Steganography analysis

ILLIA KOSTENKO

1. What is steganography?

Steganography is the art of science of embedding secret messages in cover message in such a way that no one, a part from the sender and intended recipient suspects the existence of the message.

Steganography is a Greek origin which means:

Steganos – covered,

Grafia - writing

What files can be used for the steganography?

What files can be used for the steganography:

- BMP files (.bmp)
- JPEG files (.jpg)
- GIF files (.gif)
- WAV/ MP3 files (.wav/ .mp3)
- Video files (.avi/ .mpg)
- Executable files (.exe)
- And other format types can be used

Steganography VS Cryptography

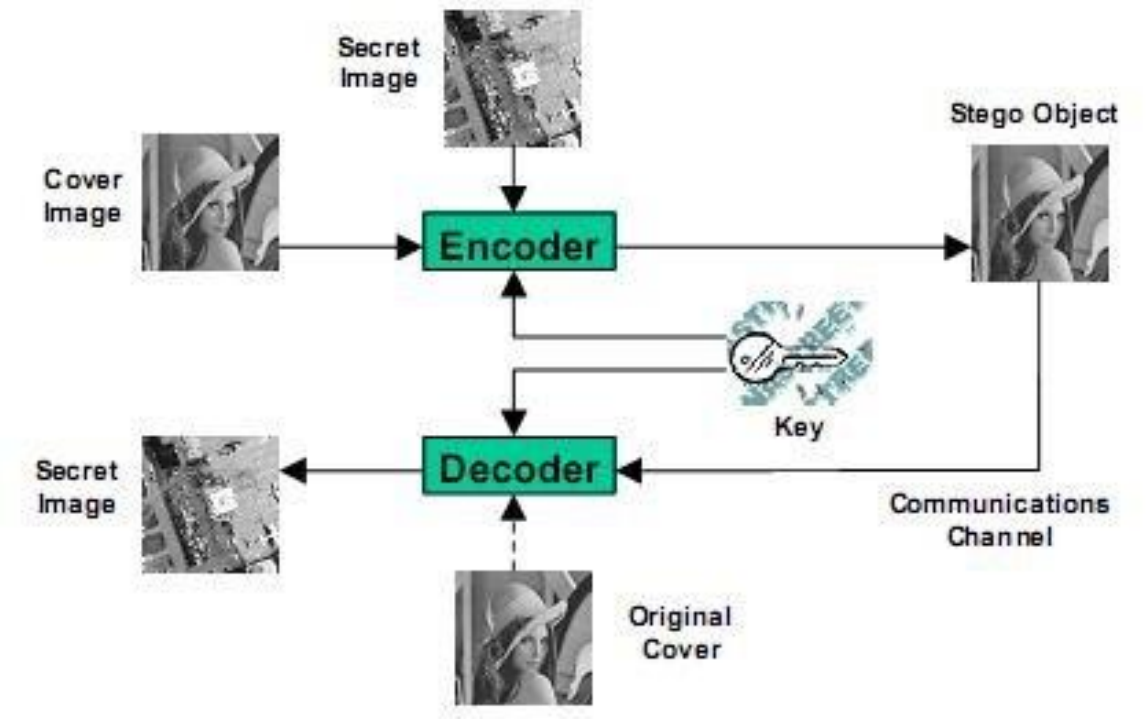
Stegano

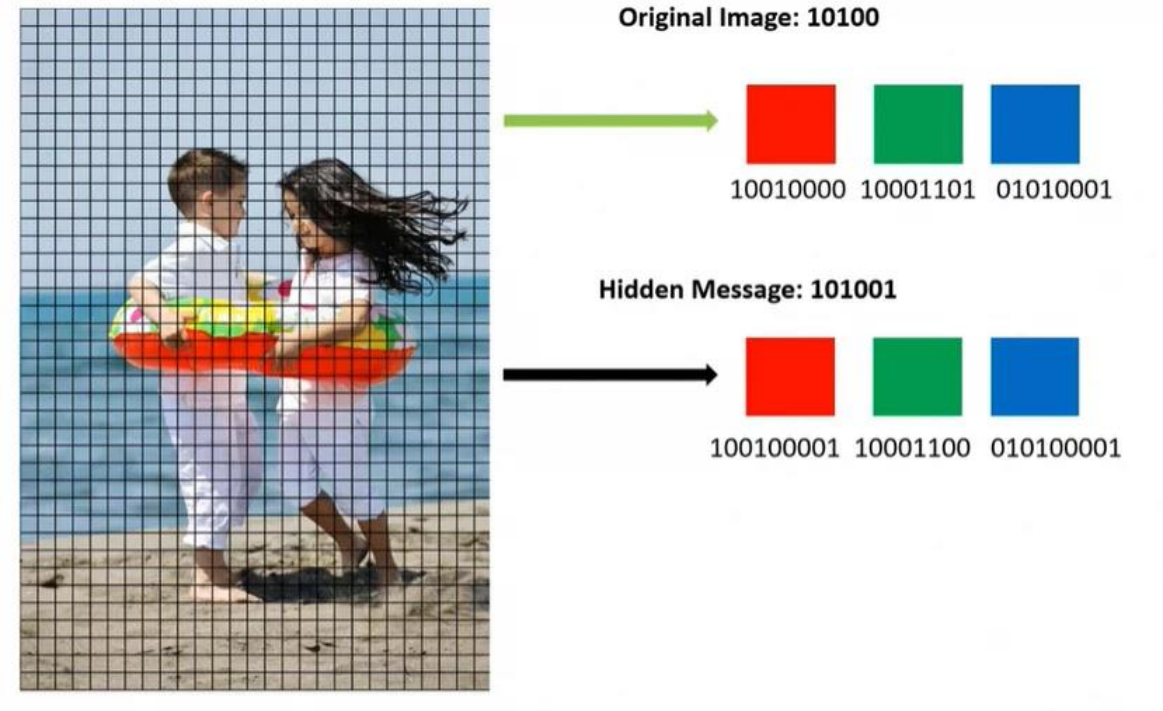
- Conceals the existence of the message
- Does not alter the structure of secret messages, but hides inside the cover file, so it cannot be seen
- Used to protect the information

Crypto

- Hiding the contents of secret messages from malicious people
- Structure of message is scrambled to make it meaningless unless the decryption key is provided
- Used to protect the information

How the steganography works?





What is steganalalysis?

- Main aim of steganalysis is identifying the existence of message, but not extracting the message
- Technically, Steganography deals with the concealment of a message, not the encryption of it.
- By identifying the existence of a message, perhaps we can identify the tool to hide it.
- If we identify the tool, we can use it to decode the message.

