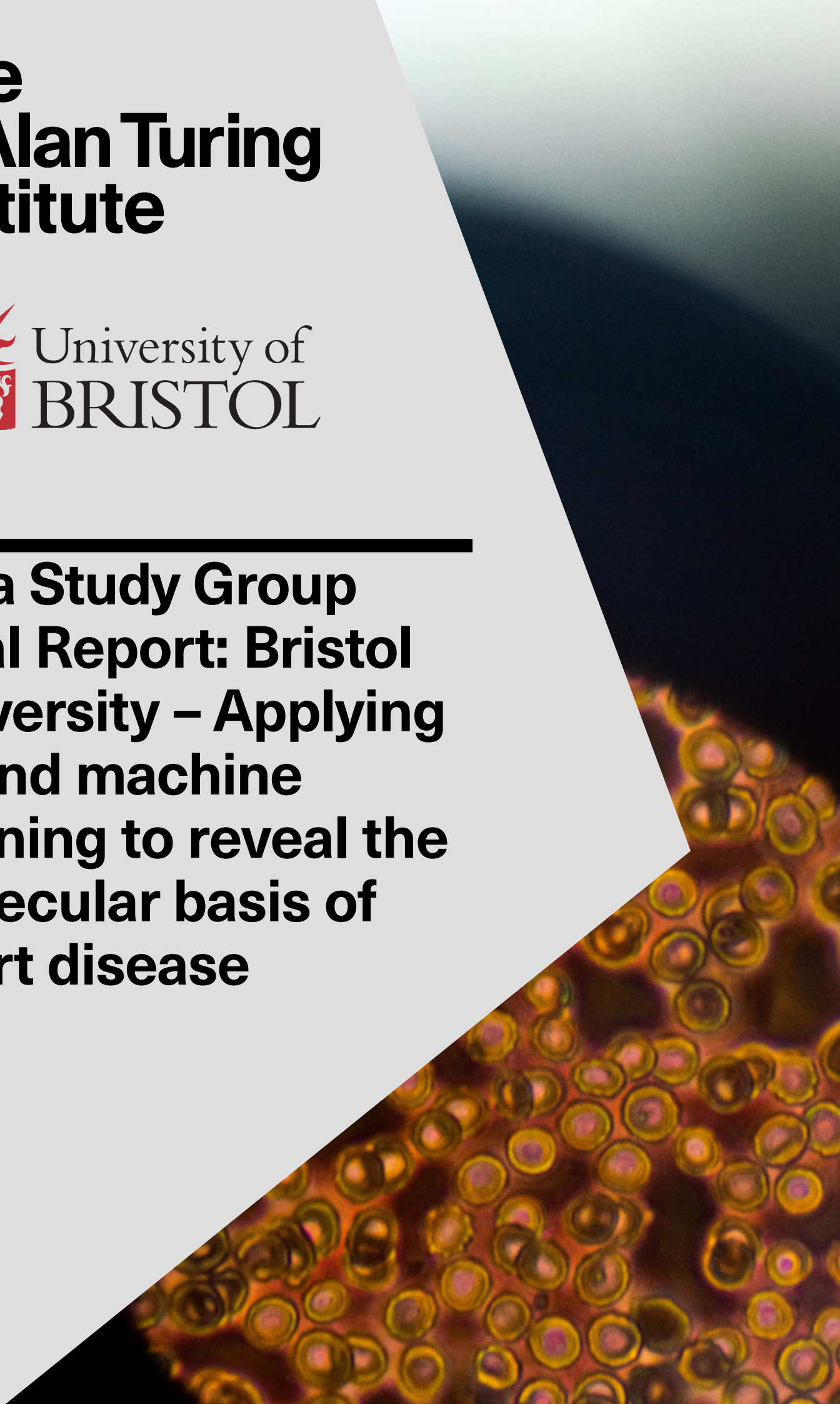
The Alan Turing Institute

University of BRISTOL

Data Study Group Final Report: Bristol University – Applying AI and machine learning to reveal the molecular basis of heart disease



Automated Actin Filament Detection in Cryo Electron Microscopy Images Using Image Segmentation Deep Neural Networks

1. Executive summary

1.1. Challenge overview

Hypertrophic Cardio Myopathy (HCM) is an inherited heart disease caused by mutations in certain proteins in the heart. The protein troponin contains 15-20% of the known mutations linked to HCM, including some of the most severe ones. It is necessary to determine positions of these mutations and establish their effect on muscle function. In Figure 1 below, Cryo Electron Microscopy (Cryo-EM) images of troponin located along an actin filament can be seen. The filament is composed of two strands of actin protein twisted around each other. With hundreds of thousands of images of troponin on its own, a high-resolution molecular model can be created. However, the detection of troponin in these images is a semi manual process. The labelled data used in this challenge represent 60GB of raw data and 6 months of data preparation. The current experiments output 40TB of data, which highlights the importance of automation. Machine learning methods has been used before to identify globular/spherical protein from images (Wagner, T.F., *et al.* 2019), but not for string-shaped objects like the actin filaments in Figure 1.

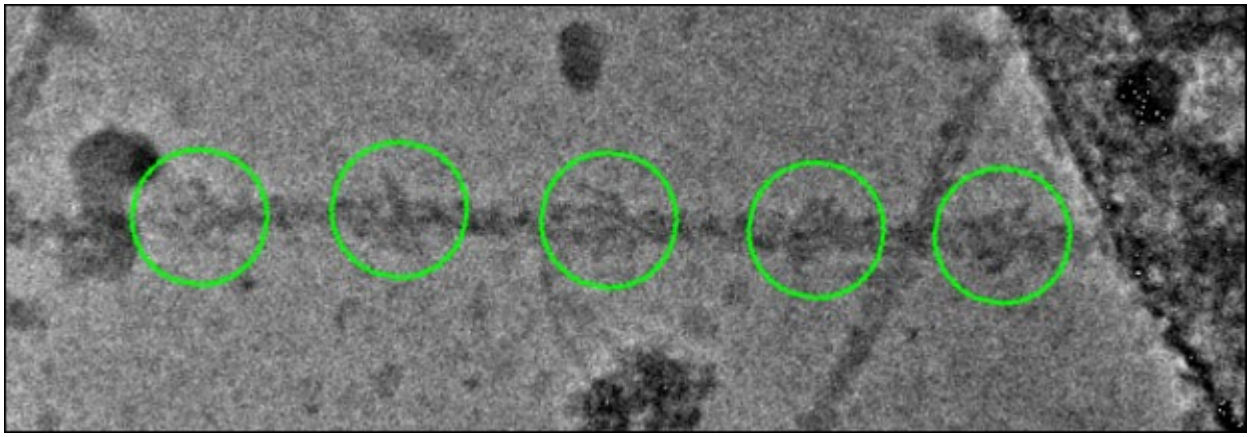


Figure 1: Cryo-EM image of troponin protein (highlighted via green circles) attached to actin filaments.

