T.C.

MANİSA CELAL BAYAR UNIVERSITY

ENGINEERING FACULTY

COMPUTER ENGINEERING DEPARTMENT

SKIN LESION CLASSIFICATION with MACHINE LEARNING

Graduation Project II

PREPARED BY

170316007 AYŞENUR BÜYÜKBAL

SUPERVISOR

ASSOC. DR. BORA CANBULA

MANİSA 2022

T.C.

MANİSA CELAL BAYAR ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Tasarım Projesi / Lisans Bitirme Tezi

KABUL VE ONAY BELGESİ

Ayşenur BÜYÜKBAL'ın "Skin Lesion Classification with Machine Learning" isimli lisans projesi çalışması, aşağıda oluşturulan jüri tarafından değerlendirilmiş ve kabul edilmiştir.

Danışman : Doç. Dr. Bora Canbula

Üye :.....

Üye :.....

Projenin Savunulduğu Tarih : .....

Bilgisayar Mühendisliği Bölüm Başkanı

TABLE OF CONTENTS

ABBREVIATIONS

CNN – Convolutional Neural Networks

R-CNN – Region Based Convolutional Neural Networks

GPU – Graphics Processing Unit

MNIST - Modified National Institute of Standards and Technology

NORB – New York University Object Recognition Benchmark

CIFAR10 - Canadian Institute for Advanced Research

ABSTRACT

Skin diseases are one of the most common diseases in society. It is also very common among cancer types. As in many diseases, early diagnosis plays a major role in saving the patient's life. Experts in skin diseases examine the image of the diseased area in their diagnosis. It has been a subject that has been studied for a long time to make this examination faster and more sensitive by machines and to use it as a decision support mechanism in health. For this purpose, the use of CNN model, which can be considered as the latest technology in image processing technologies, has become quite common in recent years. In this project, it is aimed to train the CNN model and classify skin diseases with this model using a dermoscopic image dataset which is HAM10000. It is also aimed to create a user interface to present the model to the users.

Keywords: Machine learning, CNN, classification, pigmented skin lesions, user interface

1. INTRODUCTION

Dermatology is the branch of medicine dealing with the skin [1]. Skin disorders are frequently encountered in the society and these disorders affect the quality of life considerably. The prevalence of skin diseases is more than the total prevalence of obesity, hypertension, and cancer [2]. As in many other diseases, the diagnosis phase is vital in skin diseases, so technological devices are used to make the most accurate diagnosis. In the diagnosis of skin diseases, the appearance of the affected area is the first resource for dermatologists. For this reason, various devices such as ultrasound, dermoscopy are used to better visualize the affected area. Dermoscopy is a non-invasive imaging technique that enables the visualization of submacroscopical structures invisible to the naked eye and mainly used to evaluate pigmented skin lesions [3]. This technique has been used since 1950s, and it helps to get 5–30% greater accuracy in diagnosis phase compared to naked eye evaluation. But results also depend on dermatologists' experience [4].

It has been more than 60 years since artificial intelligence was first used to generate solutions for a scientific reasoning problem. For the first time in the 1960s, the "Dendral" project is considered an expert-level system. In this project, artificial intelligence studies were carried out in organic chemistry and later formed the basis for many projects [5]. One of the early artificial intelligence projects based on the Dendral project is the "MYCIN" project, which is a clinical decision support system in healthcare. The MYCIN project was developed in 1972 at Stanford University by Edward Shortliffe et al. But the project did not achieve a routine use [6]. The 1980s and 1990s brought the enhancement of microcomputer and network connection power, the use of artificial intelligence was considered and therefore applications in health got often. At that time, artificial intelligence systems had the most difficulties due to the lack of data. With the widespread use of electronic record systems in health, data has increased, and this has had a positive effect on artificial intelligence studies. In these studies conducted in the same period, subjects such as artificial intelligence approach to the analysis of clinical data [7] and the use of artificial neural networks in the diagnosis of various diseases [8] [9] [10] [11] were discussed.

The beginning of the concept of artificial neural networks is based on calculations called "threshold logic" by Warren McCulloch and Walter Pitts in 1943 [12]. Neurons in the nervous system work on the "all or none" principle. That is, neurons show maximum response for signals above a certain threshold value, while ignoring signals below this value. This situation is similar to propositions in logic. Propositions in basic logic are evaluated in two different ways, true and false, just like in the nervous system. Later, the concept of learning was emphasized by Hebb in the late of that decade [13] . The concept of learning has been developed using various models. In 1958, Rosenblatt, also known as the "father of deep learning" [14], created the perceptron algorithm [15]. It was a classification algorithm using the mathematical notation of circuits. For a while, the work of artificial neural networks slowed down until the processing power of computers advanced. After the backpropagation algorithm was created in 1975 by Werbos [16], practical work of artificial neural networks and multi-layer networks gained momentum.

CNN (Convolutional Neural Networks) are neural network models mostly used for deep learning in images. Although their first use dates back to the 1980s, the fastest growing period after GPU improvements has been in the past 20 years. Different layers are used in CNN models. The CNN model has been used in medicine for the

purpose of disease diagnosis with visual processing since artificial intelligence's early days. An example of this is the study aimed at diagnosing breast cancer using mammogram images. [17]

In this project, a CNN model has been created for the processing of images of different types of skin diseases. I attempted to use the Dermnet dataset, but for reasons I will explain later, I replaced the dataset with HAM10000 Skin Lesions. There is a user interface in order to use the created model. The aim of the project is to create a CNN model to diagnose and classify the skin disease by scanning the affected area in the image given as input.

