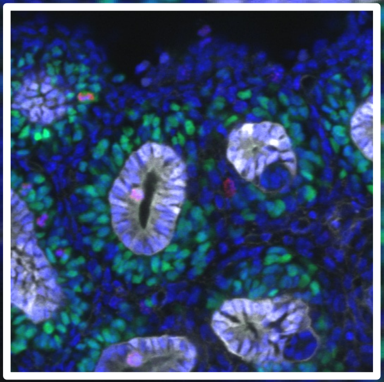
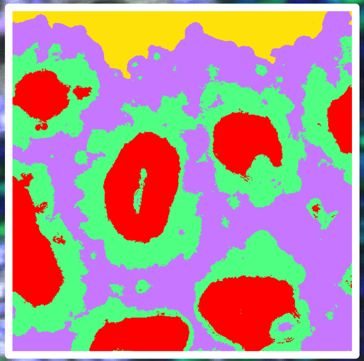
Datasets and slides:
https://github.com/doktor-nick/adv-ML-image-segmentation-with-UNET

Advanced Machine Learning Image Segmentation with UNET



Dr Nick Hamilton
Institute Bio-Mathematician
Institute for Molecular Bioscience
Research Computing Centre
Queensland Cyber Infrastructure Foundation
n.hamilton@imb.uq.edu.au

www.imb.uq.edu.au



Power corrupts.

PowerPoint corrupts absolutely.

- Edward Tufte

Images: Melissa Little Lab

Outline

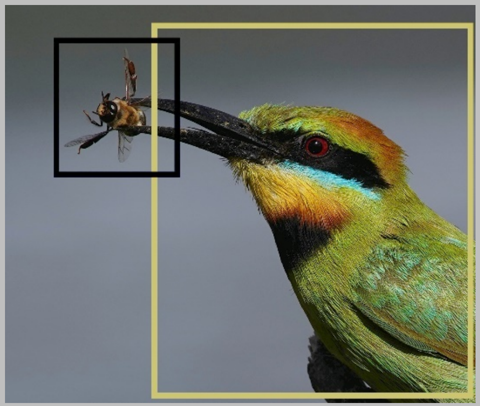
- Brief machine vision applications overview
- Semantic image segmentation
- The UNET architecture for image segmentation
- Setting up Google Colab
- Exercises
  - Introduction for UNET
  - More advanced UNET use

Applications of Computer Vision

- Classification

Rainbow bee-eater

- Object Detection

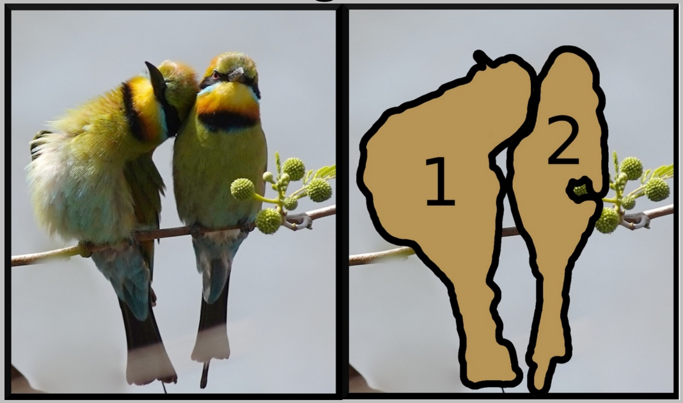


Bee-eater, bee

- Semantic Segmentation

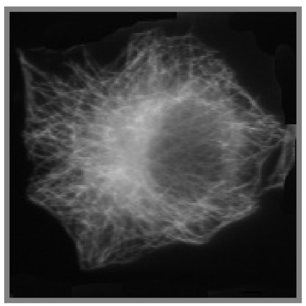
Bee-eater, bee, stick, background

- Instance Segmentation



Bee-eater 1 and 2

Image Classification



=

(0.000772532, 0.00154506, 0.00343348, 0.0139056, 0.0308155,
0.0440343, 0.0337339, 0.0433476, 0.828412, 0.000235031,
0.000352547, 0.00205653, 0.00593454, 0.0223867, 0.0351372,
0.022093, 0.0312004, 0.880604, 0.000420663, 0.00117786,
0.00496382, 0.0165741, 0.058388, 0.107017, 0.0904425,
0.110803, 0.61024)

Cell Cytoskeleton

Convolutional networks are typically good at extracting features for classification.

See “Introduction for Deep Learning and Tensorflow” Workshop.

Image Segmentation

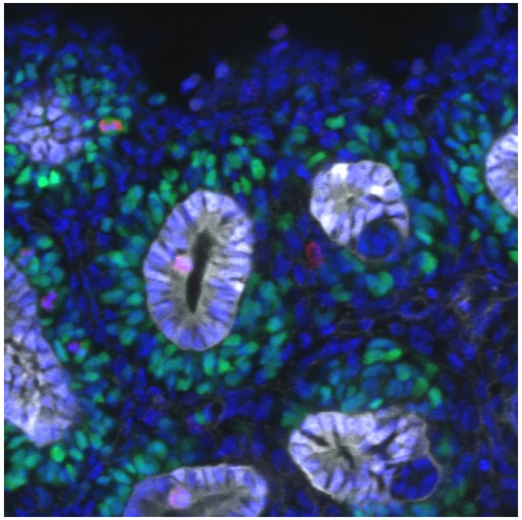
Semantic Image Segmentation

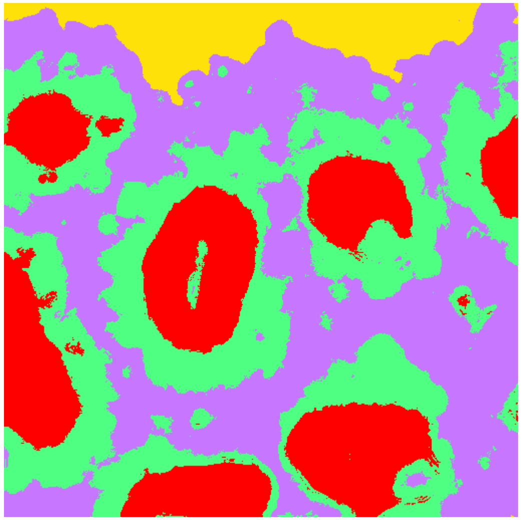
Definition

A segmentation is a partition of an image into regions that label the class of each region. e.g. foreground/background

