COMPARISON OF UNET ARCHITECTURES ON NON-MEDICAL IMAGE DOMAIN

December 11, 2020

ABSTRACT

We trained two fully convolutional neural networks and compared them to each other, namely the original UNet and MultiResUNet. Since UNet is the most popular network architecture in biomedical image segmentation, it is interesting to investigate whether the relative improvement gained in MultiResUNet compared to UNet can be achieve on a different image domain, e.g. images of cats. Since the used images in the COCO dataset contains qualities or lack thereof similar to those in biomedical images, one could suspect to see same performance difference between UNet and MultiResUNet in this domain as well. Results yielded accuracies for UNet and MultiResUNet of 61,98% and 66,02% respectively, showing that MultiResUNet were indeed able to outperform UNet by just over 4% points. Furthermore, due to lack of GPU performance, training was limited, but at the time of stopping, MultiResUNet seemed to be able to gain some extra precision had it been allowed to train for some more time.

***K*eywords** Machine Learning · Deep Learning · Convolutional Networks · UNet

1 Introduction

When it comes to image segmentation, convolutional neural networks has proven to be very successful. In particular, in the domain of biomedical images, where data is often sparse and multimodal, UNet is the most popular architecture.

Although the UNet architecture is so succesful in segmenting biomedical images, in 2019 Nabil Ibtehaz and M. Sohel Rahman came up with modified version of the UNet architecture which they called MultiResUNet. This net were supposedly able to outperform the original UNet[4], and in some cases achieve a 10% relative improvement[1]. They achieved this better result by introducing residual connections in multiple places in the network as well as altering the skip connections between the encoder and decoder. The blocks used in the encoder and decoder was also made more complex.

We would like to investigate if we are able to mimic the results above by recreating the UNet and MultiResUNet, but trained, evaluated and tested on a different image domain, namely images of cats.

We will be using the COCO dataset and suspect that the qualities of the cat images in this dataset, or lack thereof, matches in some regard those images in the biomedical domain.

2 Methods

2.1 The Dataset

Supervised image segmentation learning requires labels for each image that are actually a mask instead of, say, a single number representing a class. This mask has the same dimensions as the image but every pixel in mask has a value corresponding to the class it belongs to.

The COCO, (Common Objects in Context), dataset provides images and corresponding labels that can be used to solve 5 tasks, one of which is image segmentation[3].

2.1.1 Difficulties with COCO

COCO is a dataset with a large variety of images. Since COCO contains a lot of classes, the images of cats often also contain other objects which in some cases dominate the images. For our model training, image down-resizing is needed, and in some of these resized images, the cats are now almost impossible to spot. It is also not rare that the cats are occluded behind other objects, or text which has been inserted after the images was taken.

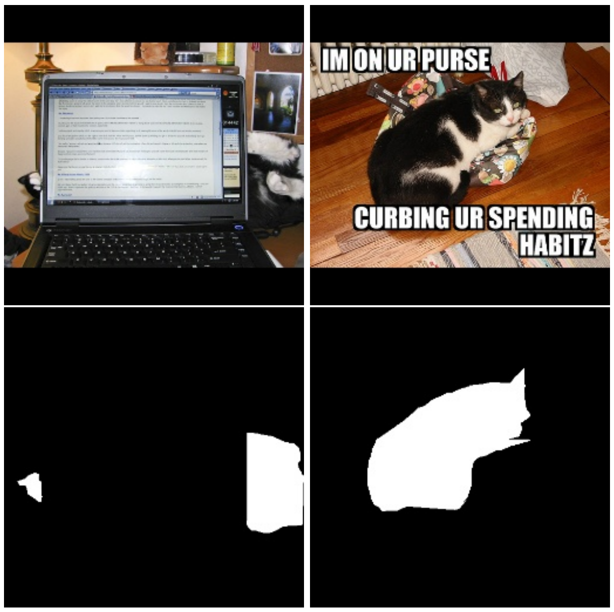


Figure 1: Left image shows an cat highly occluded behind a laptop computer. The image on the right shows a cat slightly occluded behind text inserted into the image. Below both images are their ground truth masks.

Inspecting the images further reveals that some of the images labelled with "cat" does not contain actual cats, but rather illustrations, drawings, shadows or objects of things which look like a cat. Figure 2 shows some example images and masks of this.

These "bad" images were not removed, to yield more data. In addition the network is challenged to also recognise these difficult cats. The images may have a bad effect on the network, since they actually have a valid mask, forcing the network to try to predict them.

The goal of this project is to segment real cats, hence these images impose a problem. In the future iterations these images could be removed completely, or only the mask could be removed yielding negative images.

2.2 The Architecture

In this project, we have built two UNets, one of which we will call UNet 1.0 and the other for UNet 2.0. UNet 1.0 will have an architecture as described in the next section, where UNet 2.0's architecture will be a version of UNet 1.0 where the skip connections and blocks have been modified.

2.2.1 UNet 1.0

The original UNet was created by Olaf Ronneberger, Philipp Fischer, Thomas Brox in 2015 in the paper “U-Net: Convolutional Networks for Biomedical Image Segmentation”[4], and as the paper also hints, the most common use case for UNets is for biomedical image segmentation.

