# Skin Lesion Classification Using Convolutional Neural Networks

**A.Abdukarimov, Z. Kazikhanov**
**IT-2209, 2024**

## OUTLINE

## Introduction:

Problem:

Any region of skin that differs from the surrounding skin in terms of color, shape, size, or texture is referred to as a skin lesion. Skin lesions are quite prevalent and frequently develop from localized skin injury, such as contact dermatitis or sunburns. Conversely, certain conditions may be signs of underlying illnesses such diabetes, infections, autoimmune diseases, or genetic problems. The accurate classification of skin lesions plays a crucial role in early detection and treatment of skin diseases. Traditional diagnosis methods can be time-consuming and subjective, highlighting the need for automated and efficient systems for skin lesion classification.

Literature Review:

1. This study introduces ViT-NeT, addressing interpretability issues in CNN and ViT models for image classification. Utilizing ViT as a backbone, ViT-NeT achieves a state-of-the-art (SOTA) performance without compromising interpretability, showcasing its superiority on benchmark datasets. Acknowledgments cite support from the National Research Foundation of Korea.
https://paperswithcode.com/paper/vit-net-interpretable-vision-transformers

2. This study proposes a revolutionary method for machine learning and supports models that are modular and adaptable. Continuous development is made possible by this innovative process, which smoothly combines design iterations into a single adaptable system. The paper demonstrates its effectiveness by creating a multi task machine learning model that performs very well in 124 image classification tasks, exhibiting cutting-edge quality, decreased size, and enhanced computing efficiency.
https://paperswithcode.com/paper/a-continual-development-methodology-for-large

Current Work:

In this project, we propose a CNN-based approach for skin lesion classification using the HAM10000 dataset. Our goal is to leverage deep learning to maximize accuracy of categorizing skin lesions into distinct classes for improved diagnostic results.

Steps taken to achieve the goal:

1. Data acquisition
2. Data preparation and processing
3. Training and evaluation of the model based on conventional data
4. Training and evaluation of the model based on augmented data
5. Using an existing, advanced model
6. Converting a finished model and integrating it in a webpage

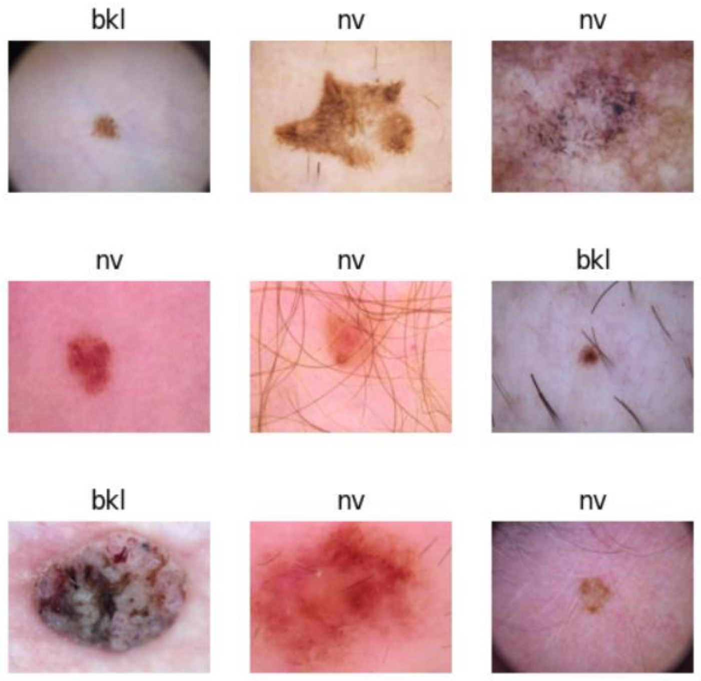
**Data and Methods:**

Information about the data:

