Politechnika Śląska

Biologically inspired artificial intelligence

Final report

„Car plate detection”

Jakub Polczyk & Grzegorz Piotrowski

1. Introduction

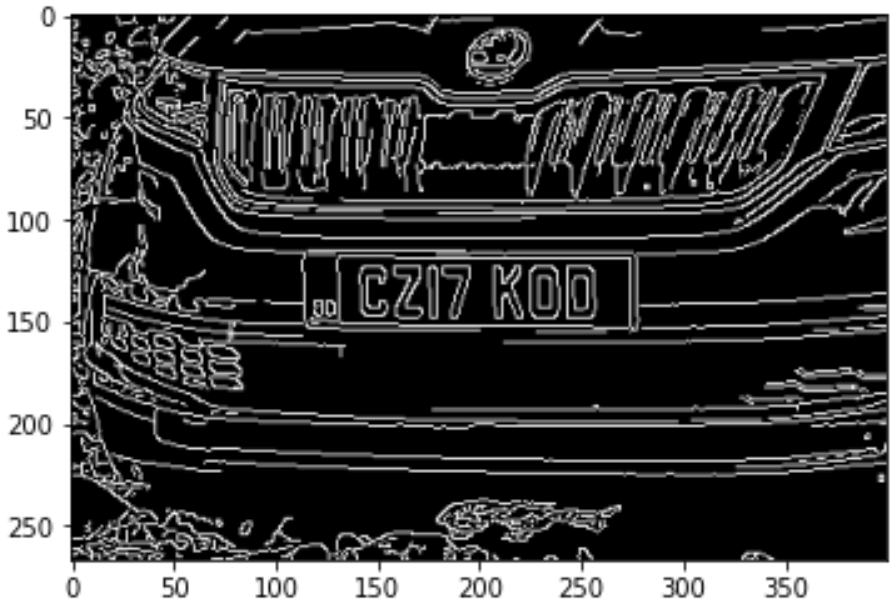
The dataset contains images of cars with bounding box annotations of the car license plates within the image. The goal is to create a detection model to localize the car license plate.

2. Analysis of the task

a. Possible approaches

i. Optical Character Recognition (OCR)

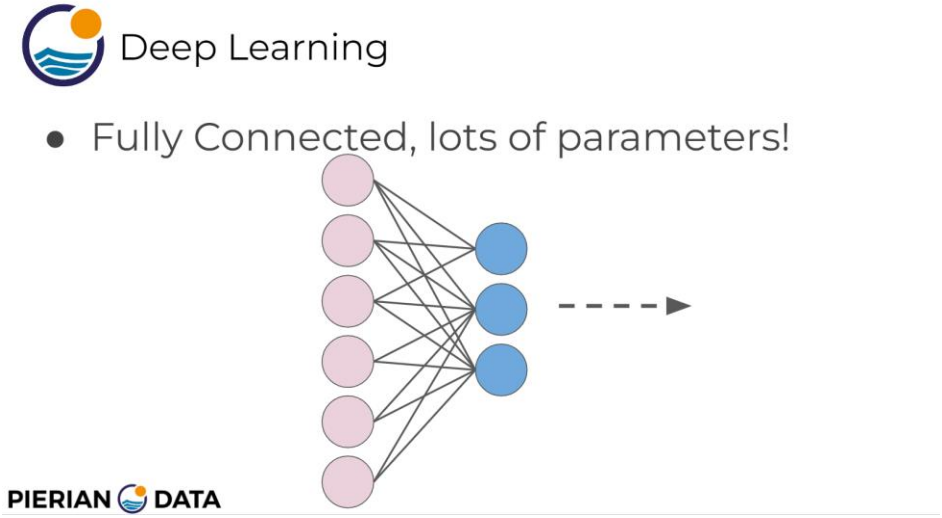
The first method relies on applying a set of filters combined with finding contours on the image. The great advantage of this approach is execution time but even worse is its disadvantage – lack of flexibility. Filters are not able to adapt properly to all conditions e.g., lighting or sizes.



ii. Artificial Neural Network (ANN)

A much better approach is utilizing an artificial neural network that can easily adapt to newer seen before images. Unfortunately, there are a few problems that exclude this method. First of all, the number of parameters increases significantly with the size of the image making it useful only

for smaller pictures. Second of all, it is not universal – its high performance is restricted to very small and centered images. What is more, ANN is converting each image to a single array making it lose all valuable information from a 2D image. Last but not least, it was not meant to be used for dealing with pictures.

