Deep Learning with MRI

Neurotech MTL

Support provided by

neuro

Institut et hôpital neurologiques de Montréal
Montreal Neurological Institute and Hospital

NEUROTECHX

CENTRE LUDMER CENTER

NEUROINFORMATIQUE & SANTÉ MENTALE | NEUROINFORMATICS & MENTAL HEALTH

MCIN MCGILL CENTRE for INTEGRATIVE NEUROSCIENCE

McGill | Centre universitaire de santé McGill McGill University Health Centre

About Us

Thomas Funck: Having originally studied philosophy and cognitive science as an undergraduate at McGill, Thomas is now a Ph.D. candidate in neuroscience at the MNI. He uses multi-modal brain imaging, signal processing, and computational simulation to study the cellular architecture of the living brain.
Twitter : @tffunck

Andrew Doyle:
During his MSc with Tal Arbel, Andrew developed machine learning algorithms to detect lesions in multiple sclerosis. He is now continuing his work in the MCIN lab at the MNI, with a special interest in using AI to automate quality control for brain imaging.
Twitter : @crocodoyle

Estefany Saurez: Estefany is a PhD student in neuroscience at the Montreal Neurological Institute. She has a bachelor's degree in Mechanical Engineering and a Master's degree in Industrial Engineering. Estefany has been working for three years at the intersection of AI, neuroscience, and engineering to better understand the relationship between structure and function in brain networks and examine how this can be used to improve the information-processing properties of neuromorphic architectures.
Twitter: LauraESaurez24

neuro

McGill | Centre universitaire de santé McGill McGill University Health Centre

Installation

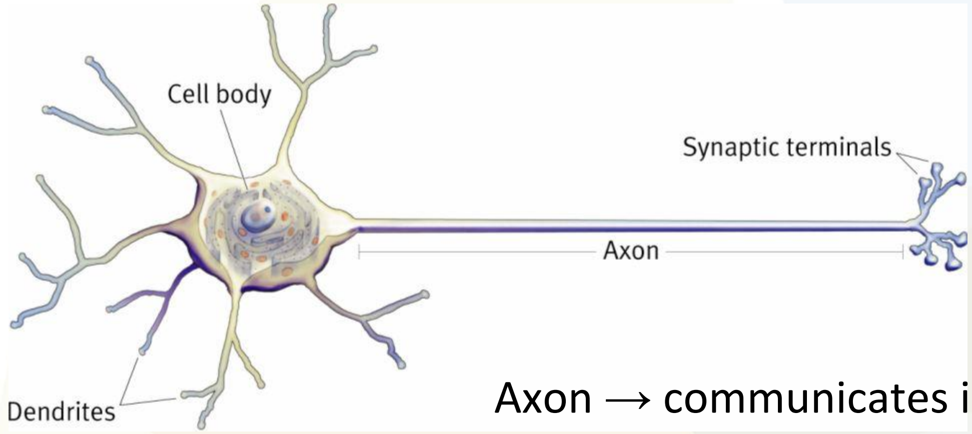
- **www.github.com/tfunck/minc_keras**
    - *has code, data, presentation, and installation instructions*
- Google Colab (super easy)
    - Create / Log-in to Google account
    - Go to https://colab.research.google.com
    - Download and load: https://tinyurl.com/yd8dd5x3
- Docker (very easy):
    - Install docker on your OS
        - https://docs.docker.com/install/#cloud
    - docker pull tffunck/neurotech:latest
    - docker run -it --rm tffunck/neurotech:latest
- DIY (pretty easy):
    - wget https://bootstrap.pypa.io/get-pip.py
        - Or go to the link and download manually
    - python3 get-pip.py
    - pip3 install   pandas numpy scipy h5py tensorflow keras
    - git clone https://github.com/tfunck/minc_keras