2. PROBLEM DESCRIPTION

Speeding up and facilitating our daily work with machines has been in our lives for a long time. Studies in the development of machines are mostly aimed at making human life easier. A long time has passed since the first use of machines in medicine. For more than 60 years, studies have been carried out more specifically on the use of artificial intelligence in medicine. These studies were carried out with the aim of helping experts in decision making. The use of machine learning models in the diagnosis of various diseases has been considered and applied even in the early times of artificial intelligence. Why did people think of getting help from machines in such a vital issue as medicine? Because decision support systems present patient-specific recommendations in a way that can save clinicians time, and they have been shown to be highly effective and sustainable tools for changing clinician behaviour [18]. According to a study, it has been seen that decision support systems improve the performance of the practitioner by 64% [19]. With the increasing population and the demand for health services, it is an important issue to be able to make accurate diagnoses in a short time. Such models both save time and give an idea to the experts. From another perspective, these models can be used as educational tools for practitioners who have not yet graduated. Upon completion of the necessary procedures, this model can serve as a pre-examination tool to shorten inspection times.

The aim of this study is to create a system that will provide decision support function to clinical experts with state-of-the-art models.

3. LITERATURE ANALYSIS

CNN, (Convolutional Neural Networks) developments in the field, which can be considered as the latest technology, started in 2011 [20], and studies on the classification of skin lesions with CNN were seen in 2014. Early and current related works on this subject are as follows:

Ciresan et al. [20] introduced a fast GPU implementation for the CNN model. This study stands out as the best object recognition model among the published studies of the period. The model achieved very low error rates in object recognition (with NORB and CIFAR10 datasets) and handwritten character recognition (with MNIST dataset). A simple back propagation algorithm was used in the training of the model. Error rates are 2.53%, 19.51%, 0.35%, respectively.

Bernart et al. [21] explained in detail the techniques used in the classification of pigmented skin lesions and the stages of these techniques and talked about the place of CNNs among these techniques. In the study, it is seen that the questions about how the images in the data set are used for training and testing, how the pre-processing of the raw images are made, how the feature extraction is done from the processed image and how the paths are followed in the dermatology literature on this issue are answered. In this study published in 2019 (conducted in 2017), the current technologies of the period were examined and the latest studies in this field were conveyed.

Jayalakshmi and Kumar [22] worked on a model that could make a binary classification to detect skin lesions as benign or malignant. The best results were obtained by using 70% of the data set in training this CNN model. The average accuracy percentages obtained with 80% and 75% data usage in training rates are 73.95% and 78.33%, respectively. In the study, model optimization with batch normalization technique is proposed. The layer architecture in the proposed model consists of 6 layers of convolutional blocks with batch normalization followed by a fully connected layer that performs binary classification. According to the experimental results, it can be said that this model is 89.30% more successful than the custom CNN.

Hameed et al. [23] instead of diagnosing and classifying skin diseases, worked on the detection and classification of different types of skin cancer. Stacked CNN and MNIST datasets are used in the model. Working with a 30% data split, the stacked CNN model in this research, together with data augmentation and image pre-processing techniques, achieves 95.2% accuracy.

Wan and Zhang [24] studied the diagnosis of skin diseases with CNN from another perspective. They suggested using Faster R-CNN (Faster Region Based CNN) architecture since CNN networks cannot work well enough in multiple object recognition situations. In order to identify multiple objects in an input, the R-NN architecture first proposes possible regions where those objects can be found and inserts each of these regions' boxes into a separate CNN model. Faster R-CNN puts these regions in a single CNN model. Faster suggested that the R-CNN model would be more effective than traditional CNN because skin problems can always be seen spread over more than one area rather than a single area.

4. METHODS AND TECHNOLOGIES

In this project, a CNN model in ResNet50 architecture was created using TensorFlow, a Python machine learning library, and its API Keras. This model was trained and tested using the HAM10000 Skim Lesion dataset. Details of the technologies used in the project are listed in the following headings in chronological order.

4.1 TensorFlow and Keras

TensorFlow is a platform for building, training, and importing models for multidimensional array-based computations, GPU and distributed data processing, machine learning. As it can be understood, TensorFlow works with multidimensional arrays, namely tensors. These tensors are defined with Tensor class objects belonging to the TensorFlow library. For example, a two-dimensional tensor:

```
In [1]: import tensorflow as tf

In [2]: x = tf.constant([[1., 2., 3.],
                         [4., 5., 6.]])

        print(x)
        print(x.shape)
        print(x.dtype)

tf.Tensor(
[[1. 2. 3.]
 [4. 5. 6.]], shape=(2, 3), dtype=float32)
(2, 3)
<dtype: 'float32'>
```

Figure 4.1.1 TensorFlow Example

TensorFlow, a popular open-source deep learning library from Google, uses Keras as a high-level API for its library. It is often referred to as tf.keras. Keras' core data structures are layers and models. The simplest type of model is the Sequential model, which is a linear stack of layers. For more complex architectures, you should use the Keras functional API, which allows creating arbitrary layer graphs or writing models completely from scratch via subclasses.

4.2 General View of CNN Architecture

The layer architecture in the CNN model to be built will be finalized during the implementation of the project. In general terms, the layers of the model to be applied and the functions of these layers are as follows:

Convolution Layer: In this layer, it is aimed to remove some features (edge, corner, width, etc.) from the image by applying a filter to the image. The multiplication of another matrix (filter) of a certain size on the matrix representing the image is called convolution. After this operation, the size of the input matrix decreases. The size of the output matrix can be calculated using the size of the input matrix and the size of the filter to be applied.

$$\text{output size} = \text{input size} - \text{filter size} + 1$$

The step size of this filter while navigating the matrix is called "stride". A shift of 1 unit at each stage of the filter means that the size of the stride is 1. The size of the stride will directly affect the size of the output matrix.

If it is desired to obtain an output matrix of a certain size, then the padding operation can be used. Padding means filling the missing places with values so that the matrix reaches the desired size. There are two methods for this:

- empty spaces are filled with 0
- empty spaces are filled with values of nearby pixels

Activation Layer: The activation layer is one of the most important layers, which will save the neural network from being a simple system that linearly transforms the value it obtains. In this layer, which enables more efficient processing of values, an activation function is selected (mostly ReLU) and the obtained values are converted using this function.