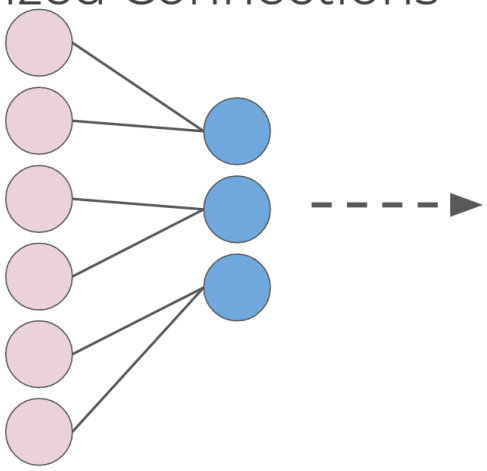


iii. **Convolutional Neural Network (CNN)**

The only right approach is to implement CNN which uses convolutional layers to help alleviate all those previous issues. Each layer consists of multiple filters. To further improve performance, pooling and dropouts are applied. The whole architecture is trained to figure out the best filter weight values.

Deep Learning

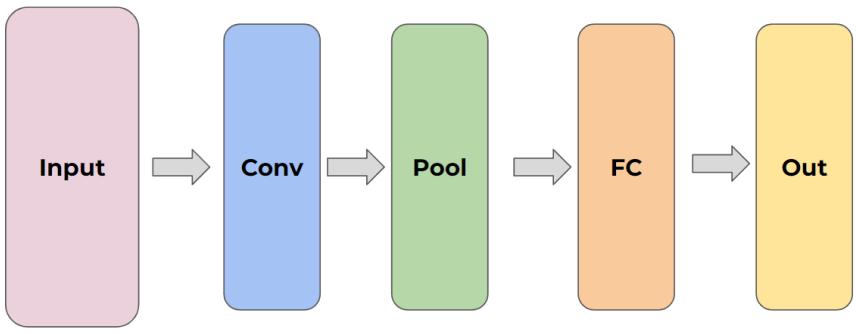
- CNN - Localized Connections



PIERIAN DATA

Deep Learning

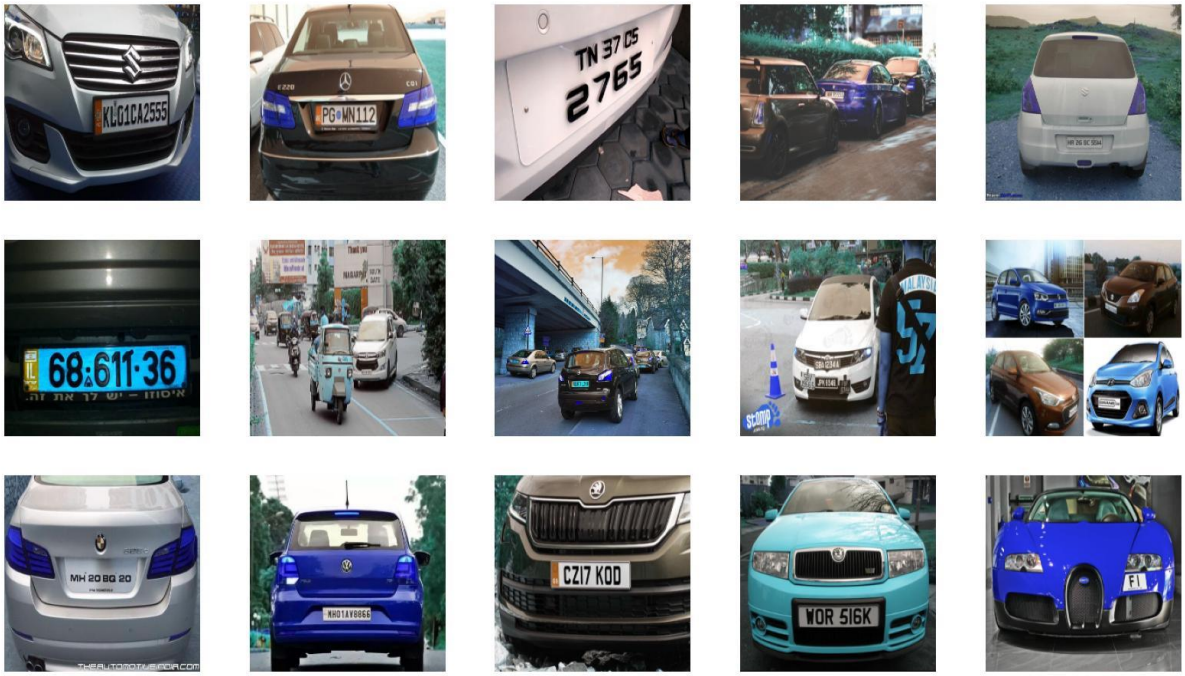
- CNNs can have all types of architectures!