<u>Outline</u>

1. Brain imaging and neuroanatomy
2. Machine Learning
3. Deep Learning
4. Keras Example

Super simple neuroanatomy

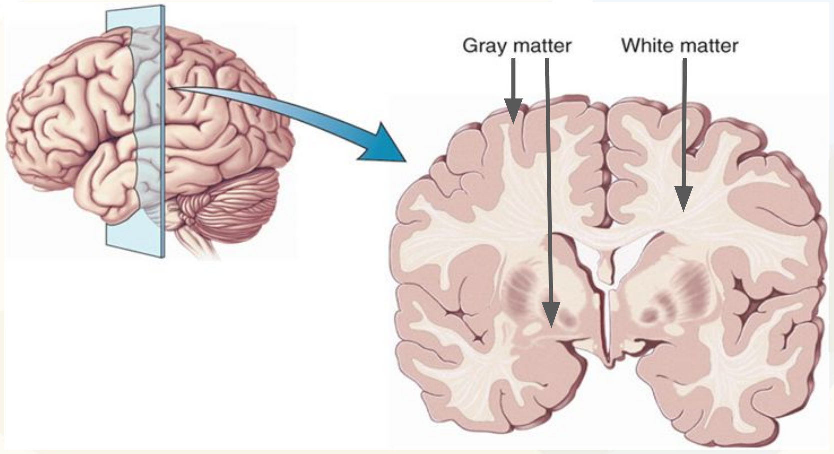
Cell body and dendrites → integrate information