Discrete cosine transform

- A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies.

$$DCT(P^b)_{kl} = \sum_{i,j=0}^{7} \frac{w_k w_l}{4} cos \frac{\pi k(2i+1)}{16} \times cos \frac{\pi l(2j+1)}{16} P_{ij}^{(b)}$$

where k,l ∈ {0,…,7} index the DCT mode and $w_0 = \frac{1}{\sqrt{2}}$, $w_k = 1$ for k > 0.

To obtain an actual JPEG image from a two-dimensional array of quantized coefficients **X** (cover) or **Y** (stego), it is required to create an (arbitrary) JPEG image of the same dimensions $n_1 \times n_2$ and then merely replace the array of quantized coefficients in this structure with **X** and **Y** to obtain the cover and stego images, respectively.

Steganalysis algorithms

- Main idea of steganography algorithms lies inside of distortion function. The distortion depends on the choice of a directional filter bank and one scalar parameter whose purpose is stabilizing the numerical computations.
- By a directional filter bank, we understand a set of three linear shift-invariant filters represented with their kernels $\beta=\{K^{(1)}, K^{(2)}, K^{(3)}\}$. They are used to evaluate the smoothness of a given image X along the horizontal, vertical, and diagonal directions by computing the so-called directional residuals $W^{(k)} = K^{(k)} \star \mathrm{X}$, where ' $\star$' is a mirror-padded convolution so that $W^{(k)}$ has again n1 × n2 elements. The mirror padding prevents introducing embedding artifacts at the image boundary.

UNIWARD distortion function (UNIversal Wavelet Relative Distortion)

$$D(X,Y) = \sum_{k=1}^{3}\sum_{u=1}^{n}\sum_{v=1}^{m}\frac{|W_{uv}^{(k)}(X) - W_{uv}^{(k)}(Y)|}{\sigma + |W_{uv}^{(k)}(X)|}$$

where σ>0 is a constant used to stabilize the numerical calculations.

The distortion function of J-UNIWARD needs to filter images three times to get HL,LH and HH subbands, and then combine three subband coefficients to calculate the embedding distortion for each element. Besides, the size of filter kernel is relatively large. Therefore, the time complexity of distortion calculation for large size image is unacceptable. The embedding distortion of J-UNIWARD is computed as a sum of relative changes of coefficients in a directional filter bank decomposition of the decompressed cover image.

- LL is the approximate image of input image it is low frequency subband so it is used for further decomposition process.
- LH subband extract the horizontal features of original image.
- HL subband gives vertical features .
- HH subband gives diagonal features .

Source: Defining Cost Functions for Adaptive JPEG Steganography at the Microscale

What is MiPOD algorithm

- As in the UNIWARD algorithm, distortion function ρ associates to each pixel the cost of modifying it. More formally, for a given cover X, let D(X) be the matrix whose elements represent the cost of increasing or decreasing by 1 the corresponding pixels. By ranking pixels according to their value in D(X), one can compute the set of pixels whose modification induces the smallest detectability.
- In MiPOD the distortion function D is obtained thanks to a probabilistic approach. More precisely, let β be the matrix defined as the probabilities to increase by 1 the image pixels. The objective of such a scheme is then to find probabilities which minimize a deflection coefficient - $\sum \sigma^{-4}\beta^2$.

$$D(X) = \ln(\frac{1}{\beta} - 2)$$

What is MiPOD algorithm

- As in the UNIWARD algorithm, distortion function ρ associates to each pixel the cost of modifying it. More formally, for a given cover X, let D(X) be the matrix whose elements represent the cost of increasing or decreasing by 1 the corresponding pixels. By ranking pixels according to their value in D(X), one can compute the set of pixels whose modification induces the smallest detectability.
- In MiPOD the distortion function D is obtained thanks to a probabilistic approach. More precisely, let β be the matrix defined as the probabilities to increase by 1 the image pixels. The objective of such a scheme is then to find probabilities which minimize a deflection coefficient - $\sum \sigma^{-4}\beta^{2}$.

$$D(X) = \ln(\frac{1}{\beta} - 2)$$

UERD algorithm - Uniform Embedding Revisited Distortion

- Followed by the concept in spirit of "spread spectrum communication", UED (Uniform Embedding Distortion) and UERD (Uniform Embedding Revisited Distortion) with low complexity uniformly spread the embedding modifications to DCT coefficients of all possible magnitudes.
- Though J-UNIWARD achieves state-of-the-art performance, it has high computational complexity. A lightweight distortion (UERD) for JPEG steganography was proposed. When it comes to the IF in UERD, quantization table with an adjustment on the DC quantization step is adopted to distinguish the impact of different positions in a block. The weighted DCT energy is defined as TD, reflecting the texture of the block and its neighbours.

UERD algorithm – Uniform Embedding Revisited Distortion

- And the DCT energy is formulated as follows:

$$D = \sum_{k=0}^{7}\sum_{l=0}^{7}|x_{kl}| * q_{kl}, k, l \in \{0, \dots, 7\}$$

where x_{kl} is DCT coefficients in the mnth block, x_{00} = 0 to avoid the influence of DC coefficient, and q_{kl} is the quantization step. Then the distortion of UERD is defined as

$$p_{mn}^{(k,l)} = \begin{cases} \dfrac{0.5 * \left(q_{(k+1)l}\right) + q_{k(l+1)}}{D_{mn} + 0.25 * \sum_{d \in \widehat{D}} d} \; if \; (k,l) mod 8 = (0,0) \\ \dfrac{q_{ij}}{D_{mn} + 0.25 * \sum_{d \in \widehat{D}} d}, otherwise \end{cases}$$

Where $\widehat{D}$ are the block energies of the neighbourhood of the mnth block. When it comes to boundary blocks, the nonexistent blocks are obtained by block symmetric padding. The distortions for the DC coefficients are defined as the mean of their neighbourhood AC coefficients in the same DCT block.