This report presents the output of the Cryo-EM challenge organised by the Alan Turing Institute and Jean Golding Institute as part of the Turing Data Study Group event in Bristol, August 2019. The aim of this challenge was to create an automated image processing workflow of actin filament identification, straightening, and extraction. This would be an important step towards detecting the troponin located on top of these actin filaments.

1.2. Data overview

The data was comprised of 100 grayscale Cryo-EM TIF images at roughly 4Kx4K resolution and the coordinates of troponin and actin. These coordinates had been obtained by visual inspection and hand labelling by people with relevant expertise.

The troponin coordinates were supplied for all images, whereas the actin locations were supplied for seventy-four images only. Also, for these seventy-four images, a corresponding "mask" image was provided where black and white pixels indicated presence and absence of actin filaments respectively.

1.3. Main objectives

The main aims of this challenge were the automation of actin filament identification, straightening, and extraction. The work in this report focused only on the filament identification part of this challenge.

1.4. Approach

The work on this report approached the filament detection as an image segmentation problem. In image segmentation, each pixel of an image is classified into one of the possible classes (e.g. "filament" and "not filament"). This task was approached five different ways, with different libraries and different pre-processing steps a slightly different neural network architectures. These different approaches were: VGG16-PSPNet, vanilla UNet, Nested-UNet, Fast AI UNet, and UNet with augmentation. The performance of the different algorithms was assessed using metrics available within the libraries used. These metrics included training and validation accuracies, training and validation losses, and F1 Scores.

It was not possible to approach this as a troponin detection challenge as the troponin is quite small relative to the size of a typical Cryo-EM image, and there were not enough labelled examples of them. Image classification approaches could tell if there is an actin filament in the image, but not necessarily its location. On the other hand, object detection algorithms like YOLO (You Only Look Once) are able to put a bounding box on an object within an image and classify it; however, the filaments are difficult to fit into a rectangle as they are long, string-shaped objects with many curves.

1.5. Main Conclusions

The work in this report demonstrates that image-segmentation models can reliably identify actin filaments in Cryo-EM images. The accuracy of these models varied between 82-91% with F1 score of 0.93, which indicates high precision and recall for the models.

1.6. Limitations

1. In order to feed images with high pixel density per image into deep neural networks, image pre-processing methods such as resizing and rescaling need to be employed. Such methods may lead to information loss.
2. Due to time restrictions, it was not possible to derive identical metrics for model comparison. For each model, different metrics are provided. An example prediction from each model is presented so that a qualitative comparison can be made.
3. The findings may not be generalisable to other datasets. The data used comes from one protein system and one instrument; therefore, all images have the same resolution and dimensionality. Better performance may be achieved with more diverse data from a wider range of biological systems.

1.7. Future work

A different number of avenues for future research were identified, including post-processing of the predicted masks, instance segmentation methods, explanatory Artificial Intelligence (AI) and neural network compression.

2. Quantitative problem formulation

2.1. Problem statement

In order to obtain high resolution spatial models of troponin, hundreds of thousands of images are required. There is no automated image processing pipeline to identify the troponin or the actin filaments. Processing 60GB of data can take 6 months of semi-manual data processing and this does not scale with 40TB of data obtained from some Cryo-EM experiments.

The identification of the actin filaments would be an important step in the image processing pipeline towards the overall aim of automated troponin detection. Computer vision methods have already been used for detection of globular/spherical proteins but not for string-shaped proteins like the actin filaments.

2.2. Data

2.2.1. Data overview

The data comprises 100 Cryo-EM images from zebrafish heart tissue. These images were greyscale, 3838×3710 pixel TIF images. These TIF images were converted to PNG during the challenge for easier ingestion into neural network packages. Seventy-four of these images have been previously annotated such that the coordinates and indices of filaments have been identified. This information was provided in SVG format, as well as "mask" image in 3838×3710 pixel PNG files. In these mask images, black and white pixels corresponded to filament and not-filament respectively. For each of the 100 images, there was an csv file, which specified the positions of troponin molecules within the images. The data available for this challenge can be seen in Table 1 below. The work in this report only used the Cryo-EM images and the filament masks, both in PNG format.

Table 1. Data Description.

Description	Data type	size (pixels)	number
Cryo-EM image	Image (TIF, PNG)	3838×3710	100
Filament mask	Image (PNG)	3838×3710	74
Filament locations	X,Y coordinates (SVG)		74
Troponin locations	X,Y coordinates (CSV)		100

2.2.2. Dataset description and visualisation

An example of the Cryo-EM images and their corresponding mask images can be seen below in Figure 2. Such side by side plots were created for many images to make sure each mask matched with the correct Cryo-EM image. This image can be reproduced via the code below:

