Comparison of neural networks for on-camera debris prediction

Goal definition

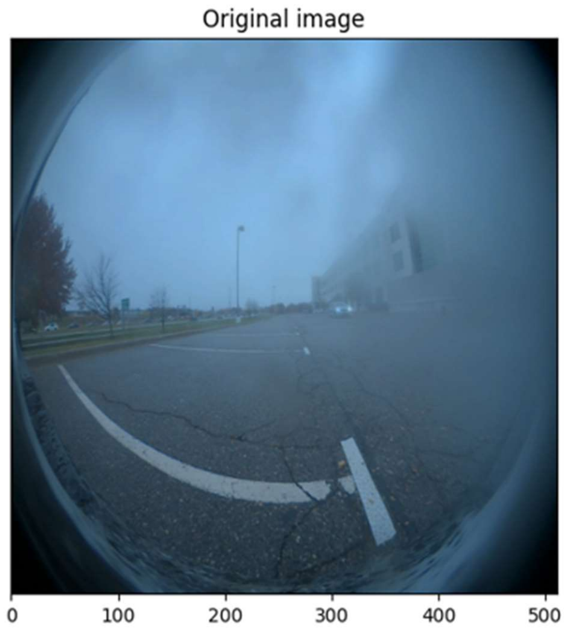
There is a dataset with images from the front cameras of cars. For each image from this dataset, an annotation is created that describes 4 states of soiling:

- Clear view – black color
- Transparent – green color
- Semi-transparent – red color
- Opaque – blue color

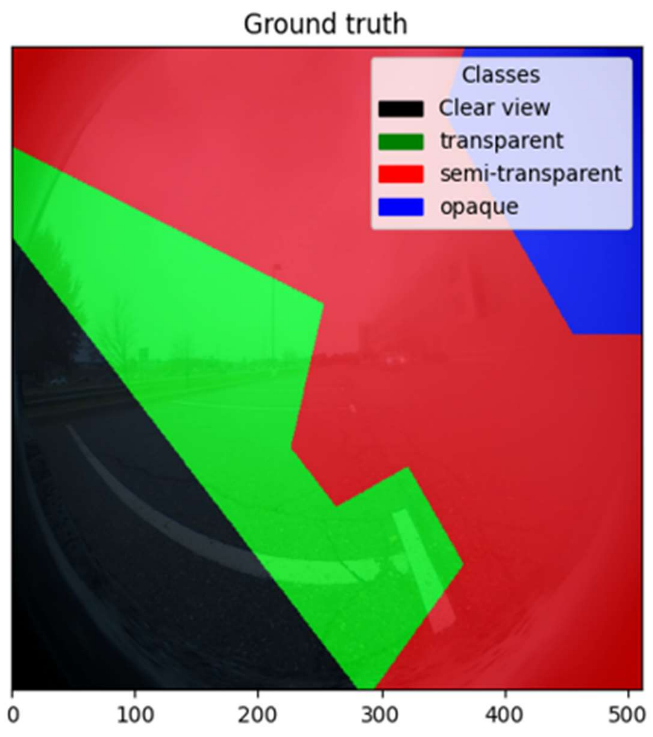
So there is a pair of images. An RGB input image that contains the scene from the camera and a pixel mask image that contains the classification of each pixel in the input image with a defined class.

The goal is to train a neural network and find out which is the best fitting one. The task is important with regard to other systems that can be affected by the state of contamination of the cameras and it is necessary to investigate their functionality in relation to Contamination.

Input image



Annotation mask



Input data a preprocessing

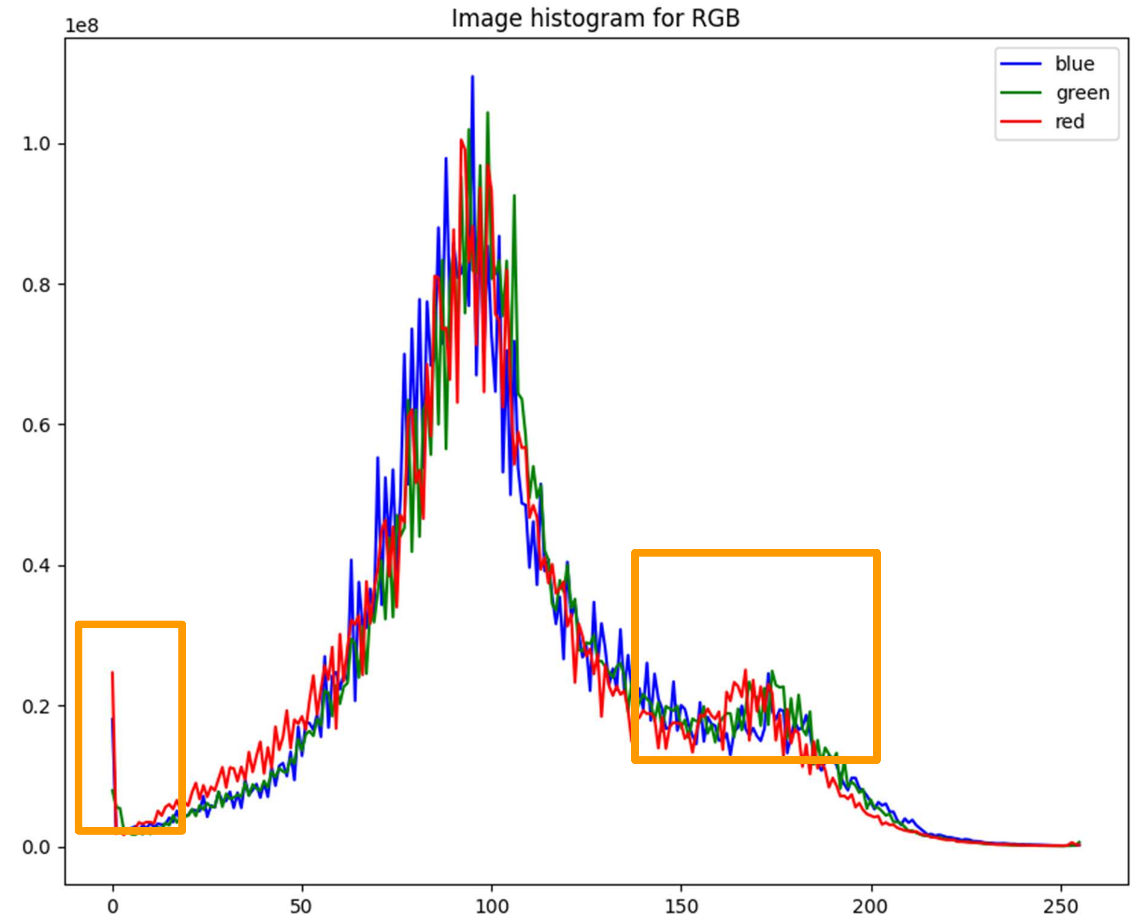
The input dataset used for training and evaluation is a publicly available valeo dataset.
https://woodscape.valeo.com/woodscape/download
The dataset contains a total of 20014 RGB images and the same number of pixel masks for these images
For further processing, the dataset is divided into the following 3 categories with their numbers.

- Train dataset = 14 776
- Validation dataset = 2 606
- Test dataset = 2 631

This dataset contains different scenes such as city, open roads etc.
During the analysis of the dataset, I discovered that some annotations are not consistent within the annotation rules, or are interpreted as errors in the annotation. It is a purely human characteristic, when each person assesses the annotations of some parts according to his discretion. This problem is the subject of an article that is currently being written. Overall, the dataset includes:

- 1 262 image that have an obvious error in the annotation
- 535 images that are inconsistent with annotation rules and other annotations

For all images, the accumulated histogram looks like following:



A peak around the zero values can be seen here, which is caused by the presence of the camera bezel. **The second peak is in the lighter spectrum of colors, but there is no explanation for it, since the impurities are of a dark nature.**

First, we calculate the histograms per image. We calculate the mean and standard deviation from the histograms. Next, we will calculate basic statistics for averages and standard deviation per image.

	blue_mean	blue_std	green_mean	green_std	red_mean	red_std	grayscale_mean	grayscale_std
mean	**105.8814**	**39.6399**	**107.5292**	**38.1279**	**103.3189**	**38.0135**	**106.0925**	**38.0199**
std	**14.3587**	**7.7880**	**14.9728**	**6.5971**	**16.4718**	**7.1554**	**15.1665**	**6.5676**
min	**48.6913**	**16.7193**	**44.3809**	**13.9988**	**27.5820**	**12.6785**	**40.3561**	**13.5606**
25%	**96.5394**	**35.0331**	**98.3342**	**34.3465**	**95.1396**	**34.0081**	**97.2120**	**34.4079**
50%	**104.3773**	**41.2560**	**106.1619**	**39.4079**	**102.6029**	**38.3725**	**104.8295**	**39.1964**
75%	**113.5052**	**44.7239**	**116.1708**	**42.5610**	**111.8251**	**42.1585**	**114.6817**	**42.4948**
max	**169.2605**	**62.6496**	**155.3433**	**50.9674**	**153.8751**	**54.9167**	**154.5322**	**50.7842**

Colors have fairly similar base stats. The single blue color is represented in a richer shade and has a larger standard deviation.

