Politechnika Śląska

Katedra Grafiki, Wizji komputerowej

i Systemów Cyfrowych

Academc year			Group	Section
2022/2023	SSI	**BIAI**	**ISMiP**	**2**
Supervisor:	dr inż. Grzegorz Baron		**Classes: (day, hour)**	
Section members:	**Jakub Hoś** **Bartłomiej Piątek**		**Tuesday**	
emails:	**jakuhos330@student.polsl.pl** **bartpia809@student.polsl.pl**		**11:30**	

Report

Subject:

Classification of furry animals

Main assumptions:

1.Short introduction presenting the project topic

Our goal was to build a model which can be used on classifying images containing some of furry animals(sheep, dog, cat, rabbit or squirrel).

Accuracy of trained model should be close to 100%. Training algorithm will be implemented in Python using tensorflow library.

2. Analysis of the task

Use of pretrained model

Pros of Pretrained Models:

- Time and resource efficiency:
  Pre-trained models are already trained on large datasets, saving a lot of time and computational resources.
- Improved performance:
  Pre-trained models often perform well out of the box because they are exposed to large amounts of heterogeneous data during training.
- Transfer learning:
  Pre-trained models enable transfer learning, where knowledge learned in one task or domain can be applied to another task or domain.

Cons of Pretrained Models:

- Lack of domain specificity:
  Pre-trained models are trained on large and diverse datasets and may not be optimized for specific niche domains or applications.
- Demand for massive computing resources:
  Using pre-trained models saves computational resources during training, but fine-tuning and adapting models to specific tasks can still be computationally intensive.

Use of own model:

Pros of own model:

- Domain Specificity:
  By training your own model, you can adapt it to the specific needs of your domain or task.
- Interpretability and controllability:
  Training your own model gives you more control over its architecture, hyperparameters, and training process.
- Privacy and Security:
  Training your own model gives you complete control over your training data. This can be important for applications that contain sensitive or proprietary information.

Cons of own model:

- Time and resource intensity:
  Training a model from scratch can be time consuming and computationally intensive, especially when dealing with large datasets and complex architectures.
- Data requirements:
  Training your own model typically requires a large amount of labeled training data.
- Knowledge and skill requirements:
  Developing and training your own models requires expertise in BIAI.

We were looking for comprehensive and diverse dataset that provides rich and valuable information for analysis at Kaggle.com We collected around 8 thousands images in training dataset and 400 images to validate models accuracy. Then We resized them and used augmentation to improve models versatility.

Tools and libraries

Used IDE: Visual Studio 2022 – used to create Python console app project

Language: Python 3.7

Used Python libraries:

- keras: It is a high-level neural networks API written in Python, which can run on top of TensorFlow, Theano, or CNTK.
- keras.models: It provides a way to create and manipulate models in Keras, a high-level neural networks API.
- keras.layers: It contains various types of layers, such as convolutional layers, pooling layers, and dense layers, that can be used to build neural network models.
- tensorflow.keras.layers: It is a module that contains layer implementations for TensorFlow, which can be used alongside the Keras API.
- keras.applications: It provides pre-trained models that can be used for various tasks, such as image classification and object detection. Examples include ResNet, Inception, and MobileNet.
- numpy: It is a fundamental package for scientific computing with Python, providing support for large, multi-dimensional arrays and matrices, along with a large collection of mathematical functions to operate on these arrays.
- pandas: It is a powerful data manipulation and analysis library. It provides data structures like DataFrames for efficient data handling and processing.
- tensorflow: It is an open-source machine learning framework that allows you to build and train neural networks. It provides a comprehensive set of tools and libraries for deep learning.
- cv2: It is a library for computer vision tasks. It provides functions for image processing, feature extraction, and object detection, among other things.

It was possible to implement own model in programing languages like C++, but we decided to use programing language which is more associated with AI algorithms – Python.

3. Internal and external specification of the software solution

Script use to train model, read input files, showing the results and learning progress - ***Cat_or_Dog.py***

Analysis of own model script functions:

- conv_relu:
  - Function parameters depend on its position in model structure
  - Function consist of three layers:
    - Conv2D
    - BatchNormalization
    - Activation – relu
- dense_relu:
  - Function parameters depend on its position in model structure
  - Function consist of three layers:
    - BatchNormalization
    - Dense; activation- relu
    - Dropout
- aspiring_mobile_net

- Function Builds a sequential model, using custom functions with keras layers.
- Follows logic:
  - Every few conv layers with set filters put Pooling layer.
  - At the end put dense layers with decreasing number of filters
  - Between every dense layer put dropout layer.