The network does image segmentation, meaning to simplify and change the representation of an image, to something that it is easier to analyse and make sense of than the original image. This is done by converting the input image into an

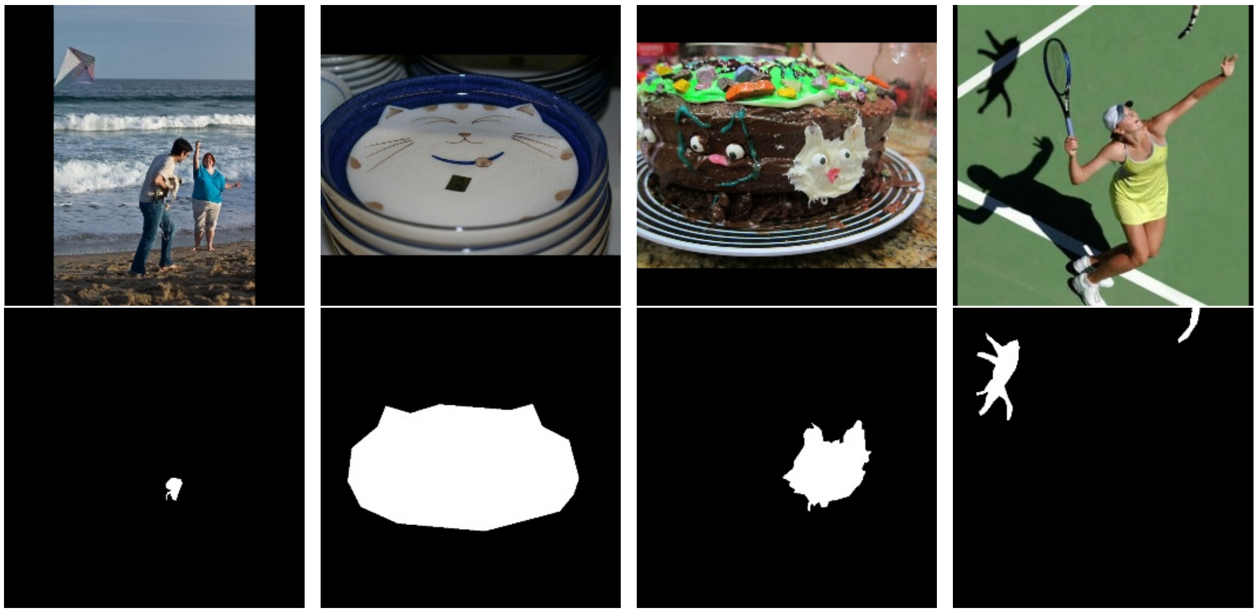


Figure 2: The first image shows a man holding a cat. The second shows some plates with cat image. The third shows a cake with cat frosting. The fourth shows the shadow and tale of a cat. Note that only the first image contains an actual cat, and a cat's tale is visible in the fourth image. In the COCO dataset, however, judging by the corresponding masks, it is still images of cats.

output mask where each pixel has been assigned a class. In the output mask, pixels that share the same class would then be assigned to the same color for instance, making it easier to see boundaries between objects of difference classes.

The network is a fully convolutional neural network, meaning that it has no dense layers as traditional CNNs, but instead has 1x1 convolutions at the end that performs the task of the dense layers.

The network is built out of so-called down- and up-blocks. The input image is first fed into a down-block and the extracted features are then downsampled and propagated on to the next down-block and so on. After a total of 4 down-blocks in succession, we have the bottleneck-block, and afterwards 4 up-blocks in succession. Contrary to the down-blocks the up-blocks upsamples the features from block to block. This means that the UNet is an autoencoder that learns to extract the most useful information from the images in the downsampling phase and then in the upsampling phase reassembles the pieces into something more meaningful, e.g. segments the image into different semantic categories.

The down-blocks consists of two convolutions followed by a max pooling layer. In the bottleneck-block (the block that connects the encoder to the decoder) a transposed convolutions follows the two convolutions instead. This is also the case in the up-blocks. Lastly a 1x1 convolution is applied.

The "output" features of a down-block before the max-pooling operation is copied over to the corresponding up-block. Here it is concatenated with the incoming features from the previous up-block just before the first convolution. For example, after the first two convolutions in the first down-block, the features are copied and concatenated with the last up-block's incoming features, see figure 3. This enables the network to propagate the spatial information that gets lost during the pooling operation from the encoder to the decoder.[4]

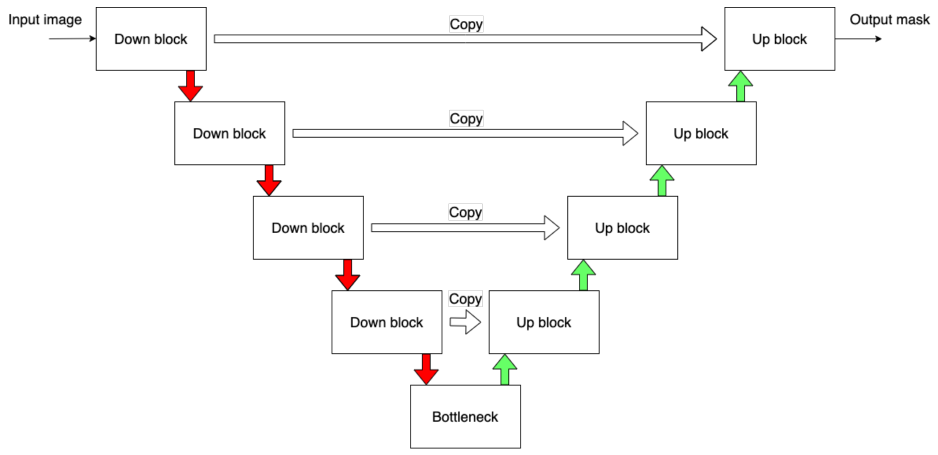


Figure 3: The general UNet architecture which is built out of blocks and when modelled this way forms a "U", hence the name. Whats common to all UNets is that the input image gets downsampled

The first convolution in the network have 64 filters, and the following layers will have the same number of filters, until the first convolution in the next down block, where the number of filters gets doubled. In the bottleneck we therefore end up with 1024 filters. In the transposed convolutions the number of filters gets halved, but with incoming features from the encoder at this step 1024 filters are fed into the first convolution in the first the up-block. Here the number of filters gets halved as well. This pattern continues until the last layer, where a 1x1 convolution generates the 1 channel output mask. The convolutions and transposed convolutions have kernel size 3x3 and uses stride 2 and padding "same". This retains the size of the feature maps between layers. The max-pooling layers have a pool-size of 2x2. See figure 4. The convolution layers are activated by ReLU, except for the last layer where a sigmoid is used to squish values in between 0 and 1.