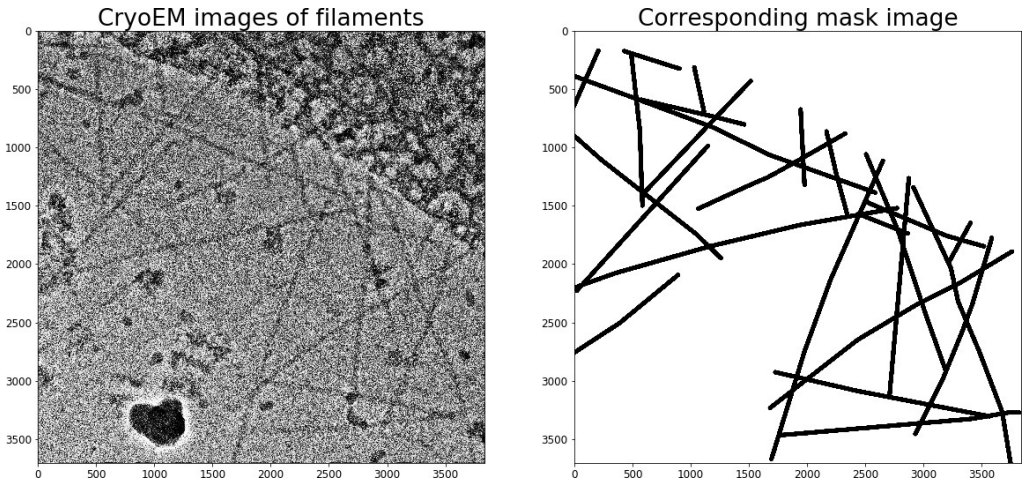


Figure 2. Left: original Cryo-EM image; **Right**: manually labelled mask of filaments.

```
import matplotlib.pyplot as plt
from glob import glob
from PIL import Image
import numpy as np

image_list = glob("/data/filament_picking/*.png")
```

```
mask_list = glob("/data/masks/*.png")
image_list = sorted(image_list)
mask_list = sorted(mask_list)
print(len(image_list))
print(len(mask_list))

img1 = Image.open(image_list[0])
img2 = Image.open(mask_list[0])

fig = plt.figure(1,figsize = (20,10))
ax1 = fig.add_subplot(121)
plt.imshow(img1, cmap='gray')
ax1.title.set_text("CryoEM images of filaments")
ax2 = fig.add_subplot(122)
ax2.title.set_text("Corresponding mask image")
plt.imshow(img2)
#plt.rcParams.update({'font.size': 22})
plt.rc('figure', titlesize=42) # fontsize of the figure title
plt.rc('xtick', labelsize=12)    # fontsize of the tick labels
plt.rc('ytick', labelsize=12)    # fontsize of the tick labels
plt.savefig("comparison_of_mask_and_image.png",format='png')
plt.show()
```

2.2.3. Data quality issues

There were limitations associated with the available data including the presence of noise and contamination. Apart from normalisation, nothing else was done to mitigate these problems. Since the data was manually annotated there might be some missing troponin. In addition, the cryo-EM data contain artefacts which can impede filament identification by presenting as false positives.

3. Exploratory data analysis

Detailed exploratory data analysis was not carried as the format of the data did not require it.

4. Image segmentation methods

Image segmentation is a collection of methods that help classify each pixel in an image to a particular class. For example, detecting cars and pedestrians in an image.

All algorithms were implemented in a Microsoft Azure VM, within the shared Linux system (standard D8s v3, 8 vcpus, 32 GiB memory), with four GPUs (standard NC24_Promo, 24 vcpus, 224 GiB memory), NVIDIA Tesla K80, NVIDIA-SMI 418.67, Driver version: 418.67, CUDA version: 10.1.

The data were split into training and test sets with an 80/20 split using `scikit-learn`'s `model_selection` module. To ensure reproducibility, the random seed was set to 42. This resulted in fifty-nine training images and masks and 15 test images and masks. The same training and test sets were used for all models described hereafter.

The following semantic segmentation methods were implemented: 1) VGG16-PSPNet; 2) Vanilla UNet; 3) Nested-UNet; 4) UNet with data augmentation; and 5) Fast AI Unet . All methods tried to separate pixels into one of two categories "filament" and "not filament".

The metrics quoted for each study is dependent on the code base in which they were deployed. Due to time restrictions, it was not possible to obtain the same metrics across all models. Specific implementation details for each study is presented within their own subsection

4.1. VGG16-PSPNet

4.1.1. Description

The Python library `keras-segmentation` (https://github.com/divamgupta/image-segmentation-keras) was used to provide an implementation of a VGG16-PSPNet Deep Neural Network. This library provides a simple interface to a number of implementations of various semantic image segmentation algorithms.

4.1.2. Dummy run

To ensure that this library functioned correctly, a VGG-Unet model with 51 classes and image heights and widths of 416 and 608, respectively, was ran as per the tutorial published in the repository. The algorithm was evaluated using training accuracy and training loss, and it yielded a training accuracy 0.8795 and training loss of 0.3847.

4.1.3. Data preparation

The training and test images were resized to 960 x 960 pixels using `scikit-image`'s `transform` module. The image masks were intensity scaled from 0 to 255 to 0 to 1, since this is a requirement of `keras-segmentation` library. Note that `rescale_intensity` function performs spline interpolation and antialiasing as part of the resizing. No other manipulations or transformations were applied to the data before it was used to train and test the models.

The following code was used to perform the resizing and rescaling:

```
import os
from skimage import io
from skimage.exposure import rescale_intensity
from skimage.transform import resize

for f in os.listdir('/data/test/masks/rescaled'):
    src = os.path.join('/data/test/masks/rescaled', f)
    dest = os.path.join('/data/test/masks/resized_rescaled', f)
    if not f.endswith('.png'):
        continue
    image = io.imread(src)
    rescaled = rescale_intensity(image, in_range=(0, 255), out_range=(0, 1))
    resized = resize(rescaled, (960, 960))
    io.imsave(dest, resized)
```

4.1.4. Model setup and execution

Having determined that the library was functioning correctly, a VGG-UNet neural network was setup and run over five and ten epochs, 512 steps per epoch, and all other settings set to library defaults (default optimiser was Adam). The two models were evaluated by the training and validation accuracies and losses. The validation metrics were obtained predicting against the test set of images and masks, whilst ensuring that this set did not influence the training of the model.

4.2. Vanilla UNet

4.2.1. Description

A GitHub repository (https://github.com/bigmb/Unet-Segmentation-Pytorch-Nest-of-Unets) that contains the Pytorch implementation of the UNet and its variations was used for this method. Besides pytorch, the code requires the installation of other packages including torchvision, tensorboardX, and natsort.

