# Mind the Gap (In Resolution): Assessing the Impact of MRI Slice Resolution on the Clinical Applicability of Lesion Mapping Models

Rebecca Bryan and Rusty Bielski

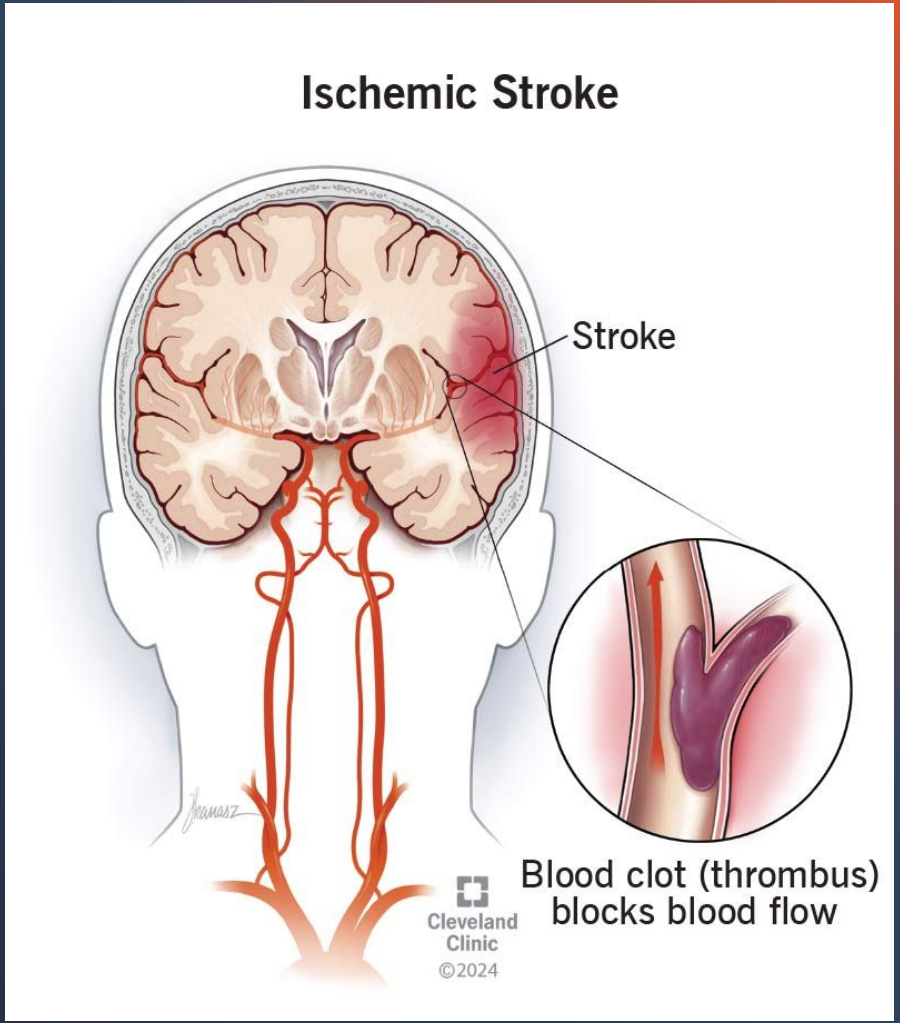
# Ischemic Stroke

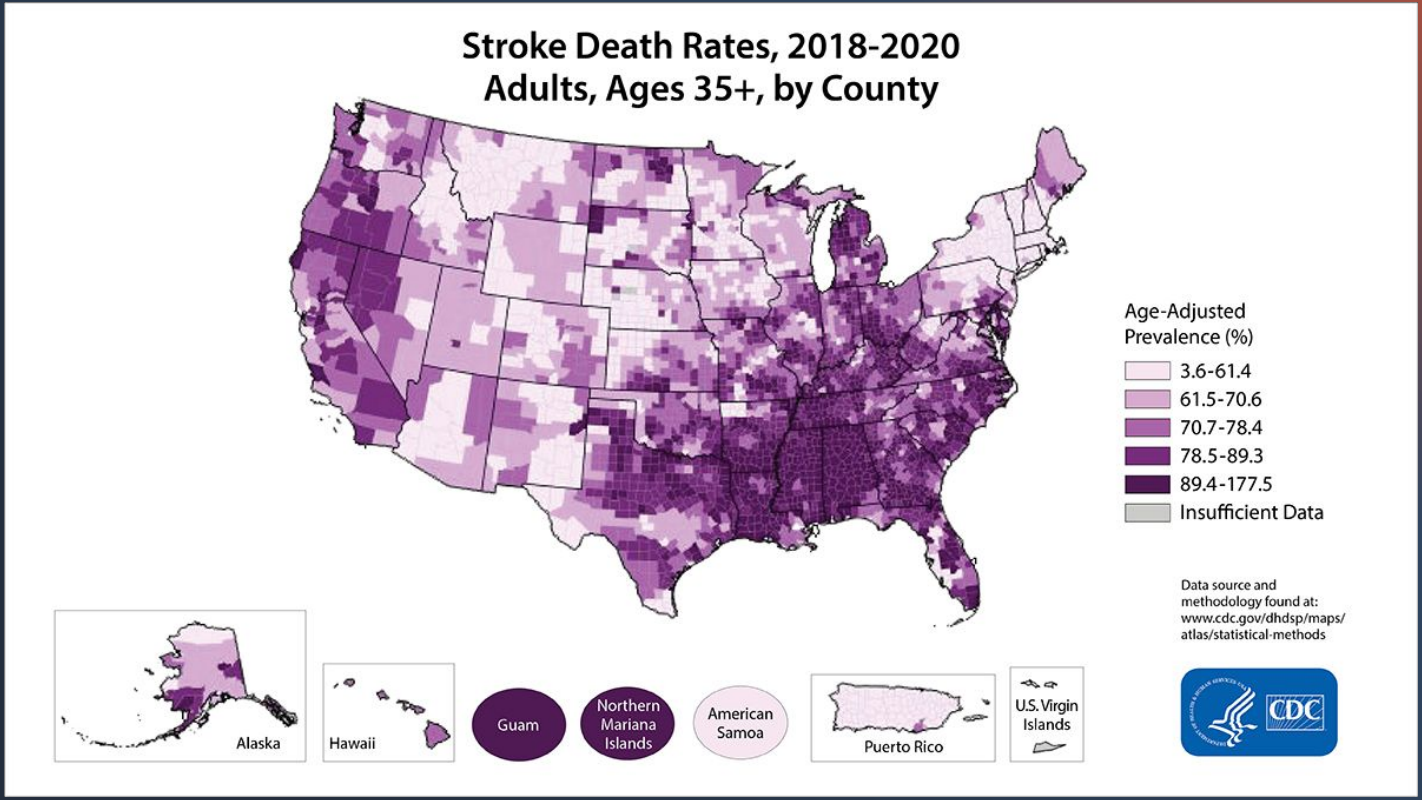
An ischemic stroke occurs when something blocks blood flow from reaching the brain