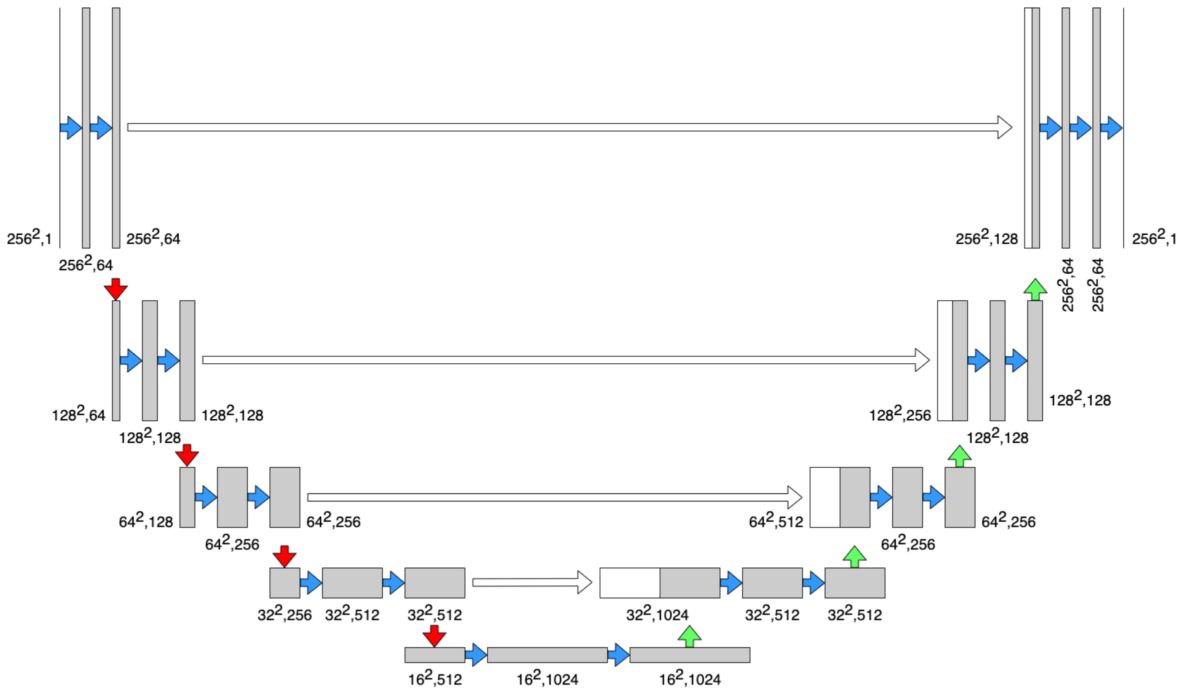


Figure 4: The UNet architecture. The tuples' first value is the size of the filters and the second value is the number of filters. Blue arrows indicate convolutions with kernel size 3, red arrows indicate 2x2 max pooling, green arrows indicate transposed convolutions also with kernel size 3, and white arrows indicate copying layers which will be concatenated with layers created by the up-sampling.

2.2.2 UNet 2.0

The architecture and ideas for our UNet 2.0 is heavily borrowed from Nabil Ibtehaz and M. Sohel Rahman's paper[1] in which they describe a MultiResUNet, a modified architecture for UNet which should allegedly be able to perform up to 10% better relative to UNet. The modified elements of UNet will be referenced in our project by the same name as they use in their paper.

2.2.2.1 Modified skip connections

A key part of UNet is its skip connections between the encoder and decoder, and although they support the decoder with spatial information about the input that gets lost during the max-pooling operations, there might be a mismatch in the semantics of the features between those coming from the encoder and those coming from the decoder at the concatenation step.

For instance, before the first max-pooling operation in the first block the features sent through the first skip connection are low level features of the input image, whereas in the corresponding up-block just after a transposed convolution, the features coming from the decoder are much more high level since it is in the deeper layers of the network. Because of this, there is a reason to suspect a semantic mismatch that could potentially impact the prediction negatively.

It should be noted, however, that as we get deeper in the encoder, that skip connections connects blocks that supposedly have a smaller and smaller semantic mismatch (if present at all), since they are closer to each other in the network, causing a smaller negative impact in the final prediction.

To reduce the semantic gap between the features from the encoder and the features from the decoder, one could introduce convolutions on this skip path. You could also argue that the number of convolutions should vary between the skip connections to accommodate the difference in the semantic mismatch of the features.

This new variable length skip connections, the ResPaths, could be built out of simple 3x3 convolutions blocks with an added residual connection combined with a 1x1 convolution. The reason for the residual connection is simply that is has been seen to ease the learning process as well proven very useful in convolution neural networks in general[1], see figure 5.

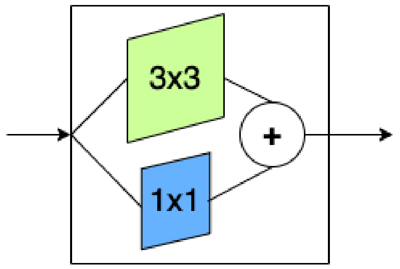


Figure 5: ResPathBlock. The building block for the modified skip connections. The input is simply convoluted with 3x3 filters and added together with itself after 1x1 convolution.

Now, the first ResPath in the network could for instance consist of four ResPathBlocks in succession, whereas the last could consist of only one. See figure for an example of a complete ResPath.

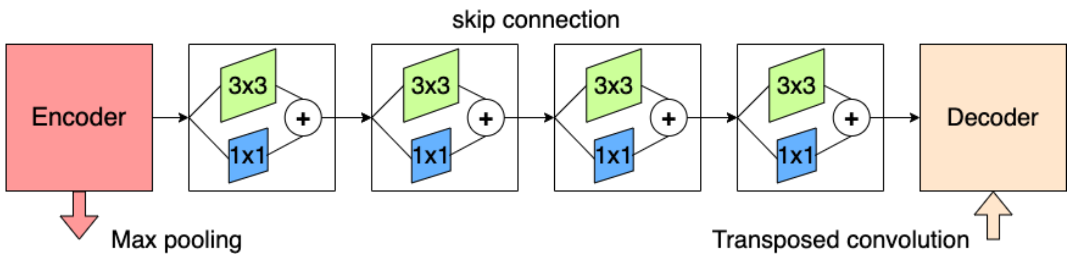


Figure 6: ResPath. Multiple ResPathBlocks in succession that adds a connection between the encoder and the decoder

2.2.2.2 Modified blocks

In the our dataset, the objects of interests were varying size, stretching from so tiny that the object were almost impossible to spot with the naked eye to so big that they take up most of the image, see figure 7.