4.2.2. Dummy run

The code from the above repository was first run using a dummy dataset that contains 59 serial section Transmission Electron Microscopy (ssMET) images from the ISBI Challenge: Segmentation of neuronal structures in EM stacks

(https://brainiac2.mit.edu/isbi_challenge/). Following the instructions in https://github.com/zhixuhao/unet, 30 images for training and 29 for testing were used and a F1 score of 0.9062 was obtained with learning rate of 0.0005 after 15 epochs.

4.2.3. Data preparation

To feed the Cryo-EM images into the model, the images were resized down to 256 pixels by 256 pixels by using the `transform.resize` function in pytorch (https://pytorch.org/docs/stable/torchvision/transforms.html). This function implements a bilinear interpolation as part of the resizing. The images were also normalised by subtracting the mean value and dividing the images with the standard variation of the dataset.

4.2.4. Model setup and execution

To run the model with Cryo-EM images, the data path specified in pytorch_run.py was updated and test set was sampled for evaluation after each epoch. The hyperparameters used can be found below in Table 2. Adam was used as the optimiser.

Table 2. Hyperparameters chosen for Vanilla UNet model.

Model No.	Batch size	learning rate	num. of epoch	algorithm
1	4	0.001	15	UNet
2	4	0.001	15	Nested-UNet

4.3. Nested-UNet

4.3.1. Description

Nested UNet is a encoder-decoder network where the encoder and decoder are linked with several nested, dense skip pathways (Ronneberger et al., 2015).
It is designed for medical image segmentation and we are using the same github repository as for Vanilla UNet that contains the Pytorch implementation of the Nested-UNet.

4.3.2. Dummy run

We tried this method on the dataset from the International Symposium on Biomedical Imaging Challenge, which is described in the previous section.

The F1 Score achieved was 0.9375 for Nested-UNet with a learning rate of 0.0005 after 15 epochs.

4.3.3. Data preparation

The data was resized and prepared as described in section 4.2.3.

4.3.4. Model setup and execution

The path in the `pytorch_run.py` script was changed as described in Sect. 4.2.4.
We run this algorithm with the Cryo-EM using the parameters described in 4.2.4. Note that Adam was used as the optimiser.

4.4. UNet with data augmentation

4.4.1. Description

For this implementation, an image segmentation repository https://github.com/zhixuhao/unet was used. It is an implementation of UNet based on this research paper by Ronneberger, *et al.* 2015. The data used in this repository was very similar greyscale medical images, similar to the data in this challenge. This model is based on Keras. TensorFlow backend was used for this report, but it might work with other backends.

The method is applicable to small datasets thanks to the data augmentation by means of standard Keras Image Pre-processing tools such as rotation, shear, zoom, horizontal shift. Augmenting the data might be good idea for us since we don't have that many unique images.

An error was encountered where all the outputs were grey. The fix was manually implemented following github issue https://github.com/zhixuhao/unet/issues/132.

4.4.2. Dummy run

The model was tested using the data provided in the repository. The only default settings changed was `steps_per_epoch`, which was changed from 2000 to 200. A loss of 0.34, accuracy of 0.8529 was obtained from the training set of the repository's data.

4.4.3. Data preparation

The train test split as defined in the methodology section. The images were not manually resized or normalised; however, images were resized to 256x256 pixels as part of Keras's own pipeline using `keras.preprocesing.image.flow_from_directory()` . By default, this function performs nearest interpolation when resizing. Only one colour channel was used, as the images were grayscale.

4.4.4. Model setup and execution

We ran the model on our training data with a few different settings for steps per epoch, number of epochs. We also ran with and without data augmentation. The different values used and their result can be seen in Table 5. Note that Adam was used as the optimiser.

4.5. Fast AI UNet

4.5.1. Description

Fast AI is based on PyTorch and allows the users to design a neural network architecture, train and validate the model with a few lines of code. This library also provides different visualisation methods for interrogation and interpretation of the results.

4.5.2. Dummy run

An example shown on Towards Data Science was followed in this test with slight extensions as some variables were missing their initialisation. More training for this library is available from Practical Deep Learning for Coders. This example used the road images from the University of Cambridge, CamVid. The preliminary test used U-Net learner and it has been successfully completed with approximately 90% training accuracy and loss of 0.39 while the validation loss was 0.32 after 10 epochs, where an encoder was chosen to be Resnet34. The initial learning rate was set to 3e-03 based on loss evaluation, which is also a functionality provided by Fast AI (it also seems that Fast AI embraces dynamic learning rates). Considering this promising output, we have tested then this approach using our image data.

4.5.3. Data preparation

The images were resized, and masks were resized and rescaled as described in Section 4.1.3. The training dataset of 59 images as described in Section 4.1.3 to have further been split into 47 images for training and 12 for validation by the algorithm. The images were also further manipulated (e.g. normalised) as shown in Towards Data Science.

4.5.4. Model setup and execution

As compared to the dummy test, we have only two main classes such as **Yes** - there are filaments on the image, or **No** - no filaments on the image. It has to be mentioned that the dummy test also had **Void** class that would define anything else except the objects of interest. In our case, we have defined this class as **Noise**, and this class was excluded from target to mask comparison when the training accuracy was calculated. There were also other changes as compared to the example settings such as the data bunch was reduced from **2** to **1** due to extensive memory consumption. The initial learning rate was chosen to be **3e-04** after a few tests but not using the Fast AI functionality for technical reasons. Note that Adam was used as the optimiser.

5. Results

5.1. VGG16-PSPNet

A VGG16-PSPNet Deep Neural Network was trained with five and ten epochs, each with 512 steps per epoch, using the training set derived as described above. Model performance was assessed using training and validation accuracies and losses as shown in Table 3. These results demonstrate that doubling the number of steps of the network gives slight increases in training and validation accuracies, a fair decrease in training and validation losses.

Table 3. Summary of metrics obtained from training a VGG16-PSPNet model with five and ten epochs.