Pooling Layer: The purpose of this layer is to reduce some dimensions of the input. In the pooling operation, the same as in the convolution operation, the matrix is navigated. It is used to obtain a smaller output from the input data with minimum loss by taking the average (average pooling) or the maximum value (max pooling) of the pixels in the examined region.

Flattening Layer: In order for artificial neural networks to process these matrices, they must be entered as one-dimensional vectors. The matrix obtained as a result of the operations carried out up to this stage is flattened into a one-dimensional vector in this layer.

Fully Connected Layer: It is the layer where the learning process is started by giving the obtained flattened vector to the neural network.

Apart from these layers, techniques such as batch normalization are used to increase the dilution and optimization of the model to prevent memorization of the model.

4.3 Kaggle and Dataset Handling

Kaggle is a community where people working in the field of data science and machine learning come together. Founded in 2010, this community was originally a platform for contests. In these competitions, which are still ongoing today, companies or individuals share their problems and solutions in the field of data science and machine learning on this platform, and users who want to solve them enter a competition for the optimum solution. Contest owners test the solutions offered to them and share their evaluations with users. The ratings made by the users in such competitions and the evaluations they received are shown in their profiles. In addition to these competitions, Kaggle has become a more comprehensive community in recent years. Users can share datasets, source codes (called notebooks) and other users can practice with them in data science and machine learning. Thanks to the comment sections, it offers an interactive experience that is open to questions and discussion.

Before creating the CNN model, I found a dataset provided by Dermnet, on Kaggle, suitable for the machine learning project I wanted to do. This data set consists of approximately 21.000 images of 23 different skin diseases. With this dataset, I tried 3 different CNN architectures with different optimization methods, epoch, and batch sizes. But I haven't had success with any model. The accuracy rates I got from the models varied erratically between 3% and 30%. The CNN architectures I tried using the Dermnet dataset were VGG16, InceptionV3, and ResNet50. Since I was not successful with any of these architectures, I started to examine my dataset. In the images in the Dermnet dataset, the area affected by the lesion was not in focus. The images were taken from various parts of the body, taken from different distances, and were not obtained in a uniform shot. Processing and learning such a dataset is quite a challenge for the model. For this reason, I realized that I needed to change my dataset and found the HAM10000 dataset.

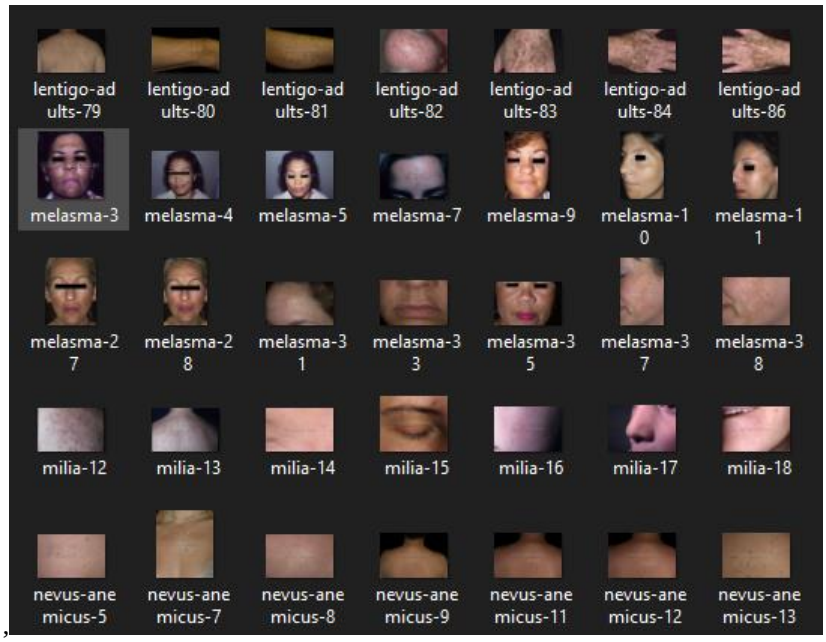

Figure 4.3.1 Dermnet Dataset file view

The HAM10000 (Human Against Machine) dataset, published in the Harvard Dataverse in 2018, is a dataset containing 10000 skin lesion images obtained from multi-sources. The collected data is also publicly published in the ISIC archive for academic purposes. Cases in the dataset are a collection of diseases that have a precautionary place in pigmented skin lesions: Actinic keratoses and intraepithelial carcinoma / Bowen's disease (akiec), basal cell carcinoma (bcc), benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses) , bkl), dermatofibroma (df), melanoma (mel), melanocytic nevi (nv) and vascular lesions (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage, vasc) [25].

Figure 4.3.2 HAMM10000 Dataset file view

The same dataset is available from the Harvard Dataverse site or on Kaggle. Firstly, I used the JupyterLab environment that comes with Anaconda for the implementation of my first CNN model. I downloaded the HAM10000 dataset as a 2 piece .zip archive file. This archive contains 10015 images and various .csv files, including the metadata file with .csv extension for these images. All of these images, which belong to 7 different classes, are systematically named, but they are not separated according to their labels in 2 parts. Therefore, I first need to scan these two folders with the help of metadata and divide my dataset into subfolders according to the tags.

4.4 Triple Services: Google Collaboratory, Drive and Kaggle

I wrote and ran the source codes which belongs my very first CNN models using Dermnet data in JupyterLab, and this was a very time-consuming process. Moreover, Dermnet data was already separated into "train" and "test", with each folder containing 23 different subfolders for 23 different diseases, meaning it was ready for direct reading, but not the HAM10000 dataset. When I decided to use HAM10000 dataset, firstly I divided the whole

dataset into subfolders by comparing them with the tags in the metadata on the cloud. I did this by using Google services such as Kaggle, Drive, Colaboratory together.