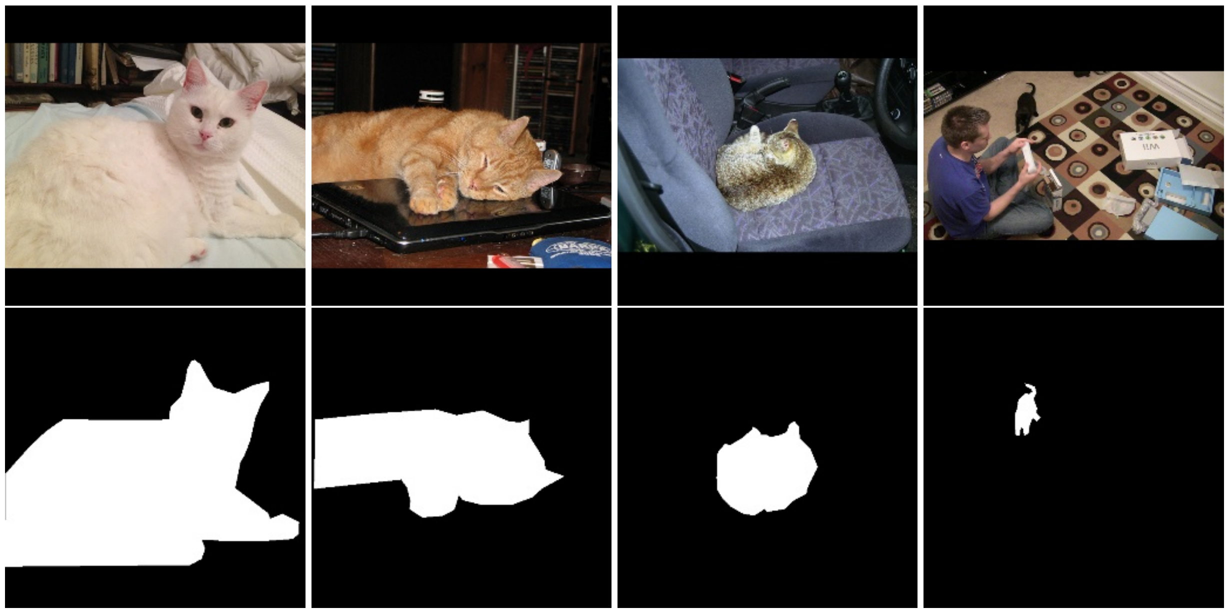


Figure 7: Four images and their masks where the object of interest takes up different amount of space in the image.

The revolutionary Inception architecture made by Szegedy et al. in 2015[5] introduced Inception blocks that used convolutions of varying kernel sizes in parallel, to inspect the points of interest in images from different scales. One can imagine that these blocks allows the network to "zoom" in and out as needed to achieve the best predictions.

An idea could be to replace all down- and up-blocks with corresponding inception blocks, or as Nabil Ibtehaz and M. Sohel Rahman call them in their paper ResBlocks.

It is known that combining multiple convolutions of some kernel size approximates a single convolution with bigger kernel size while reducing the number of parameters. In the ResBlock a 3x3, a 5x5 and a 7x7 convolution is run on the input and concatenated before being added to a residual connection with a 1x1 convolution. In reality only convolutions with kernel size of 3 are used in the ResBlock to save memory consumption, but as mentioned above, by chaining the convolutions together we can approximate convolutions with bigger kernel sizes, see figure 8.

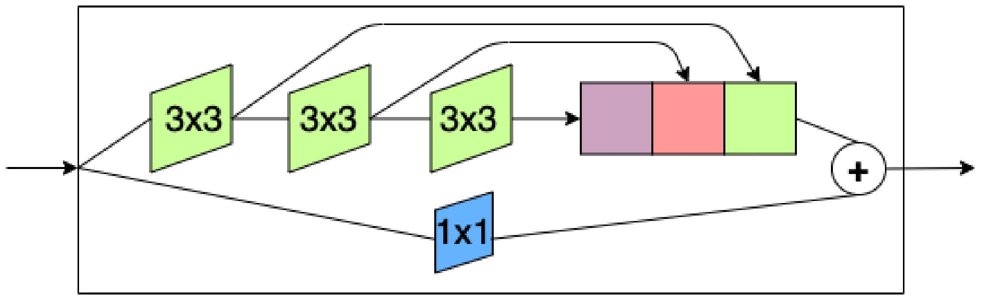


Figure 8: ResBlock. The input is convoluted by 3x3, 5x5 and 7x7 filters all approximated by 3x3 convolutions. Afterwards the features are concatenated. Purple are features extracted by an approximated 7x7 convolution, pink are features extracted by an approximated 5x5 convolution and green is features extracted from the 3x3 convolution. Lastly the concatenated features are added with the residual connection that also has a 1x1 convolution.

2.3 Data preprocessing

The objective of this project is to compare the performance of the different architectures. Therefore, no domain-specific preprocessing has been applied. The images were zero padded, though, to make their dimensions square and afterwards scaled to 256x256 pixels to fit into the network. Since only one class is used, the masks are simply black and white images, where the white pixels correspond to the cat in the image. The preprocessing has also been applied to the masks, such that they still fit the original images with respect to size and location.

The dataloader used does further preprocessing. It normalizes all images such that values are in the range $[0, 1]$. The same goes for the masks. Since they are strictly black and white, the pixels have values 1 or 0.