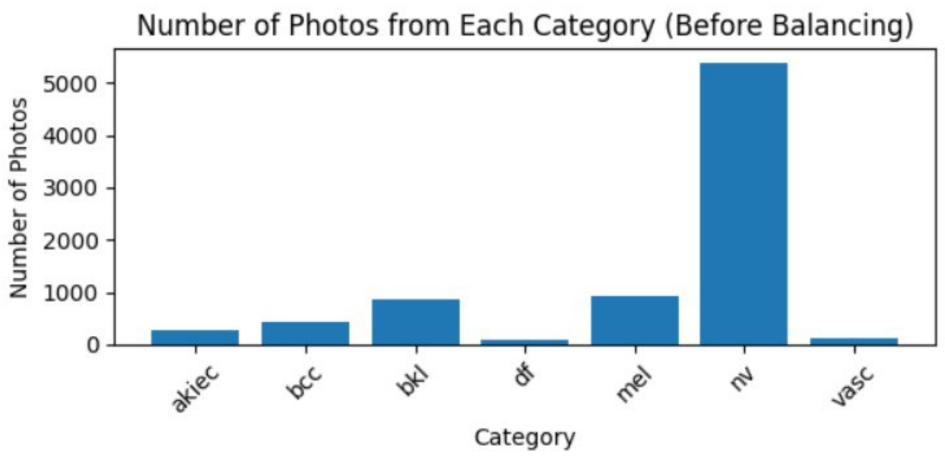
The HAM10000 dataset comprises 10,000 dermoscopic images of skin lesions, categorized into seven classes: melanoma, melanocytic nevus, basal cell carcinoma, actinic keratosis, benign keratosis, dermatofibroma, and vascular lesion. A preliminary analysis of the dataset reveals variations in lesion size, color, and texture. Dataset can be accessed by this link: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T

Dataset was downloaded locally and uploaded to Google Drive for easy access from Google Colaboratory. Here is the link to the dataset: https://drive.google.com/drive/folders/1RFX58NYHw0sHY9C62-UvPYlV0v4dVgJQ?usp=sharing

There is a matplotlib graph illustration of 9 random images from out dataset to describe what kind of material we are currently working with (picture 1).