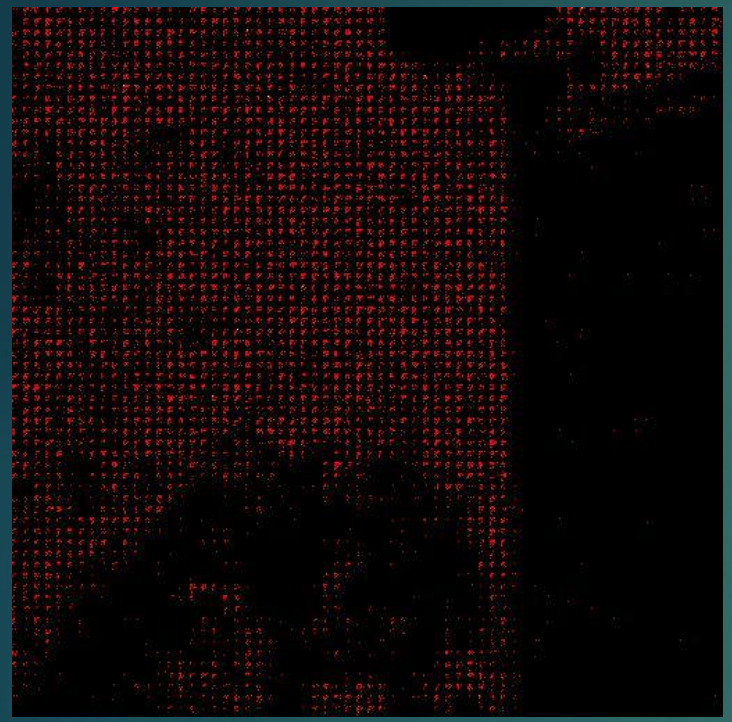
Source: Defining Cost Functions for Adaptive JPEG Steganography at the Microscale

Examples of using these algorithms

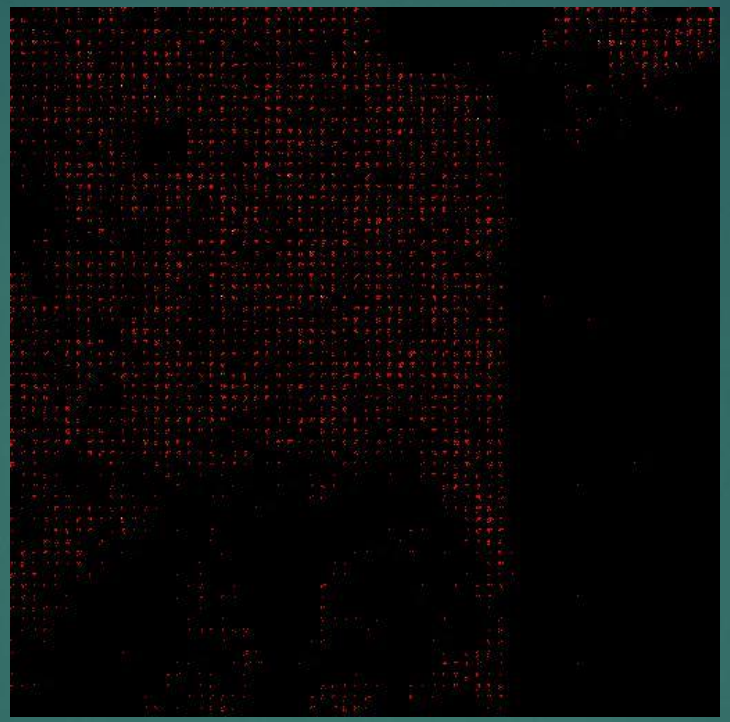
Unchanged image

Image in DCT coeficients

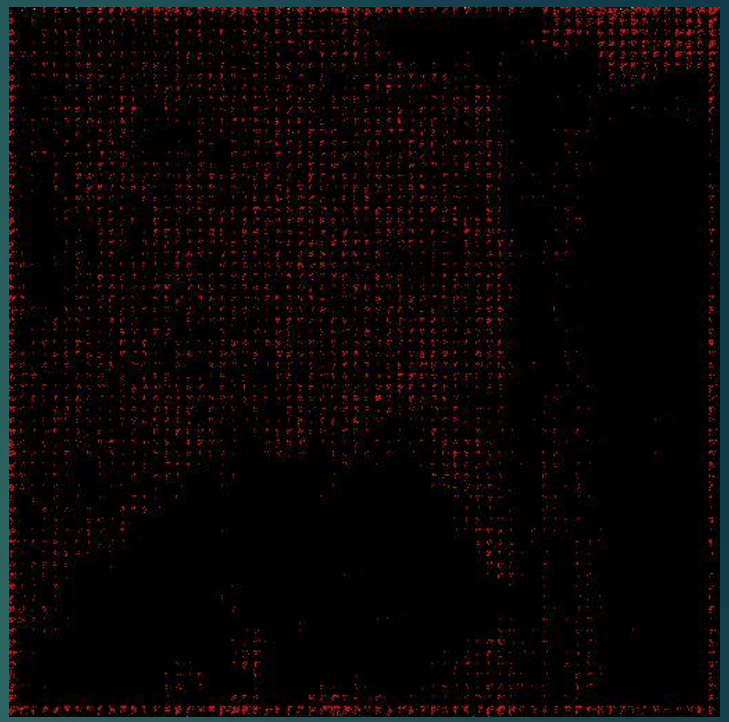
Examples of using these algorithms



JMiPOD



JUNIWARD



UERD

Used neural network models

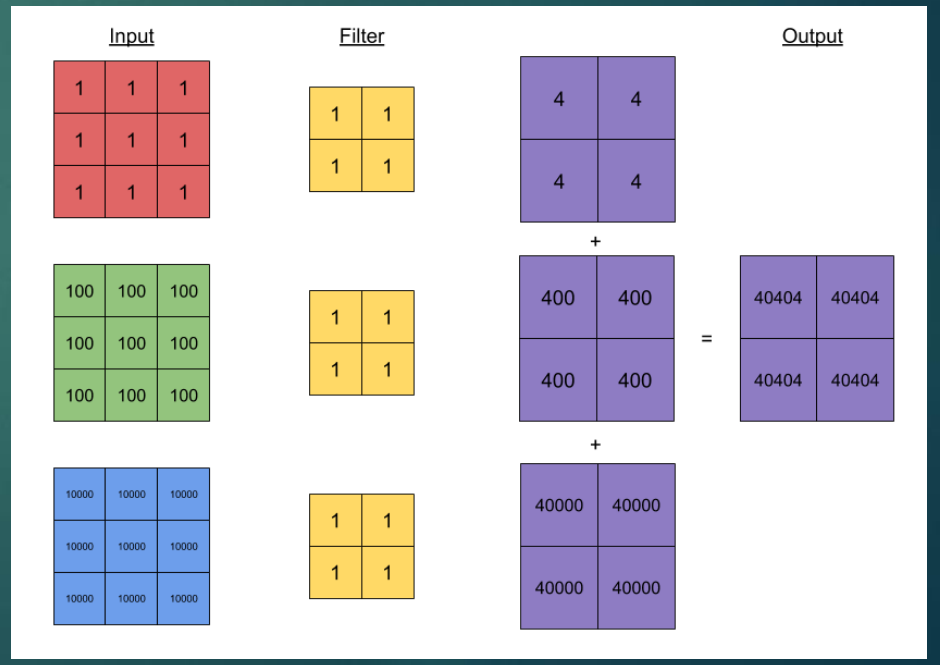
- Model created with Conv2D, maxpooling2d and BatchNormalization blocks.
- EfficientNet B0
- EfficientNet B3
- EfficientNet B7