PIERIAN DATA

b. Dataset presentation

The dataset contains 433 images with bounding box annotations of the car license plates within the image. Annotations are provided in the PASCAL VOC format.



Dataset with annotations:

c. Available and selected tools

i. *Numpy* – mathematical functions
ii. *Pandas* – data analysis
iii. *Matplotlib* – data visualization
iv. *Seaborn* – data visualization
v. *OpenCV* – computer vision
vi. *Scikit-learn* – machine learning tools
vii. *Keras Tensorflow* – deep learning

3. Internal software specification

The project is organized by multiple directories with corresponding content. The most important directories are 'annotations' and 'images' which play the role of a dataset followed by 'notebooks' folder which mainly contains Jupyter Notebook files for each model. Results are stored in 'saved_images' and 'saved_models' directories.

Both images and annotations after being loaded are resized to the same uniform format. Next, they are converted to a Numpy array which is required by the Tensorflow models' input. It is followed by data normalization by dividing each value by 255 to fit between 0 and 1 for best performance.

In the next step using SciKit-Learn utility images (X) and annotations (Y) are split into a train (80%), validation (10%) and test (10%) groups. The most important part is model creation. Each layer is added sequentially. Each model begins with different commonly known architecture followed by a flattening layer which resized multi-dimensional data into a single-dimensional array. Lastly, there are numerous dense layers with the number of neurons decreasing twice each time.

Later model is compiled and fitted to the training data to end with model evaluation on test data generating metrics and useful graphs.