The used dataset of images is still normalized at the input using "Min max scaling". This helps to eliminate "exploiting" and "vanishing" the gradient in the gradient step on the neural network parameters.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

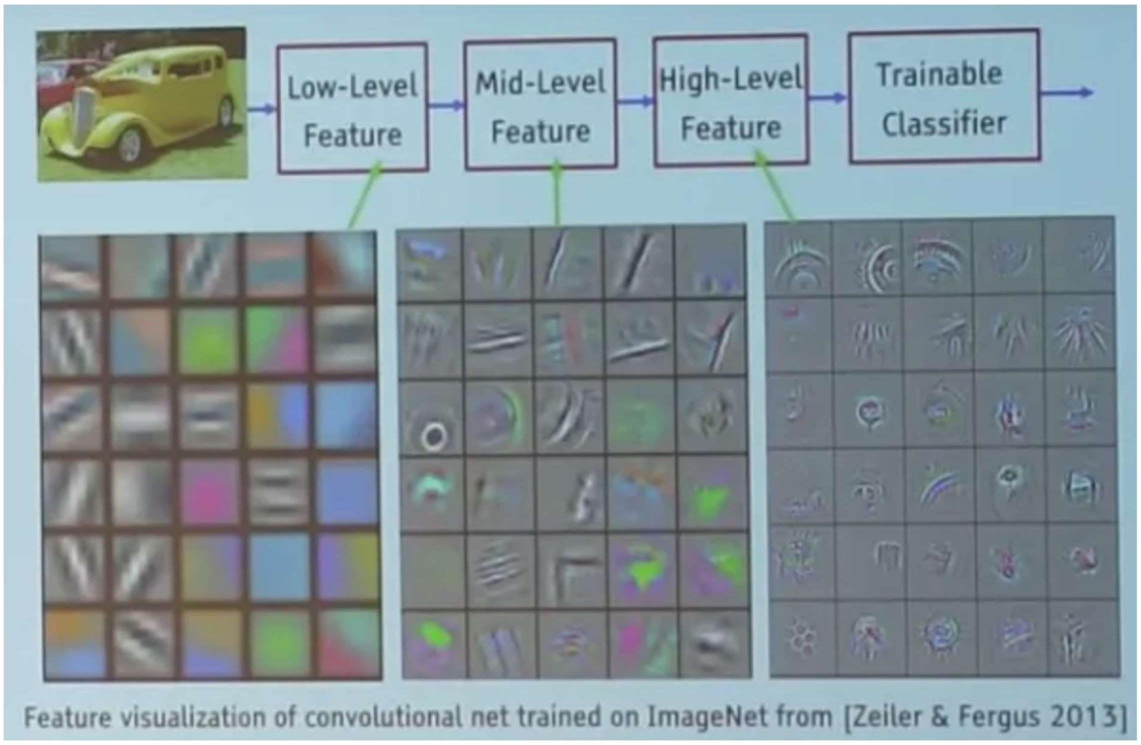
The minimum and maximum values are calculated from the training dataset to maintain consistency for the network parameters during prediction.

Used neural networks

Since the whole project was conceived as an experiment with the aim of capturing possible correlations and discovering new information for the application for similar tasks, when annotations are influenced by people and it is a very subjective assessment and at the same time the annotation is influenced by the dynamics of objects and the dynamics of light, several networks were used.

Feature extraction vs. new training

Some of the networks presented here were trained from scratch, others used "Feature extraction" methods. To explain "Feature extraction", it is good to mention how convolutions decompose the image in the architecture of neural networks.


Low-Level Feature
Mid-Level Feature
High-Level Feature
Trainable Classifier
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

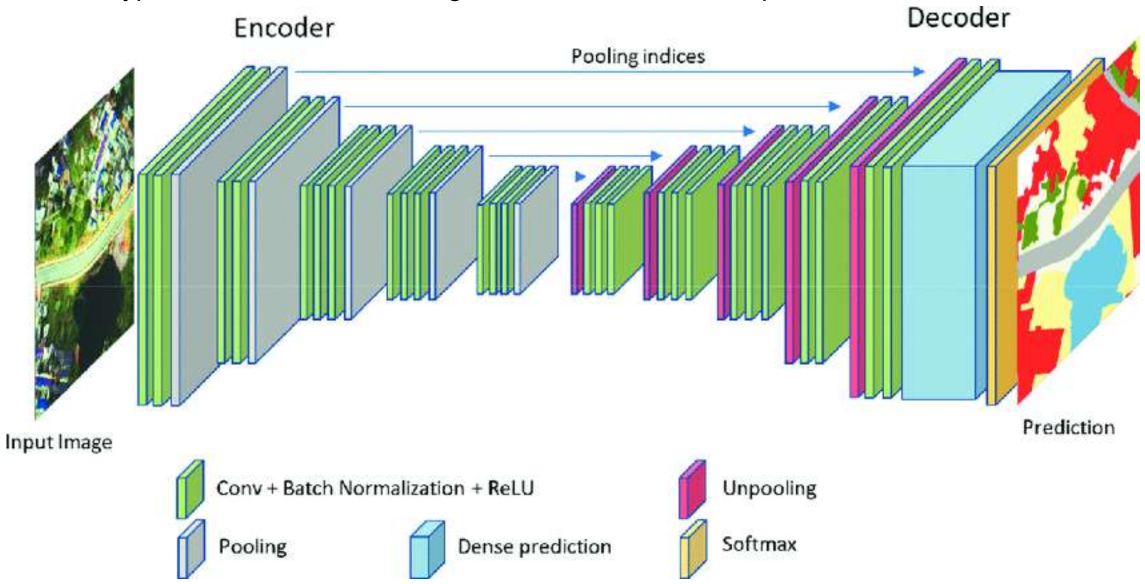
https://medium.com/@leicao.me/how-deep-learning-neural-networks-extracts-features-277244bcf66e

First, basic elements of the image such as vertical lines, horizontal lines, colors, etc. are extracted through primary convolutions. It is only at higher levels that elements specific to the given class are extracted. It is therefore possible to take a model trained on a large dataset such as "ImageNet", or others. Cut off the higher layers of the neural network, including the classifier, and replace them with new layers. A newly created network with parameters at lower levels can be quickly and efficiently trained for new purposes. *https://medium.com/@leicao.me/how-deep-learning-neural-networks-extracts-features-277244bcf66e*

The difference is that compared to new training, the network is already pre-trained on a large dataset to extract basic features, and training for specific classes is thus very fast and accurate. In the case of training from scratch, we need to balance the dataset for each class, perform many adjustments and above all have a large and robust dataset, which is not always possible.

Segnet (Resnet50) a Segnet (VGG)

The first type of network used is "Segnet". The architecture is pictured below.



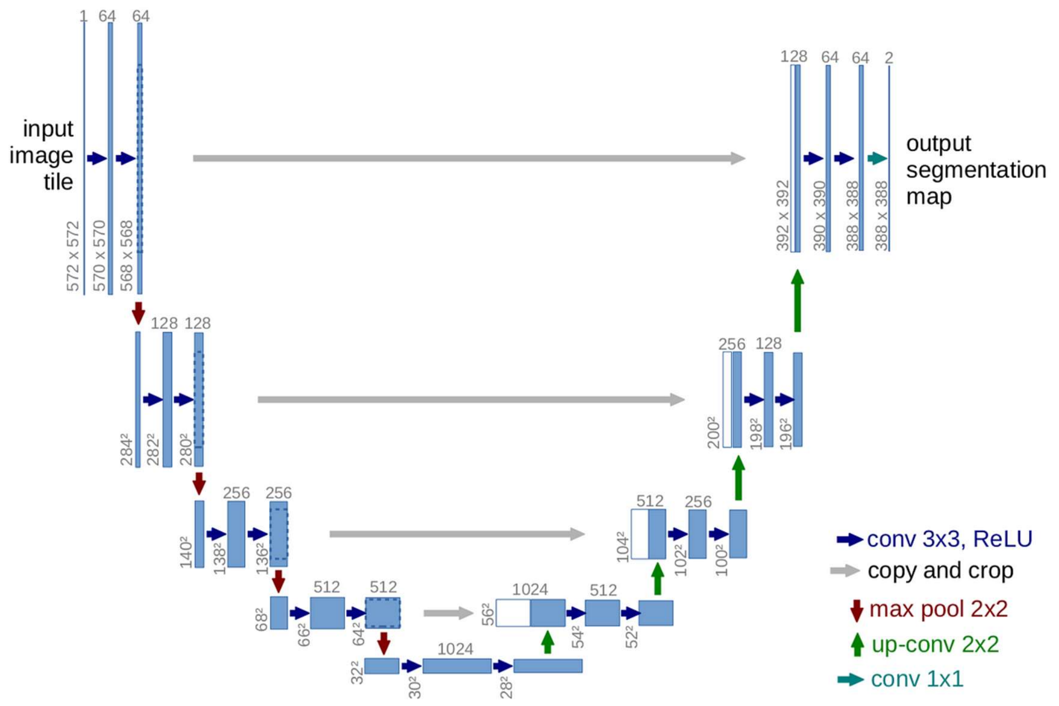
https://www.researchgate.net/figure/SegNet-architecture_fig5_343566178

As you can see, this is a very simple type of network. Through convolution, information is encoded into parameters, and further through upsampling, a pixel map is created. The network is very simple and ideal for solving simple tasks.

Resnet50 and VGG for our purposes means that for downsampling already pre-trained specific networks named Resnet50 and VGG trained on ImageNet and cut off the last layers were used. These networks are further connected to the upsampling senet and are thus an improved version of the senet network.