Approach

Instead of a whole image we want to label/classify individual pixels

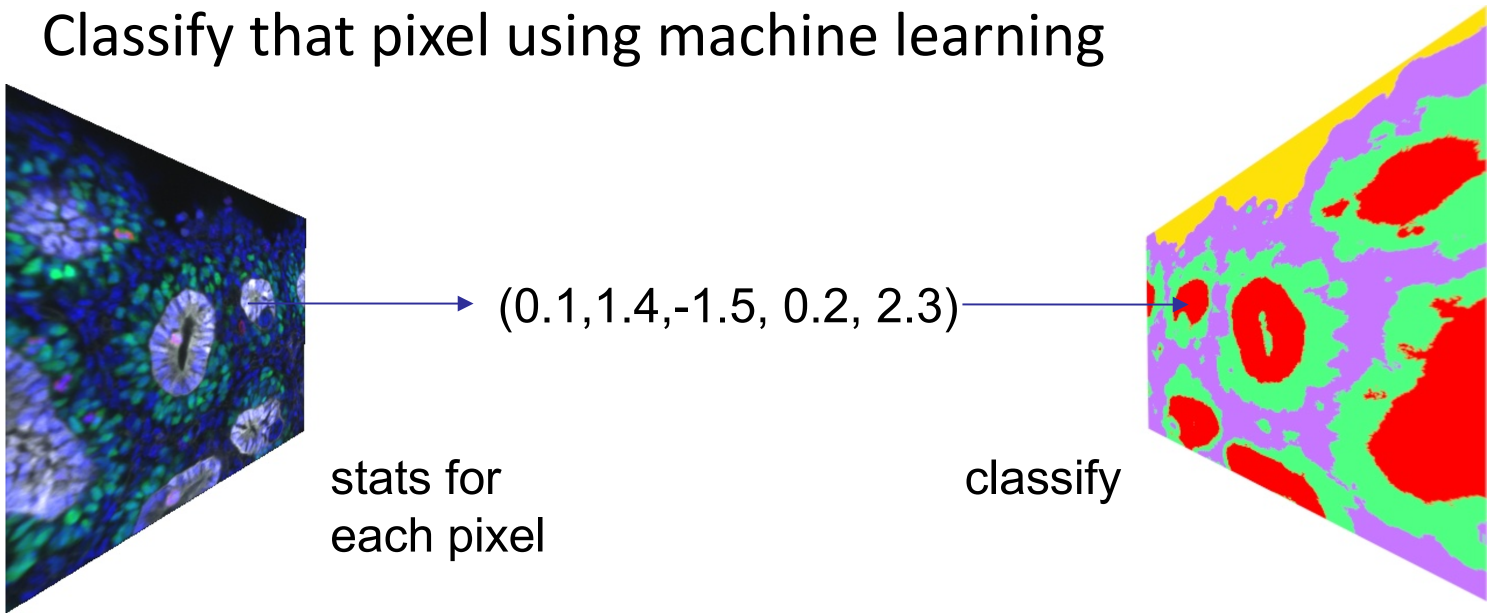




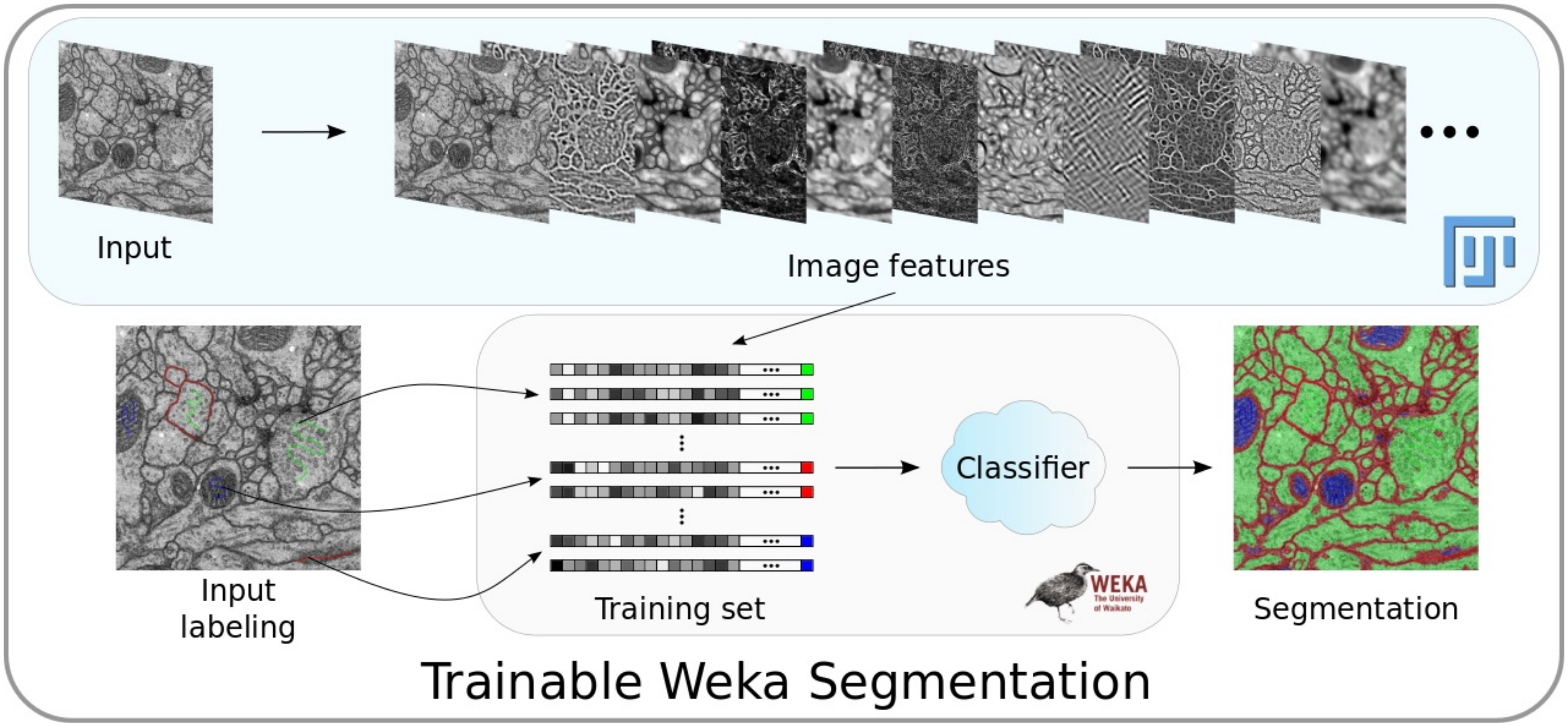
Microscopy image of a kidney by Alex Combes, Melissa Little Lab

