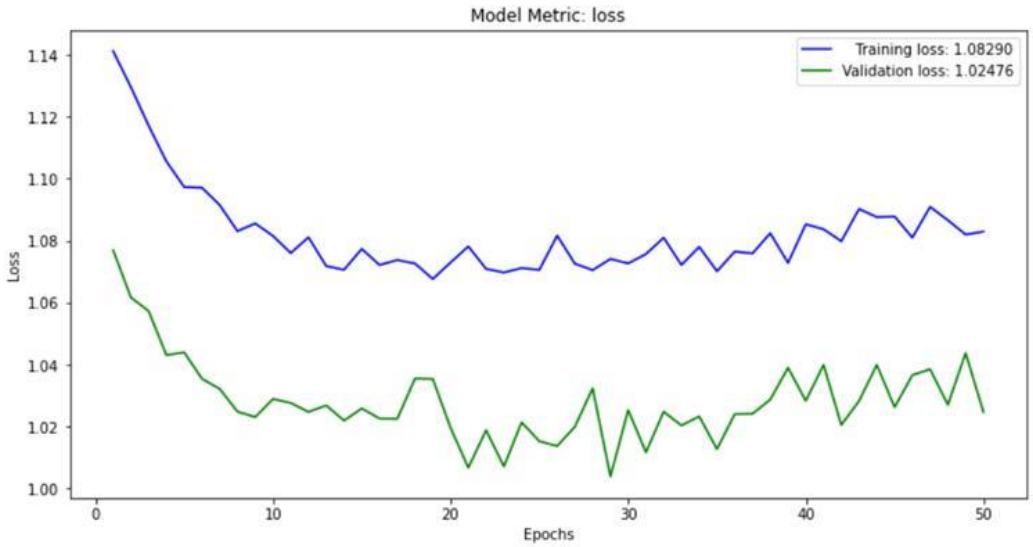


Figure 20 - Model metric: Loss

Regarding accuracy, the scores are more than great. For both validation and test dataset the accuracy performance exceeds 90%, meaning that the network was learning fast. , Moreover, the curve starts to flatten at the final steps indicating that not too many epochs are required to train the model further (Figure 21).

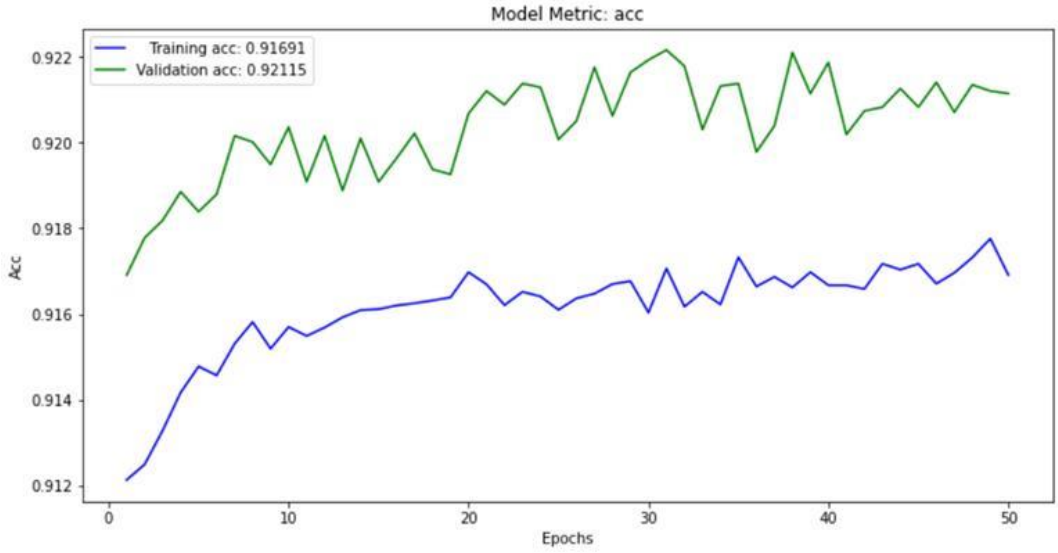


Figure 21 - Model metric: Accuracy

Prediction

Last step in order to verify that the pretrained model used has the best performance among all other models is to examine its prediction ability on the test data set. As always, we proceed with the classification report.

Class	Precision	Recall	f1-score	Support
0	0,39	0,13	0,20	130
1	0,51	0,70	0,59	499
2	0,38	0,27	0,32	394
3	0,00	0,00	0,00	36
4	0,59	0,43	0,49	678
5	0,75	0,88	0,81	1.931
6	0,40	0,02	0,04	94
7	0,70	0,18	0,29	38
Accuracy			0,65	3.800
macro avg	0,46	0,33	0,34	3.800
weighted avg	0,62	0,65	0,62	3.800

Table 5 - Classification Report

Based on the weighted average, 62% of predictions for each of the classes were actually of the predicted class (Table 5). When in all other models the precision metric for classes with short volume was equal to zero, in the pretrained model this is encountered once. The accuracy metric equal to 65%, as expected, is the best seen so far and even though is a good fit there could be a potential of improvement.

Lastly, the confusion matrix confirms that the model has achieved better predictions only by looking at the diagonal of the table as in the other models we depicted more zero cases (Figure 22).