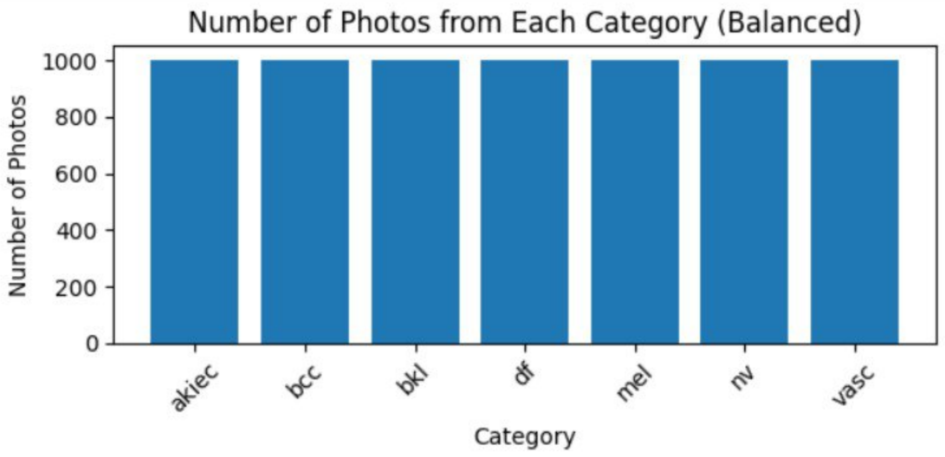


Picture 1



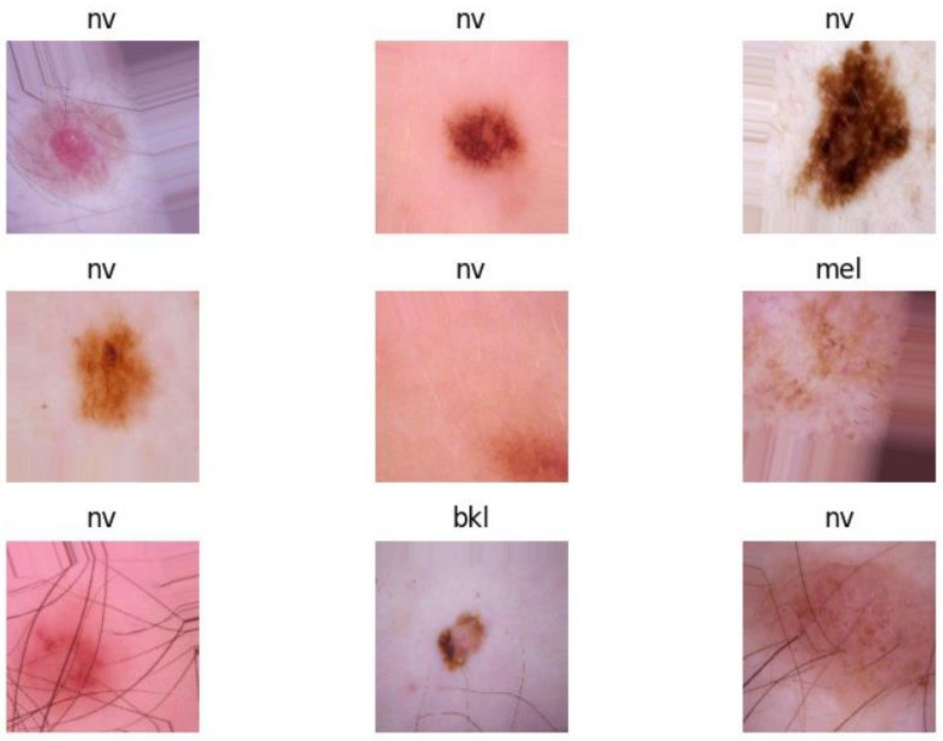
Picture 2

As you can notice in the picture above (picture 2), the difference between number of photos from each category is quite high, therefore we applied techniques of undersampling and oversampling simultaneously, so the dataset for future operations will be more stable and balanced (picture 3).



Picture 3

As the number of images from each category has stabilized, we are able to create so called “image generator” that will be taking bunch of images directly from dataset and send it to the model in the form of “batches”, each containing 32 images of random categories. The reason it is 32, which is power of 2, because the batch size needs to fit the memory requirements of the GPU and architecture of CPU.
On top of that, we have also created augmented images in order to be able to face future problems such as overfitting ( when model is useful only on training dataset and not any other ) or underfitting ( data deficiency ), here is the matplotlib graph illustrations consisting of 9 augmented images for various categories from our dataset (picture 4).