The clot may be partial or resolve itself (this is called a Transient Ischemic Stroke or TIA)

Around 87% of all strokes are ischemic

Timely intervention is crucial in stroke management





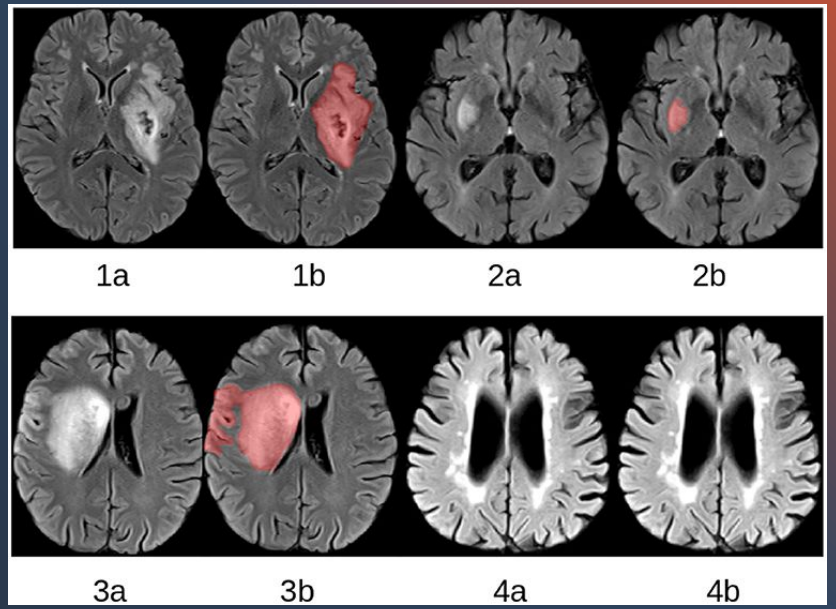

Stroke-related costs in the United States came to nearly $56.2 billion between 2019 and 2020. Costs include the cost of healthcare services, medicines to treat stroke, and missed days of work.

# Stroke Treatment and Recovery

Neuroimaging plays a crucial role in diagnosis and treatment of strokes. It can help identify the cause of the stroke, distinguish between irreversible infarcted tissues and salvageable tissue, and aid in treatment planning and outcomes predictions.

The current standard for ischemic stroke segmentation in the field is hand-drawn lesions on scans.

This is a time and labor intensive process, so researchers are building machine learning algorithms to automate lesion mapping.

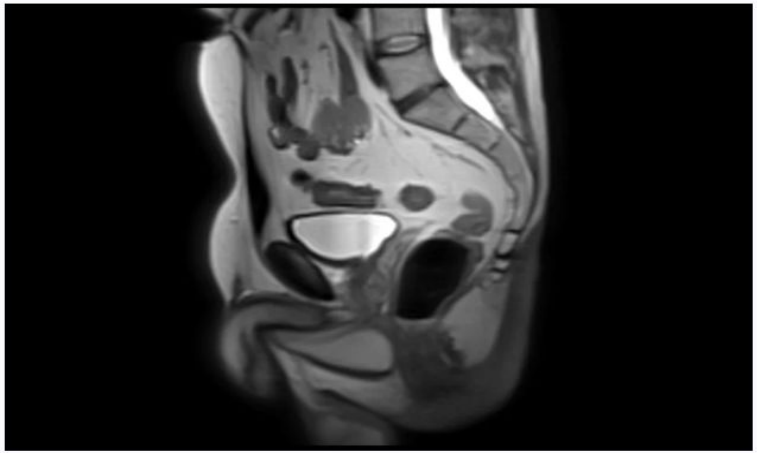



- 1a - 4a: Slices from four FLAIR MRI datasets
- 1b - 4b : Corresponding expert lesion segmentations
- The **lesions vary considerably** with respect to shape, position, and size.
- Patient 4 does not display a lesion resulting from an acute ischemic stroke but considerable **white matter hyperintensities**.

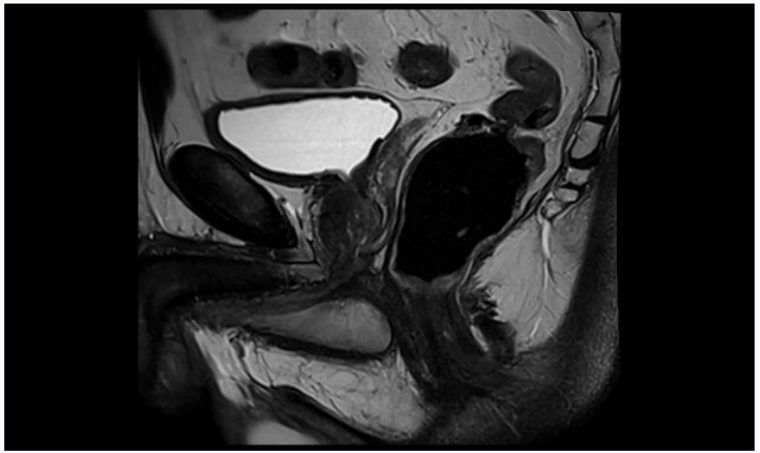
https://mrimaster.com/mri-slice-thickness/