Unet (Resnet50), Unet (VGG), Unet (Predefined), Unet (small)

The second type of network is Unet, whose architecture is shown in the figure below.



https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

The sizes of the convolutions are indicative only and can be modified, which is what happened. This type of network has one modification compared to the senet. There are skipped connections between the downsampling and upsampling layers, which helps avoid exploiting or vanishing gradients, and also preserves more information between downsampling and upsampling.

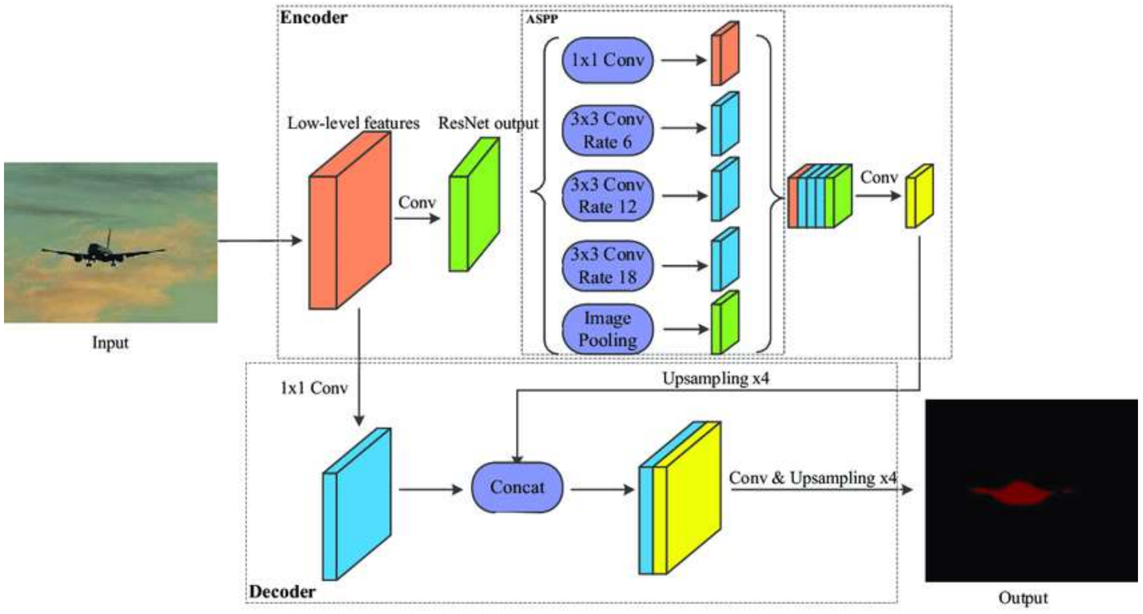
As with senet, Resnet50 and VGG for our purposes means that for downsampling already pre-trained specific networks named Resnet50 and VGG trained on ImageNet and cut off the last layers were used. These networks are further connected to the upsampling unet and are thus an improved version of the unet network.

Furthermore, there is a predefined, i.e. for our purposes, manually created copy of the architecture described in the image above. This network does not have a pre-trained model and is trained from scratch.

The last one is the small version, where the number of filters has been reduced by half and the last two layers of downsampling and upsampling are missing. This will be the simplest network. This network does not have a pre-trained model and is trained from scratch.

DeepLabV3

The last type of network is the most complex, viz. picture below.



https://www.researchgate.net/figure/Encoder-Decoder-architecture-of-DeepLab-v3_fig4_339754839

This network is by far the most complicated. It also uses resnet to extract basic features, but compared to segnet and unet, it has vertical extraction of higher features, which are inserted and stacked together, and then they are further stacked from features from the first layer and only then upsampling is applied to them. It's a mechanism that further prevents vanishinch and exploiting the gradient, but at the same time preserves a lot of information and captures finer details for the parameters. This network does not have a pre-trained model and is trained from scratch.

Training of the networks

The cross-entropy function method is used for training networks. Here is the cross-entropy function for binary classification.

$$\text{Loss} = -\frac{1}{\underset{\text{size}}{\text{output}}} \sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

https://androidkt.com/choose-cross-entropy-loss-function-in-keras/

But in this case there are a total of 4 classes. Thus, the sum in the equation will be expanded by the number of classes. The resulting equation looks like this:

$$\text{Cross Entropy} = -1/N * \sum_{i=1}^{N} \sum_{j=1}^{k} Y_{ij} * \text{Log } P_{ij}$$

where N is no of data points
k is no of classes

https://aboutdatascience.in/2022/11/04/cross-entropy-loss-function/

After calculating the loss, the gradient of the loss is also calculated. Subsequently, the gradient is combined with an optimization function. ADAM function was used here, which shows more stability and faster learning.

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$
$g_t : Gradient\ at\ time\ t\ along\ \omega^j$
$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$
$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$
$\beta_1, \beta_2 : Hyperparameters$

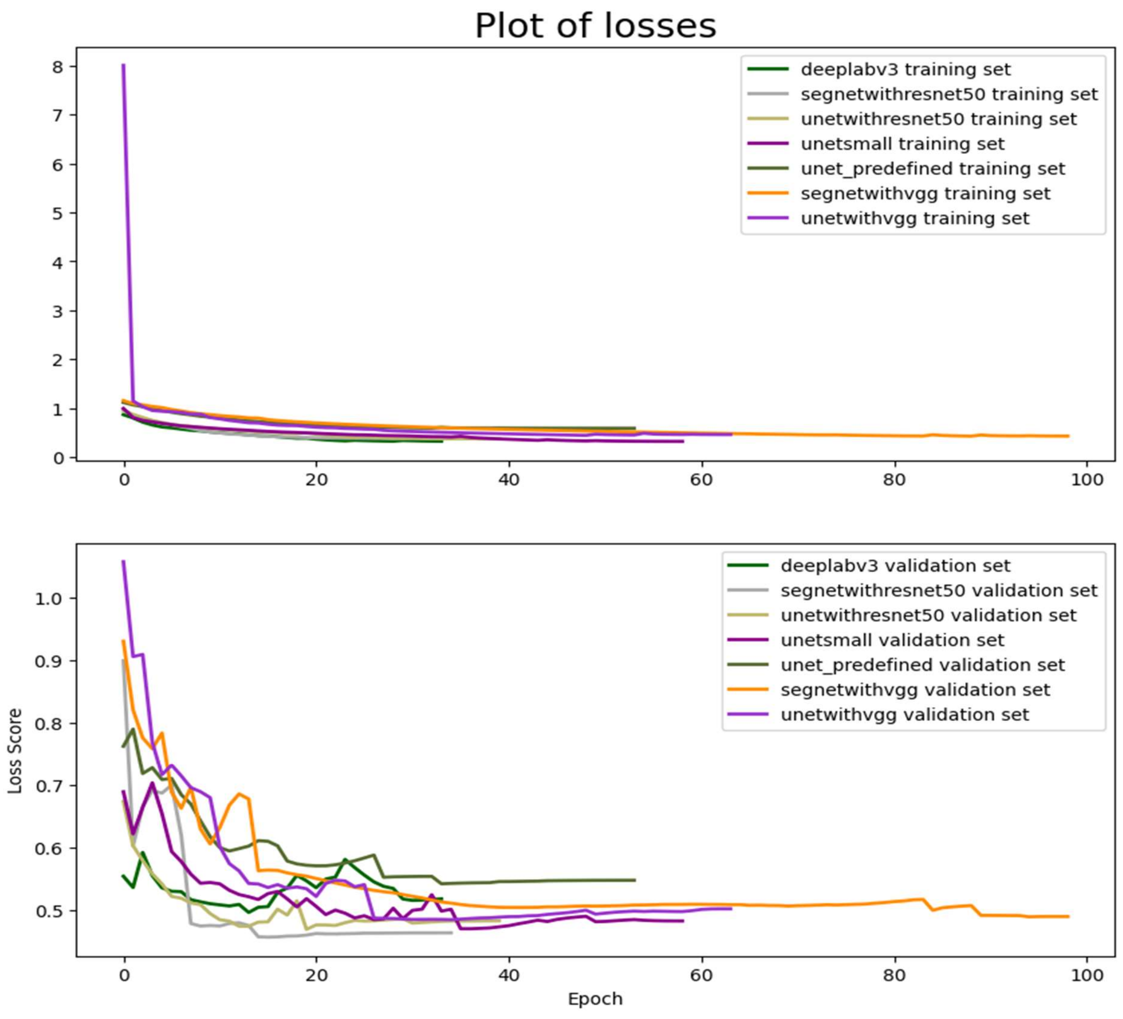
https://discuss.pytorch.org/t/how-adam-optimizer-influence-the-learning-rate/168543

The parameters for the optimization function were::