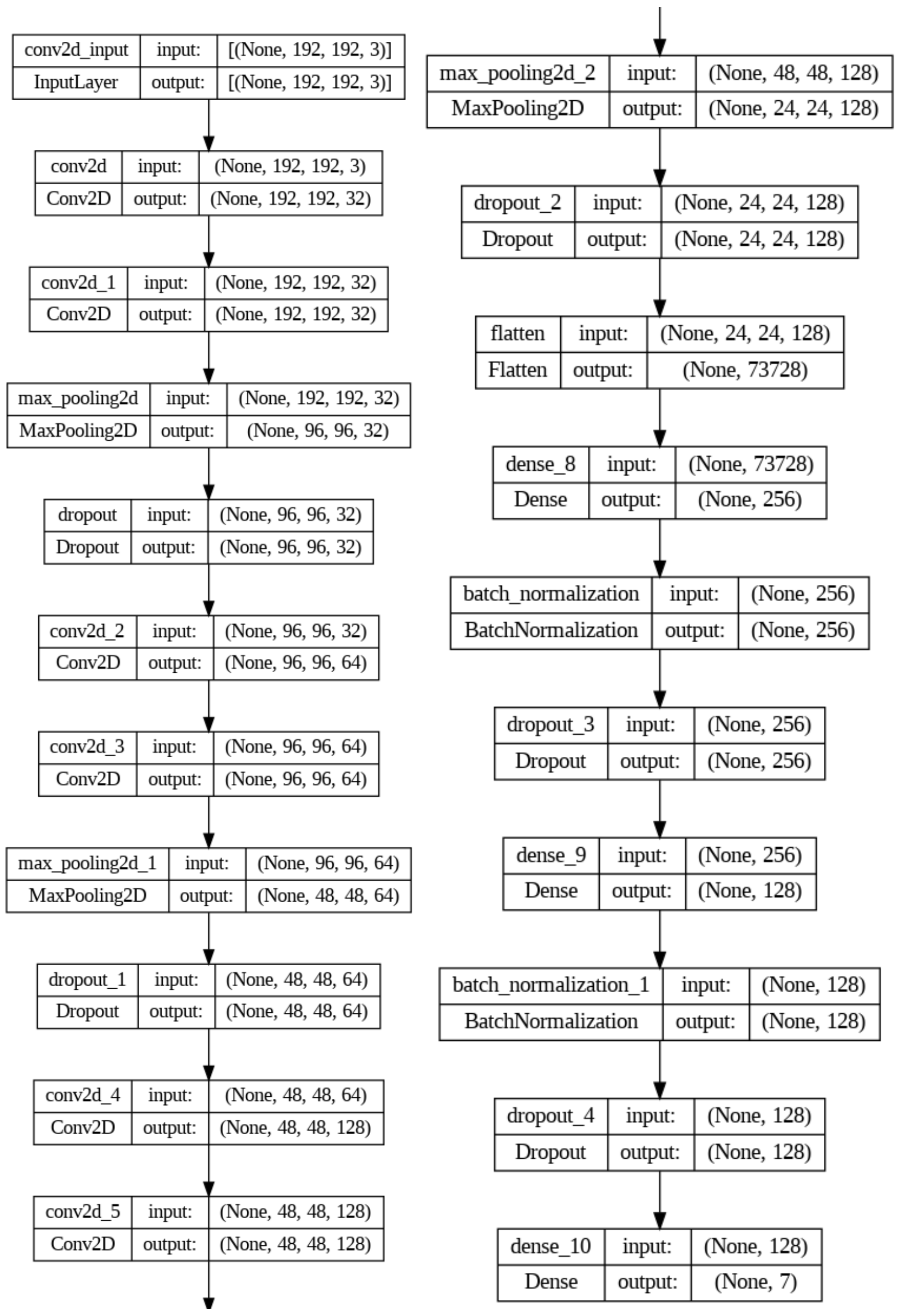
Picture 4

Description and theory:

A Convolutional Neural Network (CNN) architecture is chosen for its ability to automatically learn features from image data. The CNN model consists of convolutional layers for feature extraction, followed by pooling layers to reduce dimensionality, and fully connected layers for classification.

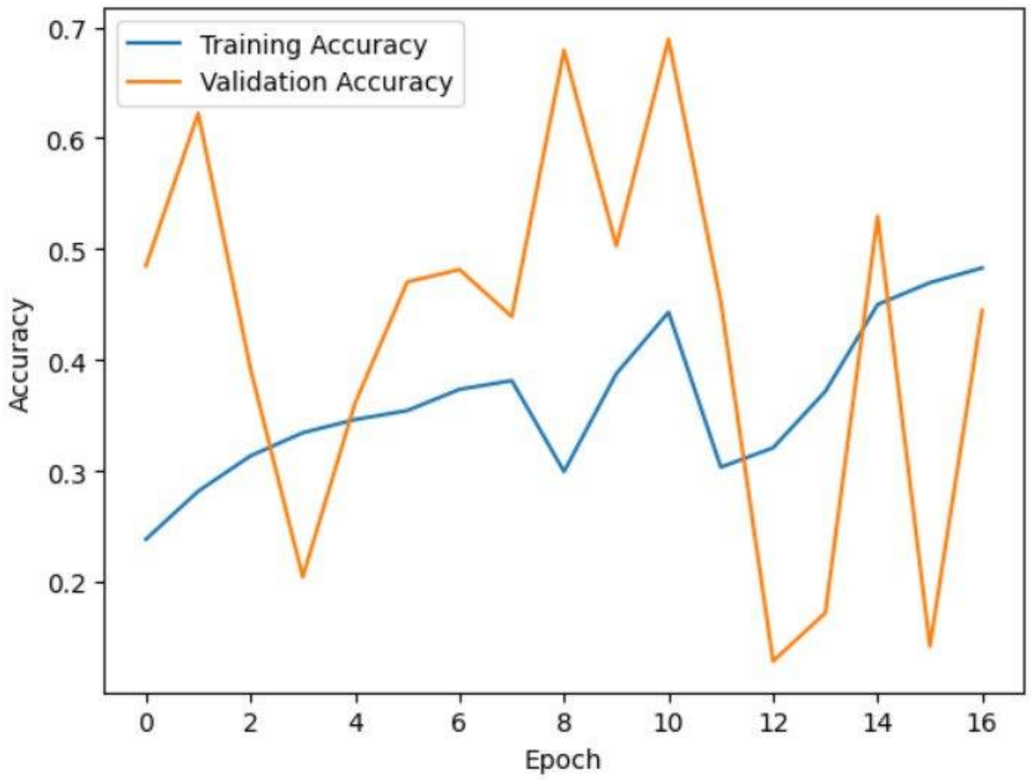
For this project, we decided to try to build our own CNN model using Tensorflow and Keras library. It is worth mentioning that every training process is repeated for 30 epochs with the patience value equal to 8 epochs, which means if the performance does not become better in interval of 8 epochs, the process will stop, and model will be ready.