Google Colaboratory gives you a certain amount of ram and disk on an edge computer, plus the most important GPU support. Running processes on the GPU during the training of the model saves time. Google Drive offers a data storage service that works in the cloud. In order to read and work with your data in Drive via Google Colaboratory, it is necessary to run the code below.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Since the dataset is in the local files of the computer, the Drive and Colab connection is not enough for me. The dataset must be in Drive. Since the dataset consists of images, it is large in size and will take time to upload to Drive, no matter how compressed it is. Therefore, it is necessary to download the dataset directly to Drive. Here Kaggle and Colab link make things easy. Firstly, I downloaded an API Token which belongs my Kaggle account. The following codes install a package for Kaggle. I uploaded my API Token (kaggle.json file) with a function. This token is necessary for account matching with Colab. Then it makes the configurations and becomes ready for use.

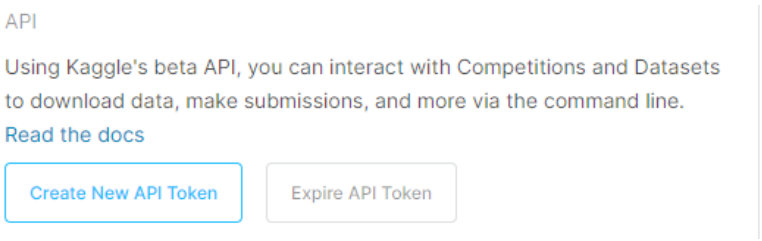

Figure 4.4.1 Kaggle API Token creation

```
!pip install --quiet kaggle
```

```
from google.colab import files
files.upload()
```

```
Dosyaları Seç  Dosya seçilmedi        Upload widget is only available when the cell h
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"ayenurbykbal","key":"6f8d2b896f2d4a10e37
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
# Check the file in its new directory
!ls /root/.kaggle/
# Check the file permission
!ls -l ~/.kaggle/kaggle.json
#Change the file permission
# chmod 600 file - owner can read and write
# chmod 700 file - owner can read, write and execute
!chmod 600 ~/.kaggle/kaggle.json
```

```
kaggle.json
-rw-r--r-- 1 root root 68 Jun  2 08:40 /root/.kaggle/kaggle.json
```

Figure 4.4.2 Getting ready to use Colab with Kaggle

After it is ready to use, it is possible to download a data set from Kaggle to the Drive directory we are in with one line. For this process, we open the page of the mentioned data set in Kaggle and receive the download command for the data. When we run the copied row in Colab, the dataset descends to the current directory. Since the dataset will download as a zip archive, unzip is required after downloading. This is done with the codes below.

```
!kaggle datasets download -d kmader/skin-cancer-mnist-ham10000
```

```
from zipfile import ZipFile
file_name= "skin-cancer-mnist-ham10000.zip"
with ZipFile (file_name ,'r') as zip:
  zip.extractall()
  print('done')
```

```
done
```

Figure 4.4.3 Dataset download and unzip on Drive from Kaggle

After the data set was acquired and the connections were established, I started the process of dividing the pictures into subfolders according to their tags. I assigned the metadata file of the images to the variable named "dataframe" with the "read_csv()" function of the Pandas library. The 'dx' column in the dataframe indicates the label of the images, that is, which of the 7 diseases they belong to.

Splitting into subfolders is done with these steps:

- Moving 5000+5015 images in Images Part1 and Images Part2 folders to a single folder (all_img).
- Converting the values in the 'dx' column of the dataframe to the "label" list without repeating them. We will get a list of 7 labels for 7 diseases. Thus, we get the names I will give to the subfolders.
- Creating the paths of the source directory (all_img) where all the images are located and the target directory (reorganized_img) that will be reorganized and placed
- Creating a for loop within the list of tags. Taking each tag and creating a folder with that tag name, capturing the images of that tag in metadata. Keeping the captured images in a temporary list and throwing them into the created folder
- Repeating this process for each tag

After this stage, the dataset is ready for use. The next step is to create the CNN model and select the appropriate architecture.

```
dataframe = pd.read_csv('HAM10000/HAM10000_metadata.csv')
print(dataframe['dx'].value_counts())
source_dir = 'HAM10000/HAM10000_images_part_1' #run again this line with part2
target_dir = 'all_img'

file_names = os.listdir(source_dir)
counter = 0
for file_name in file_names:
    shutil.move(os.path.join(source_dir, file_name), target_dir)
    counter+=1
print("done", counter)

label = dataframe['dx'].unique().tolist()

label_images = []
data_dir = os.getcwd()+ "/all_img/"
dest_dir = os.getcwd()+ "/reorganized_img/"

for i in label:
    os.mkdir(dest_dir + str(i) + "/")
    sample = dataframe[dataframe['dx'] == i]['image_id']
    label_images.extend(sample)
    print("copying starts", i)
    for id in label_images:

        shutil.copyfile((data_dir + "/"+ id +".jpg"), (dest_dir + i + "/"+id+".jpg"))
    print("copying ends", i)
    label_images=[]
```

Figure 4.4.4 Reorganizing Images

4.5 Keras Applications, Models, Layers, Optimizers

CNN models consist of several layers that perform different operations. The way these layers are ordered is important to the performance of the model. Layers process the inputs they receive and give them to another layer. An excess of these layers results in "deep" networks. Keras allows us to use these layer types with ease. In other words, you can create a CNN model by importing the layers you want and arranging them one after the other with a completely arbitrary selection. However, this model will not provide as high performance as you intend. What we refer to as a "layer" is actually a set of neurons. The coefficient of influence of the connections of this neuron cluster on the inputs is called the "weight". In artificial neural network models, the aim is to learn the most appropriate weights using the training data, and then predict the output with the highest accuracy for the inputs to be given.