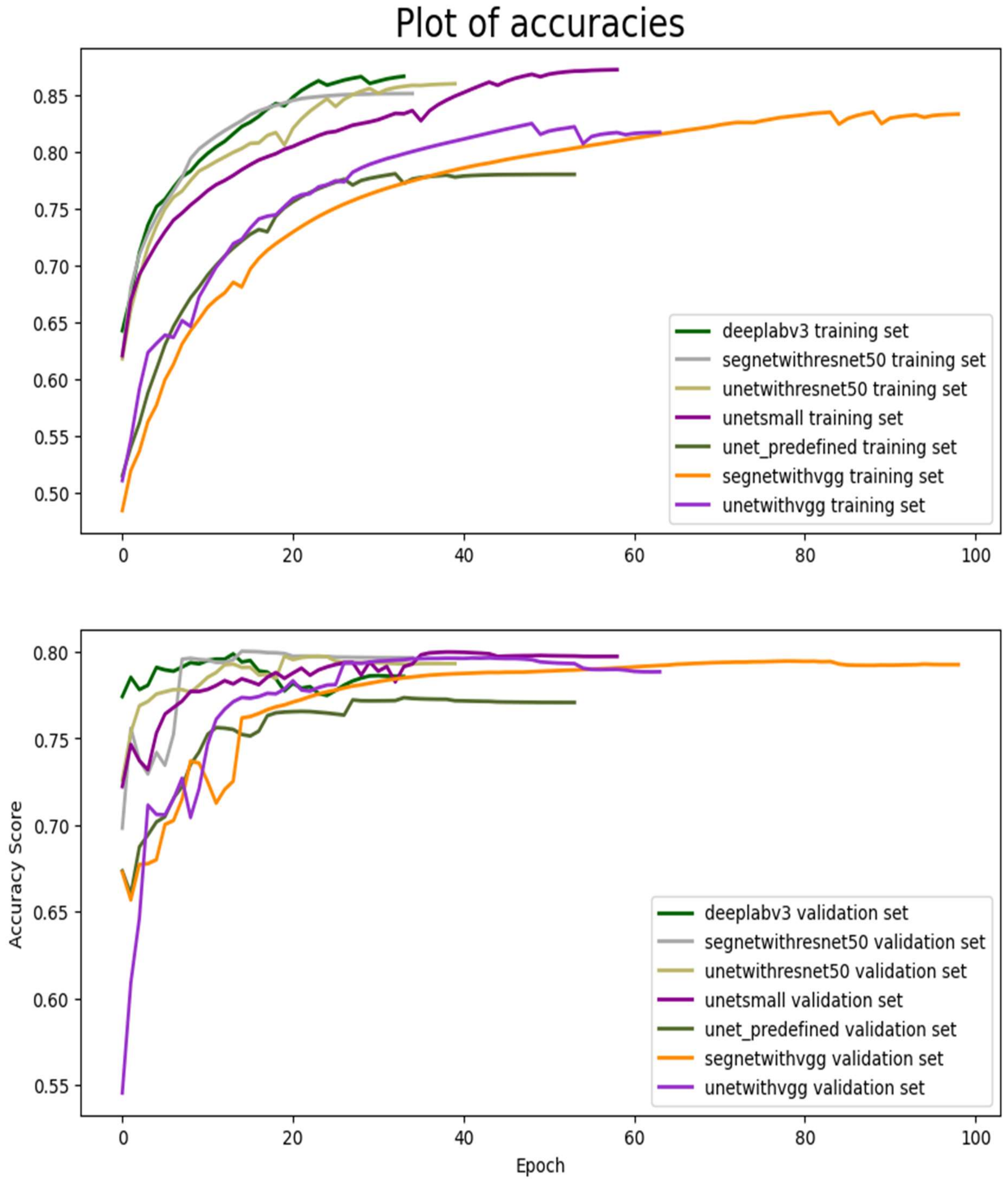
- Learning rate = 0.0001
- Beta1 = 0.9
- Beta2 = 0.999

- Epsilon = 1e-07
- Decreasing gradient by the ½ , if there is no loss reduction over 5 epochs
- Max number of epoch = 200
- Early stopping, unless there is no loss improvement on validation dataset over 10 epochs

The following graph shows the learning of all neural networks. The magnitudes of the loss functions on the training dataset above and the validation dataset below are shown. As can be seen, the networks are capable of quite fast adaptation on the training dataset. On the validation dataset, the networks behave slightly differently. Specifically, networks that use Resnet for downsampling learn patterns very quickly and then their validation loss remains almost the same.

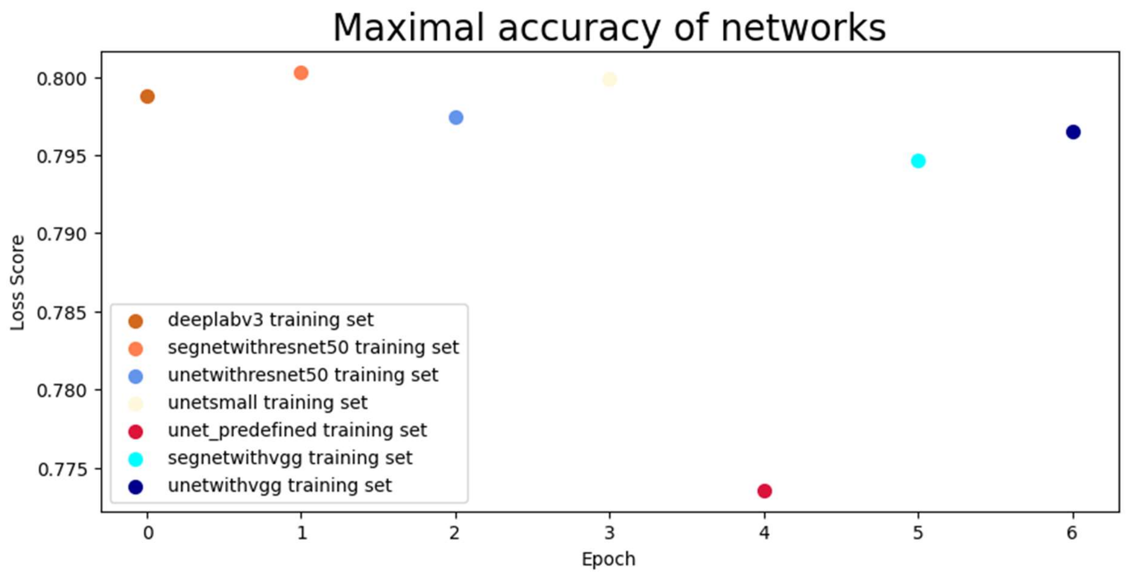


We observe a similar trend in the second metric that was calculated during training, which is accuracy.



At the end, a scatter plot with the maximum accuracy achieved on the validation dataset is attached. Two points of interest can be noted.

1. All networks that use the implementation of downsampling with a pre-trained model achieve the highest results.
2. Unet in its full version achieves worse results than its counterpart with fewer filters and layers. A possible reason is that the dataset was not sufficient to train this network.



Results on test dataset

Finally, the neural networks were run on the test dataset. The following metrics were discussed prior to the experiment:

- Precision
    - $$\text{Precision}=\frac{TP}{TP+FP}$$

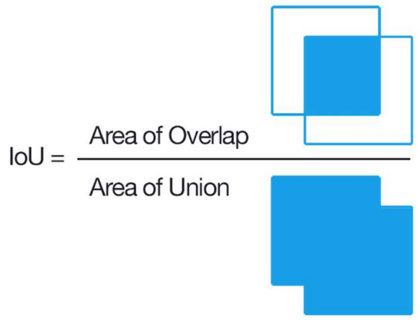
https://www.researchgate.net/figure/The-Formula-of-Precision-Recall-and-IoU_fig5_355981167

-
    - ○ Measures the classification accuracy of positive patterns if they are evaluated as positive

- Recall
    - $$\text{Recall}=\frac{TP}{TP+FN}$$

https://www.researchgate.net/figure/The-Formula-of-Precision-Recall-and-IoU_fig5_355981167

-
    - ○ Measures the accuracy of the model to evaluate positive samples as positive

- Intersection over Union
    - 

https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

    - Common area divided by the total area of positive classifications
- F1
    - $$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

https://inside-machinelearning.com/en/recall-precision-f1-score-simple-metric-explanation-machine-learning/

    - Measure trade-off between precision a recall
- Accuracy
    - ○ Describes the accuracy of the model to classify across classes when the classes are equal in weight to each other.
    - ○ Number of correctly classified pixels / total number of pixels