Model	Epochs	Training Accuracy (%)	Validation Accuracy (%)	Training Loss	Validation Loss
VGG16-PSPNet	5	87.78	87.93	0.2323	0.2615
VGG16-PSPNet	10	89.03	88.01	0.2058	0.2484

The small increase in validation accuracy between the two models, demonstrates that there is little value to using ten epochs over five and further training of this model with additional epochs would add little value to the model. Figure 3 shows a comparison between the raw Cryo-EM images, the manually produced masks and the masks predicted from two of the test images using the model trained over 10 epochs. As can be seen from the figure, the manually derived masks are reproduced to a high level in the predicted masks, indicating that this is a promising starting point. The quality of the predictions is such that resultant masks could be fed into a second post-processing step, in which the filament paths are delineated from them.

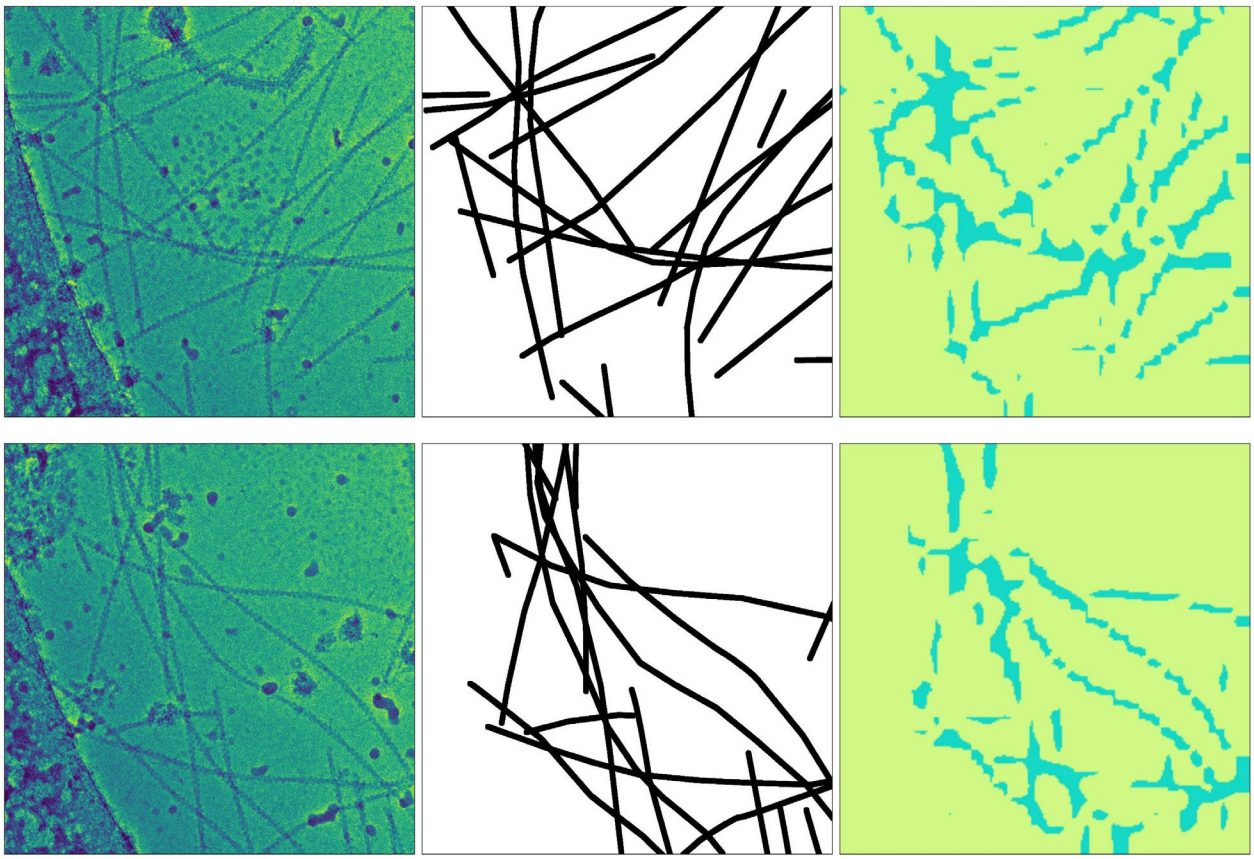
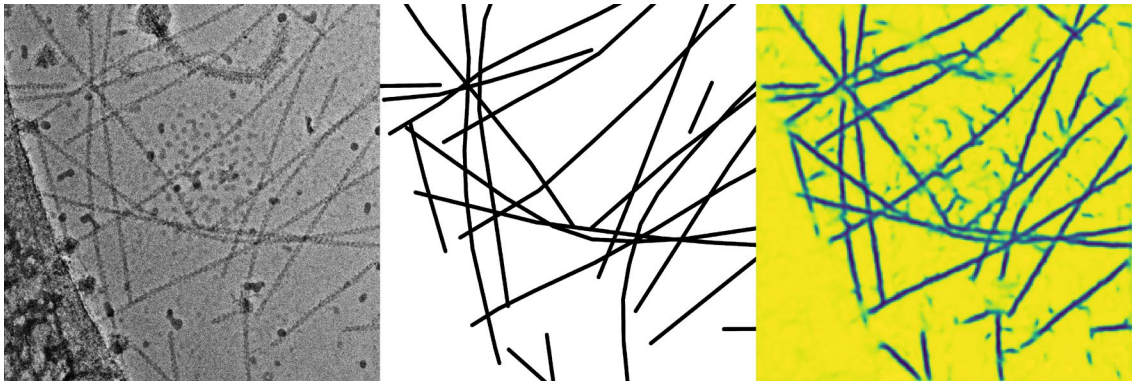


Figure 3. Comparison of original Cryo-EM images (left), manually annotated masks (center) and the masks predicted by the VGG16-PSPNet network trained over ten epochs (right).

5.2. Vanilla UNet

For Cryo-EM images, the model is trained with a batch size of 4 and learning rate of 0.001 for 15 epochs. The visual results can be found in the Figure 4 below and the quantitative results can be found in the Table 4, in Section 5.3.