Keras Applications are deep learning models that come with pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning. Currently, various versions of Inception, VGG, Resnet, Mobilenet, Densenet, NASnet, and EfficientNet are available in Keras Applications. In this project, VGG16, Inceptionv3 architectures were tried, but the results were not successful due to the irregularity in the previous dataset. Resnet50 architecture is used with the new dataset. The purpose of the applications is to train them with

large data sets containing images of objects consisting of many classes. By using these pre-trained models as a basis, adding, and removing layers can be done. The output layer is the layer that affects the prediction of the model. That is, the inputs go through the whole network and in the output layer, it gives as many outputs as there are classes. By taking a Resnet50 architecture, only the output layer can be customized to give outputs specific to our own dataset. This is a "fine-tuning" process, one of the methods called "transfer learning". In this project, "fine-tuning" was done.

Keras models are a structure that is created as a base before the CNN architecture is built. In this project, sequential model is used. Model can be created in Keras in 3 different ways:

- The Sequential Model is simply a single-input, single-output structure in the form of a list of layers.
- Functional API, on the other hand, is suitable for creating customized architectures, easy to use, and useful for many problems.
- Model subclassing is a structure where you can create everything from scratch and can be used in experimental studies for unusual problems or research.

Keras also includes layers. You can import these layers individually and add them to your model. Although Keras offers a wide variety of layers, it also lets you create your own layers. In this project, "Flatten" and "Dense" layers are used to add to the existing Resnet50 model. The flatten layer reshapes the previous layer, so it is in the reshaping layer category. Dense layer connects the incoming nodes according to the number of outputs.

The optimization process generally aims to find the lowest value of a continuous function (optimization function). In machine learning, the function that gives the error amount of the individual prediction is the "loss function". The function that gives the overall error rate of the model is called the "cost function" and this function is subjected to the optimization process, so that the most ideal model is tried to be created with minimum error. There are many

different optimization algorithms available in Keras. The most commonly used ones are Adam, SGD (Stochastic Gradient Descent) and RMSprop. The Adam method, which performs well in image classification, was used in this project [26].

4.6 CNN Architecture Selection

ResNet architectures were developed by the Microsoft Research team and are available in several versions with different number of layers, such as 50, 101, 152 [27]. Pre-trained ResNet architectures have been used for classification of skin lesions even by a few ISIC participants [28] [29]. So I decided to use Resnet50 architecture for this classification problem. Resnet50 architecture has 50 layers and looks like the following Figure 4.6.1 [30]

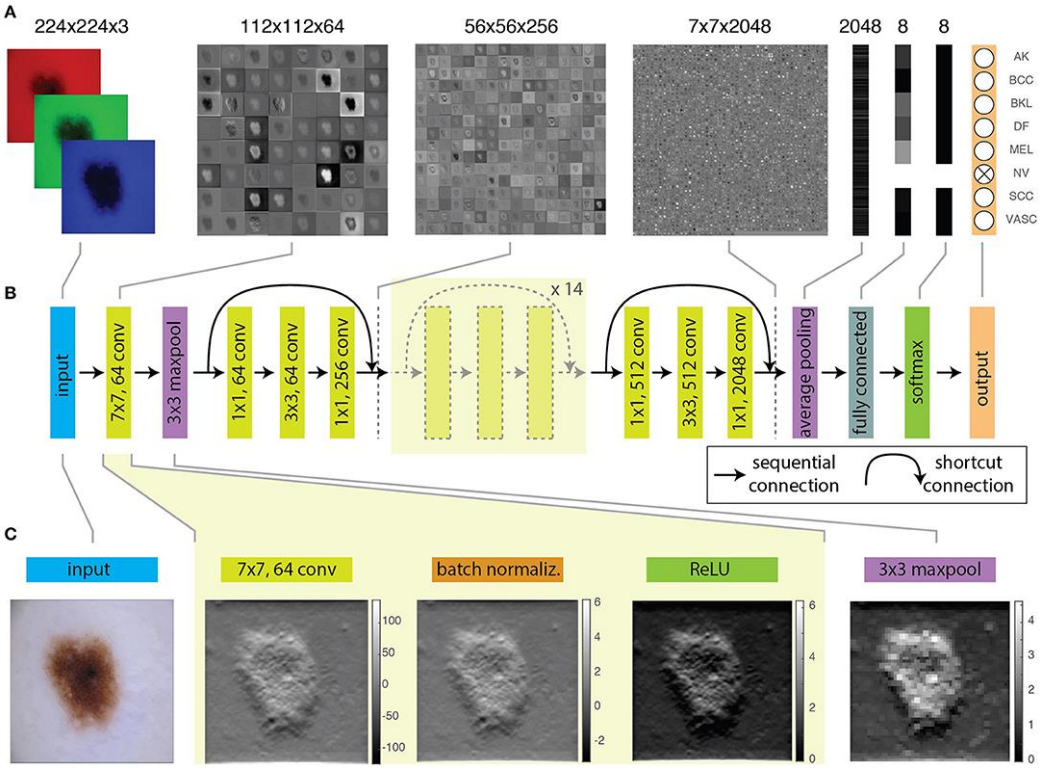

Figure 4.6.1 Resnet50 arcitecture

(A) Feature maps produced by the model at different layers throughout the network, from the RGB image to the 8-output probability vector

(B) Shapes of layer entries and order of layers

(C) Effects of layers on the image

4.7 Train & Test Data Split

Most of the data to be used in a neural network model is in the training set. The remaining data is included in the test set. While creating the neural network model, there is no exact value for parameters such as some functions, optimizers, data split ratio or batch size, epoch size. In this case, the values that give the best results are found by making trials according to the data set, the problem or the number of classes. In addition, there are commonly used values. For example, in the train-test split, 70-80% of the data is mostly used for training. The remainder is used in the testing phase. Test data should be data that the model did not see during the training phase. Which data from the data set will be used in the test phase and which data will be used in the training phase is usually randomly determined. In this project, the train-test split was used as 75% to 25%. Batch is usually given to you as powers of 2. Specifically, it varies between 64-512. It has to fit in GPU memory. Batch value represents the number of data to be processed in iteration. It is not desirable for each training set element to enter one by one or for all elements to enter. Batch size can be selected from the largest possible number of RAM supported among [1, number of training set elements]. I have chosen 32 batch size which means my training dataset will be split into batches each has 32 images, and 470 iterations will be done. Epoch means how many times does the model see your dataset entirely. It would be ideal if the epochs continue until the learning rate is constant or nearly constant with very minor changes. Depending on the batch size, usually 50-100 epochs are made. But I observed that the minor changes shown in early phase. So, I decided 8 epochs. Batch size and epoch size also vary depending on the amount of data you have. If your dataset is small and you have chosen large batch, epoch size, you may run out of data during training.