Axon → communicates information downstream

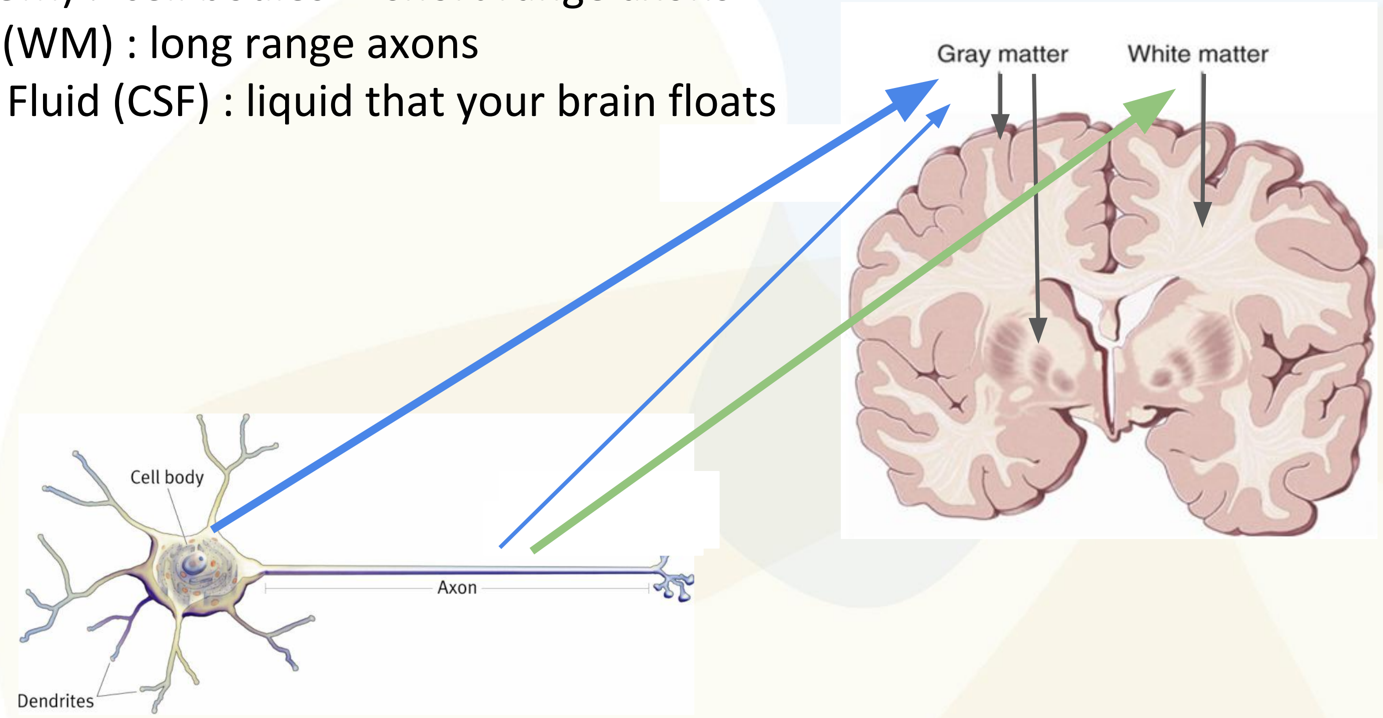
abp

Super simple neuroanatomy



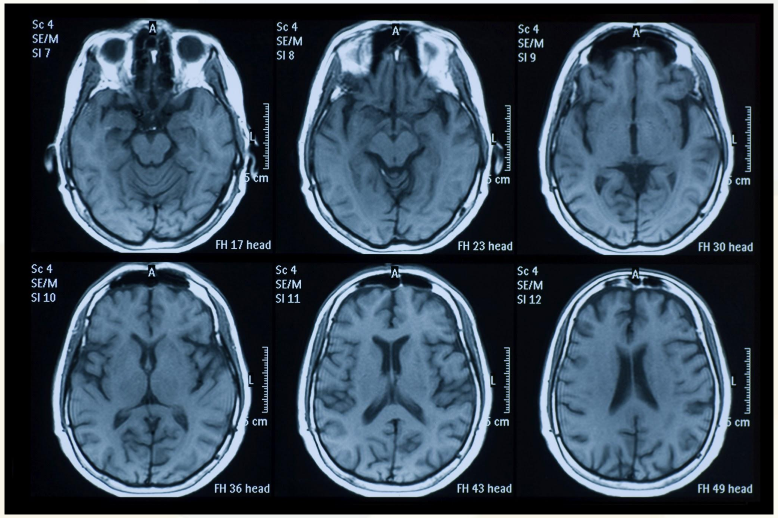
Super simple neuroanatomy

Grey Matter (GM) : cell bodies + short range axons
White Matter (WM) : long range axons
Cerebrospinal Fluid (CSF) : liquid that your brain floats in



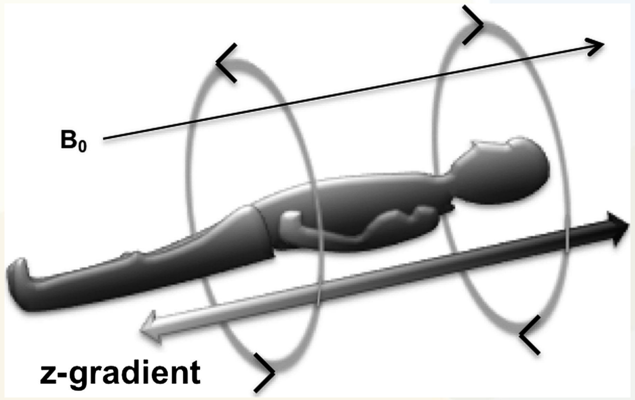
Magnetic Resonance Imaging (MRI)

- Uses powerful magnets to create images of biological tissue (e.g., brains)



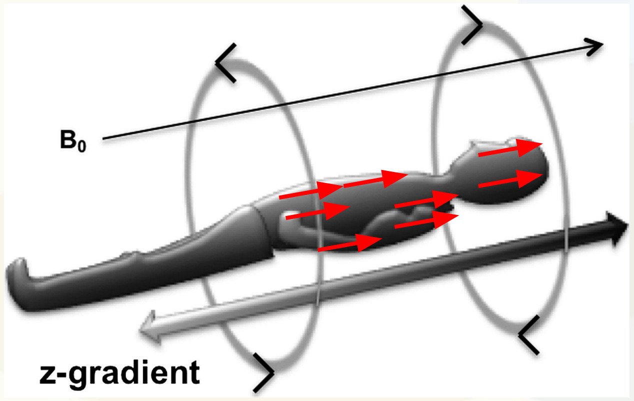
Magnetic Resonance Imaging (MRI)