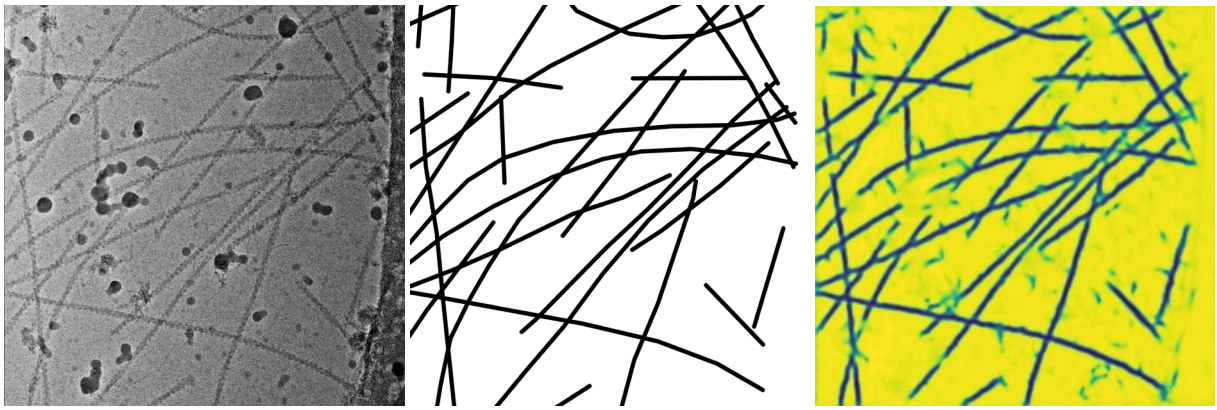


Figure 4. Qualitative results produced by vanilla UNet on test images. From left to right: original image, input mask, and generated mask.

5.3. Nested-UNet

The accuracies, F1 scores and validation losses on the test set for our 15 epoch-run for Vanilla-UNet and Nested UNet are shown in the Table 4 below.

Table 4. The table shows the performance metrics for Vanilla- and Nested-UNet.

Model No	Validation Loss	F1 Score	Accuracy [%]
Vanilla-UNet	0.1384	0.9296	82.20
Nested-UNet	0.1214	0.9322	82.29

The comparison between the test image, the test mask and the predicted image is shown below in Figure 5.

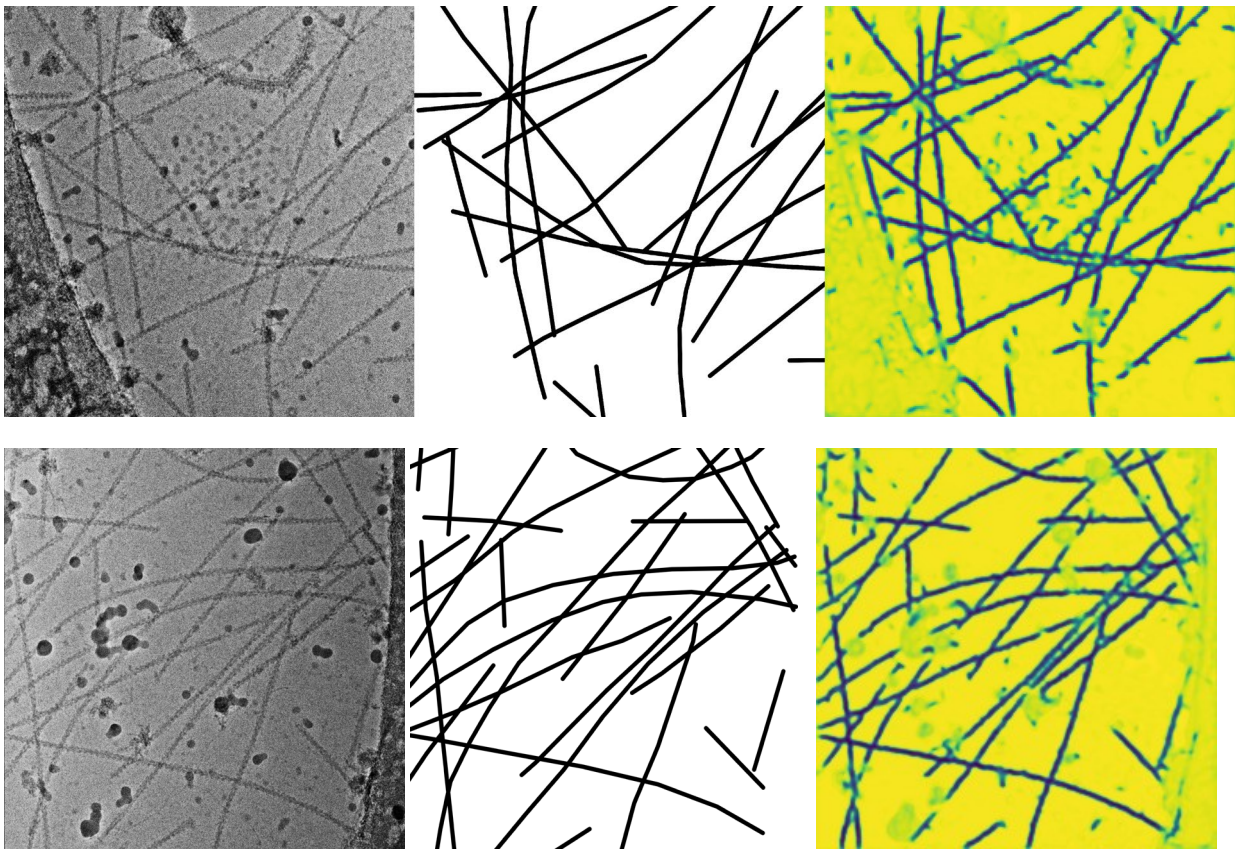


Figure 5. *Left column:* original test image, *middle column:* mask image, *right column*: Nested-UNet predicted mask for two different test images.

5.4. UNet with data augmentation

Training time for models were no longer than 30 minutes running on one GPU.

Batch size of 2 images was used. It is not clear with `steps_per_epoch=59` if this results in training on 59 images per epoch or 118.

Due to unfamiliarity with Keras generators, it was not possible to calculate good metrics such as test and validation accuracy or F1 scores. The results can be seen in Table 5 below. The best result seems to appear with data augmentation. Since the test data accuracy was not calculated, it is likely that overfitting is present. Model 4's predictions can be seen in Figure 6.

Table 5. Summary of some UNet models attempted. The best result seems to appear with data augmentation.