Conv2D + MaxPooling2D + BatchNormalization neural network

- At the beginning, it was decided to use Convolutional Neural Network. This model is a multi-layer sequential model consisting of Conv2D, maxpooling2d and BatchNormalization blocks. Loss can be measured using the mean square error and is used Adam as our optimizer. These block are fron Keras Tensorflow library.

Conv2D layer

- This layer forms a convolutional core, which is convolvated with the input of the layer and creates a tensor of the outputs. If use_bias is true, a bias vector is created and added to the outputs. Finally, if there is no activation, it is also applied to the outputs.
- Example of applying one filter to RGB image:

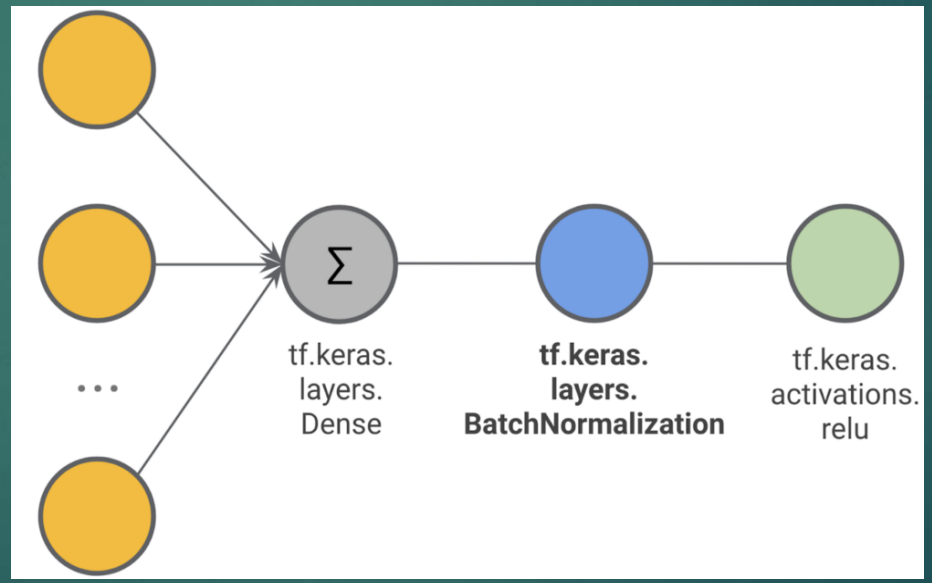


MaxPooling2D layer

- This layer obtains Maximum Association operation for 2D spatial data.
- It downsamples an input along its spatial dimensions (height and width) by putting a maximum value across the input window (the size defined by pool_size) for each channel input. The window is moved by steps along each dimension.

BatchNormalization layer

- Dose normalization applies a transformation that keeps the mean output close to 0 and the output standard deviation close to 1.
- Importantly, dose normalization works differently during training and during inference.



Obtained results from Conv2D neural network

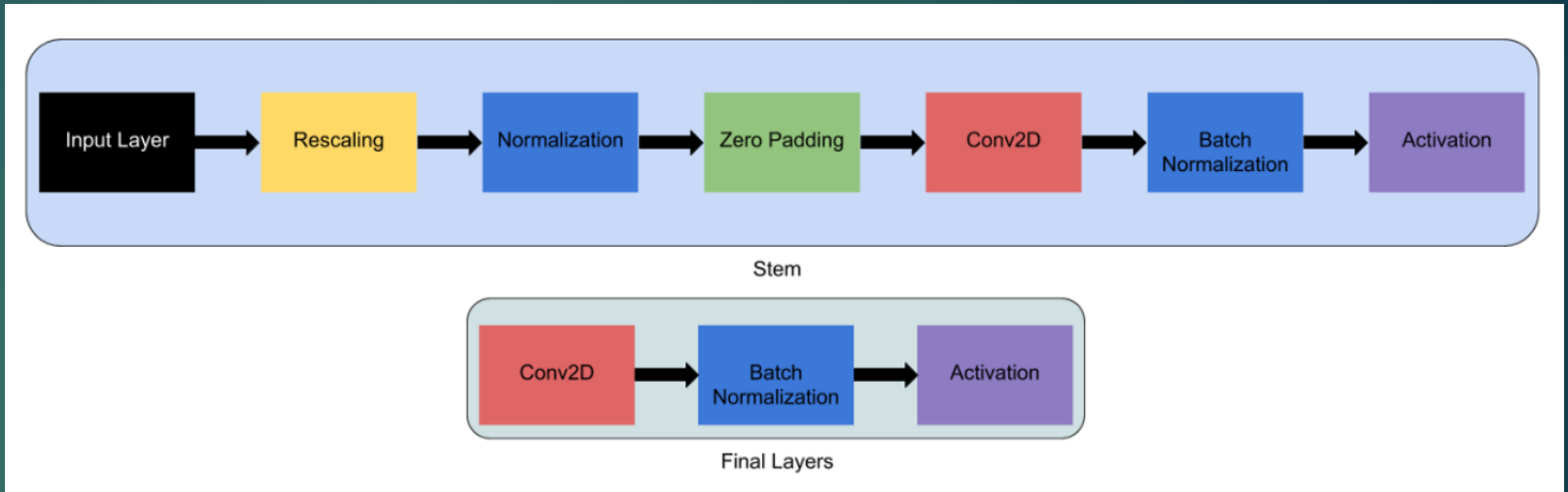
- As main metrics for created model was decided to use Loss, and mean absolute error.