Model Architecture Code:

```
conv_relu(filters=32, kernel_size=(3, 3), strides=(1, 1))

conv_relu(filters=32, kernel_size=(3, 3), strides=(1, 1))

MaxPooling2D(pool_size=(2,2))

conv_relu(filters=64, kernel_size=(3, 3), strides=(1, 1))

conv_relu(filters=64, kernel_size=(3, 3), strides=(1, 1))

MaxPooling2D(pool_size=(2,2))

conv_relu(filters=128, kernel_size=(3, 3), strides=(1, 1))

conv_relu(filters=128, kernel_size=(3, 3), strides=(1, 1))

MaxPooling2D(pool_size=(2,2))

conv_relu(filters=256, kernel_size=(3, 3), strides=(1, 1))

conv_relu(filters=256, kernel_size=(3, 3), strides=(1, 1))

conv_relu(filters=256, kernel_size=(3, 3), strides=(1, 1))

MaxPooling2D(pool_size=(2,2))

conv_relu(filters=512, kernel_size=(3, 3), strides=(1, 1))

conv_relu(filters=512, kernel_size=(3, 3), strides=(1, 1))

conv_relu(filters=512, kernel_size=(3, 3), strides=(1, 1))

MaxPooling2D(pool_size=(2,2))

conv_relu(filters=512, kernel_size=(3, 3), strides=(1, 1))

conv_relu(filters=512, kernel_size=(3, 3), strides=(1, 1))

conv_relu(filters=512, kernel_size=(3, 3), strides=(1, 1))

GlobalAveragePooling2D()

Dense(4096,activation='relu')

Dropout(dropout)

Dense(4096,activation='relu')

Dropout(dropout)

Dense(1024,activation='relu')

Dropout(dropout)

Dense(256,activation='relu')

Dropout(dropout)
```

Function connected with managing data:

- collect_filepaths(root_path:str, class_names:list, shuffle:bool = True) -> list:

  Function collects image paths from all classes and returns a shuffled list/list of all the file paths.

  Returns list of collections of all the paths.

- load_image(image_path:str, size:tuple = (256,256), normalize:bool = True):

  The function takes in the image path and loads the image, resize the image to the desired size.

  Returns a Tensorflow 32Bit-Float Tensor of the Image.

- load_data(paths:list, mapping:dict, size:tuple = (256,256), tf_data=False, BATCH_SIZE:int=32):

  This function starts with creating a space for the images & labels to be loaded. Then it iterates through the file paths and load the image & the respective label. It stores these loaded image & label. If needed, it converts this numpy array data into a tensorflow data. At last, it returns the generated data.

- show_images(images:list, labels:list, class_names:list, model:None = None, GRID:tuple = (6,10), SIZE:tuple = (30,25)) -> None:

  This function is responsible for creating a plot of multiple images & labels from the given dataset.

We used data structures from imported libraries and standard Python list, dict, tuple etc.

Our program doesn't provide rich user interface. While building the model, the information about learning progress are loged to the text file.

Then model parameters are also logged to the file(when learning ended).

```
249/249 [==============================] - ETA: 0s - loss: 0.1040 - sparse_categorical_accuracy: 0.9688
249/249 [==============================] - 1067s 4s/step - loss: 0.1040 - sparse_categorical_accuracy: 0.9688
- val_loss: 0.1480 - val_sparse_categorical_accuracy: 0.9557
Epoch 28/50

  1/249 [..............................] - ETA: 19:14 - loss: 0.0355 - sparse_categorical_accuracy: 1.0000
  2/249 [..............................] - ETA: 18:27 - loss: 0.0610 - sparse_categorical_accuracy: 1.0000
  3/249 [..............................] - ETA: 18:15 - loss: 0.0975 - sparse_categorical_accuracy: 0.9896
```

Img.1 „Part of example log file"