1. MRI scanners first create a magnetic field along the axis of the scanner



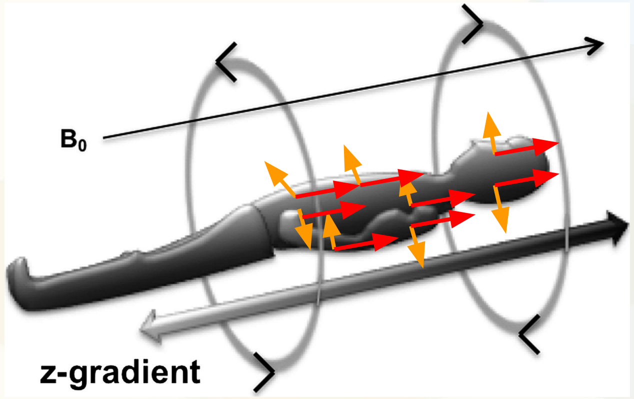
Magnetic Resonance Imaging (MRI)

1. MRI machines first create a magnetic field along the axis of the scanner
2. Aligns hydrogen (H) atoms in the body to the magnetic field



Magnetic Resonance Imaging (MRI)

1. MRI machines first create a magnetic field along the axis of the scanner
2. Aligns hydrogen (H) atoms in the body to the magnetic field
3. MRI emits magnetic pulse that knocks H atoms out of alignment

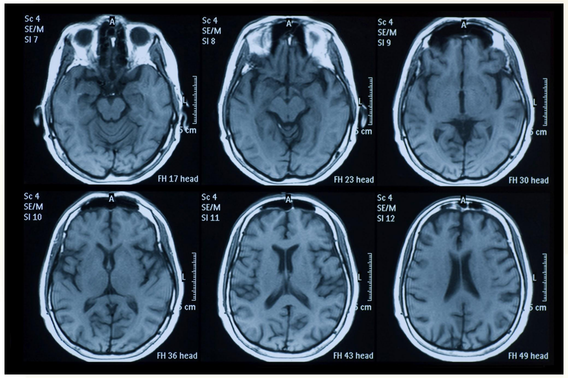


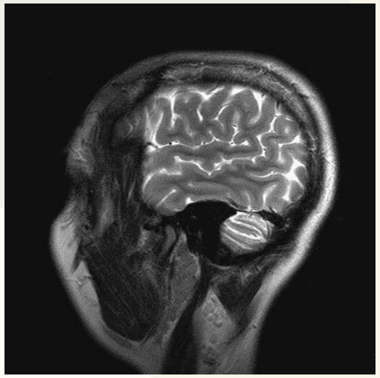
Magnetic Resonance Imaging (MRI)

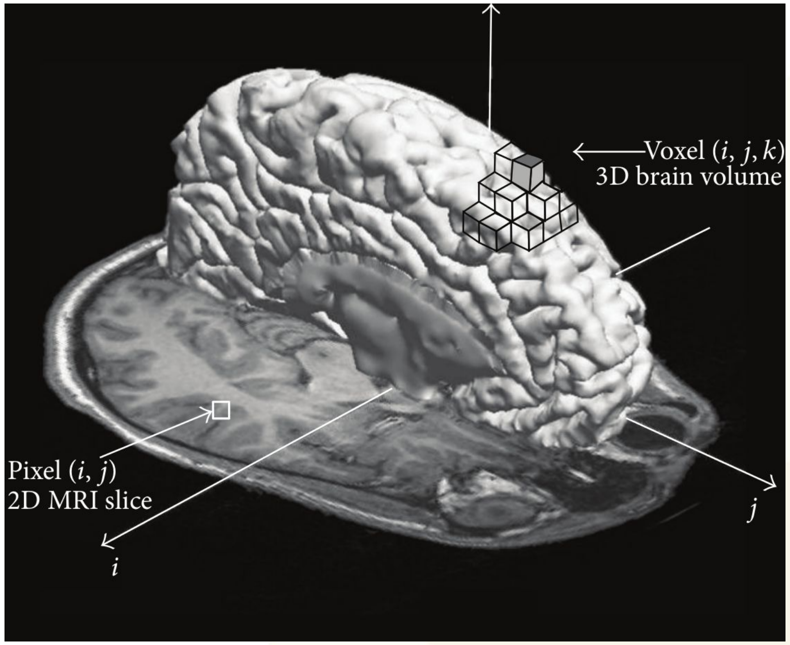
1. MRI machines first create a magnetic field along the axis of the scanner
2. Aligns hydrogen (H) atoms in the body to the magnetic field
3. MRI emits magnetic pulse that knocks H atoms out of alignment
4. Time it takes for H atoms to regain alignment depends on biological tissue