After training on 30 epochs:

- Loss value: 0.233
- Val_loss value: 0.39
- Mean_absolute_error: 0.46

```
Epoch 17/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2415 - mean_absolute_error: 0.4670 - val_loss: 0.4706 - val_mean_absolute_error: 0.6809
Epoch 18/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2411 - mean_absolute_error: 0.4672 - val_loss: 0.4870 - val_mean_absolute_error: 0.6954
Epoch 19/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2410 - mean_absolute_error: 0.4674 - val_loss: 0.3944 - val_mean_absolute_error: 0.6216
Epoch 20/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2397 - mean_absolute_error: 0.4680 - val_loss: 0.4923 - val_mean_absolute_error: 0.6997
Epoch 21/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2391 - mean_absolute_error: 0.4668 - val_loss: 0.4169 - val_mean_absolute_error: 0.6437
Epoch 22/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2376 - mean_absolute_error: 0.4661 - val_loss: 0.3648 - val_mean_absolute_error: 0.6022
Epoch 23/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2385 - mean_absolute_error: 0.4675 - val_loss: 0.4320 - val_mean_absolute_error: 0.6543
Epoch 24/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2379 - mean_absolute_error: 0.4681 - val_loss: 0.4419 - val_mean_absolute_error: 0.6632
Epoch 25/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2372 - mean_absolute_error: 0.4662 - val_loss: 0.4297 - val_mean_absolute_error: 0.6519
Epoch 26/30
120/120 [==============================] - 16s 136ms/step - loss: 0.2371 - mean_absolute_error: 0.4671 - val_loss: 0.4073 - val_mean_absolute_error: 0.6360
Epoch 27/30
120/120 [==============================] - 16s 137ms/step - loss: 0.2377 - mean_absolute_error: 0.4669 - val_loss: 0.3740 - val_mean_absolute_error: 0.6089
Epoch 28/30
120/120 [==============================] - 16s 137ms/step - loss: 0.2357 - mean_absolute_error: 0.4670 - val_loss: 0.4100 - val_mean_absolute_error: 0.6383
Epoch 29/30
120/120 [==============================] - 16s 137ms/step - loss: 0.2359 - mean_absolute_error: 0.4666 - val_loss: 0.4324 - val_mean_absolute_error: 0.6565
Epoch 30/30
120/120 [==============================] - 16s 137ms/step - loss: 0.2361 - mean_absolute_error: 0.4666 - val_loss: 0.3968 - val_mean_absolute_error: 0.6254
INFO:tensorflow:Assets written to: my_model_30_epochs/assets
```

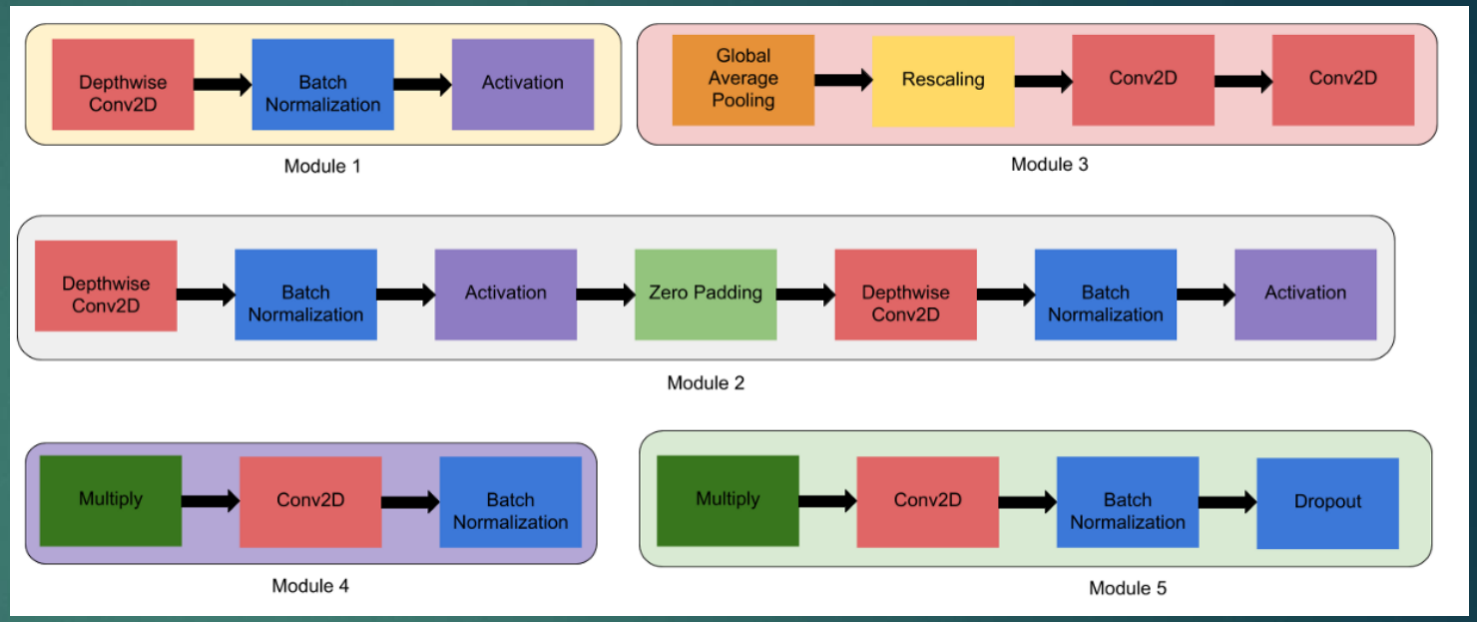
Family of EfficientNet models



Architecture of EfficientNet models starts which is common in all the eight models and the final layers. Each of them contains 7 blocks. These blocks further have a varying number of sub-blocks whose number is increased as changing model from EfficientNetB0 to EfficientNetB7.