4.8 Data Handling Method #2

Another way to make the dataset available to the model is to use metadata. In order to do this, I first encoded the labels of the diseases as [0,1,...,6]. It was necessary for some comparisons while working. Then I browsed the images one by one, using the filenames in the image_id column and created a column specifying the path of each image. A new column was named "image", where the images in this path were kept in the form of an array. While dividing the dataset, I set up a structure like x -> image , y-> label. I used these columns while training and testing the model. After preliminary training trials of the model, I used the Spyder IDE to better analyse the variable types. The codes after this stage are written in Spyder.

```
In [12]: dataframe = pd.read_csv('HAM10000_metadata.csv')

In [13]: SIZE=224

In [14]: le = LabelEncoder()
    ...: le.fit(dataframe['dx'])
    ...: LabelEncoder()
    ...: print(list(le.classes_))
    ...: dataframe['label'] = le.transform(dataframe["dx"])
['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']
```

Figure 4.8.1 CSV Reading Label Encoding

```
In [15]:
    ...: print(dataframe['label'].value_counts())
    ...: image_path = {os.path.splitext(os.path.basename(x))[0]: x
    ...:                       for x in glob(os.path.join('data/', '*', '*.jpg'))}
5    6705
4    1113
2    1099
1     514
0     327
6     142
3     115
Name: label, dtype: int64

In [16]: dataframe['path'] = dataframe['image_id'].map(image_path.get)

In [17]: dataframe['image'] = dataframe['path'].map(lambda x:
np.asarray(Image.open(x).resize((SIZE,SIZE))))
```

Figure 4.8.2 Adding new columns

```
In [18]: X = np.asarray(dataframe['image'].tolist())

In [19]: Y=dataframe['label'] #Assign label values to Y

In [20]: Y_cat = to_categorical(Y, num_classes=7)

In [21]: x_train, x_test, y_train, y_test = train_test_split(X, Y_cat,
test_size=0.25, random_state=42)
```

Figure 4.8.3 Data split

4.9 Model Creation

I first created a Sequential Model to use the ResNet model available in Keras. I assigned the pre-trained Resnet Model to a variable. I cycled through the layers of this model in a loop, changing the trainability value of each layer to "False". This process is used when "fine-tunning". I froze the weights of the pre-trained model in order not to lose it while training with the new dataset. I added these frozen layers to the Sequential Model I created. Since the number of output classes in my dataset is 7, I created the Flatten and Dense layers according to my class count. I used the softmax activation function, which is often used in multiclass classification processes. While compiling the model, I chose Adam as the optimizer and the commonly used value 0.01 as the learning rate. Again, since it is a multiclass classification, I chose categorical_crossentrophy as the loss function. Since the batch and epoch sizes are also determined, the model is ready to be fitted.

```
resnet_model = Sequential()

pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                   input_shape=(SIZE,SIZE,3),
                   pooling='max',classes=7,
                   weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False

resnet_model.add(pretrained_model)

resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation='relu'))
resnet_model.add(Dense(7, activation='softmax'))
```

Figure 4.9.1 Creation

```
In [25]: resnet_model.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 dense_1 (Dense)             (None, 7)                 3591

=================================================================
Total params: 24,640,391
Trainable params: 1,052,679
Non-trainable params: 23,587,712
```

Figure 4.9.2 Model summary

4.10 Model Training and Test

Since I split the data, all I had to do during the training phase was to give the necessary variables to the fit() function.

```
In [26]:
resnet_model.compile(optimizer=Adam(learning_rate=0.01),loss='categorical_cro
ssentropy',metrics=['accuracy'])
    ...: batch_size = 16
    ...: epochs = 50
    ...: history = resnet_model.fit(
    ...:     x_train, y_train,
    ...:     epochs=epochs,
    ...:     batch_size = batch_size,
    ...:     validation_data=(x_test, y_test),
    ...:     verbose=1)
Epoch 1/50
 12/470 [..............................] - ETA: 27:30 - loss: 122.7110 -
accuracy: 0.4531
```

Figure 4.10.1 Model compile and fit

```
    ...:     batch_size = batch_size,
    ...:     validation_data=(x_test, y_test),
    ...:     verbose=1)
Epoch 1/50
470/470 [==============================] - 2227s 5s/step - loss: 4.8877 -
accuracy: 0.6617 - val_loss: 0.8995 - val_accuracy: 0.6745
Epoch 2/50
 10/470 [..............................] - ETA: 28:25 - loss: 0.8541 -
accuracy: 0.7250
```

Figure 4.10.2 Trainning

After the model training was completed, I measured the test score and saved the model.

```
score = resnet_model.evaluate(x_test, y_test)
print('Test accuracy:', score[1])
```

```
79/79 [==============================] - 543s 7s/step - loss: 1.0136 -
accuracy: 0.6661
Test accuracy: 0.6661341786384583
```

```
resnet_model.save('saved_model')
```

4.11 User Interface

I used the PyQt5 library for the UI. Since I made a simple interface, it did not have a very complex design. The purpose of the interface would be to load a file and display the model's prediction on the screen. PyQt5 offers a design tool called QtDesigner. I designed the desired design, buttons and texts in QtDesigner with drag-and-drop. I uploaded this .ui file to the Python file I am working with, thanks to a "loadUi" function. I wrote a class to connect the buttons I designed with the file upload and prediction functions. To write the estimation function, I reloaded and used my previously trained and saved model.