Here is the illustration that describes the work principle of our first model, which is a base model that trains based on regular data from the given dataset:

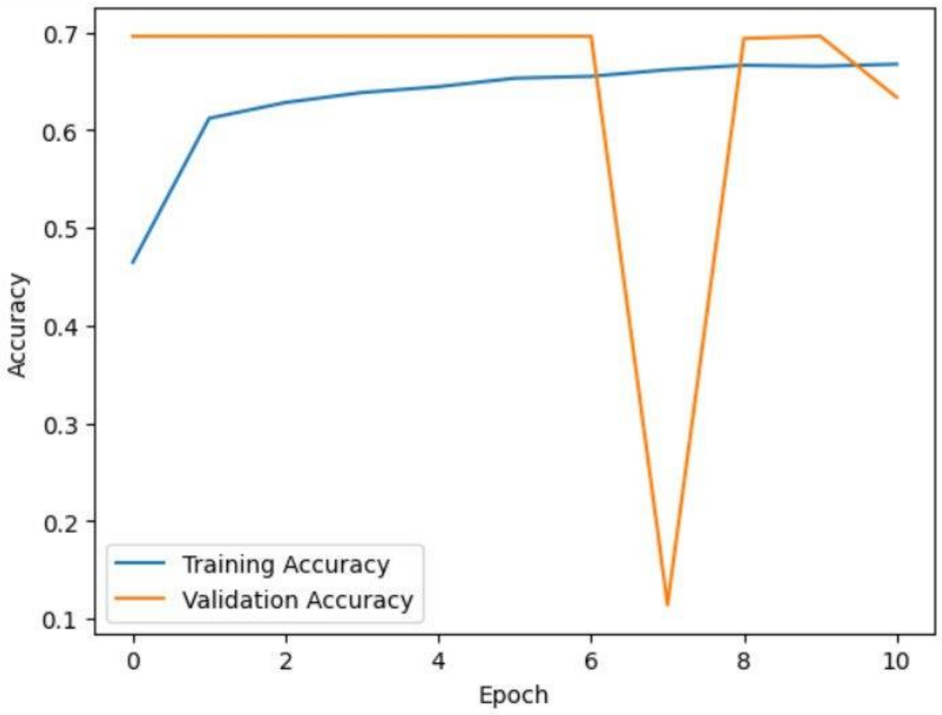

conv2d_input | InputLayer | input: [(None, 192, 192, 3)] | output: [(None, 192, 192, 3)]
conv2d | Conv2D | input: (None, 192, 192, 3) | output: (None, 192, 192, 32)
conv2d_1 | Conv2D | input: (None, 192, 192, 32) | output: (None, 192, 192, 32)
max_pooling2d | MaxPooling2D | input: (None, 192, 192, 32) | output: (None, 96, 96, 32)
dropout | Dropout | input: (None, 96, 96, 32) | output: (None, 96, 96, 32)
conv2d_2 | Conv2D | input: (None, 96, 96, 32) | output: (None, 96, 96, 64)
conv2d_3 | Conv2D | input: (None, 96, 96, 64) | output: (None, 96, 96, 64)
max_pooling2d_1 | MaxPooling2D | input: (None, 96, 96, 64) | output: (None, 48, 48, 64)
dropout_1 | Dropout | input: (None, 48, 48, 64) | output: (None, 48, 48, 64)
conv2d_4 | Conv2D | input: (None, 48, 48, 64) | output: (None, 48, 48, 128)
conv2d_5 | Conv2D | input: (None, 48, 48, 128) | output: (None, 48, 48, 128)
max_pooling2d_2 | MaxPooling2D | input: (None, 48, 48, 128) | output: (None, 24, 24, 128)
dropout_2 | Dropout | input: (None, 24, 24, 128) | output: (None, 24, 24, 128)
flatten | Flatten | input: (None, 24, 24, 128) | output: (None, 73728)
dense_8 | Dense | input: (None, 73728) | output: (None, 256)
batch_normalization | BatchNormalization | input: (None, 256) | output: (None, 256)
dropout_3 | Dropout | input: (None, 256) | output: (None, 256)
dense_9 | Dense | input: (None, 256) | output: (None, 128)
batch_normalization_1 | BatchNormalization | input: (None, 128) | output: (None, 128)
dropout_4 | Dropout | input: (None, 128) | output: (None, 128)
dense_10 | Dense | input: (None, 128) | output: (None, 7)