Basic Image Segmentation

- Generate **statistics** for **each pixel** in the image
- E.g. intensity, local median intensity, gradient, …
- Classify that pixel using machine learning



The ImageJ/Fiji Trainable Weka Pipeline



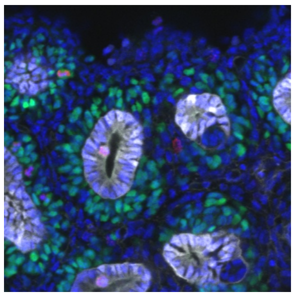
See "Introduction to Machine Learning for Imaging" workshop

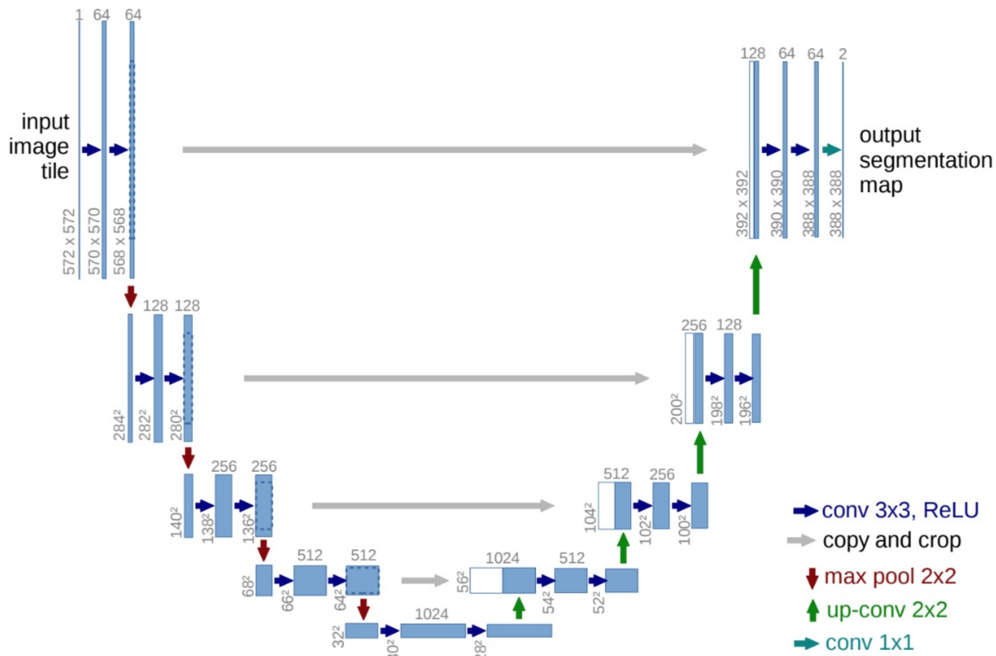
Image: https://imagej.net/Trainable_Weka_Segmentation

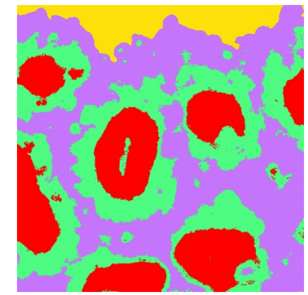
The UNET Architecture for Image Segmentation

Convolutional Deep Learning with U-net

- **U-net** was a new type of neural network architecture introduced in 2015
- It down-scales then up-scales an image to create a segmentation





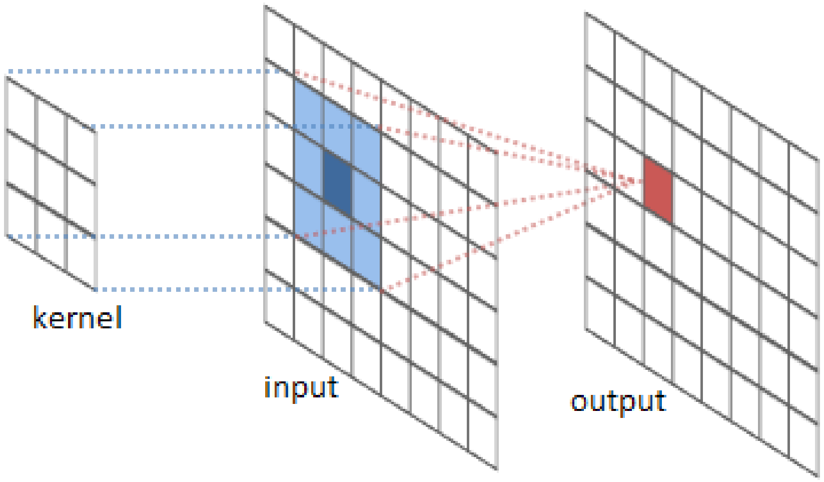


- U-Net and its variants are the state of the art in segmentation
- U-net paper: https://arxiv.org/pdf/1505.04597.pdf

Principle UNET Components

- 3x3 Convolution followed by ReLU (rectified linear activation function)
- 2 x 2 Maxpooling with stride 2 (halves image size)
- Double number of feature channels (convolutions) 64, 128, 256, 512, at each down sampling to compensate
- Dropout layers
- Skip layers
- Transpose convolutions / up convolutions

Convolutions



$$Output = \sum Kernel_{i,j} * Input_{i,j}$$

- The kernel represents the pattern to be detected

- The more the input matches the kernel, the more positive the output response would be

- Convolutions are pattern detectors

- The kernel weights are learned during training
  i.e. it learns what patterns to detect and respond to