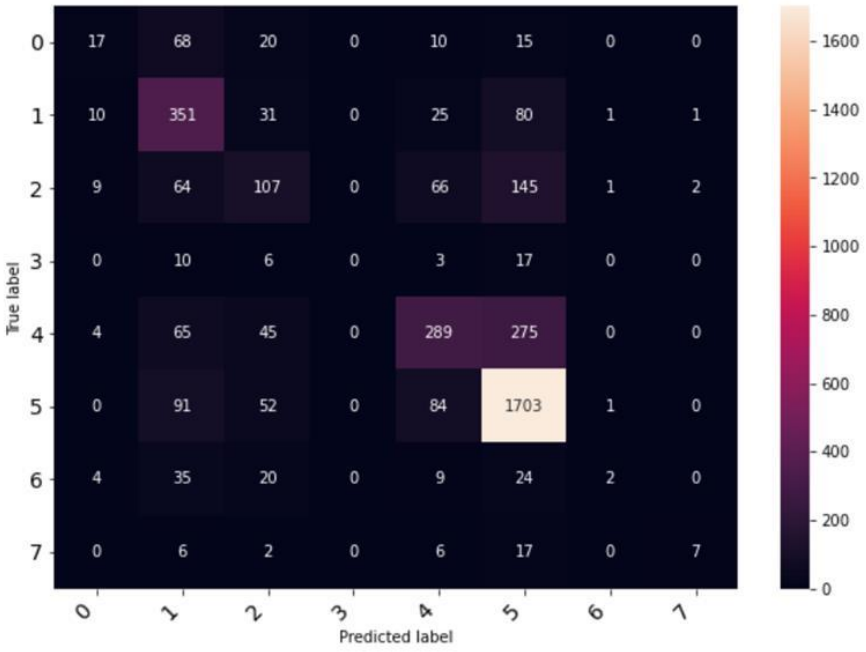


Figure 22 - Confusion Matrix

Conclusions

With digitalization taking over most industries through the past decades, the ability to capture and share data is becoming a high priority. Of course, the healthcare sector does not make the exception, as electronic health records are constantly updated. In light of this, our thought was to take advantage of this great volume of data and make a profit out of it.

Given that cancer is widely known as one of the most serious and fatal diseases, we decided to illustrate the importance of supervised machine learning techniques by developing predictive models for skin cancer diagnosis.

The critical factor in skin cancer treatment is early diagnosis and the conventional skin cancer detection includes painful and time-consuming process. This makes computer-based technology even more necessary for both doctors and patients, while computer systems can provide a comfortable, and speedy diagnosis of skin cancer symptoms, instead.

The learning methods developed by Machine Learning procedures offer tremendous potential to enhance medical research and clinical care and in this report we attempted to build three different models for skin lesion classification.

To summarize we constructed a CNN model from the initial dataset, a CNN model based on augmentation techniques and a pretrained VGG16 model. The first model was overfitted and we overcome this issue with the second model that brought more slightly different images to train. However, the performance of both models was not desirable. That is why we proceeded with a third model.

By taking into consideration classification metrics, such as accuracy, loss and precision measurement in order to evaluate the models, we concluded that the pretrained model had the best performance. However, and in any case the model could be better if the data were balanced and not mostly distributed to specific types of class.

Timeplan

Tasks	**21/06 - 24/06**	**28/06 - 18/07**	**19/07 - 01/08**	**02/08 - 15/08**	**16/08 - 29/08**	**30/08 - 05/09**
Bibliography Research	✔					
Dataset Selection	✔					
Data Processing		✔				
Build CNN model			✔			
Build CNN model with data augmentation				✔	✔	
Bulid VGG16 pretrained model				✔	✔	
Model Evaluation						✔
Report						✔

Citations

- [1] Tschandl P., Rosendahl C. & Kittler H. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5, 180161 doi.10.1038/sdata.2018.161 (2018)
- [2] Noel C. F. Codella, David Gutman, M. Emre Celebi, Brian Helba, Michael A. Marchetti, Stephen W. Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, Allan Halpern: "Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC)", 2017; arXiv:1710.05006.
- [3] Marc Combalia, Noel C. F. Codella, Veronica Rotemberg, Brian Helba, Veronica Vilaplana, Ofer Reiter, Allan C. Halpern, Susana Puig, Josep Malvehy: "BCN20000: Dermoscopic Lesions in the Wild", 2019; arXiv:1908.02288.
- [4] © 2021 American Cancer Society, Skin Cancer Image Gallery, <https://www.cancer.org/cancer/skin-cancer/skin-cancer-image-gallery.html?filter=Basal%20Cell%20Carcinoma,Kaposi%20Sarcoma,Melanoma,Merkel%20Cell%20Carcinoma,Skin%20Lymphoma,Squamous%20Cell%20Carcinoma>
- [5] Jenni A. M. Sidey-Gibbons & Chris J. Sidey-Gibbons, *Machine learning in medicine: a practical introduction*, < https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/s12874-019-0681-4>
- [6] Image Data Generator, *tf.keras.preprocessing.image.ImageDataGenerator* <https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator >
- [7] Analytics Vidhya, *Image Augmentation on the fly using Keras ImageDataGenerator!* <https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/ >
- [8] IBM, *Neural Networks*, < https://www.ibm.com/cloud/learn/neural-networks>

- [9] IBM, *Convolutional Neural Networks* < https://www.ibm.com/cloud/learn/convolutional-neural-networks>
- [10] Analytics Vidhya, *Fundamentals of Deep Learning – Activation Functions and When to Use Them?* < *https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/* >
- [11] Towards data science, *Building a Convolutional Neural Network (CNN) in Keras I* <https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5>
- [12] YellowBrick, *Classification Report*, < https://www.scikit-yb.org/en/latest/api/classifier/classification_report.html>
- [13] Towards data science, *VGG Neural Networks* < https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c >
- [15] Xiangyu Zhang, Jianhua Zou, Kaiming He, Jian Sun, *Accelerating Very Deep Convolutional Networks for Classification and Detection*, < https://arxiv.org/pdf/1505.06798.pdf >