2.4 Training methodology for UNets

2.4.1 Implementation

All of the UNets have been implemented in Python 3, using Keras as the deep learning framework. This made the implementation of the networks compositional, simple and fast. The source code can be found here.

To accelerate training, the code is written using Colaboratory by Google. This made it possible to utilise cloud computing with GPU acceleration. Furthermore Google Colab allowed us to store and load the models parameter between epochs easily.

During the training phase, the model's weights are saved when it achieves a new highscore. The "score" can be measured either as a low loss or a high accuracy. We opted for achieving the highest accuracy, since the problem at hand is a classification problem, where the network classifies every pixel either as a cat, or non-cat.

2.4.2 Batch normalization

Batch normalization is known to combat vanishing or exploding gradients, and hereby ease and improve the learning process. It has been seen, though, that UNet 1.0 can perform worse with batch normalization[1], and so it has been omitted in this architecture.

We have tested UNet 2.0 with and without batch normalization and found that the best performance is obtained with batch normalization.

This means there's the additional difference in the architectures of UNet 1.0 and UNet 2.0 of batch normalization.

2.4.3 Dataset split

From the COCO dataset all images containing cats were extracted and used. There were a total of 4290 images that contained pixels that were classified as cat.

These images were split into a training, validation and test set. To achieve an equal split, an additional 30 images were added to the image pool before the split, giving a total of 4320 images. These are random images from the rest of the COCO set and does not contain a cat, i.e. their mask is a black image.

The splits are as follows:

Set	Images	Fraction
train	2880	$4/6$
val	720	$1/6$
test	720	$1/6$

Doing it this way, makes it possible to fine tune the network on the validation and training data, and still have unseen data for proper evaluation.

The reason the $\frac{1}{6}$ skip, is essentially that if the network performs good on the validation, we would like for it to perform good on the test data. Having the same amount of images in validation and train, make them more representative. Furthermore, since the dataset is dificult, we would like equal amounts of images, and a large fraction, to limit the difference in distribution. This means that tuning the network for the validation set also makes it better on the test, without it ever training on these images.

2.4.4 Class imbalance

In image segmentation, a typical problem is having class imbalance in the dataset. Class imbalance is when some class or classes are overrepresented in general in the datasets. Unfortunately, this is also this case for our dataset of cats.

Here having the cat as one class, and the rest of the image as a second class, may pose problems if there is much more non-cat than cat in the images. This may affect the model negatively as its primary goal is to minimise the loss in general and not maximise correct predictions, so to speak.

To visualise our skewed dataset, the distribution of cats in the image is calculated. This is done, by summing over the pixels in the masks that are 1 and dividing by the size of the image.

$$\textit{cat\_fraction} = \frac{\#pixels = 1}{\#pixels}$$

This gives the fraction of how much 'cat' there is in the image. These are plotted in a histogram with groups defined by the following elements $[0, 0.01, 0.02, \ldots, 0.99, 1]$.

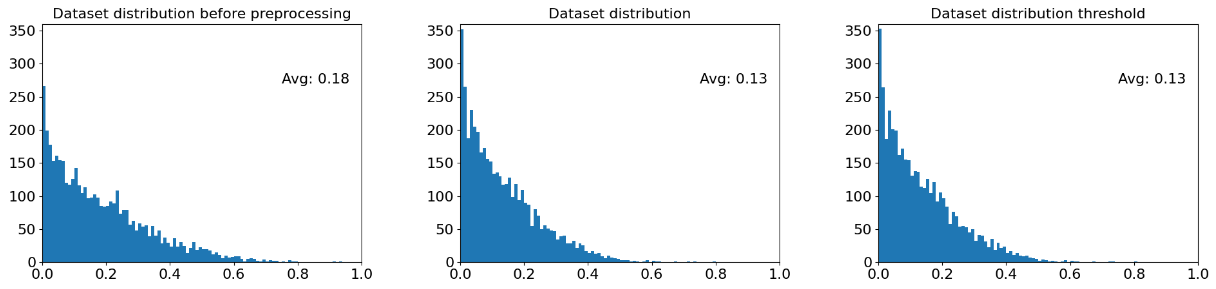


Figure 9: Distribution of cat-per-image. The first figure shows the distribution before the preprocessing. The next two figures shows the distribution without threshold, and with threshold.

Figure 9 shows the distribution of the cat class compared to the entire image. Its clear that in the majority of the images there is very little cat compared to non-cat. As the sub-figures also show, the cats take up 18%, 13% and 13% of the images on average respectively.

It also worth mentioning that the imbalance becomes greater after the preprocessing. This is due to many of the images being padded with zeroes (non-cat pixels) along top and bottom or in the sides, to make the image have square dimensions.

During the preprocessing the images are scaled down which causes aliasing along the edges of the segmentations. This means that some values of the mask are not strictly 0 and 1. The third figure is there to illustrate, that this not effect the distribution of cat-per-image in the dataset greatly.

2.4.5 Loss function and accuracy

Our task at hand is to predict which pixels represent a cat and which that does not. This task is therefore a binary classification problem for each pixel. Intuitively the binary cross-entropy function could then have been the choice of loss function.

The binary cross-entropy function, though, can cause problems when having imbalance in the labels. As Figure 7 shows that the cat portion of the images may be very small, causing this imbalance. Here binary cross-entropy would not penalise labelling all pixels as non-cat very much.

As mentioned earlier our dataset is skewed. Figure 9 shows that the distribution of cats is skewed towards the low end. This would mean, that the binary cross-entropy loss could perform bad and the network would learn to create black images with white blobs, matching the distributions.

Therefore the Dice coefficient loss has been chosen instead.

$$\textit{dice\_loss} = 1 - \frac{2 \cdot A \cap B}{A + B}$$

The Dice coefficient is a measure of the intersection over prediction and the ground truth. This way, the network will have to learning to classify cat pixels correctly and not just classify everything a non-cat. [2].