Slide courtesy Titus Tang, Monash

A 3x3 Convolution Max

- **Convolution** of an image by a 3x3 matrix replaces each pixel in an image by a weighted sum of those pixels adjacent to it, i.e.

$$
\begin{aligned}
I'(x,y) = {} & a\, I(x-1,y+1) + b\, I(x,y+1) + c\, I(x+1,y+1) \\
& d\, I(x-1,y) \quad + \; e\, I(x,y) \quad + f\, I(x+1,y) \\
& g\, I(x-1,y-1) \;+ h\, I(x,y-1) \;+ i\, I(x+1,y-1)
\end{aligned}
$$

Local Image Region

I(x-1,y+1)	I(x,y+1)	I(x+1,y+1)
I(x-1,y)	I(x,y)	I(x+1,y)
I(x-1,y-1)	I(x,y-1)	I(x+1,y-1)

Convolution matrix

a	b	c
d	e	f
g	h	i

Convolution Example

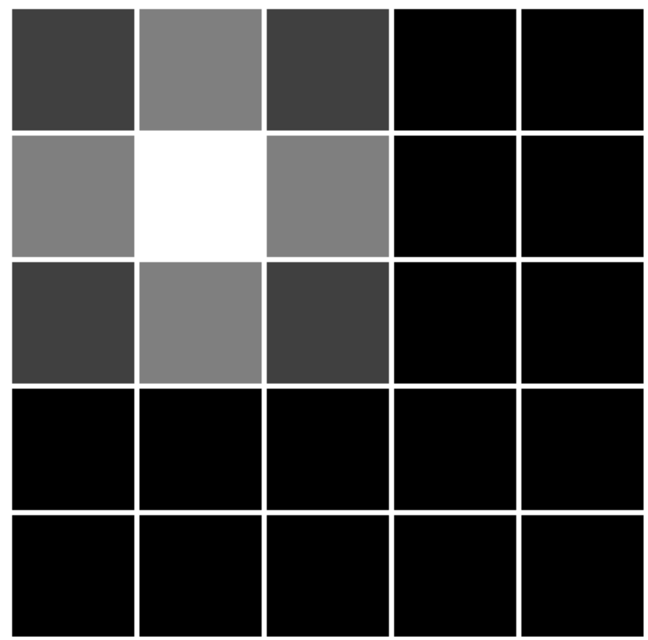
Convolution

0	0	0
0	1	1
0	1	1

Image

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Image x Convolution



Max Pooling with step size 2

10	6	6	0
5	-3	14	-2
0	2	-7	-3
0	0	-5	-5

Input 4x4 image

10	14
2	-3

Output 2x2 image

- Slide a window across the input and pick a value at every window position.
- Max pooling – take the max value.
- Average pooling – take the average value.
- Pooling layers are information filters.
- Images are reduced in scale, typically by ½ x ½

Slide courtesy Titus Tang, Monash

Spatial Dropout Layers

- Dropout regularisation is a simple computational method to prevent over-fitting to data
- SpatialDropout2D in Keras drops (sets to 0) entire feature maps with a given probability during training
- Helps promote independence between feature maps

Up Convolutions (scaling up)

10	14
2	-3

10	6	6	0
5	-3	14	-2
0	2	-7	-3
0	0	-5	-5

- Also known as **transpose convolutions**
- Max pooling scales an image down (½ x ½ smaller)
- Up Convolutions scale an image up (e.g. 2 x 2 bigger)
- Want 4 pixels to become 16
- Roughly (??) it works by:

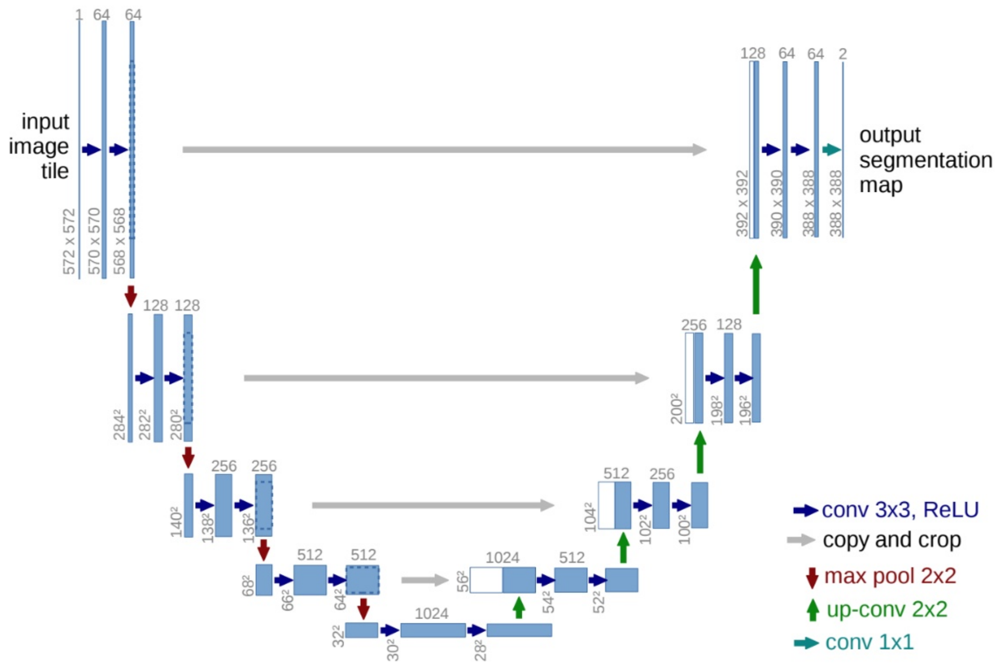
  Rearrange 2x2 input to be 4x1

  Rearrange a 3x3 kernel into a 16x4 by repeating entries

  The 16x4 then maps a 4x1 to a 1x16

  Rearrange 1x16 to a 4x4 output

Skip Connections



- Copy some of the outputs from the scaling down to the scaling up side
- Crop middle and copy, add/combine with up side of the network