Model No	Data aug.	Steps per epoch	Epochs	Training loss	Training accuracy
1	No	59	5	0.3358	0.8769
2	No	59	15	0.6623	0.8767
3	Yes	200	5	0.3150	0.8775
4	Yes	500	15	0.1197	0.9581

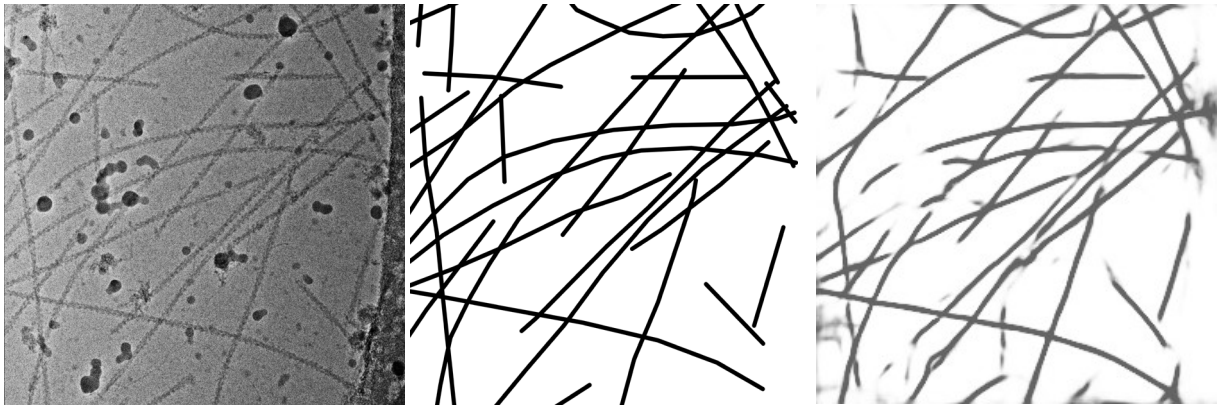


Figure 6. *Left column:* original test image, *middle column:* mask image, *right column*: predicted mask using model 4.

5.5. Fast AI UNet

It was challenging to run this model due to extensive GPU memory consumption, especially, when only two classes were considered. The large amount of memory seemed to be required for model evaluation, and it was reduced when a new class 'Noise' was introduced and excluded from this evaluation. Due to unfamiliarity with the library, it was not clear how data is batched and processed. The preliminary results that we received from the Fast AI UNet model with three classes can be seen in Table 6 below. The mask prediction output can be seen in Figure 7 below.

Table 6. This table shows the performance of Fast AI using U-Net learner.

Epochs	Training loss	Validation loss	Accuracy
10	0.24	0.20	0.91

Figure 7. This figure shows the original image (left), its original mask (centre) and Fast AI UNet predicted mask (right).

6. Limitations

1. A common method for image detection tasks is to train with bounding boxes, one for each object. However, filaments shapes vary greatly, and overlap, which makes their bounding boxes challenging to unequivocally identify. Therefore, this method is not suitable for our case.
2. The dataset may be biased, which may negatively impact the performance of classification and segmentation models, due to the fact that high quality images may have been intentionally selected.
3. The use of manually annotated images could compromise the accuracy of the model since we cannot guarantee that all images have been completely and comprehensively annotated.
4. In order to feed images with high pixel density per image into deep neural networks, image pre-processing methods such as resizing and rescaling need to be employed. Such methods may lead to information loss.
5. Our findings may not be generalisable to other datasets as we used one dataset from one protein system recorded on one instrument, therefore all images have the same resolution and dimensionality. Better performance may be achieved with more diverse data from a wider range of biological systems.
6. The lack of sufficient GPU memory may be challenging for some deep learning packages, such as Fast AI. Generators could be used to train, validate, and predict models in order to overcome this problem.
7. Each method provides different metrics and therefore direct comparison across the methods implemented was not possible due to the time effort required to derive identical metrics for comparison.

7. Discussion

We used several different semantic segmentation algorithms to detect actin filaments in Cryo-EM micrographs of zebra fish heart tissue. We considered both classification and segmentation algorithms. Classification approaches would require cutting images into bounding boxes, standardising size, test and training set split, as well as labelled filaments and noise.

We have demonstrated the utility of several different methods for the reliable identification of actin filaments in the cryo-EM data. Visual comparison of the predictions with test data show that all the models can identify large portions of the filaments. The models obtain between 82% and 95% accuracy; however, the large class imbalance in this dataset mean that perhaps the more important metric is the F1 score. For the Nested U-net model, the F1 score was 0.93, which indicates high precision and recall. This metric combined with the visual comparison shows that image segmentation can be the correct tool for this problem.

All of these models used in this study were not optimised to a high degree, rather they were evaluated, on the whole, using default parameters supplied by their implementations. In some cases, hyperparameter tuning was attempted, e.g. with the Vanilla and Nested UNet implementations. Further optimisation could yield superior performance. Moreover, systematic comparison between the methods with the same parameters would furnish us with an unbiased view of model performance. Finally, further work could possibly benefit from larger convolution filter size to account for presence of long filaments.

Due to time constraints, troponin detection was not attempted, but potential methods were explored in the future work section below. In addition, the limited time did not allow us to explore the identification of individual instances of filaments. Algorithms

such as YOLOv3 could identify troponin.

Following the prioritisation of segmentation methods to solve this challenge, we considered the following methods: reseg and DeepLabv3. These methods were considered but were not fully implemented due to some technical limitations described below.

ReSeg is a recurrent neural network for semantic segmentation (Vasin et al., 2016, https://arxiv.org/abs/1511.07053). The implementation of ReSeg was not very user friendly and the weights were not provided. The relevant repositroy is: https://github.com/fvisin/reseg.

We also attempted to implement DeepLabv3+ (Chen et al., 2018). However, due to the memory bottleneck on the hardware, this method could not be executed.

Finally, we considered alternative methods such as CapsNet, InceptionV3, and Transfer Learning. However, we elected to prioritise segmentation methods rather than classification methods.

8. Future work and perspectives

Future work may involve systematic evaluation of the aforementioned semantic segmentation methods, selection of best performing model, and validation to an independent larger dataset.