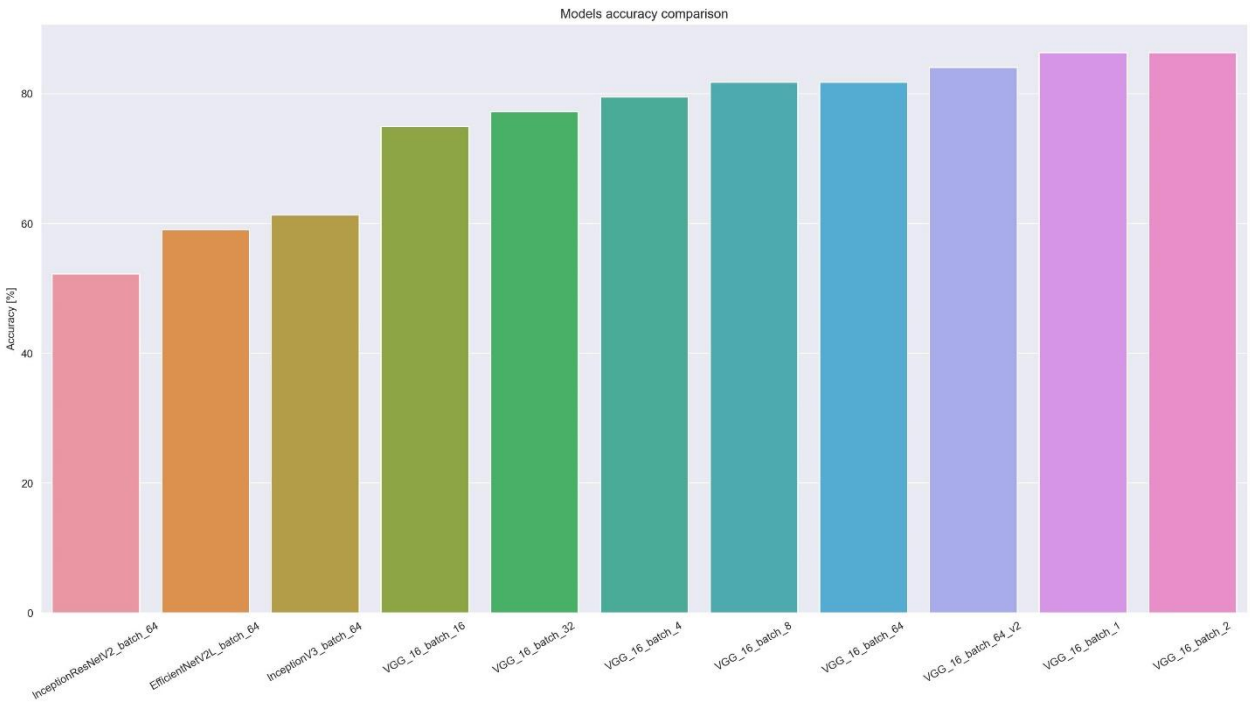
4. External software specification

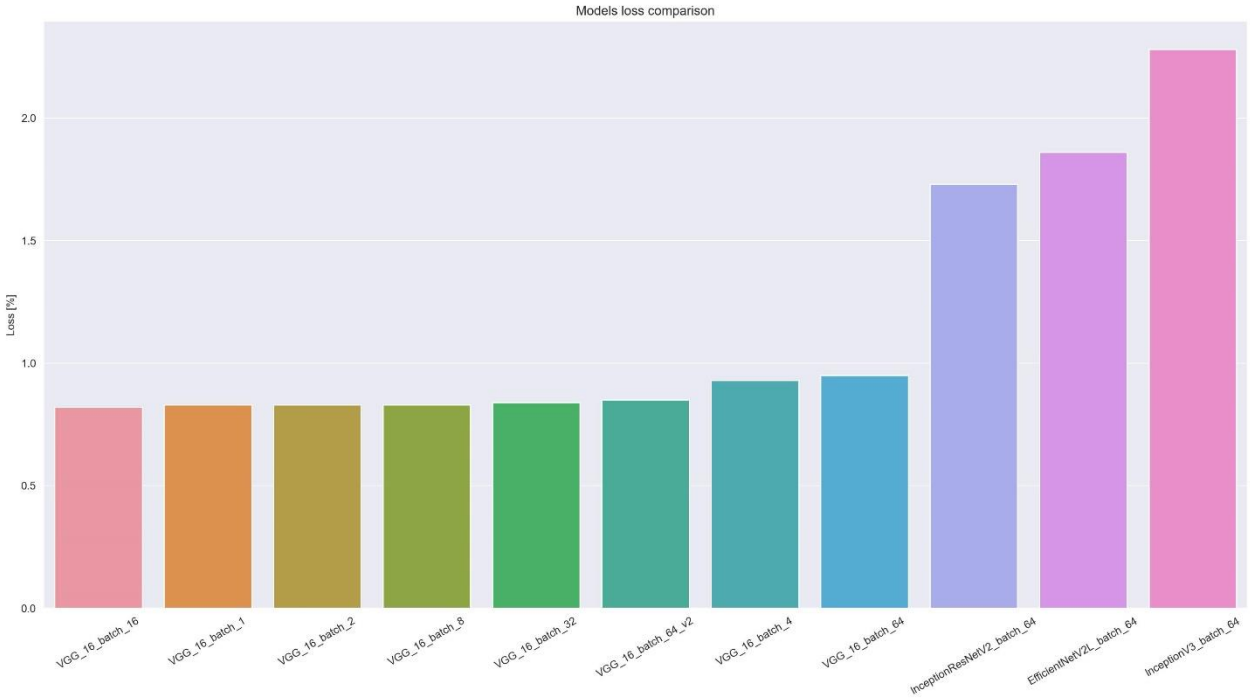
Each model is saved in the commonly used '.h5' format. It can be loaded easily to quickly predict never seen before images.