Let’s breakdown the structure of the model above:
Input Layer: The input shape is determined by variable input_shape, which is equal to (192, 192, 3), 192x192 pixels and 3 color channels.
Convolutional Layers: Convolutional layers are used to extract features from the input images. Each convolutional layer applies a set of filters to the input images, which helps the model learn features.
The first argument (32, 64, 128) specifies the number of filters to be applied. Each filter acts as a feature detector and extracts specific features from the input data. The choice of the number of filters determines the depth of the feature maps produced by the convolutional layers.
Activation Function: Rectified Linear Unit activation function is used after every convolutional layer. “ReLU” helps the model to learn complex patterns in the data in non-linear way.
Pooling Layers: Max pooling layers are added after some convolutional layers to reduce the dimensions of the feature maps, making the model more efficient and helps to reduce overfitting.
Dropout Layers: Dropout layers are inserted after convolutional and dense layers. Dropout is a regularization technique used to prevent overfitting in neural networks by randomly dropping (setting to zero) a portion of input units during training. For example, in our model – we used a dropout rate of 0.25, which means that 25% of the input units will be randomly set to zero during each epoch.
Flatten Layer: The flatten layer converts the 3D feature maps into a 1D vector, preparing the data for input into the dense layers.
Dense Layers: Dense layers are connected layers that learn from the features extracted by the convolutional layers. They help the model to learn complex patterns and relationships in the data.
Batch Normalization: Batch normalization is applied after dense layers. It normalizes the activations of the previous layer, improving training stability.
Output Layer: The output layer uses the softmax activation function, which is suitable for multi-class classification problems. It outputs a probability distribution over the classes, with each neuron representing the probability of belonging to a particular class.
Model Output: The model outputs class probabilities for each input image, with the number of output neurons matching the number of classes (7 in this case).

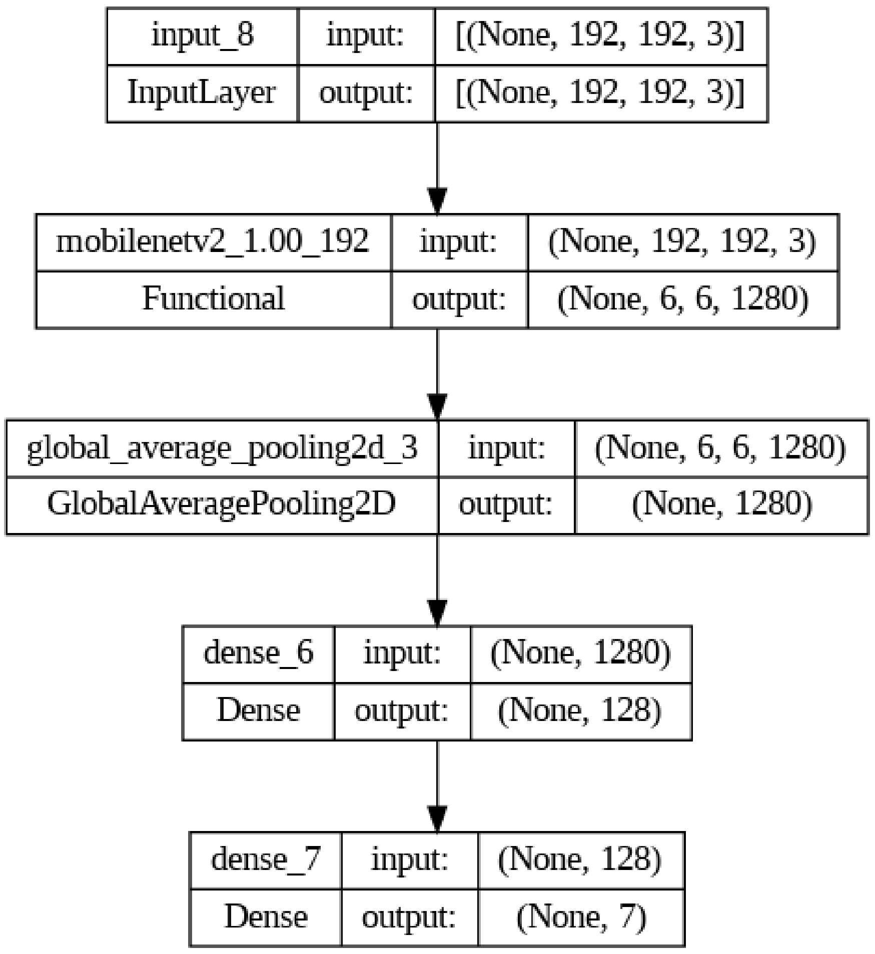
Also, this is the result of training of our first CNN model:



Here is the illustration that describes performance results of the same first model trained on augmented database using “ImageDataGenerator”:



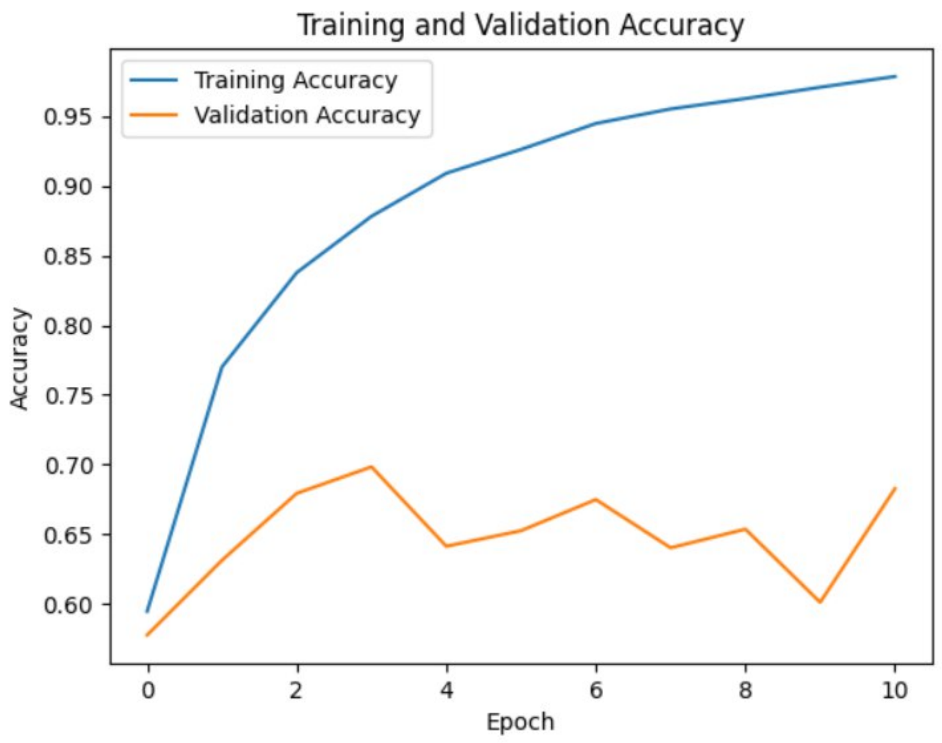
Here is also the structure of the MobileNetV2 pre-trained model that will be trained and tested on our dataset regarding skin lesions, the results will be discussed later in this document:



In this case, we are again using input layer of (192, 192, 3) and output layer of 7, which is the number of classes in our dataset.
Global Average Pooling is applied to reduce the spatial dimensions of the feature maps obtained from the pre-trained model.
Dense layer with 128 units is added after the Global Average Pooling layer.

**Results:**

As the result of our project and research, we created a Convolutional Neural Network by using transfer learning method based on pre-trained model named "MobileNetV2", which has shown great performance on our dataset HAM10000, which is stored on Google Drive and is available by link above. Here are the performance statistics of final model:



It is not perfect, but the model showed almost 100% accuracy on training dataset and around 70% on evaluation dataset. On testing dataset, it scored almost 72% accuracy, which is pretty decent.

**Discussion:**

Critical review of results :

Due to poor performance of the first model on regular data, our next decision was to train it on augmented dataset, which might have been a great solution to solve previously mentioned model training problems (underfitting).