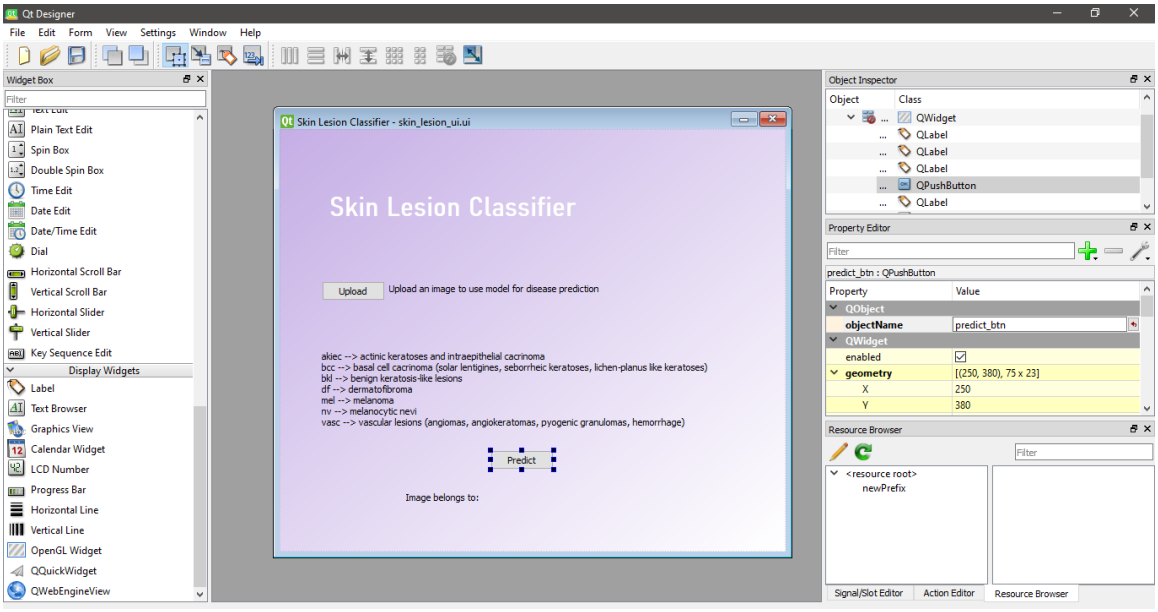

Figure 4.11.1 QtDesigner View

```
class ProgramUI(QDialog):

    def __init__(self):
        super(ProgramUI,self).__init__()
        loadUi("skin_lesion_ui.ui",self)
        self.upload.clicked.connect(self.button_handler)
        self.path = ""
        self.predict_btn.clicked.connect(self.button_handlerr)
        self.dataframe = pd.read_csv('HAM10000_metadata.csv')
        self.size =224
        self.model = tf.keras.models.load_model('saved_model')
        self.labels = []
        le = LabelEncoder()
        le.fit(self.dataframe['dx'])
        LabelEncoder()
        self.dataframe['label'] = le.transform(self.dataframe["dx"])
        labels = self.dataframe['dx'].unique().tolist()
        self.labels = sorted(labels)
```

Figure 4.11.2 Class constructor

```
def button_handler(self):
    self.open_dialog()

def open_dialog(self):
    filename = QFileDialog.getOpenFileName(filter='Image files (*.jpg)')
    path = filename[0]
    print(path)
    self.path = path

def button_handlerr(self):
    self.predict()

def predict(self):
    image_path = self.path
    print(image_path)
    img = image.load_img(image_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_batch = np.expand_dims(img_array, axis=0)
    img_preprocessed = preprocess_input(img_batch)
    prediction = self.model.predict(img_preprocessed)
    index_max = np.argmax(prediction)
    pred = self.labels[index_max]
    self.disease.setText(str(pred))
    print("prediction has done")
```

When the user uploads a file using the upload button, the file path is assigned to the class variable. This path is then used by the predict() function and given to the model after the relevant file is loaded and the necessary pre-processing operations are performed. Label information is assigned to the empty string in the interface by comparing the value with the highest prediction rate from the (1.7) dimensional prediction array that the model outputs. Thus, the user sees the output.

```
app = QApplication(sys.argv)
main_window = ProgramUI()

widget = QStackedWidget()
widget.addWidget(main_window)
widget.setFixedHeight(500)
widget.setFixedWidth(600)
widget.show()

try:
    sys.exit(app.exec_())

except:
    print("Exiting")
```

Figure 4.11.3 Application initialize

5. PROJECT TEAMS

This project will be carried out by a single person, not with a team, under the supervision of a consultant. Project will be carried out within the scope of the "Graduation Project II" course. This project is also intended to be a study in which the knowledge gained within the scope of "Vocational Training in Workplace" will be used.

6. INTERDISCIPLINARY STUDY AREA

Since this project is related to the health as well as engineering, it has an interdisciplinary structure.

Interdisciplinary Area: Health

REFERENCES

[1] Random House Inc., Random House Webster's Unabridged Dictionary, Random House, 2001.

[2] M. Hamblin, P. Avci ve G. Gupta, «Introduction to Imaging in Dermatology,» %1 içinde *Imaging in Dermatology*, Boston, Academic Press, 2016, pp. 1-4.

[3] A. Lallas, Z. Apalla, E. Lazaridou ve D. Ioannides, «Dermoscopy,» %1 içinde *Imaging in Dermatology*, Thessaloniki, Academic Press, 2016, pp. 13-28.

[4] S. Campos ve A. Lencastre, «Dermatoscopic Correlates of Nail Apparatus Disease: A Look at the Nail From Another Scope, the Dermatoscope,» %1 içinde *Imaging in Dermatology*, Lizbon, Academic Press, 2016, pp. 43-58.