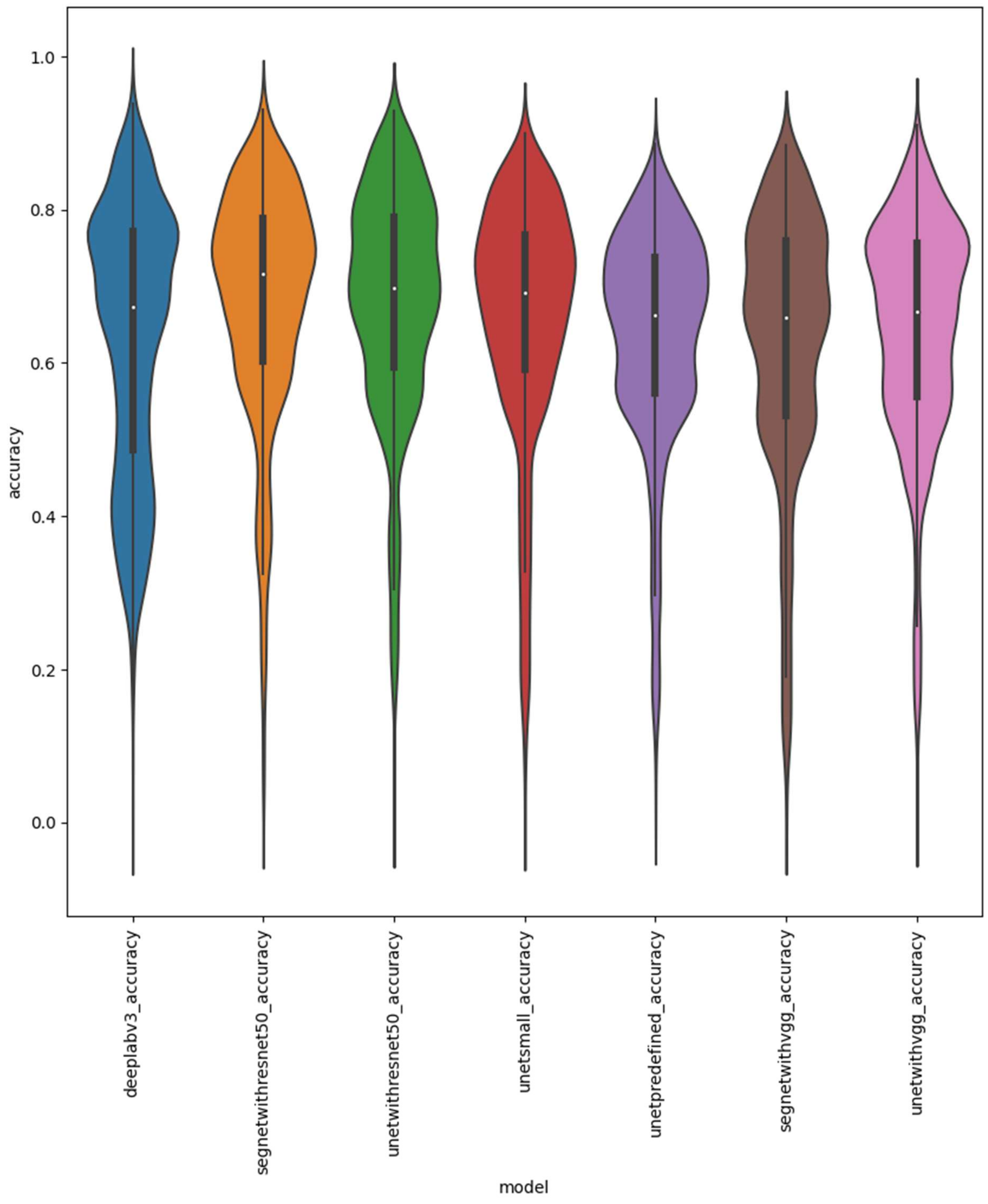
I have decided to focus on Accuracy and IoU as it fits our purposes.

Subsequently, the results were inserted into the table viz. below:

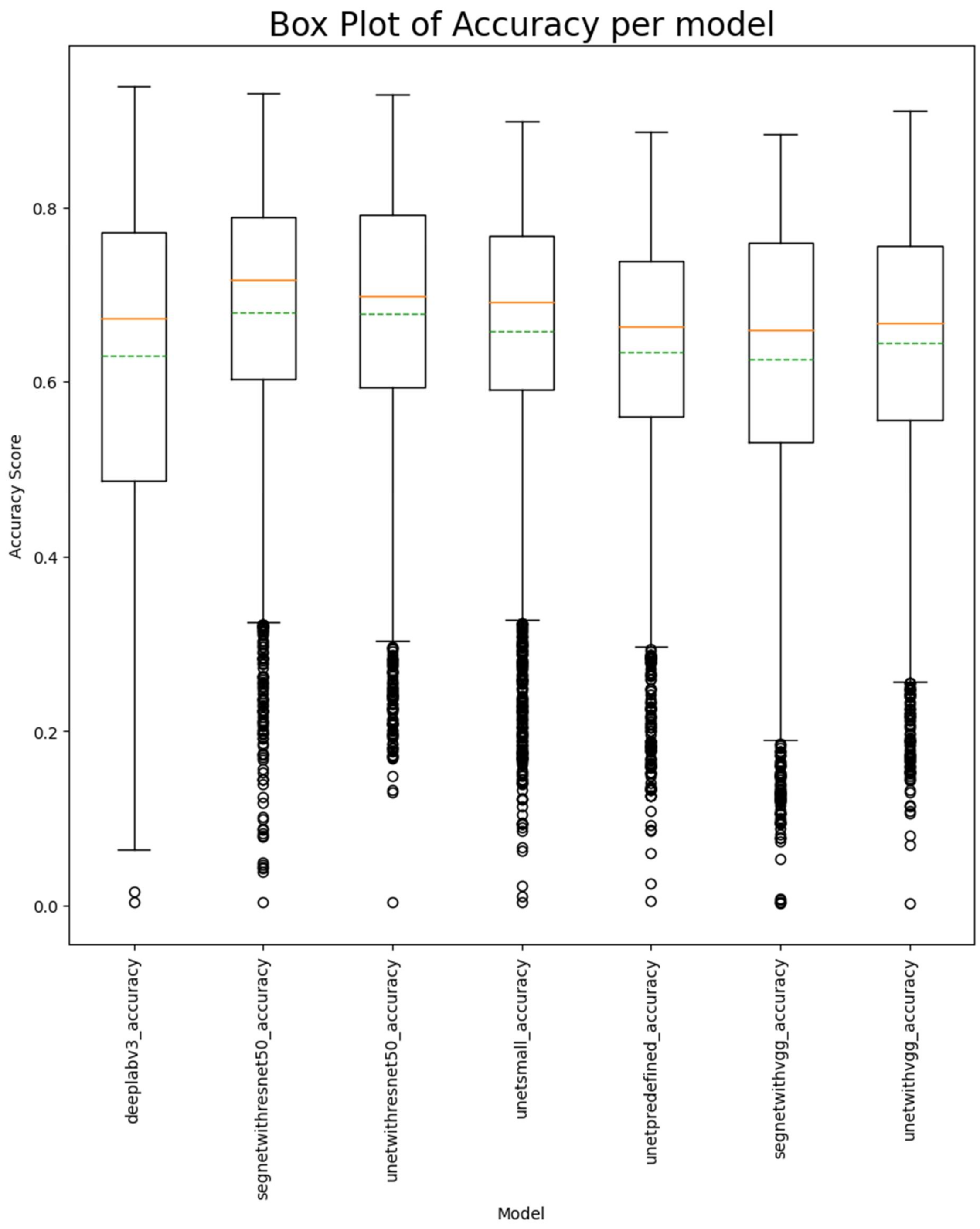
Model	IoU	Pixel accuracy
DeepLabV3	0.402	0.630
Segnet with Resnet50	**0.454**	**0.679**
Segnet with VGG	0.408	0.626
Unet small	0.438	0.657
Unet predefined	0.405	0.634
Unet with Resnet50	0.445	0.678
Unet with VGG	0.440	0.644

In turn, the results show that networks that use predefined networks, specifically Resnet, achieve better results. At the same time, the simplest network also achieves interesting results.

However, the results can also be visualized so that they display, for example, the distribution of accuracy, or display percentiles, etc. The first visualization is the violin, which displays both the distribution of the results, but also the median and interquartile (25%-75% distribution of all results).



It is possible to observe differences in the results, but at first glance they do not look very convincing. The black thick line is the interquartile and the white dot in the middle is the median.



When visualizing using a box plot. Wheels below the minimum indicate outliers. The green line indicates the median and the orange is the mean. The median and average can also be seen here at similar values.

Results comparison and testing

We will compare the results firstly by using confidence interval applied on mean both sided and then with several hypotheses tests.

Confidence interval of mean – both sided

We will want to determine with 95% confidence the interval of the mean values of the accuracy of the model. We will use the equation below for the calculation.