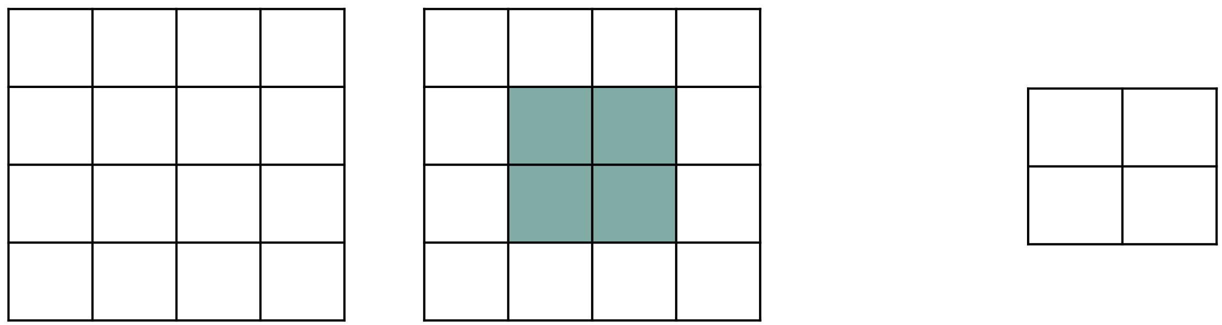
UNET Output Segmentation

- Output is a binary image that selects the object region
- May be multiple binary images, one for each object class

- Note that while the input is 572x572 output is 388x388
- Lose some of the edges due to convolutions
- Later versions of U-Net use **padding** for convolutions to maintain original image size

Padding Convolutions

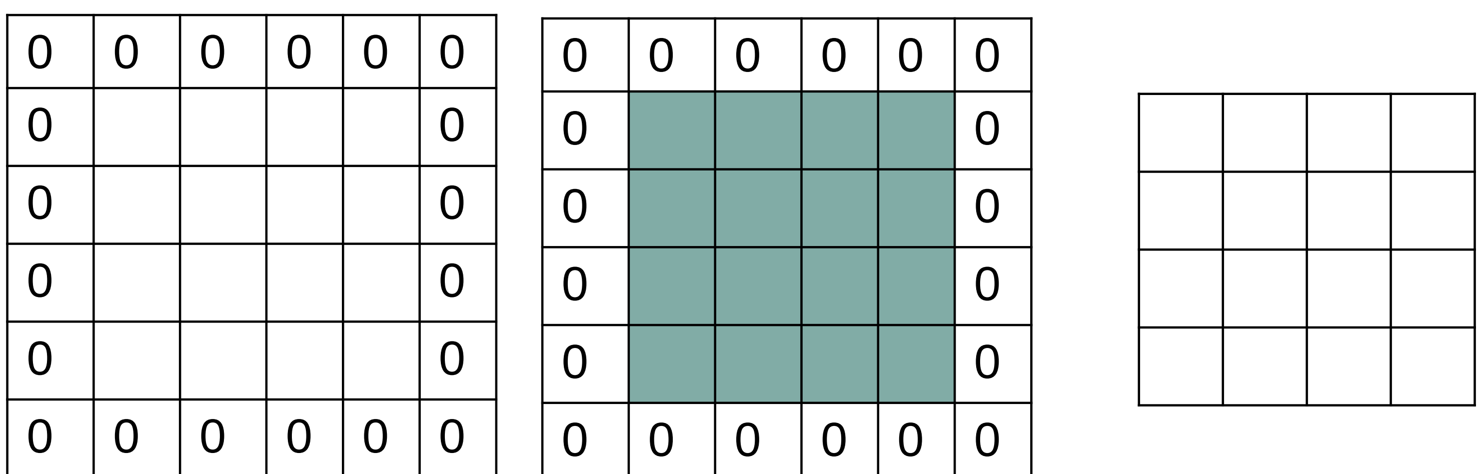
Suppose we had a 4 x 4 image & we applied a 3x3 convolution



Input 4x4 image

Output 2x2 image

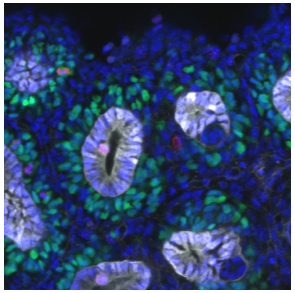
The centre of 3x3 conv can only fit in the green squares above