[5] R. Lindsay, B. Buchanan ve J. Lederberg, «DENDRAL: A case study of the first expert system for scientific hypothesis formation,» *Artificial Intelligence,* cilt 61, no. 2, pp. 209-261, 1993.

[6] R. Duda ve E. Shortliffe, «Expert Systems Research,» *Science,* cilt 220, no. 4594, pp. 8-261, 1983.

[7] J. DeRoach, «Neural networks—An artificial intelligence approach to the analysis of clinical data,» *Austr. Phys. Eng. Sci. Med,* cilt 12, no. 2, pp. 100-106, 1989.

[8] W. Baxt, «Use of an artificial neural network for the diagnosis of myocardial infarction,» *Annals of Internal Medicine,* cilt 115, no. 11, pp. 8-843, 1991.

[9] D. Bounds ve P. Lloyd, «A multilayer perceptron network for the diagnosis of low back pain,» *Proceedings of the IEEE International Conference on Neural Networks ,* cilt 2, pp. 483-489, 1988.

[10] B. Mulsant, «A neural network as an approach to clinical diagnosis,» *MD Comput.,* cilt 7, no. 1, pp. 25-36, 1990.

[11] M. P.S, J. Dempsey, J. Brooks ve J. Rand, «Using neural networks to diagnose cancer,» *Journal of Medical Systems,* cilt 15, no. 1, pp. 9-11, 1991.

[12] W. McCulloch ve W. Pitts, «A logical calculus of the ideas immanent in nervous activity,» *Bulletin of Mathematical Biophysics,* cilt 5, pp. 115-133, 1943.

[13] D. Hebb, The Organization of Behavior: A Neuropsychological Theory, Psychology Press, 1949.

[14] C. Tapert, «Who Is the Father of Deep Learning?,» %1 içinde *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, 2019.

[15] F. Rosenblatt, «The Perceptron: A Probabilistic Model For Information Storage And Organization In

The Brain,» *Psychological Review,* cilt 65, no. 6, p. 386–408, 1958.

[16] P. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Washington: Harvard University, 1975.

[17] W. Zhang, «Computerized detection of clustered microcalcifications in digital mammograms using a shift-invariant artificial neural network,» *Medical Physics,* cilt 21, no. 4, pp. 24-517, 1993.

[18] T. H. Payne, «Computer Decision Support Systems,» *CHEST,* cilt 118, no. 2, pp. 47S-52S, 2000.

[19] M. Amit X. Garg, M. Neill K. J. Adhikari, M. Heather McDonald ve e. al, «Effects of Computerized Clinical Decision Support Systems on Practitioner Performance and Patient Outcomes,» *JAMA,* cilt 293, no. 10, p. 1223–1238, 2005.

[20] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella and J. Schmidhuber, "Flexible, High Performance Convolutional Neural Networks for Image Classification," in *International Joint Conference on Artificial Intelligence*, Manno, 2011.

[21] E. Bernart, E. Flores ve S. J., «Macroscopic Pigmented Skin Lesion Pre-screening,» %1 içinde *'Encyclopedia of Biomedical Engineering*, Porto Alegre, Elsevier, 2019, pp. 561-573.

[22] G. Jayakashmi ve V. Kumar, «Performance analysis of Convolutional Neural Network (CNN) based Cancerous Skin Lesion Detection System,» %1 içinde *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*, Chennai, 2019.

[23] A. Hameed, M. Umer, U. Hafeez, H. Mustafa, A. Sohaib, A. Siddique ve H. Madni, «Skin lesion classification in dermoscopic images using stacked Convolutional Neural Network,» *Journal of Ambient Intelligence and Humanized Computing,* 2021.

[24] Z. Wan ve T. Zhang, «Skin Lesions Detection via Convolutional Neural Networks,» %1 içinde *2021 2nd International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, 2021.

[25] P. Tschandl, C. Rosendahl ve H. Kittler, «The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions.,» *Sci Data,* cilt 5, pp. 1-9, 2018.

[26] S. Gu, M. Pednekar ve R. Slater, «Improve Image Classification Using Data Augmentation and Neural Networks,» *SMU Data Science Review* , cilt 2, no. 2, p. 16, 2019.

[27] K. He, X. Zhang, S. Ren ve S. J., «Deep residual learning for image recognition,» %1 içinde *Proc IEEE Comput Soc Conf Comput Vis Pattern Recognit*, 2016.

[28] D. Gutman, N. Codella, E. Celebi, B. Helba, M. Marchetti, N. Mishra ve e. al., «Skin lesion analysis toward melanoma detection.,» %1 içinde *A Challenge at the International Symposium on Biomedical*

Imaging (ISBI) 2016, 2016.

[29] S. Han, M. Kim, W. Lim, G. Park, I. Park ve S. Chang, «Classification of the clinical images for benign and malignant cutaneous tumors using a deep learning algorithm,» *J Invest Dermatol,* cilt 138, no. 7, pp. 1529-1538, 2018.

[30] M. Cullell-Dalmau, S. Noé, M. Otero-Viñas, I. Meić ve C. Manzo, «Convolutional Neural Network for Skin Lesion Classification: Understanding the Fundamentals Through Hands-On Learning,» *Front. Med,* no. 644327, p. 8, 2021.

CIRRICULUM VITAE

Personal information

Nationality : Turkish

Place and date of birth : Ankara 03.11.1999

Marital status : Single

Foreign language : English (Upper Intermediate)

Driving license : None

Contact information

Phone : +90 (546) 466 57 96

E-mail : aysenurbuyukbal@gmail.com

Education

2017 September - today	Celal Bayar University Engineering Faculty Computer Science and Engineering Department (English)
2013 September – 2017 June	Ankara Highschool (Anatolian)

Experience

2021 July – 2021 August	**SECURIFY** Information Technologies & Security Corporation - Ankara Teknopark