5 main models in EfficientNet

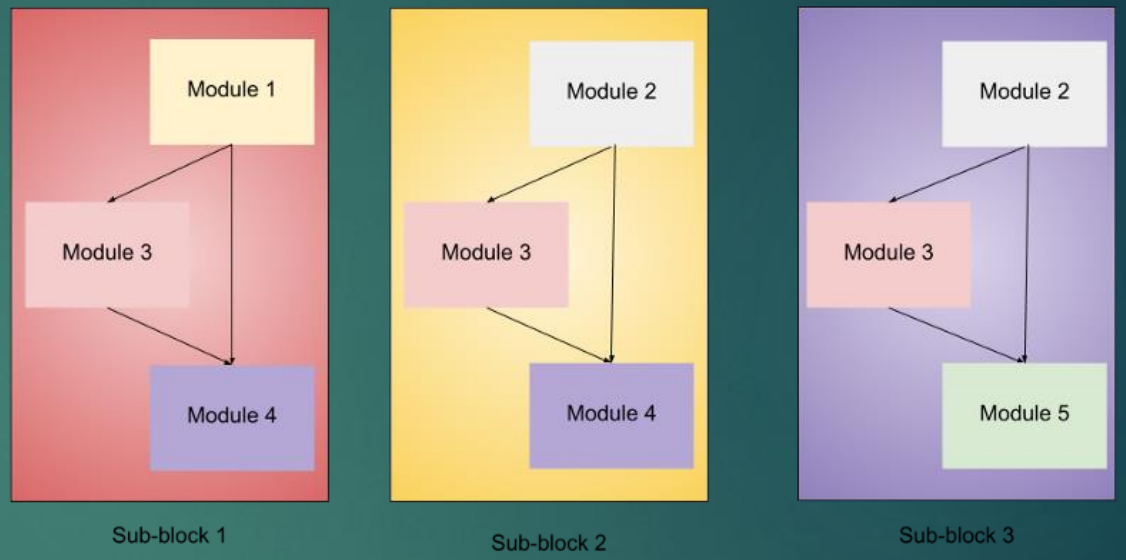
- **Module 1** — This is used as a starting point for the sub-blocks.
- **Module 2** — This is used as a starting point for the first sub-block of all the 7 main blocks except the 1st one.
- **Module 3** — This is connected as a skip connection to all the sub-blocks.
- **Module 4** — This is used for combining the skip connection in the first sub-blocks.
- **Module 5** — Each sub-block is connected to its previous sub-block in a skip connection and they are combined using this module.



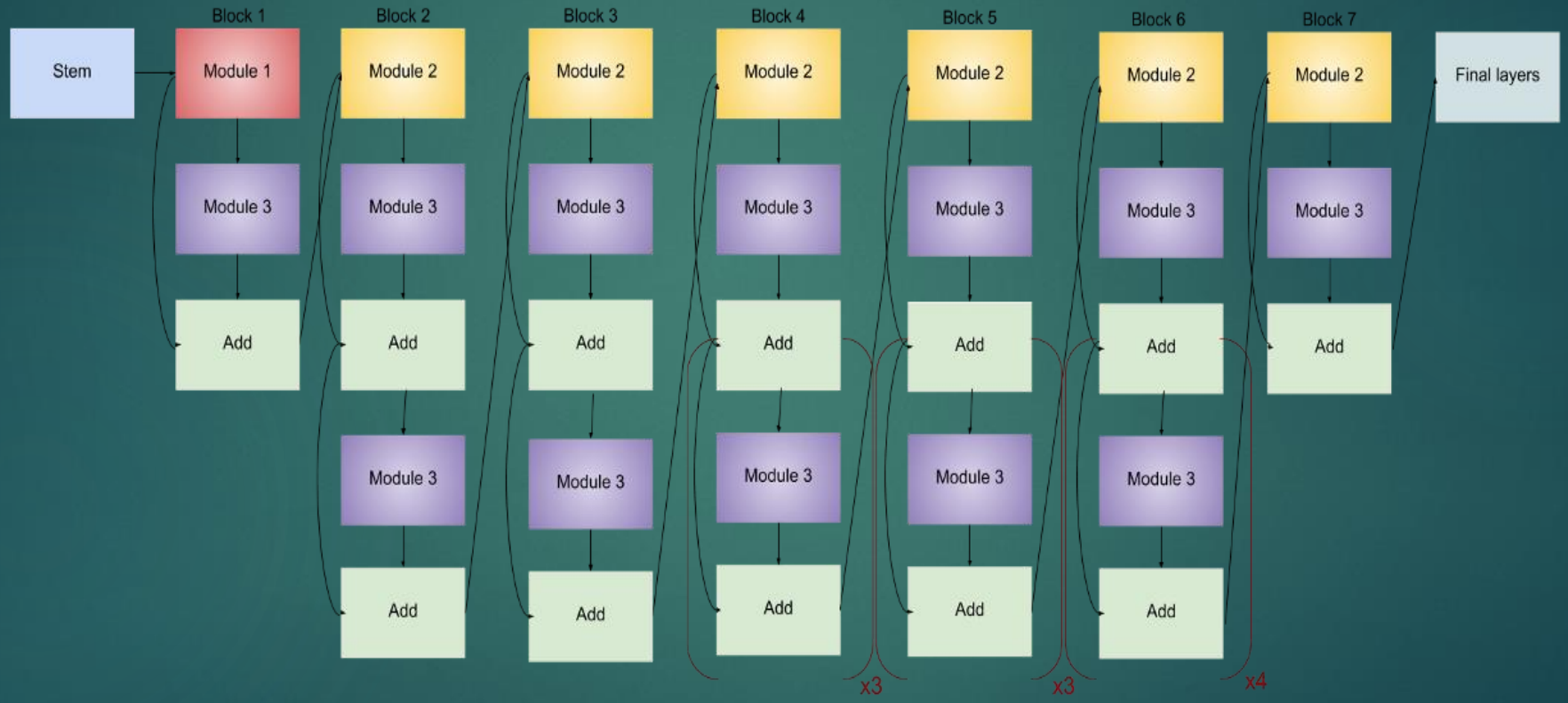
SubBlocks in EfficientNet

•These modules are further combined to form sub-blocks which will be used in a certain way in the blocks.

Sub-block 1 — This is used only used as the first sub-block in the first block.
Sub-block 2 — This is used as the first sub-block in all the other blocks.
Sub-block 3 — This is used for any sub-block except the first one in all the blocks.