# The Problem

PELVIS IMAGE WITH 8MM SLICE THICKNESS

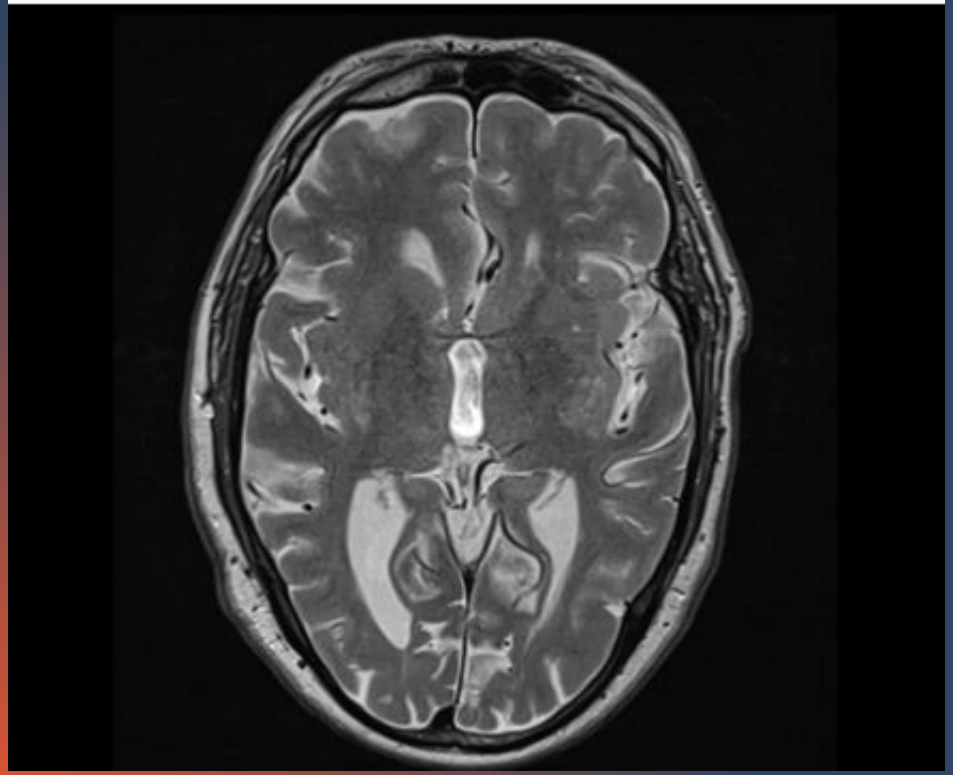


PELVIS IMAGE WITH 3MM SLICE THICKNESS

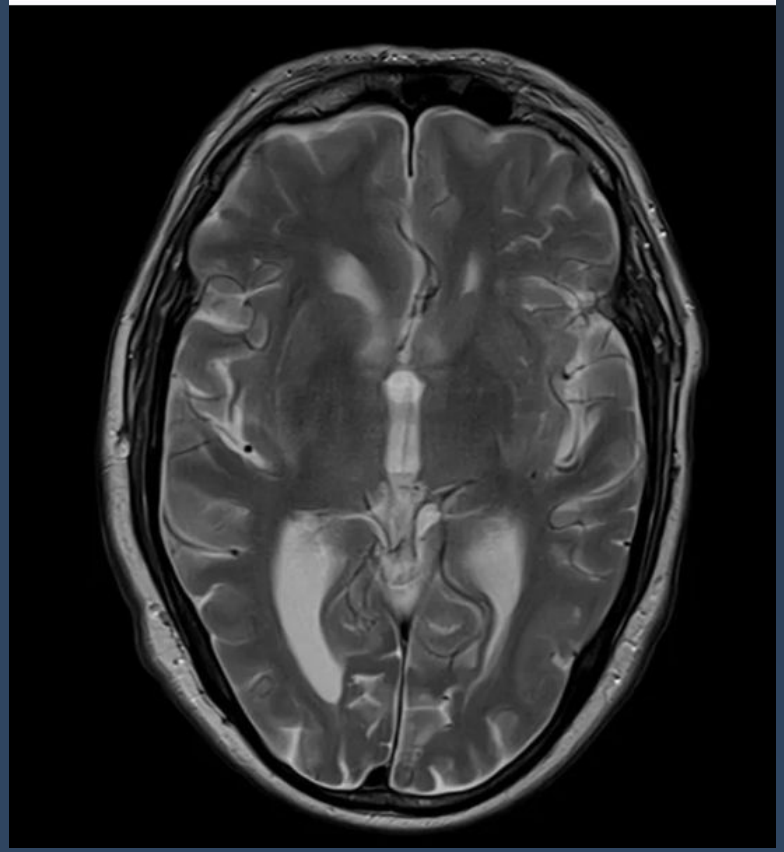


**Clinical MRIs** are often **lower spatial resolution** compared to **Research MRIs** (10mm vs. 1mm slice thickness).
Automated segmentation models are generally trained on the smaller resolution MRIs
- but does that lead to a loss of information when applied in clinical settings with smaller?
- increasing the risk of false negatives?

SLICE THICKNESS 3MM



SLICE THICKNESS 5MM



**When training ML models for Ischemic Stroke Lesion Mapping:**

Q: How does MRI spatial resolutions effect model performance?

H: Applying a **Lower** spatial resolution model will:

- Decrease Dice Similarity Coefficients
- Increase False Positives and False Negatives
- Reduce Boundary Decision Detection Precision
- Inaccurately Estimate Mask Volumes

Goal: To determine the extent to which lower resolution MRI scans affect lesion detection accuracy when using automated models trained on higher resolution scans.

# The Data

- Research study between University of Montana and University of California, Irvine
  - Goal: automate lesion mapping of MRI scans
- Dataset: Anatomical Tracings of Lesions After Stroke (ATLAS v2.0)
  - USC Stevens Neuroimaging and Informatics Institute
  - 955 T1-weighted MRI scans,
  - Training set (n=655 T1w MRIs with manually-segmented lesion masks)
  - Test dataset (n=300 T1w MRIs only; lesion masks not released)
- Using the training set → Training the model on 500 scans, testing on 154 (one scan removed during preprocessing)
  - Test the accuracy of the high resolution (1 mm) scans vs. the manually downsampled (10 mm) scans.