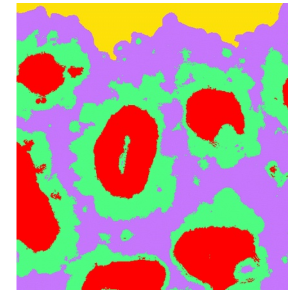


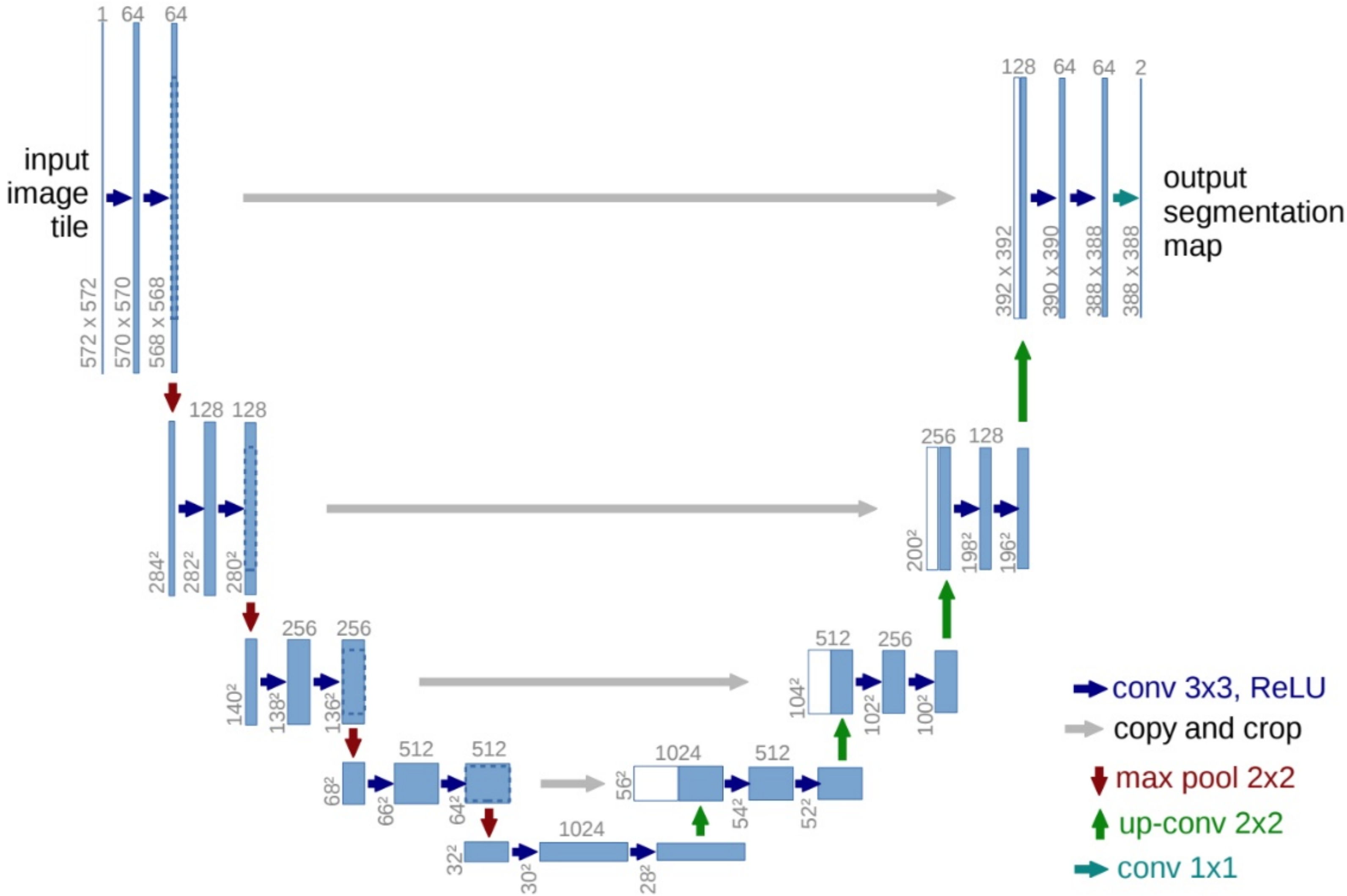
Padded 6x6 input image

Output 4x4 image

U-net







1
64
64
input image tile
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
56²
54²
512
52²
512
104²
102²
256
100²
256
200²
128
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

UNET Advantages and Disadvantages

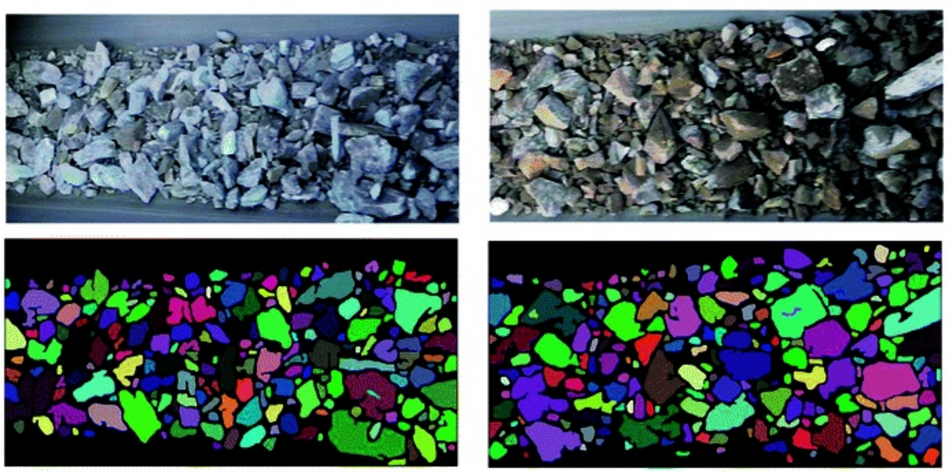
- Fast to segment
- Field-leading accuracy for segmentation
- Do not need large numbers (1000s) of examples
- Does not separate objects of same type
- Labor intensive to create manual segmentations for training
  - Every training image needs to be completely segmented
  - Conversely, a basic pixel classifier approach does not this
- Now implemented in Keras / tensorflow for easy use

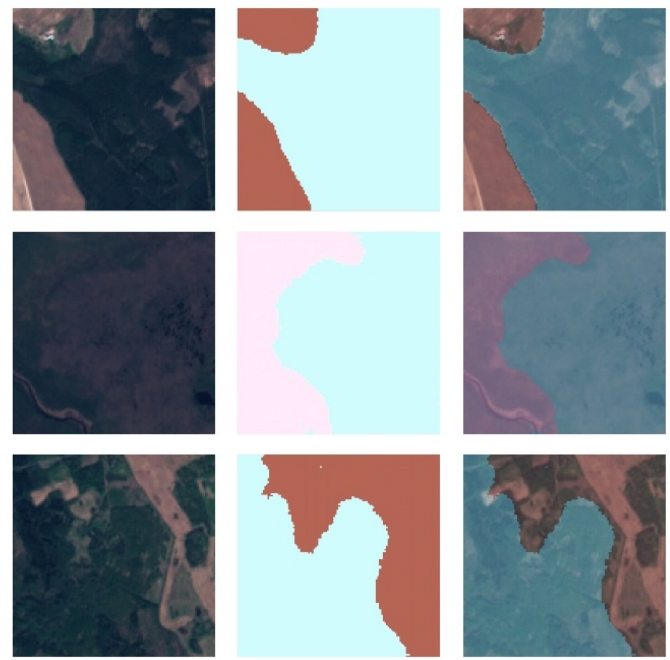
Real-World Applications of U-Net

Medical images https://arxiv.org/pdf/2011.01118.pdf

Ore segmentation

https://pubs.rsc.org/en/content/articlelanding/2020/ra/c9ra05877j





Satellite Images

https://arxiv.org/pdf/2003.02899.pdf

U-Net in Keras

- Three U-net networks are built into Keras
  Vanilla: based in the original implementation of U-Net
  Custom: a customisable U-Net architecture
  Satellite: optimised for satellite imaging

- There are also several utility functions to help with training and visualisation of the data and model outputs

- Documentation: https://pypi.org/project/keras-unet/

- Keras U-Net model python code:
  https://github.com/karolzak/keras-unet/tree/master/keras_unet/models