B3 neural network architecture

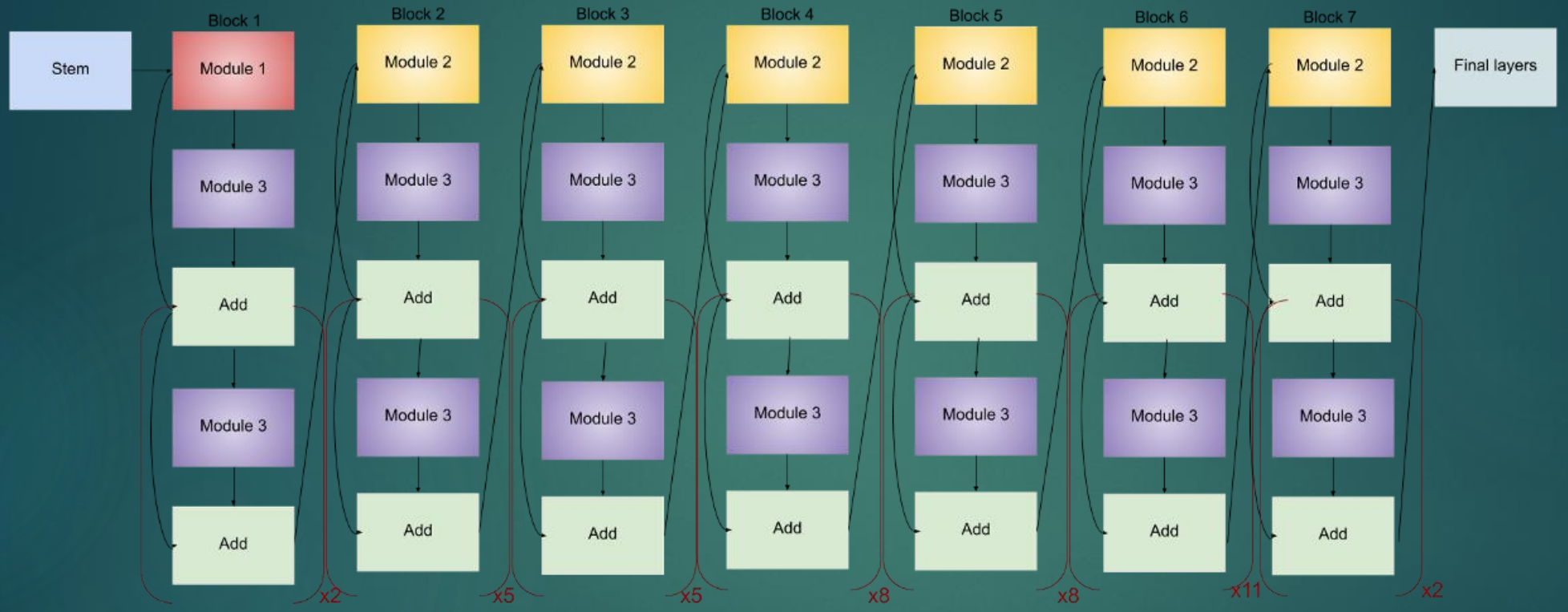


B3 neural network obtained results

- As for the task of classification was decided to use as main measurment metrics accuracy. After training model on 10 epochs, was reached accuracy 66%.
- Loss value: 0.64
- Val_loss value: 0.61

```
Train for 398 steps, validate for 71 steps
Epoch 1/10
398/398 [==============================] - 2929s 7s/step - loss: 0.6789 - accuracy: 0.5412 -
val_loss: 0.6582 - val_accuracy: 0.5778
Epoch 2/10
398/398 [==============================] - 142s 357ms/step - loss: 0.6480 - accuracy: 0.5827
- val_loss: 0.6349 - val_accuracy: 0.6052
Epoch 3/10
398/398 [==============================] - 142s 357ms/step - loss: 0.6299 - accuracy: 0.6092
- val_loss: 0.6563 - val_accuracy: 0.5761
Epoch 4/10
398/398 [==============================] - 142s 358ms/step - loss: 0.6171 - accuracy: 0.6218
- val_loss: 0.6216 - val_accuracy: 0.6179
Epoch 5/10
398/398 [==============================] - 142s 357ms/step - loss: 0.6062 - accuracy: 0.6329
- val_loss: 0.6114 - val_accuracy: 0.6181
Epoch 6/10
398/398 [==============================] - 142s 357ms/step - loss: 0.5984 - accuracy: 0.6434
- val_loss: 0.6157 - val_accuracy: 0.6200
Epoch 7/10
398/398 [==============================] - 142s 357ms/step - loss: 0.5907 - accuracy: 0.6503
- val_loss: 0.6150 - val_accuracy: 0.6118
Epoch 8/10
398/398 [==============================] - 142s 357ms/step - loss: 0.5848 - accuracy: 0.6563
- val_loss: 0.6304 - val_accuracy: 0.6251
Epoch 9/10
398/398 [==============================] - 142s 357ms/step - loss: 0.5785 - accuracy: 0.6611
- val_loss: 0.6765 - val_accuracy: 0.5824
Epoch 10/10
398/398 [==============================] - 142s 356ms/step - loss: 0.5717 - accuracy: 0.6678
- val_loss: 0.6891 - val_accuracy: 0.5891
```

B7 neural network architecture



Obtained results from B7 neural network

- As in previous example was used same metrics.

Obtained accuracy: 74%.

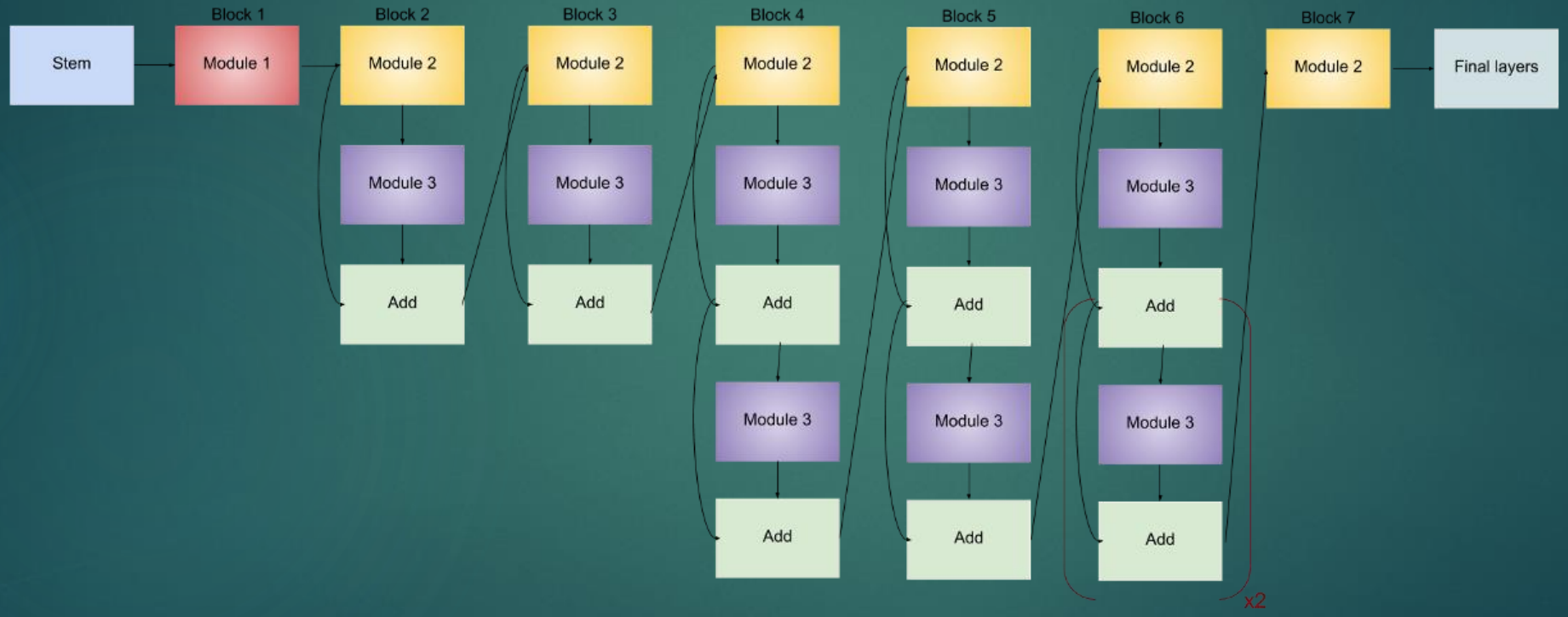
- Loss value: 0.56
- Val_loss value: 0.59

```
Train for 250 steps, validate for 63 steps
Epoch 1/5
250/250 [==============================] - 4240s 17s/step - loss: 0.5623 - accuracy: 0.7466
- val_loss: 0.5932 - val_accuracy: 0.7483
Epoch 2/5
250/250 [==============================] - 1232s 5s/step - loss: 0.5374 - accuracy: 0.7495 -
val_loss: 0.5320 - val_accuracy: 0.7483
Epoch 3/5
250/250 [==============================] - 1209s 5s/step - loss: 0.5252 - accuracy: 0.7493 -
val_loss: 0.5220 - val_accuracy: 0.7483
Epoch 4/5
250/250 [==============================] - 1235s 5s/step - loss: 0.5182 - accuracy: 0.7490 -
val_loss: 0.5285 - val_accuracy: 0.7418
Epoch 5/5
250/250 [==============================] - 1257s 5s/step - loss: 0.5100 - accuracy: 0.7491 -
val_loss: 0.5255 - val_accuracy: 0.7449
```