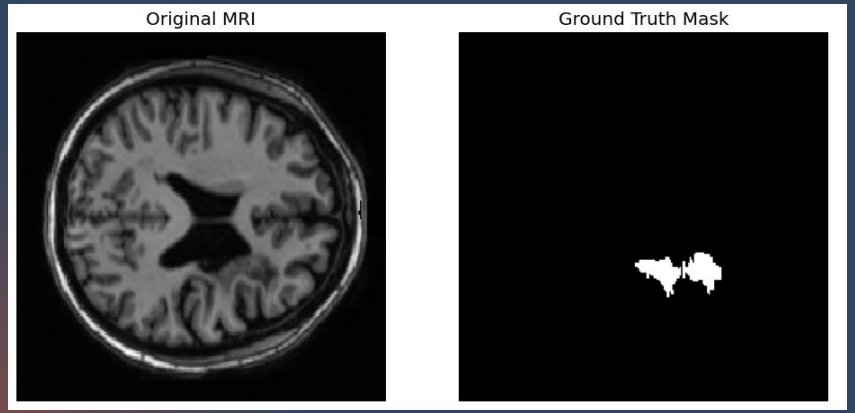
# Data Preprocessing

Download data from ATLAS → Separate 500 training scans and 154 testing scans

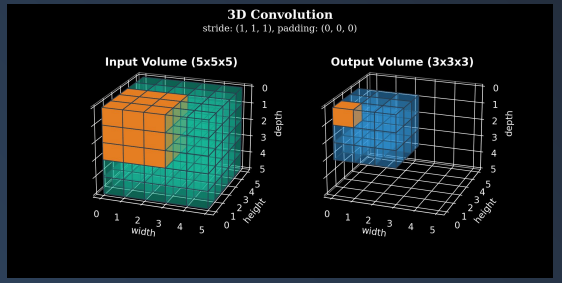
Normalize → $$\text{Normalized Image} = \frac{(\text{Image} - \text{Min Value of ROI})}{(\text{Max Value of ROI} - \text{Min Value of ROI} + 1e - 7)}$$

- The mask is also checked to ensure that it's binary (1s for lesions, 0s for background).

Padding → Making the model so that it will work with any dimension divisible by 16. This step adds extra empty layers to the MRIs to meet that requirement

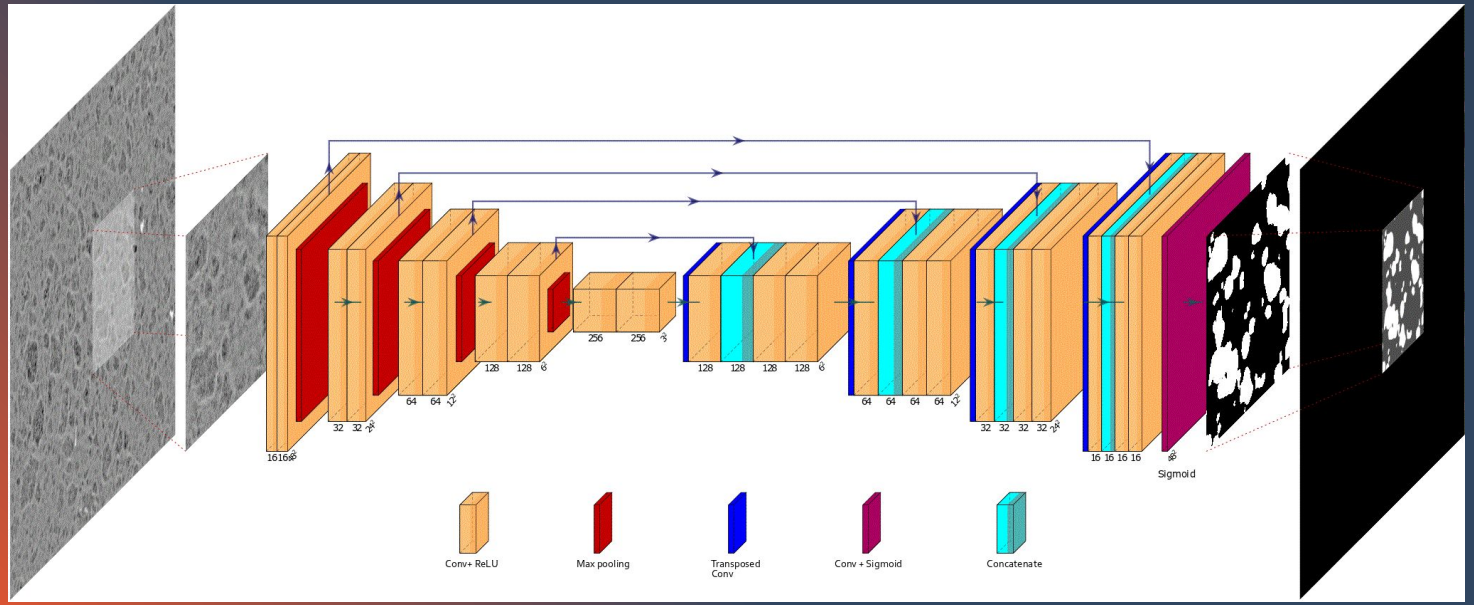


# 3dU-Net The best ML model for image segmentation...




**Input:** The model learns from pairs of 3D MRI images and corresponding masks (the "answer key" showing the true lesion location).