As an accuracy measure Intersection over Union have been used. This is almost identical to the Dice coefficient with the only difference being that Dice emphasises on hitting true positives more. IoU is more intuitive to understand as it is simply the ratio between what the ground truth and prediction have in common and the union of the ground truth and prediction.

The intersection over union is computed as $\boldsymbol{IoU} = \frac{A \cap B}{A \cup B}$.

Since the final layer of the networks is activated by a sigmoid, the output pixel values are in the interval $[0, 1]$ where the ground truth only contains values in the set $\{0, 1\}$. For the predicted mask to match the ground truth a threshold of 0.5 will be applied to the output mask to map values into $\{0, 1\}$.

2.4.6 Hyperparameters

There have been several tests for different usage of hyperparameters, such as learning rates. The best results came from using a learning-rate finder from keras combined with an exponential-decay-learning-rate scheduler. This scheduler is used for decaying the learning rate by every fixed step size. In this network the step size is 10 epochs The reason for having this kind of scheduler, is to lower the learning rate as the training progresses, which will also allow the network to become more accurate. As it forces the network to learn slow after a number of iterations, this makes sense intuitively, that a network quickly can learn the basics, but then needs to be more careful, when learning more specific things in the later iterations. Another benefit from using schedulers is that the initial learning rate is not as important as if it was a fixed constant being used in each iteration.

2.4.7 Optimizer

As an optimizer in both of the UNets Adam has been used, combined with the learning rate scheduler described in the *Hyperparameter* section. Adam is the best general optimizer, there will of course be cases where SGD could perform better. But generally Adam is considered the best option, and was also chosen for this implementation.

2.4.8 Training aproach

The model was trained using Keras with a sequence data loader that splitted the training data into batches. Keras sequencers is a secure way of doing multiprocessing as it makes sure the network will only train on each sample once per epoch. Using other dataloaders can not guarantee this.

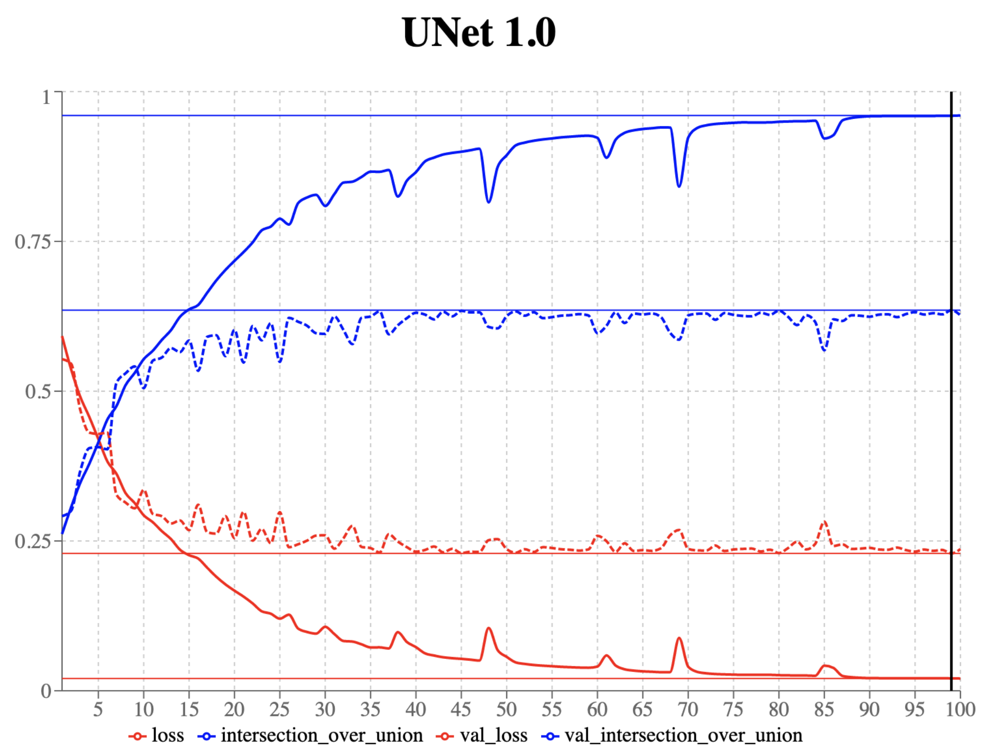
Each epoch consist of all the training-data meaning 2880 images, these were split into eight batches consisting of 360 images per batch. At the end of each epoch the validation scores was measured on the validation data, this consisted of 90 images per batch. The results reached on this data by the network was then validation score for that specific epoch. The weight for the best intersection over union validation score was then saved, as a way of minimizing the disk space used on Colab, compared to saving all the weights after every epoch.

The data in the test split will only be used for testing after the best weights have been found.

3 Results

We will compare results between the two UNets and not benchmark on the COCO dataset as we have handpicked the images used for training, validation and testing as well specified our own task, namely segmenting images into cat and non-cat.

3.1 UNet 1.0



At best epoch (99):	loss :	0.021	intersection_over_union :	0.960	val_loss (dashed):	0.229	val_intersection_over_union (dashed):	0.635
Individual best:	loss :	0.020	intersection_over_union :	0.960	val_loss (dashed):	0.229	val_intersection_over_union (dashed):	0.635

Figure 10: Training overview of UNet 1.0. Loss and accuracy scores are tracked for every epoch on both the training and validation data. Validations scores uses dashed lines. A black vertical line marks the epoch where the model performed the best.