Testing steganalysis for different classes of images

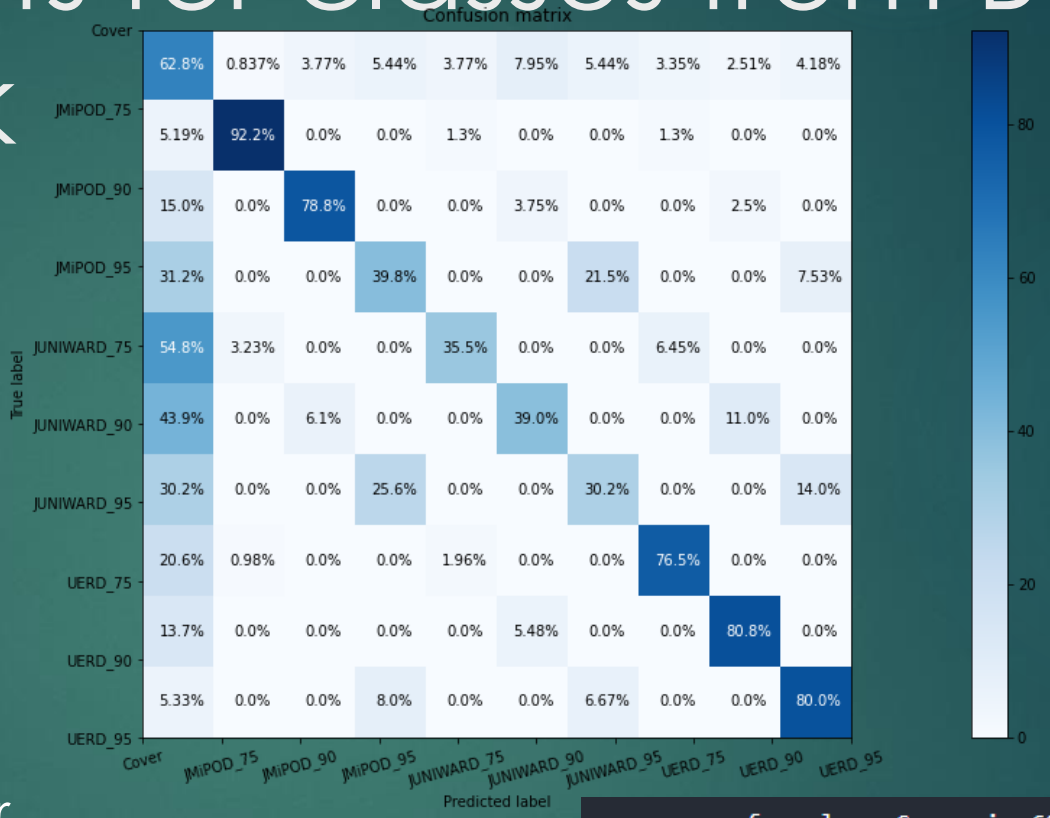
- Main idea was to test hypothesis of quality steganalysis for different class pictures. Also, is required to test dependency between quality of image and accuracy of classification steganography image as true.
- For the purposes of analysis was decided to use network EfficientNet B0.

EfficientNet B0 architecture



Obtained results for classes from B0 neural network

- Highest accuracy for class JMiPOD is 92% for the PIL index quality 75.
- Highest accuracy for class UERD is 80% for the PIL index quality 90
- Highest accuracy for class JUNIWARD is 39% for the PIL index quality 90.




```
accuracy for class Cover is 62.76150627615063
accuracy for class JMiPOD_75 is 92.20779220779221
accuracy for class JMiPOD_90 is 78.75
accuracy for class JMiPOD_95 is 39.784946236559136
accuracy for class JUNIWARD_75 is 35.483870967741936
accuracy for class JUNIWARD_90 is 39.02439024390244
accuracy for class JUNIWARD_95 is 30.23255813953488
accuracy for class UERD_75 is 76.47058823529412
accuracy for class UERD_90 is 80.82191780821918
accuracy for class UERD_95 is 80.0
Val Loss: 11.6, Weighted AUC:0.883, Acc: 60.9
```

Summary

- As the image quality increases, classification accuracy decreases
- CNN seems to put very high emphasis on border areas. In case of visualization all the results then can be found the hidden messages seems to occur on the border areas more often.
- The hidden messages seem to almost always at the edges of objects. If the object has smooth texture, the hidden message is glaringly absent in the smooth part.
- Obtained results show the effectiveness usage of EfficientNet B0, but the accuracy can be improved in case of usage bigger number of epochs and more efficient models.