Some UNET Resources

- A line by line explanation/construction of UNET in tensorflow
  https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5
- The original U-net paper
  https://arxiv.org/pdf/1505.04597.pdf
- U-net in Keras
  https://pypi.org/project/keras-unet/
- Keras documentation on layer types

  Convolution: https://keras.io/api/layers/convolution_layers/convolution2d/

  Max Pooling: https://keras.io/api/layers/pooling_layers/max_pooling2d/

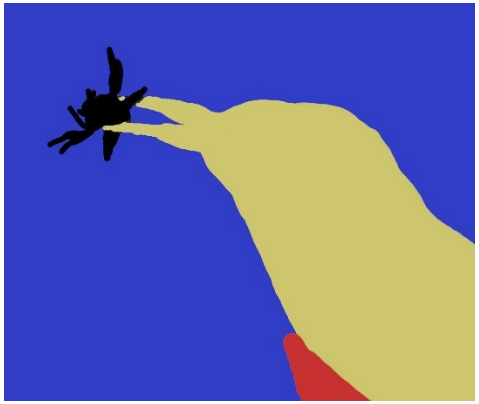
  Conv Transpose: https://keras.io/api/layers/convolution_layers/convolution2d_transpose/

  Dropout: https://keras.io/api/layers/regularization_layers/dropout/
- An explanation of Up Sampling
  https://naokishibuya.medium.com/up-sampling-with-transposed-convolution-9ae4f2df52d0

Data Preparation, Training UNET and Evaluating the Results

Creating a Training Set

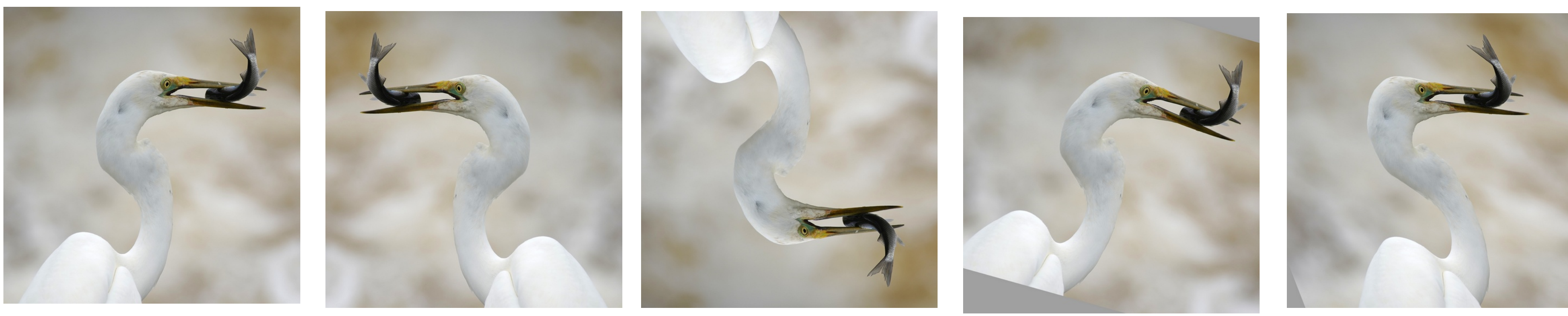
- In to create a U-net segmenter, you will need to create a set of labeled examples
- This represents the **ground truth** that U-net will learn
- This is typically a laborious and boring process drawing outlines on numerous images



- How many you will need will depend on the problem
- **Data augmentation** during training can reduce the number

Data Augmentation for Imaging

- Creating image sets is time consuming and there may be limited numbers of examples available
- One approach to extending a set of images is to apply geometric transformations to create new examples



Original Flip horizontal Flip vertical Rotate Shear

- Keras has **ImageDataGenerator** and **flow_from_directory**
- Keras-Unet has **get_augmented** function

Image Batch Generators for Training

- Usually, image sets are too large to present all at once during training as they will not fit in CPU/GPU memory
- Also calculations to update network weights may get too large
- Hence training usually occurs in **batches**, where subsets of images of a fixed size are selected and trained on
- Subsets are repeatedly drawn from the training set, and the network trained, until all of the training set had been presented
- The process is usually repeated multiple times

- In Keras you can set **batchsize** as a parameter to **model.fit** and pass all of your training data to **model.fit** (memory expensive)

  Or you can use one of the batch generators, and pass that to the model.fit function (we will see examples in the code).

Using Google Colab

- Colab is a web-based iPython Notebook service
- You can play/run blocks of python code interactively
- Your code can run on Googles CPU/GPU or TPU in the cloud
- It is free but has usage limitations

- Using Tensorflow with image applications on Colab with GPU can be 10 or 20 times faster than with CPU

  (In Colab set Runtime / Change Runtime Type → GPU)
- TPUs are Tensorflow Processing units – designed for Tensorflow

- Colab Code is interchangeable with Jupyter Notebooks
- Colab basics:

  https://colab.research.google.com/drive/1YKHHLSlG-B9Ez2-zf-YFxXTVgfC_Aqtt

Setting up Google Colab for the Workshop

- Create a folder in your Google Drive called **Colab Notebooks**
- Download code and slides for this workshop:
  https://github.com/doktor-nick/intro-to-ml-for-imaging/archive/master.zip
- Unzip it and copy the folder into **Colab Notebooks**
- Go into the folder **intro-to-ml-for-imaging-master**
- Right-click on Unet_in_Keras_introduction.ipynb
- Select **Open with / + Connect more apps**
- Search for and install Colaboratory
- Double clicking on the Unet_in_Keras_introduction.ipynb should open it in Colab

- When running the code for the workshop you will need to link and authorise your Google Drive to be used by the notebook. The provided notebook has code/instructions for doing this.