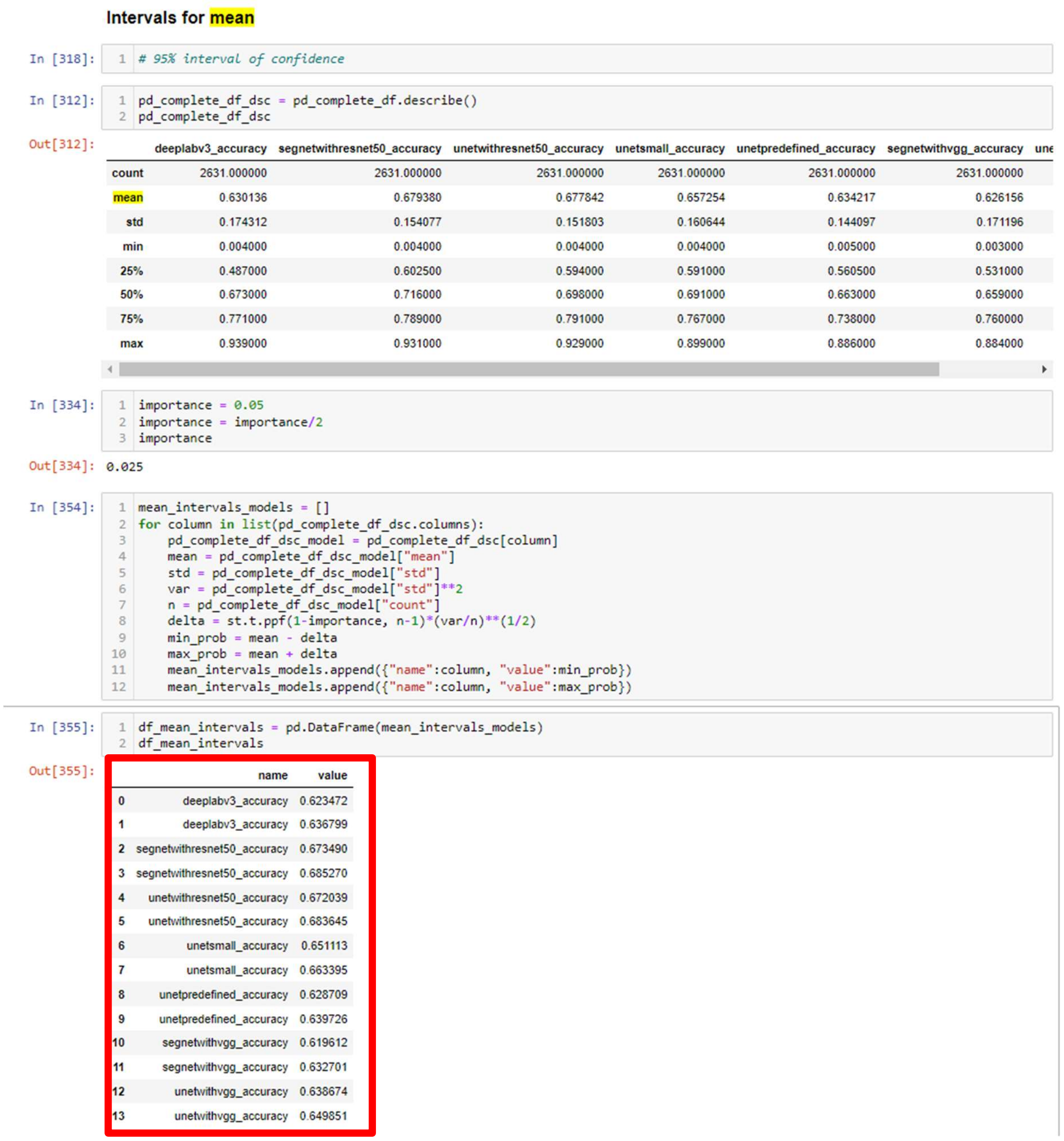
Confidence Interval Formula

$$\text{Confidence Interval} = \left(\overline{X} - Z \times \frac{\sigma}{\sqrt{n}}\right) \text{ to } \left(\overline{X} + Z \times \frac{\sigma}{\sqrt{n}}\right)$$

$$\text{Confidence Interval} = \overline{X} \pm Z \times \frac{\sigma}{\sqrt{n}}$$

https://www.educba.com/confidence-interval-formula/

Implementation is done in jupyter notebook using python as bellow:



Intervals for mean

In [318]:
```
# 95% interval of confidence
```

In [312]:
```
pd_complete_df_dsc = pd_complete_df.describe()
pd_complete_df_dsc
```

Out[312]:

	deeplabv3_accuracy	segnetwithresnet50_accuracy	unetwithresnet50_accuracy	unetsmall_accuracy	unetpredefined_accuracy	segnetwithvgg_accuracy	une
count	2631.000000	2631.000000	2631.000000	2631.000000	2631.000000	2631.000000	
mean	0.630136	0.679380	0.677842	0.657254	0.634217	0.626156	
std	0.174312	0.154077	0.151803	0.160644	0.144097	0.171196	
min	0.004000	0.004000	0.004000	0.004000	0.005000	0.003000	
25%	0.487000	0.602500	0.594000	0.591000	0.560500	0.531000	
50%	0.673000	0.716000	0.698000	0.691000	0.663000	0.659000	
75%	0.771000	0.789000	0.791000	0.767000	0.738000	0.760000	
max	0.939000	0.931000	0.929000	0.899000	0.886000	0.884000	

In [334]:
```
importance = 0.05
importance = importance/2
importance
```

Out[334]: 0.025

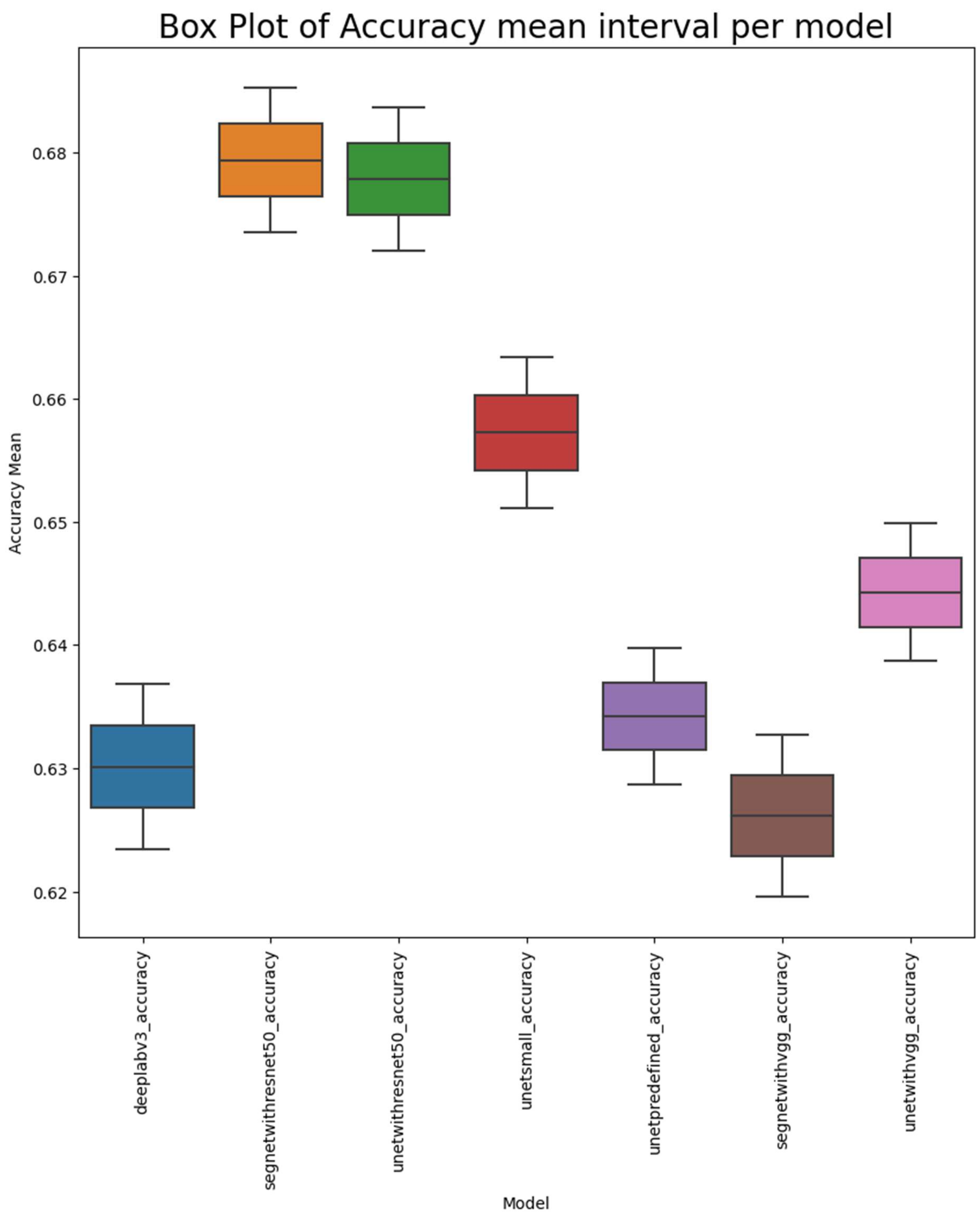
In [354]:
```
mean_intervals_models = []
for column in list(pd_complete_df_dsc.columns):
    pd_complete_df_dsc_model = pd_complete_df_dsc[column]
    mean = pd_complete_df_dsc_model["mean"]
    std = pd_complete_df_dsc_model["std"]
    var = pd_complete_df_dsc_model["std"]**2
    n = pd_complete_df_dsc_model["count"]
    delta = st.t.ppf(1-importance, n-1)*(var/n)**(1/2)
    min_prob = mean - delta
    max_prob = mean + delta
    mean_intervals_models.append({"name":column, "value":min_prob})
    mean_intervals_models.append({"name":column, "value":max_prob})
```

In [355]:
```
df_mean_intervals = pd.DataFrame(mean_intervals_models)
df_mean_intervals
```

Out[355]:

	name	value
0	deeplabv3_accuracy	0.623472
1	deeplabv3_accuracy	0.636799
2	segnetwithresnet50_accuracy	0.673490
3	segnetwithresnet50_accuracy	0.685270
4	unetwithresnet50_accuracy	0.672039
5	unetwithresnet50_accuracy	0.683645
6	unetsmall_accuracy	0.651113
7	unetsmall_accuracy	0.663395
8	unetpredefined_accuracy	0.628709
9	unetpredefined_accuracy	0.639726
10	segnetwithvgg_accuracy	0.619612
11	segnetwithvgg_accuracy	0.632701
12	unetwithvgg_accuracy	0.638674
13	unetwithvgg_accuracy	0.649851

The result is a table where we have the minimum and maximum interval for the average for each network. We put these values in a box plot and we can see the results.



We can say with 95% confidence that both segnet and unet using resnet for downsampling achieve the highest results.

We would like to confirm results with by hypothesis testing.

Hypothesis Testing – All networks are having equal resutls

The first hypothesis we want to test is that all networks are the same, since the medians and means are similar with a significance level of 5%.