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DataAugmetor (Sequential)    (None, 256, 256, 3)       0
_________________________________________________________________
batch_normalization (BatchNo (None, 256, 256, 3)       12
_________________________________________________________________
mobilenet_1.00_224 (Function (None, None, None, 1024)  3228864
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 1024)        4096
_________________________________________________________________
global_average_pooling2d (Gl (None, 1024)              0
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 1025
=================================================================
Total params: 3,233,997
Trainable params: 3,210,055
Non-trainable params: 23,942
_________________________________________________________________
```

Img.2 „End of example log file”

4. Experiments

We decided to solve the problem in two other ways.

The first solution was based on pretrained models. As we expected the gained accuracy was really satisfying. We were delighted to discover that some of our top-performing models achieved an impressive accuracy rate of approximately 99%.

It was a great score in comparison to our own trained model, which exhibited an accuracy of around 80%.

Pretrained models

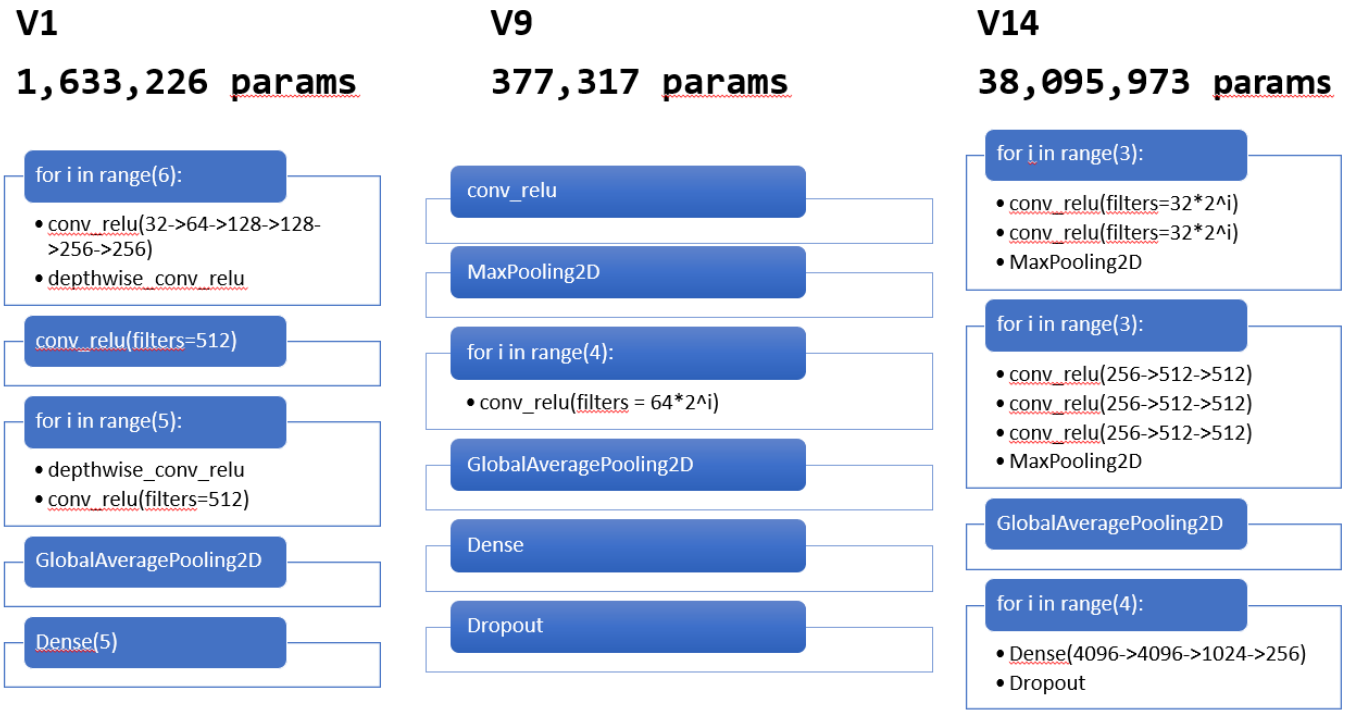
We used two different pretrained models, as core of our models:

- MobileNet
- ResNet

We generated seven different models using pretrained ones. The models primarily differed in terms of the base architecture, the augmentation techniques applied to the input images, and the utilization of Dropout. Regardless of changes results were rather consistent

Own models

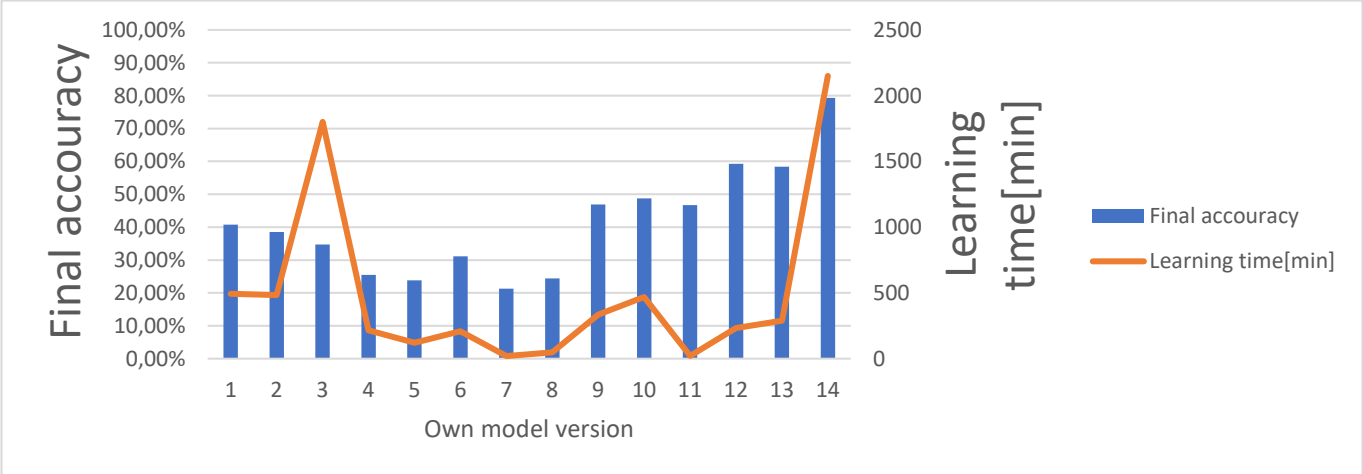
We created 14 different versions of our own model. We started with an initial accuracy of approximately 40%. However, through continuous iterations and improvements, we were able to enhance the performance of our model significantly. The latest version achieved an accuracy rate of 80%.



Img.3 „Mile Stones of our own models architecture"

The key differentiating factors among the versions primarily revolved around several crucial architectural components, including the number of conv_relu filters, the arrangement and quantity of layers, the Dense parameters, and the utilization of MaxPooling2D.

Observed results