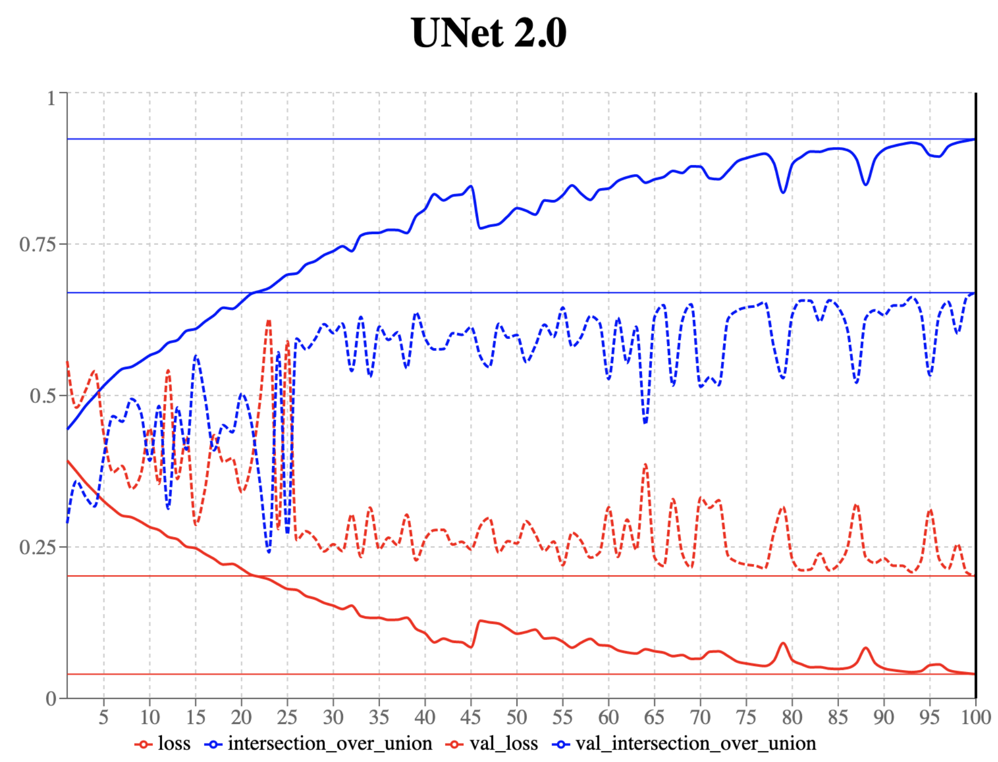
The result for UNet 1.0 shows an accuracy measured by intersection over union on 63,5 %. The graph shows that the validation accuracy barely improves after 35 epochs, the loss also stays approximately the same, but with some spikes, matching the spikes on the accuracy graph. Training accuracy continues to improve until it reaches 100 epochs, where it achieves an accuracy on 96%, if the graph for the training accuracy continues it looks as it will come close to 100% accuracy.

Measure	Score
Dice loss	0.2423
IoU	0.6198

Table 1: Dice loss and intersection over union score for UNet 1.0 evaluated on the test set.

To measure the results on the test set, the best weights for the network have been saved. The best weights had a score on 0.635 on the validation data, on the test set it scored slightly lower at a 0.612. Corresponding to approximate 1.5 % points.

3.2 UNet 2.0



At best epoch (100):	loss : 0.040	intersection_over_union : 0.923	val_loss (dashed): 0.202	val_intersection_over_union (dashed): 0.670
Individual best:	loss : 0.040	intersection_over_union : 0.923	val_loss (dashed): 0.202	val_intersection_over_union (dashed): 0.670

Figure 11: Training overview of UNet 2.0. Loss and accuracy scores are tracked for every epoch on both the training and validation data. Validations scores uses dashed lines. A black vertical line marks the epoch where the model performed the best.

The results for UNet2.0, shows a validation accuracy at 67 %, it was achieved at the very last epoch. The validation score is getting better and better further into the training, not improving on every epoch, but kept eventually getting better scores. The training score improved throughout the 100 epochs and in the end achieved a score at 0.923, this was again reached in the last epoch.

Measure	Score
Dice loss	0.2102
IoU	0.6602

Table 2: Dice loss and intersection over union score for UNet 2.0 evaluated on the test set.

The results on the test set is measured with the best weights, and achieved a score on 0.660, only 1 % point worse than the score on validation set.

3.3 Image examples

The 4 strange images from figure 2 and the different size cat images from figure 7 has been collected and run through both networks. The predictions can be seen in appendix A.

In the images where there is an actual cat, UNet2.0 seems to perform better. Comparing to the ground truth, the predicted masks from UNet2.0 fits better in terms of size and shape than UNet1.0.

In the 4 images which does not contain actual cats, both UNets perform bad. The masks does not fit the ground truths.

It should be noted that the masks shown are not the true masks. Since the network has sigmoid as final activation, the predicted masks are not strictly 0 or 1. Our intersection over union, takes this into account and uses a threshold. Therefore the shown masks in appendix A are also thresholds masks with threshold ≥ 0.5

4 Discussion

One thing that certainly is impacting the accuracy for the network, is the difficulty of the dataset in general and the large number of "outliers", but also the downscaling of the input images that was necessary due to lack of GPU performance. The images from the COCO dataset is of varying sizes, and so to achieve same sized images, all images were scaled up or down to 256x256. It so happens that all images were larger than this, so unfortunately all images had to be downscaled resulting in loss of information. It naturally also led to the absolute number of cat pixels to be lowered, making it harder for the network to achieve good prediction scores, as it gets penalised a lot by being just some pixels off, since the cat rarely takes up much of the image.

The padding on the images didn't help the model either, since it skewed the dataset even more. Instead of adding padding one could have zoomed in on the images, to maintain high resolution on the features while increasing the cat-per-image ratio.

The graphs showing the results found, is close to identical in the form that after an amount of epochs the validation scores stop rising, while the training scores continue to rise until it classifies almost perfectly on the training data. Because of this it should not be expected that the validation scores would improve a lot, even if it was to be trained on a hundred epochs more. The graphs does not show any sign of the model overfitting, which makes the network quite robust and it is hard to tell when the network will start overfitting. One may wonder if considerably better scores could be achieved if the models had run for another hundred epochs.

It is interesting to see that the learning process of UNet 1 was more stable than that of UNet 2.0, as well UNet 1.0 reaches 75% accuracy mark faster than UNet 2.0, but still ends up performing a bit worse.