H0 = All networks have the same accuracy
H1 = At least one network has differential accuracy
Significance level = 0.05

We will use a one-factor anova where we only consider the accuracy of predictions per image.

$$SS_B = \sum_k (\bar{x}_k - \bar{x})^2$$
$$SS_w = \sum_{i,k} (\bar{x}_{i,k} - \bar{x}_k)^2$$

$$MS_B = \frac{SS_B}{k-1}$$

$$MS_w = \frac{SS_w}{n-k}$$

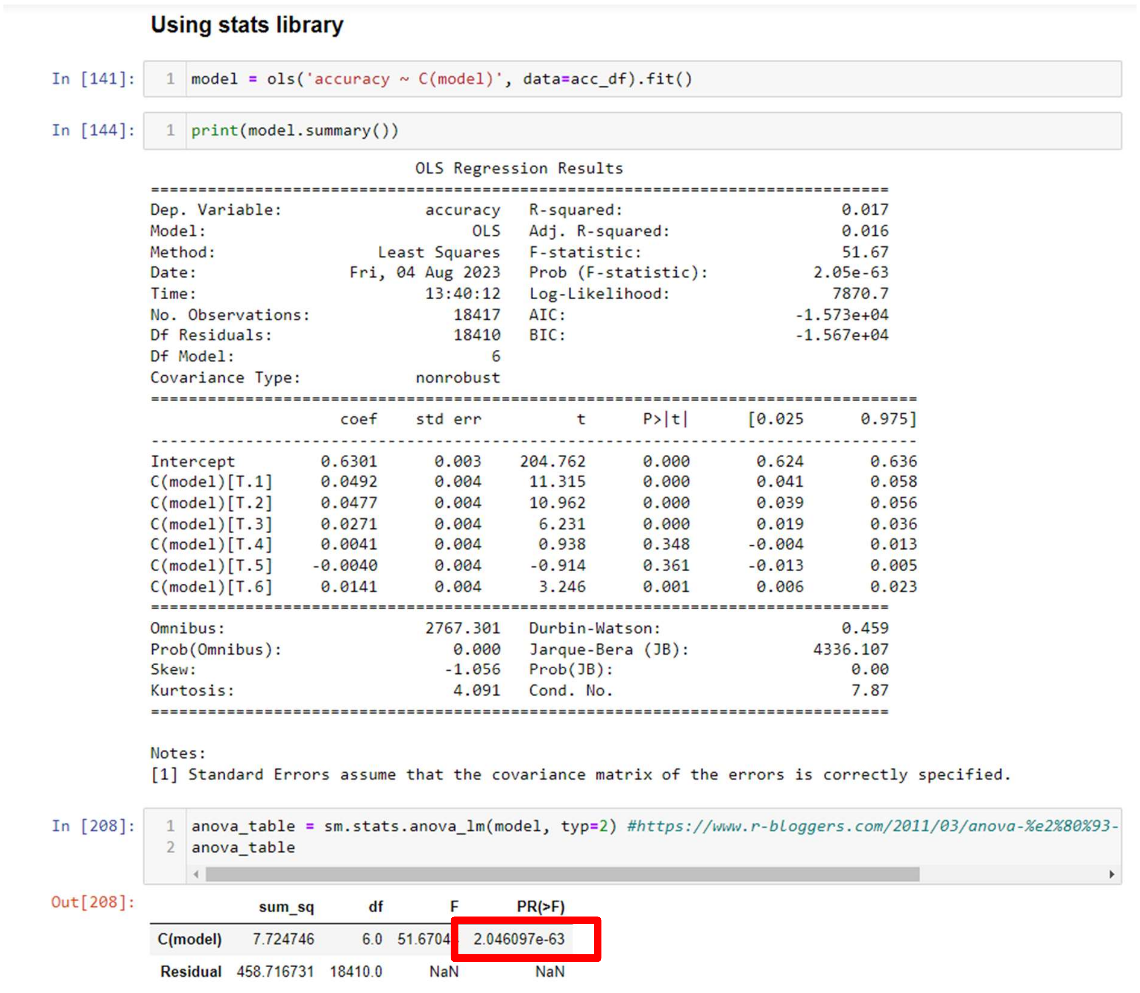
$$F - statistic = \frac{MS_B}{MS_w}$$

K *is the number of groups*
n *is the total number of observations in all groups*

https://towardsdatascience.com/anova-explained-for-beginners-with-the-bachelorette-tv-show-8503c4aaba10

MSb - mean square between groups -> Intergroup product of squares
MSw - mean square within group -> Intragroup product of squares
MSt - mean square total = MSb + MSt

For our purposes, we performed the implementation both manually and through the stats model library, which enables accelerated calculation, through jupyter notebook.



Using stats library

```
In [141]: model = ols('accuracy ~ C(model)', data=acc_df).fit()
```

```
In [144]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               accuracy   R-squared:                       0.017
Model:                            OLS   Adj. R-squared:                  0.016
Method:                 Least Squares   F-statistic:                     51.67
Date:                Fri, 04 Aug 2023   Prob (F-statistic):           2.05e-63
Time:                        13:40:12   Log-Likelihood:                 7870.7
No. Observations:               18417   AIC:                        -1.573e+04
Df Residuals:                   18410   BIC:                        -1.567e+04
Df Model:                           6
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
Intercept         0.6301      0.003    204.762      0.000       0.624       0.636
C(model)[T.1]     0.0492      0.004     11.315      0.000       0.041       0.058
C(model)[T.2]     0.0477      0.004     10.962      0.000       0.039       0.056
C(model)[T.3]     0.0271      0.004      6.231      0.000       0.019       0.036
C(model)[T.4]     0.0041      0.004      0.938      0.348      -0.004       0.013
C(model)[T.5]    -0.0040      0.004     -0.914      0.361      -0.013       0.005
C(model)[T.6]     0.0141      0.004      3.246      0.001       0.006       0.023
==============================================================================
Omnibus:                     2767.301   Durbin-Watson:                   0.459
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             4336.107
Skew:                          -1.056   Prob(JB):                         0.00
Kurtosis:                       4.091   Cond. No.                         7.87
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
In [208]: anova_table = sm.stats.anova_lm(model, typ=2) #https://www.r-bloggers.com/2011/03/anova-%e2%80%93-
          anova_table
```

Out[208]:

	sum_sq	df	F	PR(>F)
C(model)	7.724746	6.0	51.6704	2.046097e-63
Residual	458.716731	18410.0	NaN	NaN

The result is: 2.046097e-63

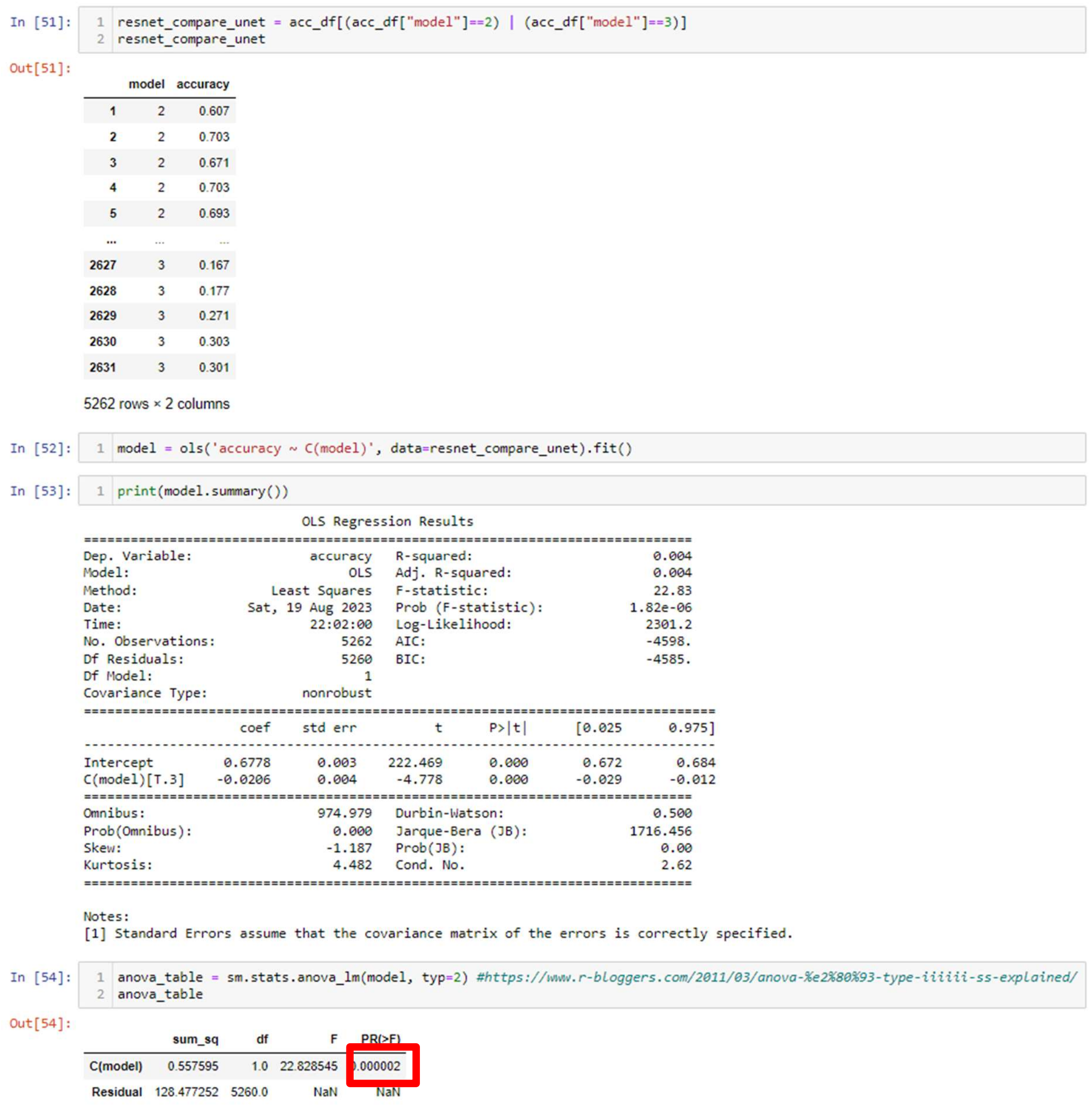
Since PR is below the significance level of 0.05, we can reject H0 and H1 is valid.