All presented methods involved pixel classification into a category of objects (e.g. cars, people or in our case filaments). A further avenue to explore would be instance rather than semantic segmentation. Instance segmentation further assigns each pixel to one object instance. In other words, it distinguishes between one filament and another. Possible examples of instance segmentation algorithm are Mask RCNN (He et al., 2017), ildoonet (https://github.com/ildoonet/data-science-bowl-2018) and PARMAGroup UNet Instance Cell segmentation code (https://github.com/PARMAGroup/UNet-Instance-Cell-Segmentation).

Another potential future step would be the isolation of individual filaments from the masks predicted from our evaluated semantic segmentation models (e.g. Hough transform).

Alternative methods to address this segmentation problem could be Generative Adversarial Networks (GAN) that are able to produce realistic images by optimising a mini-max game between the generator and the discriminator; PixelVAE is a variational autoencoder method, which could be used for image segmentation (I. Gulrajani et al. 2016); and the probabilistic U-Net (https://arxiv.org/abs/1806.05034; https://arxiv.org/pdf/1905.13077.pdf;https://github.com/SimonKohl/probabilistic_unet).

It could be possible to improve the accuracy of the outputs from the image segmentation models by using post-processing steps such as the filters available on skimage.

Interpretability of deep learning methods is an important topic. Future work can incorporate state-of-the-art methods for interpreting DNNs such as LIME (Ribeiro et al., 2016) and an information theoretic approach (Yu et al., 2018). The information theoretic approach uses the concept of Renyi entropy to explain optimality of parameters with which a neural network yields the best results.

Future work can also address computational limitations associated with applying deep learning, which could be overcome by compressing the neural networks by connection pruning (Liu et al. 2017), weight clipping, weight quantization (Kwok et al. 2018), activation quantization (Choi et al., 2018), gradient quantization (Aji et al. 2017) or regularization. These methods can enable more efficient training and memory usage.

9. References

Wagner, T.F., *et al.* 2019. SPHIRE-crYOLOisafastandaccuratefully automatedparticlepickerforcryo-EM. Communications Biology

Aji, A.F., *et al.* 2017. Sparse communication for distributed gradient descent. arXiv preprint arXiv:1704.05021.

Chen, L.C., *et al.* 2018. Encoder-decoder with atrous separable convolution for semantic image segmentation. In Proceedings of the European conference on computer vision (ECCV) (pp. 801-818).

Choi, J., *et al.* 2018. Pact: Parameterized clipping activation for quantized neural networks. arXiv preprint arXiv:1805.06085.

Chollet, F., 2017. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258).

Gulrajani, I., *et al.* 2016. Pixelvae: A latent variable model for natural images. arXiv preprint arXiv:1611.05013.

He, K., *et al.* Mask RCNN, arXiv preprint arXiv:1703.06870.

Hou, L., *et al.* ICLR 2019. Analysis of Quantized Models.

Hou, L., *et al.* 2018. Loss-aware Weight Quantization of Deep Networks, International Conference on Learning Representations.

Liu, Z., *et al.* 2017. Learning Efficient Convolutional Networks through Network Slimming, ICCV.

Ribeiro, M.T., *et al.* 2016, August. Why should i trust you?: Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144). ACM

Ronneberger, *et al.*, 2015 U-Net: Convolutional Networks for Biomedical Image Segmentation, arXiv preprint arXiv:1505.04597

Visin, F., *et al.* 2016. Reseg: A recurrent neural network-based model for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 41-48).

Yu, S., *et al.* 2018. Understanding Convolutional Neural Networks with Information Theory: An Initial Exploration. arXiv preprint arXiv:1804.06537.

Zhou, Z., *et al.* 2018. Unet++: A nested u-net architecture for medical image segmentation. In Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support (pp. 3-11). Springer, Cham.

10. Team members

Eron Cemal is a Data Science consultant at Techmodal ltd in Bristol. Eron has a physics PhD and is efficent and data analysis, visualisation and is familiar with some deep learning tools and methodologies. Eron helped define the problem by using some python, linux, and deep learning knowledge. He worked on implementation of the data augmented UNet model with Maria.

Vasiliki Chatzi is a Research Associate at the University of Cambridge. Vasiliki uses statistical and machine learning approaches to study brain alterations in psychiatric disorders. Vasiliki facilitated focused and fruitful discussion regarding methods and decisions, helped the team to remain task-oriented, document the implemented methods, troubleshoot technical errors and work together.

Andrin Fluetsch is a PhD student in Astrophysics at the University of Cambridge. In his PhD, he studies the evolution of galaxies using statistical models on multi-wavelength spectroscopic data. Andrin helped prepare the data, ran several UNet and Nested-UNet models, evaluated them and looked into ideas for future work.

Valeriia Haberland is a Senior Research Associate at the University of Bristol. She holds Ph.D in Computer Science from King's College London, and worked in a number of projects with close industry collaboration. Her interests include relational and graph-based database, causal inference and natural language processing. Valeriia helped to identify research directions, suitable implementations of deep neural networks, important limitations and tested the UNet model using Fast AI.

Simon Skinner is a former Structural Biologist, now Software Engineer working at BJSS Ltd and a Data Science hobbyist. He is a Python programmer of 10 years experience, has a PhD in Classical and Paramagnetic NMR spectroscopy from the Universiteit Leiden, and he has published multiple software solutions for biophysical protein analysis. Simon helped to define the problem, pre-processed the data and masks, and ran and evaluated an implementation of VGG16-PSPNet.

Maria Veretennikova is an Assistant Professor at the National Research University Higher School of Economics in Moscow. Maria's research interests include brain data analysis, data mining for medicine, Markov processes and fractional calculus. Maria helped with searching for methods and understanding the algorithms, identifying some useful techniques for further work, and contributed to the augmented U-Net code implementation.

Haoyi Wang is a PhD student in Computer Science at the University of Warwick. His research interests include deep supervised learning, generative models, computer vision and face recognition. He is also a research assistant at Warwick Business School. Haoyi helped to prepare the data, fixed bugs for the pytorch implementation of UNet and Nested-UNet, ran experiments, and evaluated these two models.

turing.ac.uk
@turinginst