Hands On

Unet_in_Keras_introduction.ipynb

Colab Notebook

Optimisers, loss functions, and tiling

Measuring Success: Loss functions for imaging

- A common measure of error/loss in machine learning is root mean square error (RMSE), i.e. the sum of the square of the differences between the real and predicted values

- But predicting a segmentation is a **binary** problem,

  1 = in object,   0 = not in object
- Binary Cross-entropy a bit is like RMSE, but it is more punative on getting a prediction in the wrong class.
- The formula looks like this

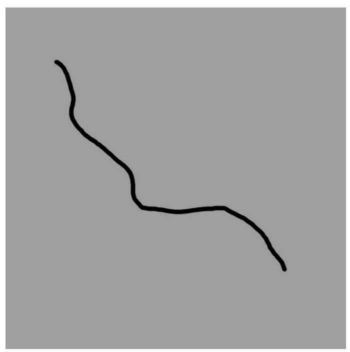
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

  More explanation here:

  https://towardsdatascience.com/undeJrstanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Unbalanced problems and loss functions

- One problem with RMSE and binary cross-entropy is that the not good when the problem is **unbalanced,** that is if there is a lot more of one class than another.
- For instance,



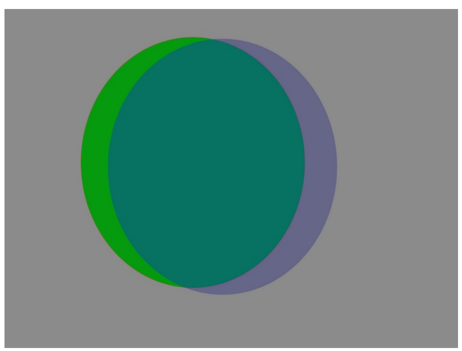
Target

Prediction

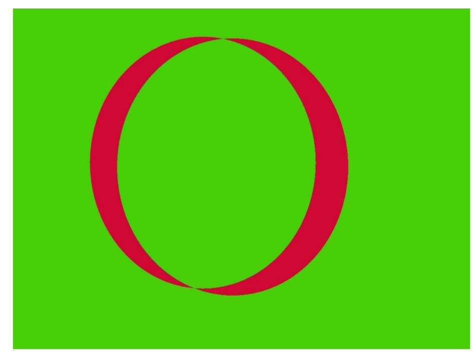
the RMSE between these two images is actually quite small because they agree a lot on the background

Jaccard loss : “intersection over union”

- One problem with RMSE and binary cross-entropy is that the not good when the problem is **unbalanced,** that is if there is a lot more of one class than another.
- Jaccard:intersection over union (|X & Y|)/ (|X|+ |Y| - |X & Y|)



object prediction



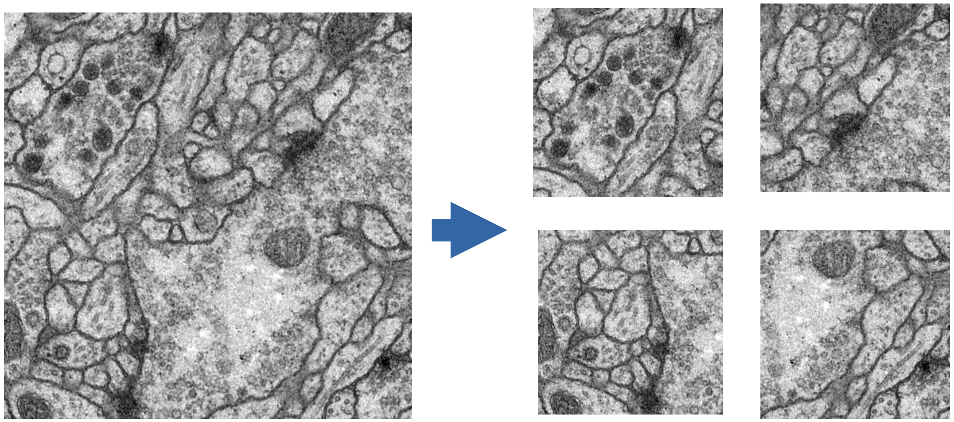
Agree

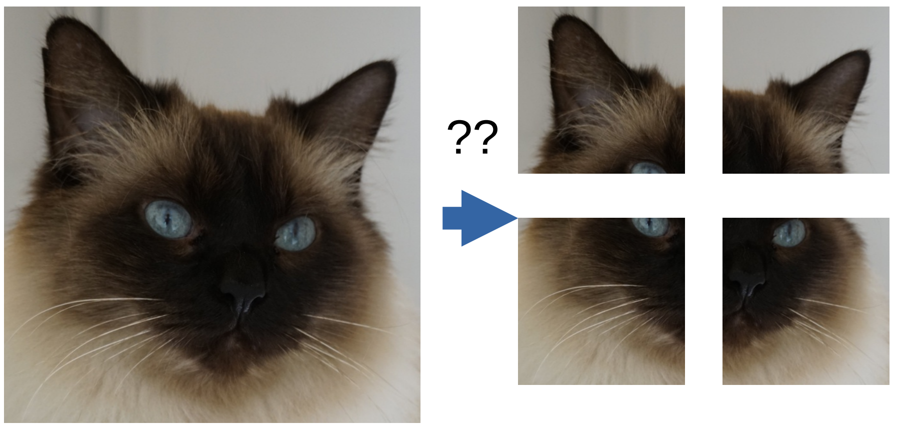
Agree + Agree

- More discussion on image metrics:
  https://towardsdatascience.com/image-segmentation-choosing-the-correct-metric-aa21fd5751af

Image Tiling

- Available memory on a GPU can often be an issue for imaging
- During training, **reducing batch** size may help or **scaling down**
- **Tiling**, that is chopping an image into smaller pieces is another approach
- Care needs to be taken that each tile contains enough information so that it can be accurately segmented





- Usually want to use **overlapping tiles** to avoid edge effects
- The Keras Unet libraries can generate patches easily

Which Optimiser for Semantic Image Segmentation?

- In Keras there are a range of optimisers available including SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam Ftrl
- Performance may vary according to the application
- Generally Adam and SGD are reasonable choices
- For a discussion of different optimiser types see https://ruder.io/optimizing-gradient-descent/
- For optimisers available in Keras see https://keras.io/api/optimizers/

Overview of choices in network/evaluations

- Optimiser: Adam
- Loss function: Jaccard distance
- Reduce batchsize, scale images down, or tile if size is a problem

- Sometimes the optimiser will get stuck. It is worth running the fitting functions more than once to ensure better results

Hands on

Unet_in_Keras_further_topics.ipynb

Colab Notebook

The End