Hypothesis testing – Second best network of resnet is having similar results as third network

Another hypothesis we want to test is that resnets are equal to the second best network, which is unet small. We will perform the test with a significance level of 5%.

H0 = Resnet networks have the same accuracy as unet small
H1 = Networks differ in accuracy
Significance level = 0.05

We will use a single factor anova, where we only consider the accuracy of predictions per image.



```
In [51]: resnet_compare_unet = acc_df[(acc_df["model"]==2) | (acc_df["model"]==3)]
         resnet_compare_unet
```

Out[51]:

	model	accuracy
1	2	0.607
2	2	0.703
3	2	0.671
4	2	0.703
5	2	0.693
...	...	...
2627	3	0.167
2628	3	0.177
2629	3	0.271
2630	3	0.303
2631	3	0.301

5262 rows × 2 columns

```
In [52]: model = ols('accuracy ~ C(model)', data=resnet_compare_unet).fit()
```

```
In [53]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               accuracy   R-squared:                       0.004
Model:                            OLS   Adj. R-squared:                  0.004
Method:                 Least Squares   F-statistic:                     22.83
Date:                Sat, 19 Aug 2023   Prob (F-statistic):           1.82e-06
Time:                        22:02:00   Log-Likelihood:                 2301.2
No. Observations:                5262   AIC:                            -4598.
Df Residuals:                    5260   BIC:                            -4585.
Df Model:                           1
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
Intercept         0.6778      0.003    222.469      0.000       0.672       0.684
C(model)[T.3]    -0.0206      0.004     -4.778      0.000      -0.029      -0.012
==============================================================================
Omnibus:                      974.979   Durbin-Watson:                   0.500
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1716.456
Skew:                          -1.187   Prob(JB):                         0.00
Kurtosis:                       4.482   Cond. No.                         2.62
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
In [54]: anova_table = sm.stats.anova_lm(model, typ=2) #https://www.r-bloggers.com/2011/03/anova-%e2%80%93-type-iiiiii-ss-explained/
         anova_table
```

Out[54]:

	sum_sq	df	F	PR(>F)
C(model)	0.557595	1.0	22.828545	0.000002
Residual	128.477252	5260.0	NaN	NaN

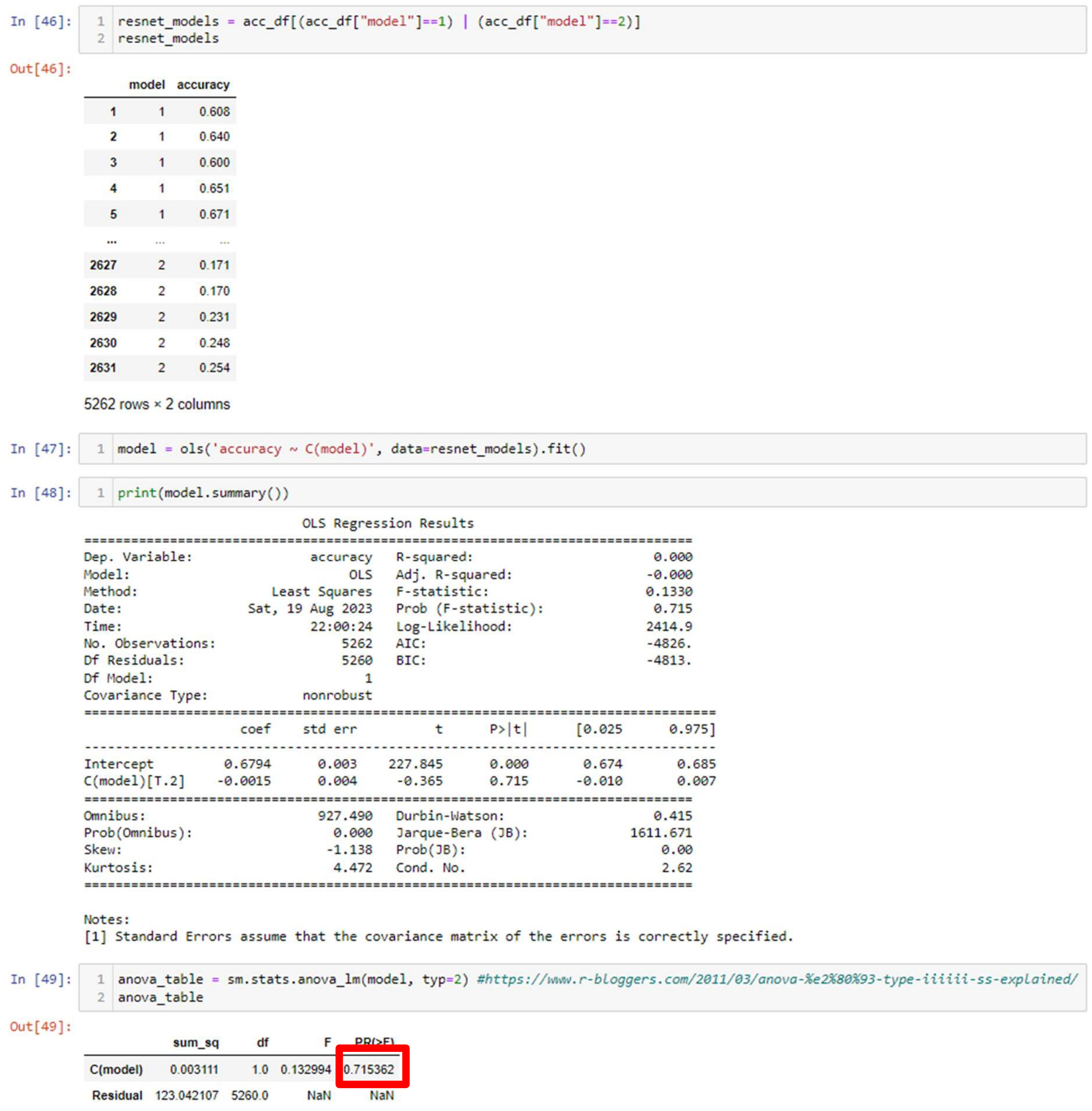
The P value is equal to 0.000002, so we reject the null hypothesis at the level of significance and accept H1, which says that the resnet network and the unet small network, i.e. other networks with a worse result, are different.

Hypothesis testing – Resnet networks are having similar results

Another hypothesis we want to test is that the resnet networks are the same, as the medians and means are similar with a significance level of 5%.

H0 = Resnet networks have the same accuracy
H1 = Networks differ in accuracy
Significance level = 0.05

We will use a one-factor anova where we only consider the accuracy of predictions per image.



```
In [46]: resnet_models = acc_df[(acc_df["model"]==1) | (acc_df["model"]==2)]
         resnet_models
Out[46]:
```

	model	accuracy
1	1	0.608
2	1	0.640
3	1	0.600
4	1	0.651
5	1	0.671
...	...	...
2627	2	0.171
2628	2	0.170
2629	2	0.231
2630	2	0.248
2631	2	0.254

5262 rows × 2 columns

```
In [47]: model = ols('accuracy ~ C(model)', data=resnet_models).fit()
```

```
In [48]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               accuracy   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                 -0.000
Method:                 Least Squares   F-statistic:                    0.1330
Date:                Sat, 19 Aug 2023   Prob (F-statistic):              0.715
Time:                        22:00:24   Log-Likelihood:                 2414.9
No. Observations:                5262   AIC:                            -4826.
Df Residuals:                    5260   BIC:                            -4813.
Df Model:                           1
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
Intercept         0.6794      0.003    227.845      0.000       0.674       0.685
C(model)[T.2]    -0.0015      0.004     -0.365      0.715      -0.010       0.007
==============================================================================
Omnibus:                      927.490   Durbin-Watson:                   0.415
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1611.671
Skew:                          -1.138   Prob(JB):                         0.00
Kurtosis:                       4.472   Cond. No.                         2.62
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
In [49]: anova_table = sm.stats.anova_lm(model, typ=2) #https://www.r-bloggers.com/2011/03/anova-%e2%80%93-type-iiiii-ss-explained/
         anova_table
Out[49]:
```

	sum_sq	df	F	PR(>F)
C(model)	0.003111	1.0	0.132994	0.715362
Residual	123.042107	5260.0	NaN	NaN

The P value in this case is 0.72, so we accept the null hypothesis and with 95% confidence the networks achieve the same accuracy.

Conclusion

The goal was to train a neural network, find out which is the best and use it for predictions.

During training, it was found that networks that use pre-trained networks for downsampling tend to learn faster.

Furthermore, during testing, we evaluated which networks perform the best. With 95% confidence, we can state that networks that use pre-trained resnet provide the highest reliability. It is thus possible to choose either senet with resnet or unet with resnet from the network.

Attachements:

Image analysis: Analýza obrázků.pdf

Network training analysis: Analýza učení sítí.pdf

Confidence intervals and ANOVA implementation: Anova with intervals.pdf