Although, performance of the model on the second attempt was better, the results (65% of correct guesses) are still unsatisfying for our project. Therefore, we decided to apply transfer learning, which is considered to use pre-trained models such as MobileNet or ResNet to leverage existing learned features.

In our case, our choice was to use MobileNetV2, which has efficient architecture, combined with its ability to maintain high accuracy, makes it an ideal choice for resource-constrained devices, for example: web-service like in our case.

Next step is to create a localhost web-page using technologies such as HTML, CSS and JavaScript. We save, transform and download our model in order to integrate it within our webpage and the result looks like this:

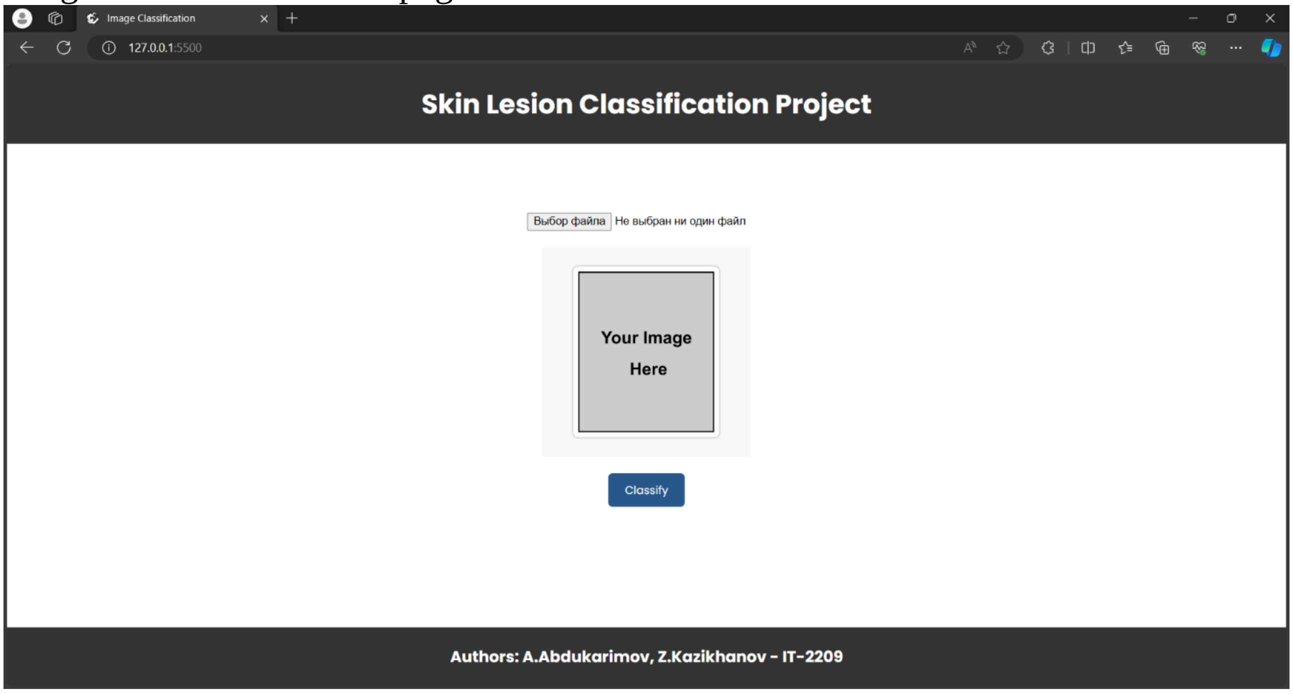


Image Classification
127.0.0.1:5500
Skin Lesion Classification Project
Выбор файла Не выбран ни один файл
Your Image Here
Classify
Authors: A.Abdukarimov, Z.Kazikhanov - IT-2209

Here is the Github link to download and try out this web-page:
- **https://github.com/don-juann/classification-skin-lesion**

Instructions to launch web-site:
1)Download repository to your local machine
2)Open Command Line(Win+R, cmd) and enter the location of the web-site directory
3)Enter command “python -m http.server 3000”
4)Open “localhost:3000” from your browser

**References:**

- https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7
-tf.keras.preprocessing.image.ImageDataGenerator | TensorFlow v2.15.0.post1
-MobileNet, MobileNetV2, and MobileNetV3 (keras.io)
-The Functional API | TensorFlow Core