5. Experiments

a. Impact of different models

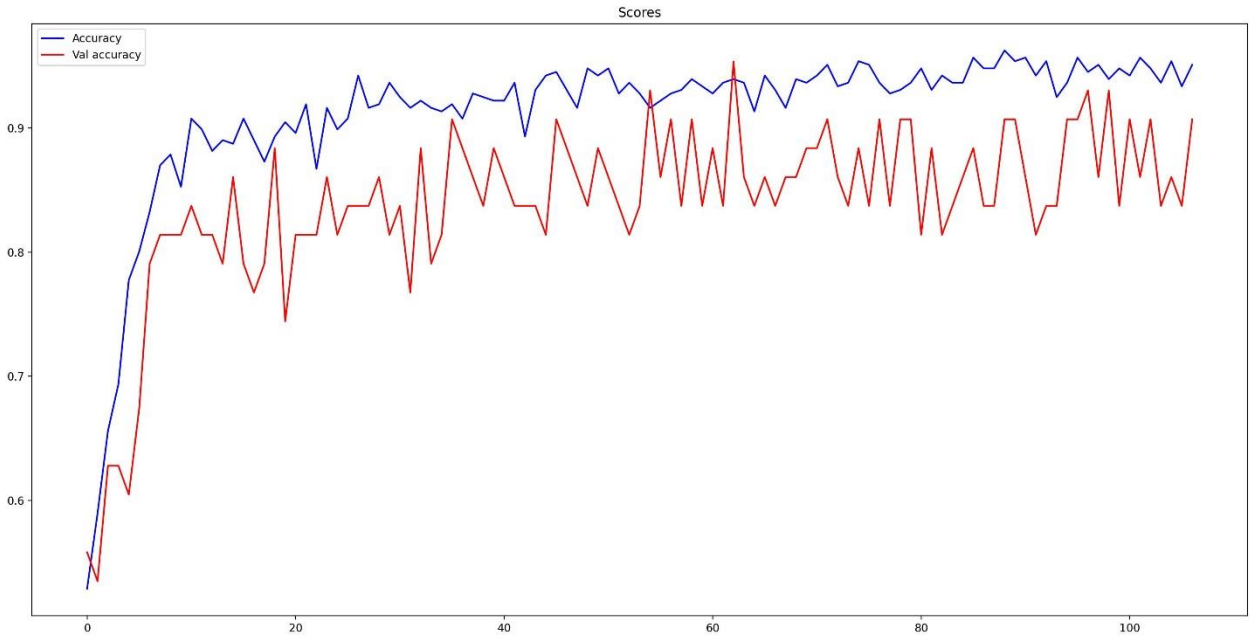
The thesis was put forward that the newer and more complex the architecture of the neural network the better the overall score would be but it occurred to be the total opposite. The best score was achieved using VGG16 reaching 88.36% accuracy with 0.83% loss using mean squared error and "Adam" optimizer. On the other side, there is InceptionResNetV2 achieving only 52.27% accuracy and 1.73% loss with the same metrics. The next thesis was put forward that the higher the batch size the higher the accuracy would be but there could not be seen any correlation.