Compared to UNet1.0, UNet2.0 have spikes with regular interval at approximate 10 epochs. This might be caused by combination of the learning rate finder, batch normalization, Adam and the learning rate scheduler. Since UNet1.0 does not have any batch normalization, that might be what is causing these spikes. It also looks as the initial learning rate for UNet2.0 was too high, when seeing the big spikes in the 25 first epochs.

UNet 2.0 did use batch normalization and it may have come in handy, but also came with the price of a slower training phase, namely half as fast as training UNet 2.0 without batch normalization. After training and validation, a test was done on the same test data, where UNet 2.0 was more than 4,04 % points better than the result for UNet 1.0, a 6,52 % relative improvement. Another and perhaps more useful test-case for these networks, is in the biomedical domain, as this is what UNets in general are being used and designed for. Having UNet 2.0 tested in that domain would make it possible to compare our results to other UNets benchmarked on the same data.

It should also be noted that the training was conducted only once per network, meaning that there's potential for improvement on this end as well, especially considering the turbulent start of UNet 2.0's training phase.

The 8 image examples shows little improvement versus the two networks, but the results are positive. Even with the improvements yielding UNet2.0 its increased capacity still only becomes better at recognizing cats, without recognizing the "bad" examples from figure 2.

5 Conclusion

It can be concluded that UNet 2.0 was able to outperform UNet 1.0 by more than 4 % points with test scores of 66,02 % and 61,98 % respectively. One may wonder if the picture would remain the same had they trained for longer time, since UNet 2.0 fluctuated a lot in the later epochs albeit with an improving tendency. Even though the image domain was changed from biomedical images to cat images, it seems that the concepts used in MultiResUNet (UNet 2.0) made a positive impact nonetheless.

References

[1] N. Ibtehaz and M. S. Rahman. Multiresunet : Rethinking the u-net architecture for multimodal biomedical image segmentation. *Neural Networks*, 121:74 – 87, 2020.

[2] J. Jordan. An overview of semantic image segmentation., Nov 2020.

[3] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015.

[4] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[5] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

A Comparison of UNet 1.0 and UNet 2.0

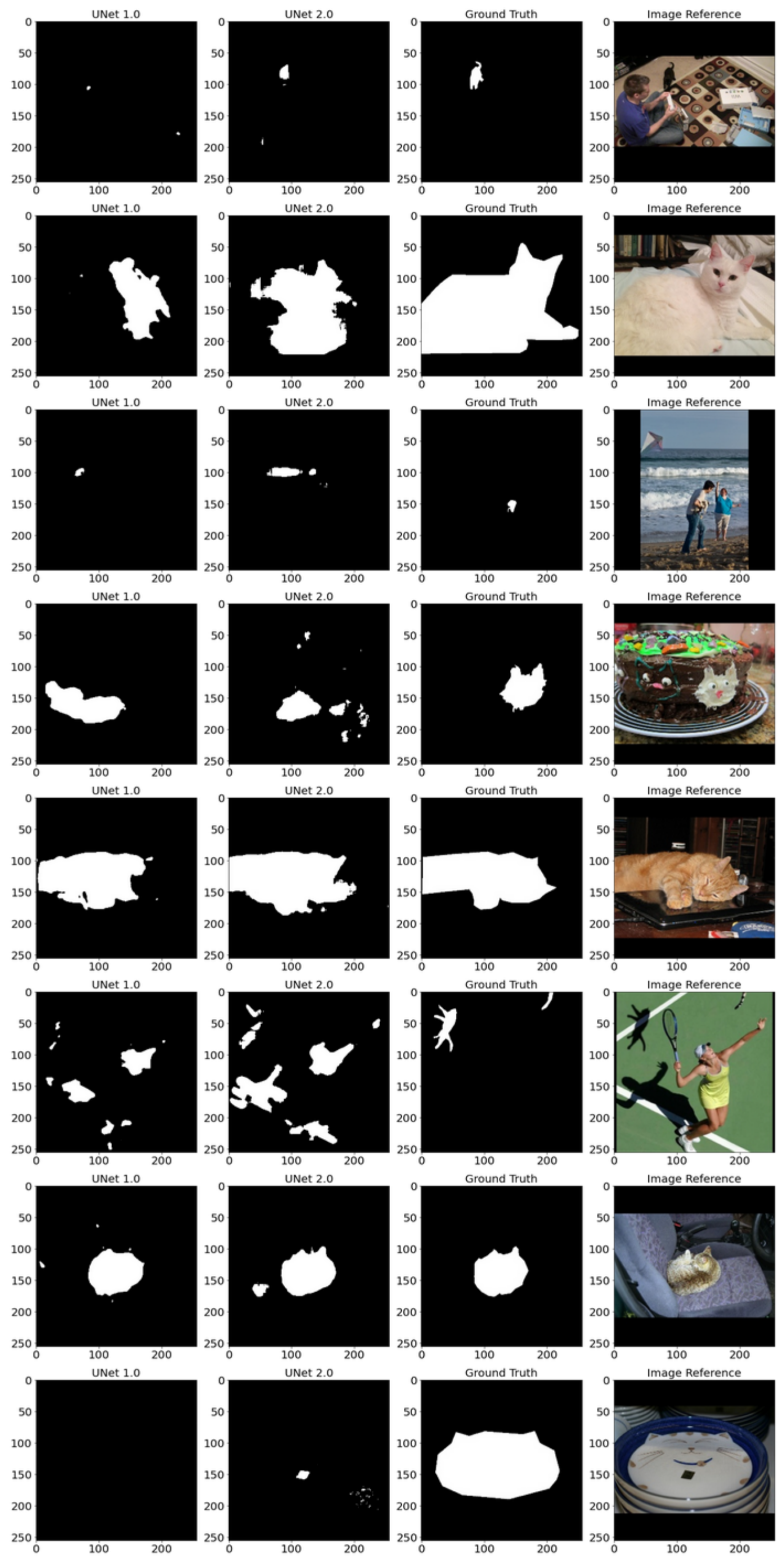


Figure 12: The 4 bad images from figure 2 and the 4 different size images from figure 7, evaluated on UNet1.0 and UNet2.0