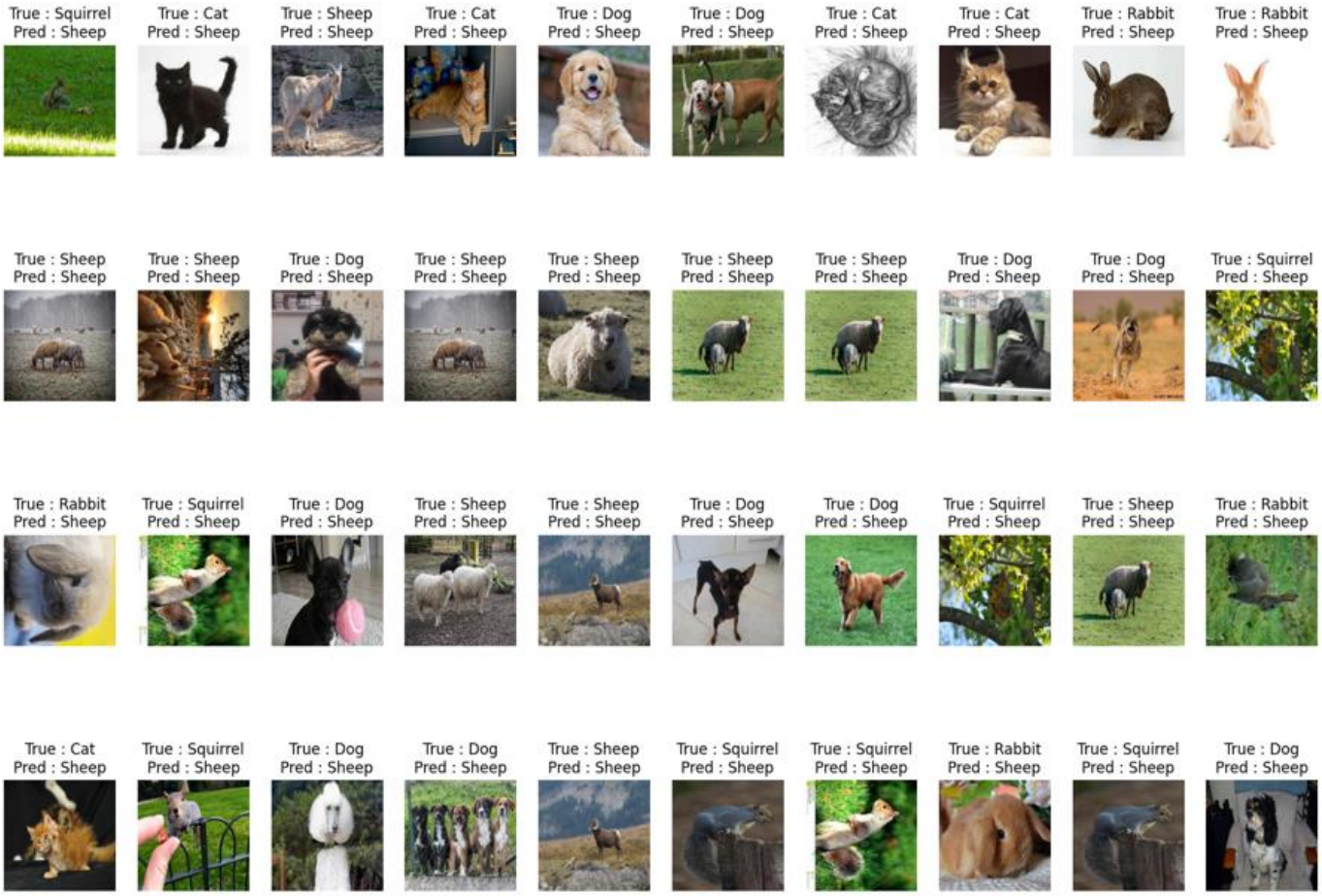


Img.4 „Accuracy and learning time of different versions of our model"

We started with not large amounts of layers and single Dense(5). We obresved, that the number of parameters is strongly connected to model accuracy and learning time:

Version	Params	Learning time	Accuracy
V1	1633226	493	41%
V9	377317	336	47%
V14	38095973	2150	79%

All information about models are read from log file generated by our script. Script also generates an image with selected images from our dataset and models predictions:



Img.5 „Generated image, after learning ended"

Dataset

As it was mentioned, our dataset includes around 8000 images:

Training set:

- Cat: 1845
- Dog: 1655
- Rabbit: 967
- Sheep: 1740
- Squirrel: 1782

Testing set:

- Cat: 100
- Dog: 100
- Rabbit: 40
- Sheep: 80
- Squirrel: 80

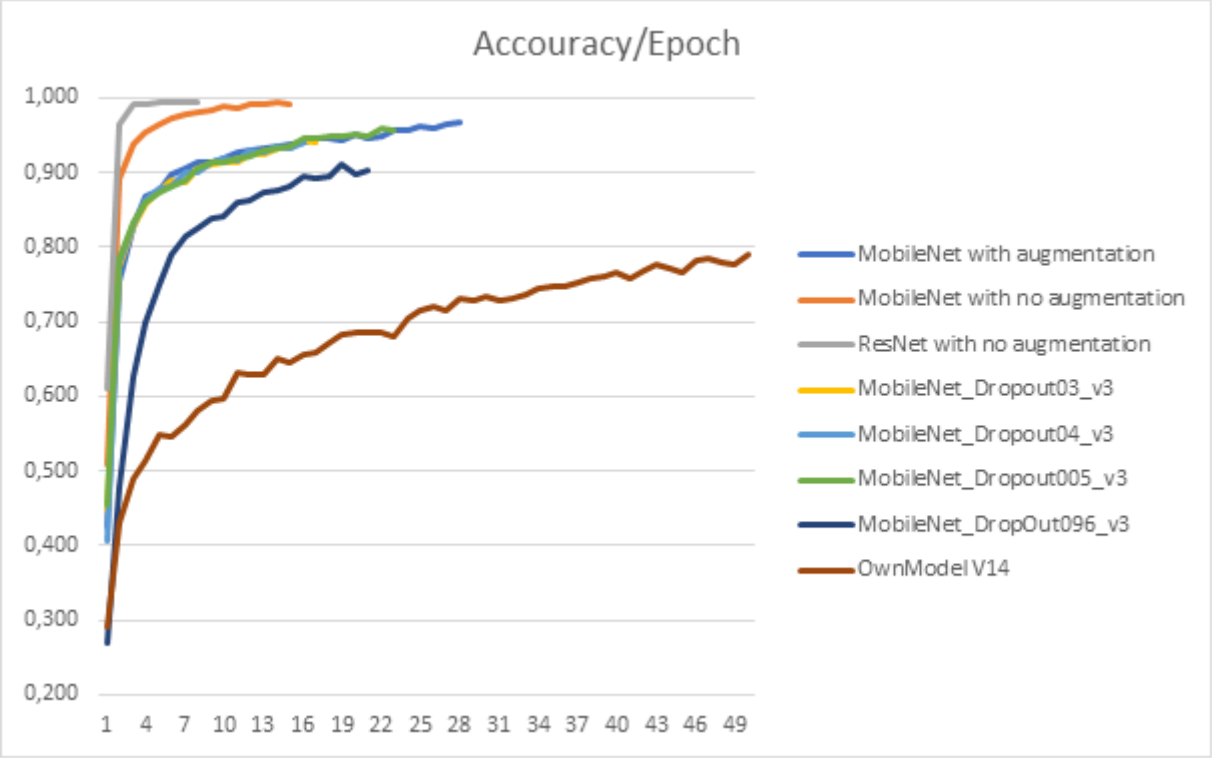
Each image was scaled to 300x300px.

Data augmentation

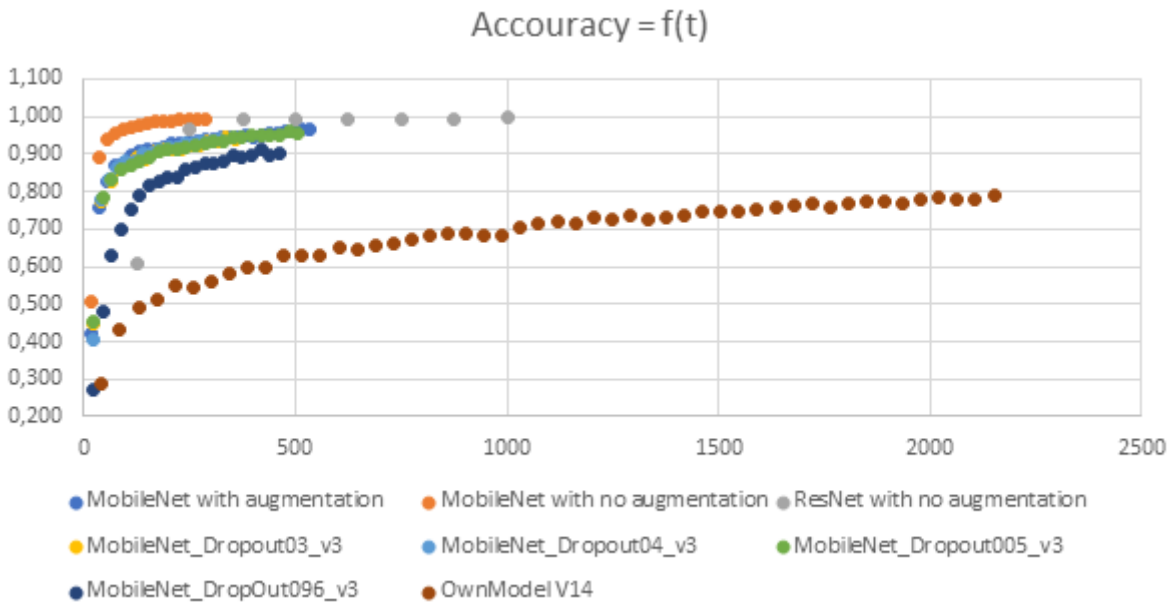
Some of our models were trained with augmented data – overall conclusions:

- Learning time for models with augmented data was higher
- Accuracy on not augmented testing set was similar or lower
- Accuracy on augmented testing set was higher
- Model trained on augmented dataset was much more robust
- Augmentation can be used to diverse and expand dataset if needed

Accuracy of our custom model



Img.6 „ The dependence of model accuracy on epoch"



Img.7 „ The dependence of model accuracy on time"

5.Summary

As anticipated, developing our own model proved to be a formidable and arduous task. It consumed over 100 hours to train the final model, yielding an accuracy of approximately 80%.

By leveraging pre-trained models, we were able to construct highly accurate models within a relatively brief timeframe, excluding ResNet. This process took roughly 70 hours to complete.

Based on our calculations, we estimated a power consumption of approximately 31 kWh, which translates to an approximate coal usage of 14 kg.

6.References

Convolutional Neural Networks (CNNs), Deep Learning, and Computer Vision

Convolutional Neural Networks, Explained

Breaking down Convolutional Neural Networks: Understanding the Magic behind Image Recognition

MobileNet V1 Architecture

An Overview on MobileNet: An Efficient Mobile Vision CNN

kaggle – cats vs dogs

kaggle – animals dataset

7.Link to files

GitHub