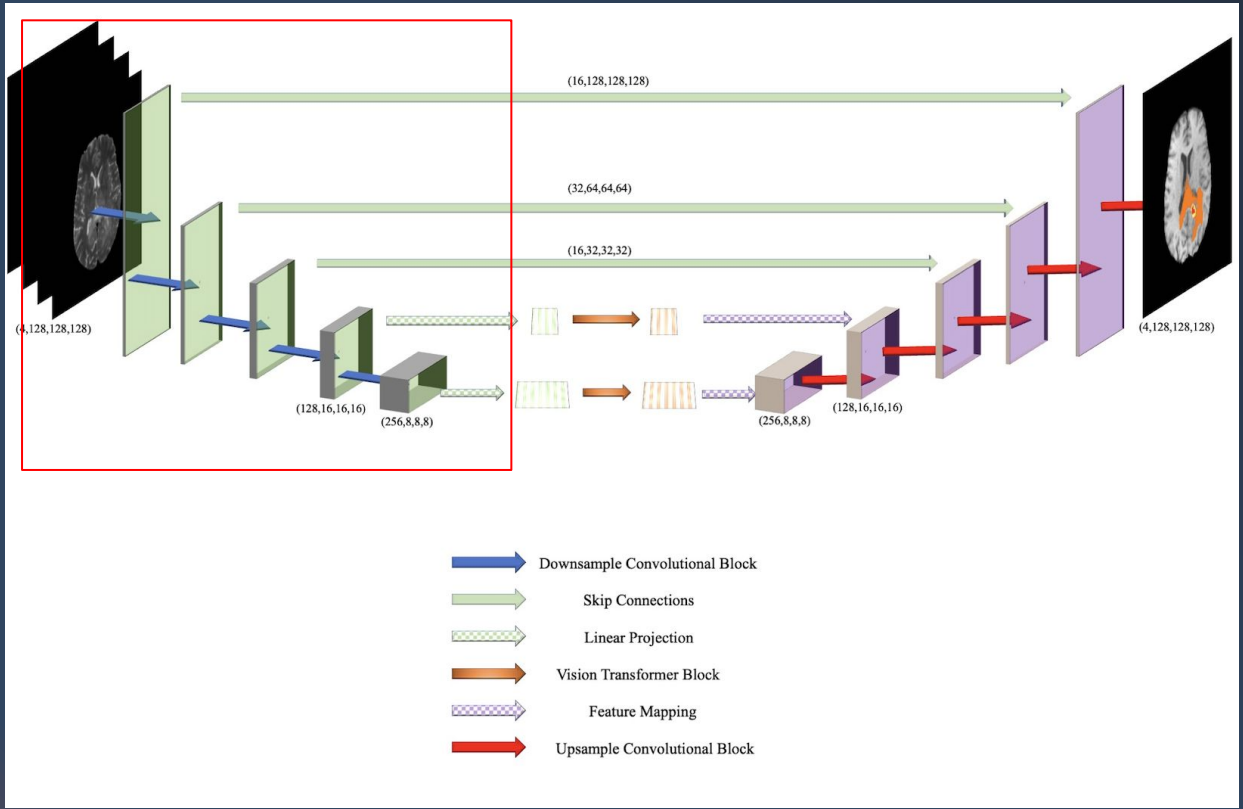
**Output:** a 3D "probability map" where each voxel's value (0-1) is the model's confidence that it belongs to the lesion.




The image here depicts a 2D model, but imagine the input and output as the 3D convolution above.

## Phase 1: Encoder

- Repeated convolutional layers and max pooling layers that extract intermediate features
- **Shrink & Summarize:** Progressively creates smaller, lower-resolution versions of the input MRI.
- **Find Features:** Uses filters at each step to detect patterns. Starts simple (edges) and gets more complex (combinations of features) in deeper layers.
- **Result:** Understands *what* features are present, but loses precise location detail.



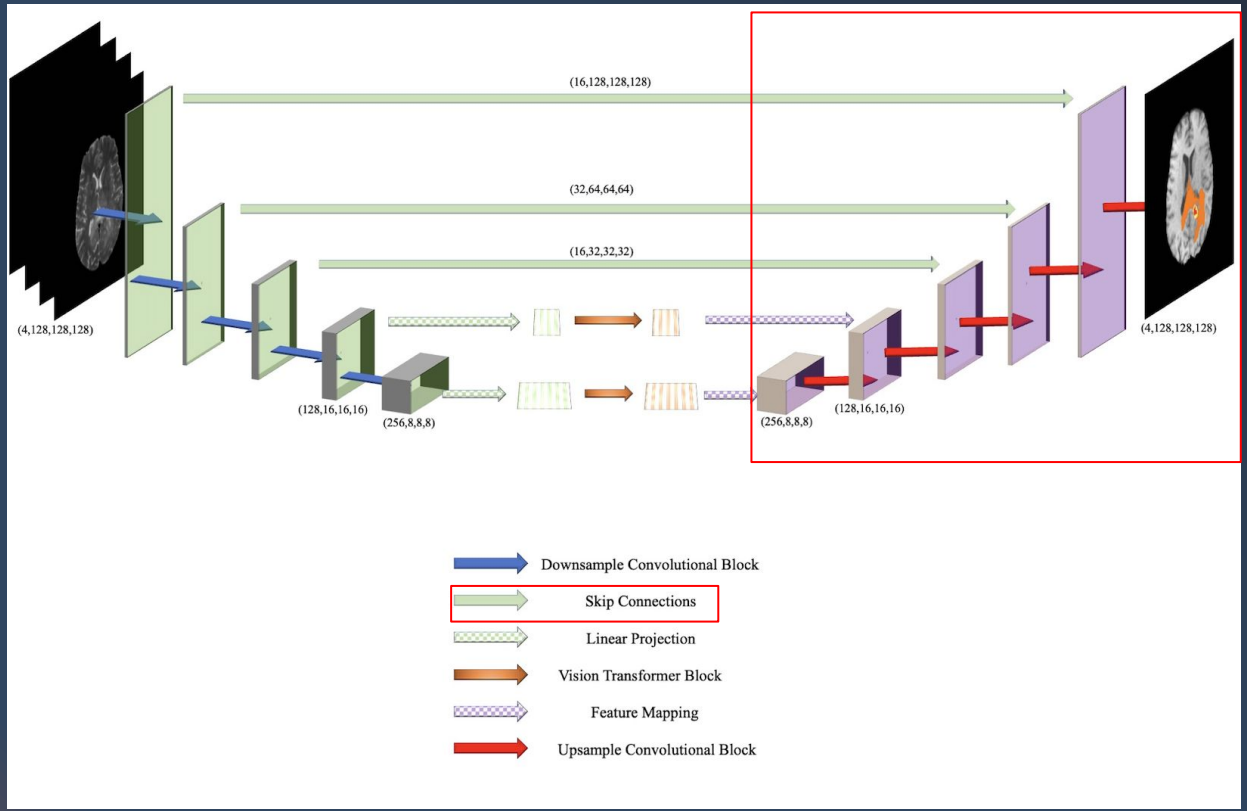
# Phase 2: Decoder

- **Expand & Detail:** Takes the summarized features from the bottom and progressively enlarges the representation back to the original size.
- **Rebuild with Detail (Skip Connections):**

  Merge the expanding features with corresponding detailed feature maps saved during the encoding phase.