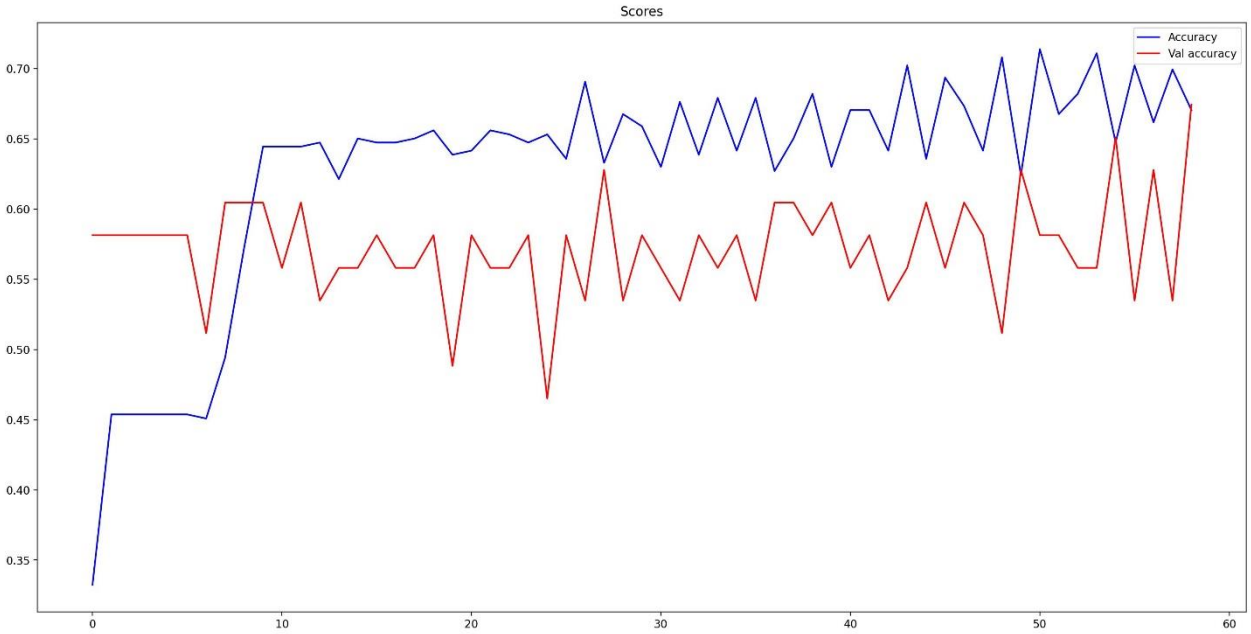


Below there are presented graphs of the 'Accuracy versus number of epochs' for each model architecture:

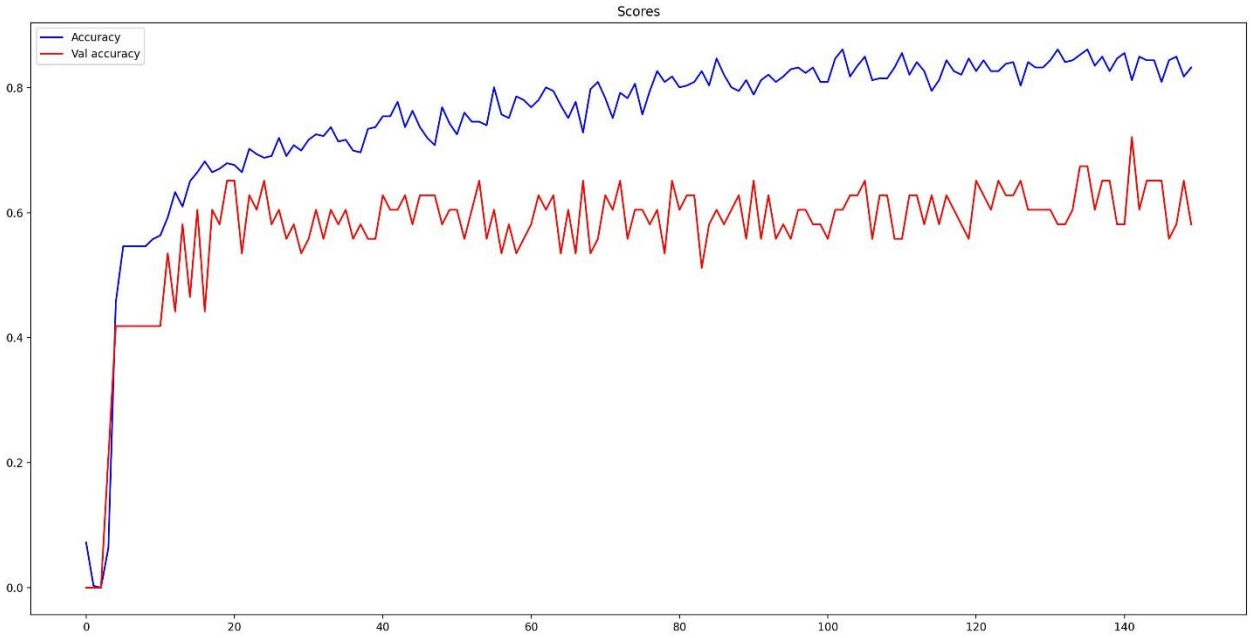
i. **VGG16:**



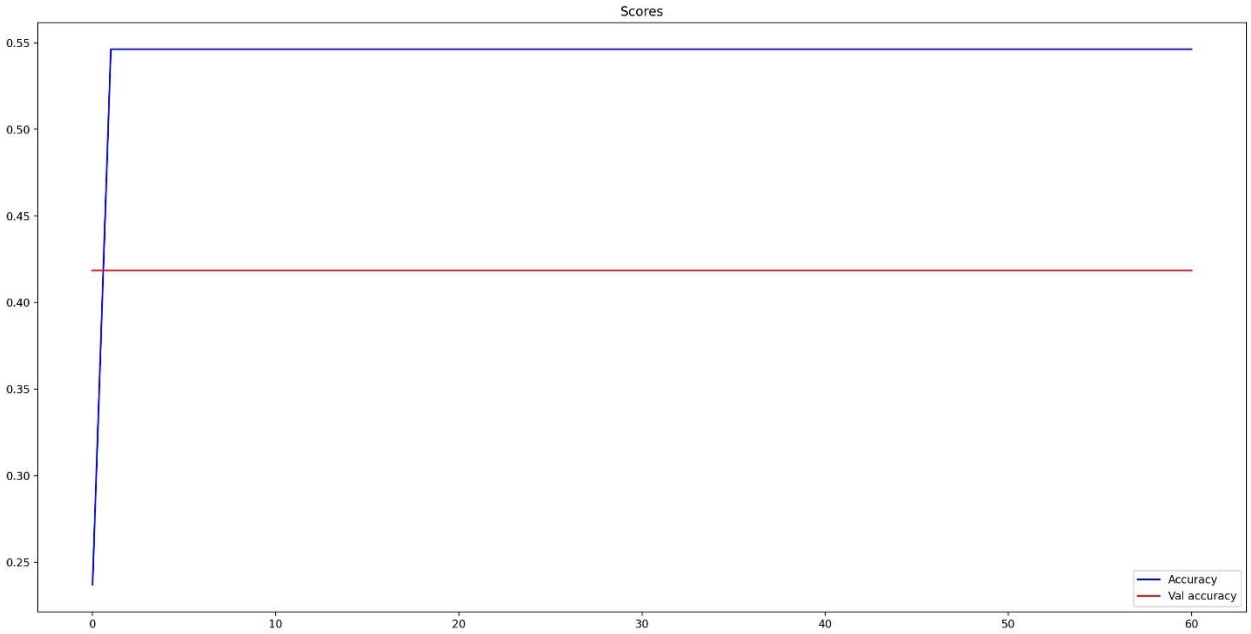
ii. InceptionResNetV2:



iii. InceptionV3:



iv. EfficientNetV2L:



b. Model predictions by each architecture:

i. VGG16:

ii. InceptionResNetV2:

iii. InceptionV3:

iv. EfficientNetV2L:

6. Summary

a. Overall conclusions

Our model works satisfactorily on images containing only one license plate within the image, especially those plates that cover most of the area of the picture. Unfortunately, the model was not trained to detect multiple car number plates as the dataset assumed detection of only one plate per image. The most challenging task for this model is detecting plates that are seen from a high angle vertically or horizontally. Nonetheless, we are truly satisfied with the achieved results.

b. Possible improvements

When thinking of implementing such a model in real life in our country, some things have to be considered. First of all, each image could be taken with the same camera and aspect ratio allowing us to skip resizing part. Second of all, results could be immensely improved by increasing the resolution to FHD minimum so that plates in distance could also be detected. What is more, each region of the world has a different style of its plates so it would be great if the dataset would contain mostly pictures of our country as the current dataset's diversity makes it nearly impossible to correctly detect them all. Last but not least, pictures for the dataset could be taken right in the place where it would have to work in the future like a dashcam so the model could be easily prepared for future use.

7. References

a. **Dataset:**
https://www.kaggle.com/datasets/andrewmvd/car-plate-detection?datasetId=686454

b. **Keras Tensorflow available models:**
https://keras.io/api/applications/,
https://keras.io/api/applications/#usage-examples-for-image-classification-models,

c. **Great source of information about Computer Vision:**
https://viso.ai/blog/

d. **YouTube tutorials:**
https://www.youtube.com/watch?v=yqkISICHH-U,
https://www.youtube.com/watch?v=0-4p_QgrdbE,
https://www.youtube.com/watch?v=COlbP62-B-U&list=PLQVvvaa0QuDcNK5GeCQnxYnSSaar2tpku&index=1,

e. **Custom object detection:**
https://neptune.ai/blog/how-to-train-your-own-object-detector-using-tensorflow-object-detection-api,
https://towardsdatascience.com/custom-object-detection-using-tensorflow-from-scratch-e61da2e10087, https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html,

f. **ML vs AI:**
https://www.jcchouinard.com/machine-learning/

g. **Udemy courses:**
https://www.udemy.com/course/python-for-machine-learning-data-science-masterclass/,
https://www.udemy.com/course/complete-tensorflow-2-and-keras-deep-learning-bootcamp/,
https://www.udemy.com/course/python-for-computer-vision-with-opencv-and-deep-learning/,

8. Repository

https://github.com/Kubson900/biai_car_plate_detection