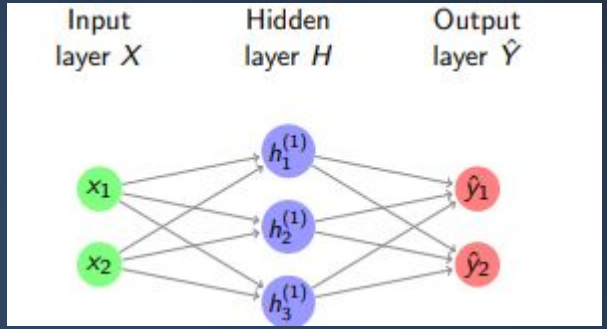
  This restores precise location information.

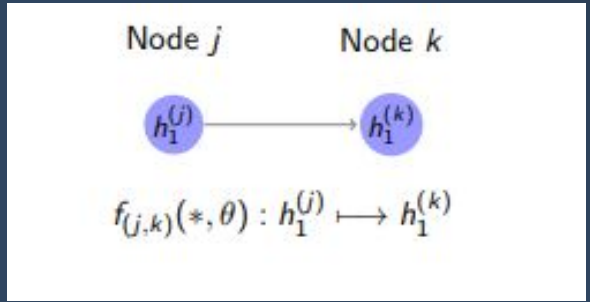
  - *Analogy:* Starting with a rough sketch and adding details using a picture for reference.



# The Neural Network

- The goal of an artificial neural network (ANN) is to approximate some function $f^*$ . In our case, this is a classification function $y = f^*(x)$, where $x \in R^n$ is an input vector.
- The simplest architecture for an ANN is a directed graph composed of three layers of nodes, an input layer, a hidden layer, and an output layer, along with the connections between those nodes (the arrows in the graph).
- The architecture of the network can be a lot more complicated than the above example with many more layers, numbers of nodes in layers, and types of connections.
- The network architecture will create a collection of formula relationships between the nodes defined by by connecting functions $f_{(j,k)}(x,\theta)$ for nodes $(j,k)$ with parameters $\theta$




*U-Net: Convolutional Networks for Biomedical Image Segmentation O. Ronneberger, P. Fischer, and T. Brox*

# The Neural Network

- So the specific architecture of an ANN defines a mapping $y\hat{} = f(x; \theta)$ and then learns the value of the parameters $\theta$ that result in the best function approximation to $f^*(x)$ given that graph and those connecting functions.

- Often there will also be a response function $g$, either at the end of the network, or between layers, to further modify the outputs. Common examples are the RELU function $g(z) = max(0, z)$, or the Sigmoid function $\sigma(z) = 1/(1 + e^{-z})$.

- Convolutional Neural Networks (CNNs) are commonly used for the analysis of 2D (or 3D) images having either gray level pixel intensities, or multi-channels pixel intensities, such as Red/Blue/Green.

- A CNN is an ANN where some of the primitive functions $f(j,k)(x, \theta)$ act on their layers as a convolution operation (often called a convolution filter).

- In CNNs, convolution operations act on whole sections of an image file, so they are usually not single primitive functions $f(j,k)(x, \theta)$, but rather are made up of such functions working together.

# Convolution

The definition of a convolution in the context of functions:

Given two functions *k* and *f*, their convolution is defined $(k * f)(x) := \int_{-\infty}^{\infty} k(y) f(x - y)\, dy.$

- the two functions are first offset by y, then multiplied together, and then their multiple is integrated over all y in their domain.

Convolutions over an image file *X* are similar, except that the domain is the 2D discrete coordinate space, so the integral is actually a summation

Let V be the square "window" of integer coordinates *(m, n)* such that $|m| \leq S$ and $|n| \leq S$ for some integer *S*.

Fix a "function" of arbitrary real numbers *K(m, n)* defined for all *(m, n)* in *V*. The *(2S + 1)x(2S + 1)* matrix *K* defines the "kernel" of a linear convolution filter with support included in *V.*

This convolution filter acts linearly on an image *X* to generate another image *G* denoted $G = K * X$, computed as follows for each pixel *(i, j)*

$$G(i,j) = K * X(i,j) = \sum_{(m,n)\in V} K(m,n)X(i-m, j-n)$$

# Convolution Layers and Non-Linear Convolution Operations in ANNs

Fix a kernel $K$ with support in a square window $V$, and fix a single threshold parameter "$b$". Then $(K, V, b)$ defines a convolution layer of size $N \times N$, with one "neuron" (or "node") $NOD_{ij}$ placed at each pixel position $(i, j)$.

When the current input is an $N \times N$ image $X$, the state $Y(i, j)$ of neuron $NOD_{ij}$ is then computed by $Y(i, j) = g(b + K * X(i, j))$

- Here $g$ is a response function. When this is non-linear, it makes the convolution layer into a non-linear convolution layer.

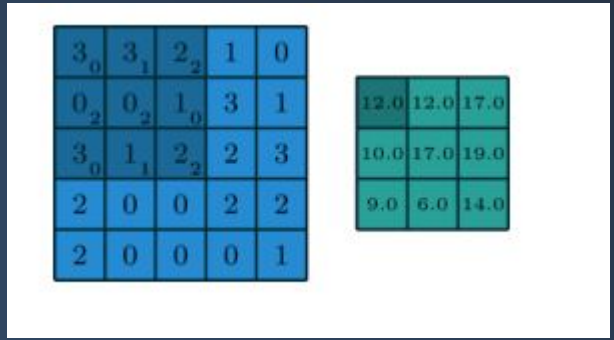
# 3x3 Convolution with RELU

Let’s say we have a standard 3×3 convolution operation with RELU.

A 3 × 3 kernel *K* (in grey) ranges over all contained intersections with an image *X* (in blue), i.e., those completely contained within the image.

The 3×3 kernel operation reduces the size of the image by 2 in both dimensions.

The RELU function (*g(z) = max(0, z)*) then takes the outputted matrix and retains only the positive part.



An important aspect of the U-Net’s architecture is the large number of feature channels in the up-sampling part. As a consequence, the expansive path is more or less symmetric to the contracting path, and yields the u-shaped architecture.

The key insight here is that deep features can be obtained when going deeper, but spatial location information is also lost when going deeper, so output from shallower layers have more local information. We want to combine both to enhance the results.

We do this through the copying and cropping operation. Every step in the expansive (upwards) path is concatenated with the correspondingly cropped feature map from the contracting (downwards) path. The cropping is necessary due to the loss of border pixels in every convolution.

# Max Pooling and Downsampling

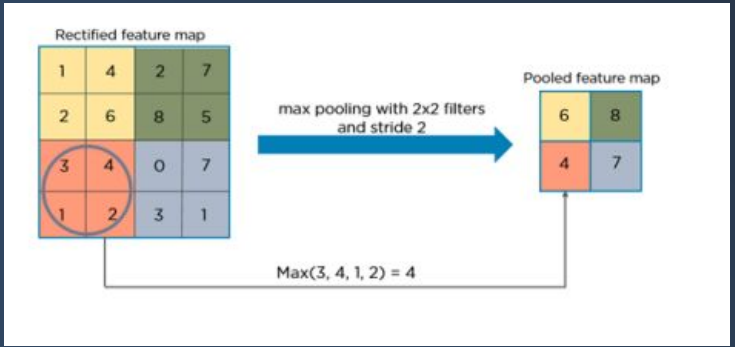
The max pooling operation partitions an input image *X* up into 2 × 2 squares, then takes the maximal element from each square and arranges them in a new image *Y* that is half the size in both dimensions.

It can be thought of as a form of data compression that ″picks the local highlights″.

″At each down-sampling step we double the number of feature channels.

The max pooling operation involves increasing the number of feature channels.

Each layer of data in a CNN is actually a three-dimensional array of size *h* × *w* × *d,* where *h* and *w* are spatial dimensions, and d is the feature or channel dimension



# Convolution and 2x2 Up-Sampling

Up-sampling is a method that gets the output size larger.

This can be a convolution operation that allows the kernel *K* (in grey) to range over all intersections with an image *X* (in blue), not just those contained within the image.

This is made concrete by expanding the image *X* to *X′* (all the white squares) by some method.

The expansion of the image *X* to ′ is done by mirroring the image where necessary.

"Every step in the expansive path consists of an up-sampling of the feature map followed by a 2×2 convolution ("up-convolution") that halves the number of feature channels"

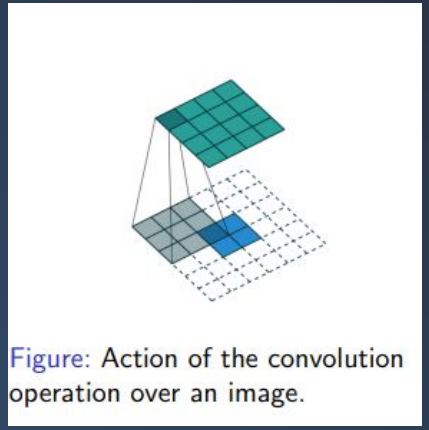


Figure: Action of the convolution operation over an image.

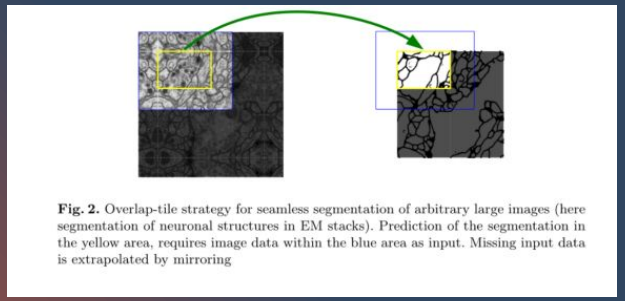


**Fig. 2.** Overlap-tile strategy for seamless segmentation of arbitrary large images (here segmentation of neuronal structures in EM stacks). Prediction of the segmentation in the yellow area, requires image data within the blue area as input. Missing input data is extrapolated by mirroring

At the final layer a 1×1 convolution is used to map each 64-component feature vector to the desired number of classes.

A 1 × 1 convolution maps an input pixel, along with all it's channels (features), to an output pixel, not looking at anything around itself.

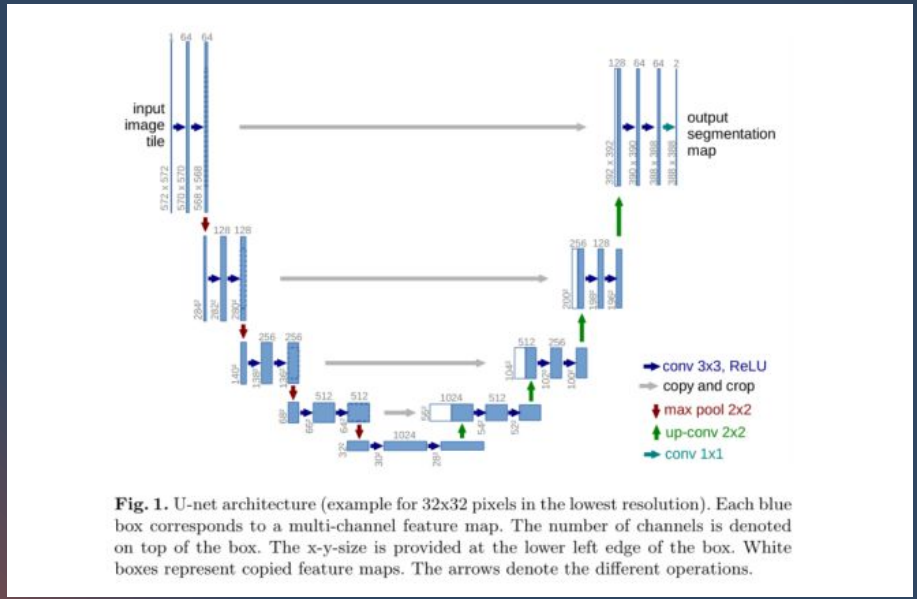



**Fig. 1.** U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

# The 3D U-Net

1. **3D Convolution**

Each layer in the encoder and decoder uses **3D convolutional filters**, which operate over the entire volume:

$$y_{ijk} = \sum_{m,n,p} x_{i+m,j+n,k+p} \times w_{m,n,p} + b$$

Where:

- x : input volume
- w: 3D filter/kernel weights
- b: bias term
- y: output feature map (activation map)

**2. Downsampling: 3D Max Pooling**

After convolution, each layer reduces spatial dimensions using 3D max pooling:

$$y_{i,j,k} = \max_{\substack{m=0 \\ n=0 \\ p=0}}^{s-1} x_{i+m,\, j+n,\, k+p}$$

Where s is the stride (usually 2), shrinking the feature maps by half while preserving the most salient features.

### 3. Upsampling: Transposed Convolution (Deconvolution)

The decoder upsamples feature maps using transposed convolutions (a.k.a. deconvolution):

$$y = x \times W^T$$

This expands the volume size and allows for learned upsampling as opposed to fixed interpolation.

### 4. Skip Connections

Skip connections concatenate the encoder feature maps to the decoder maps at each level:

$$z = \text{concat}(f_{encoder}, f_{decoder})$$

This helps retain spatial detail lost during downsampling, which is critical in medical image segmentation.

## 5. Final Activation: Sigmoid Function

The final layer outputs a voxel-wise probability using a sigmoid activation:

$$\mathrm{P}(\mathrm{lesion}) = \frac{1}{1 + e^{-z}}$$

Each voxel's output is a number between 0 and 1 representing the model’s confidence that it belongs to the lesion.

# Training the 3D U-Net: Backpropagation & Optimization

1. **The Forward Pass**

During the forward pass, the input 3D MRI is convolved through the U-Net layers to generate a voxel-wise probability map:

$$\hat{y} = f_\theta(x)$$

Where

- x: Input 3D MRI volume
- ŷ: predicted lesion mask (probabilities)
- $f_\theta$: the 3D U-Net function with learnable parameters (weights) $\theta$

## 2. The Loss Calculation

We compute the loss between the predicted output (ŷ) and the ground truth lesion mask (y).

For binary segmentation, we use Binary Cross-Entropy (BCE) loss:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

/Dice Loss:

$$\mathcal{L}_{\text{Dice}} = 1 - \frac{2\sum_{i=1}^{N} y_i \hat{y}_i}{\sum_{i=1}^{N} y_i + \sum_{i=1}^{N} \hat{y}_i}$$

$\mathcal{L}$

**Breakdown:**

- $\mathcal{L}_{\text{BCE}}$: Binary cross-entropy loss
- $N$: Total number of voxels
- $y_i$: Ground truth label for voxel $i$ (0 or 1)
- $\hat{y}_i$: Predicted probability that voxel $i$ belongs to the lesion

- $\mathcal{L}_{\text{Dice}}$: Dice loss (lower is better)
- $y_i$: Ground truth label at voxel $i$
- $\hat{y}_i$: Predicted probability at voxel $i$
- The numerator is **twice the intersection**, and the denominator is the **sum of prediction and ground truth volumes**

This version assumes soft Dice loss (continuous values between 0 and 1). Let me know if you need the **hard**

### 3. Back Propagation

Using the computed loss, we update the weights $\theta$ using backpropagation, which involves:

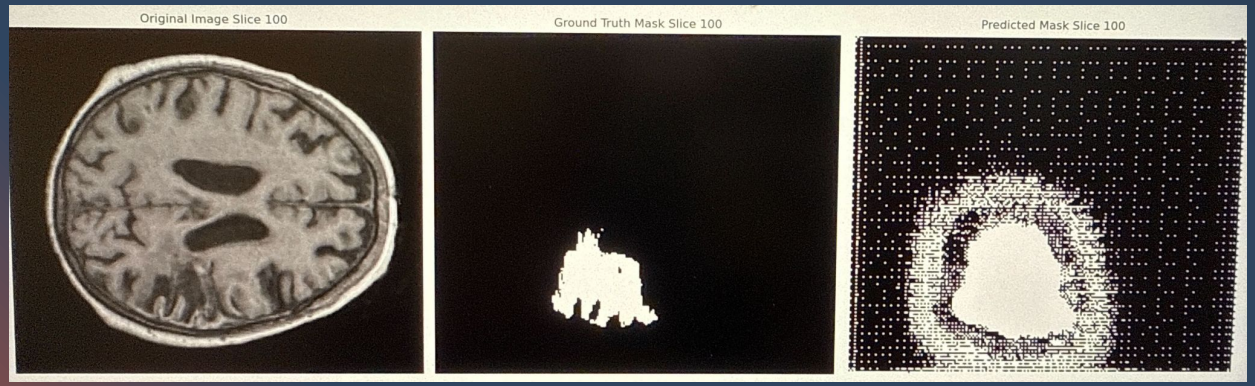
1. Gradient calculation: Compute the derivative of the loss with respect to each parameter using the chain rule:
$$\frac{\partial \mathcal{L}}{\partial \theta}$$
2. Weight update (via Adam optimizer): $\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}$

Where:

- $\theta_t$: parameters at time step t
- Ƞ: learning rate
- $\nabla_\theta \mathcal{L}$: gradient of the loss with respect to parameters

# Results

We are still waiting on our model to run




Preliminary results of our model have a training accuracy > 80%, but our most recent testing accuracy was only around 24%, so we are attempting to improve our model before we downsample our testing data.

Even big GPU cant handle that →

```
--- Loading and Combining Files ---
Detected image shape: (192, 224, 176, 1)
Detected mask shape: (192, 224, 176, 1)
  Loaded pair 50/500...
  Loaded pair 100/500...
  Loaded pair 150/500...
  Loaded pair 200/500...
  Loaded pair 250/500...
  Loaded pair 300/500...
  Loaded pair 350/500...
  Loaded pair 400/500...
  Loaded pair 450/500...
  Loaded pair 500/500...

Stacking 500 valid pairs...
Final combined images shape: (500, 192, 224, 176, 1)
Final combined masks shape: (500, 192, 224, 176, 1)
```

# Next Steps

- Implement skull stripping → this should improve model performance significantly
- Get the model trained
- Predict the model on the 154 MRI scans set aside from the ATLAS data for testing at full resolution (1mm thick slices)
- Manually downsample each of the 154 testing MRI scans (10 mm thick slices) to model data from clinical settings
- Compare the accuracy and determine the next steps from there

# Questions?