Magnetic Resonance Imaging (MRI)

1. MRI machines first create a magnetic field along the axis of the scanner
2. Aligns hydrogen (H) atoms in the body to the magnetic field
3. MRI emits magnetic pulse that knocks H atoms out of alignment
4. Time it takes for H atoms to regain alignment depends on biological tissue
5. MRI image is based on this realignment time and this reflects type of tissue
    a. Realignment speed : WM (bright) > GM (medium) > CSF (dark)

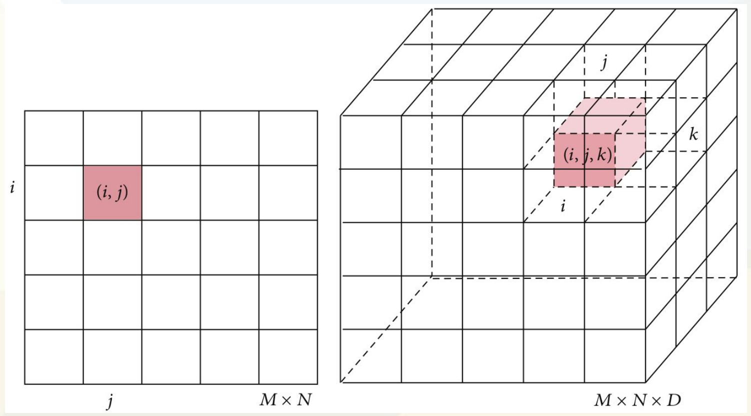






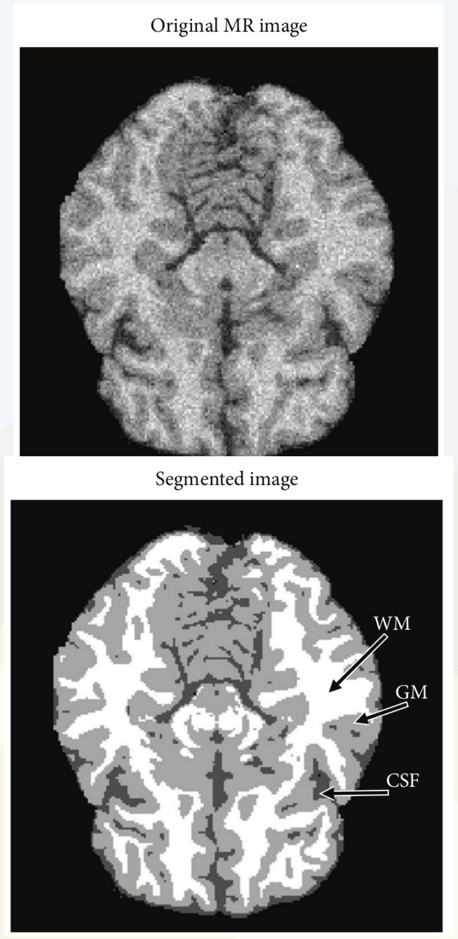
Pixel (2D image)

Voxel (3D volume)



MRI Segmentation

- Lots of ways to analyze MRI
  - Brain size
  - Cortical thickness
  - GM/WM intensity ratio
- Segmenting MRI very useful
  - Segmenting into GM and WM is common processing step
  - Helps to quantify brain metrics measured in these regions
- ML can be used to perform segmentation!

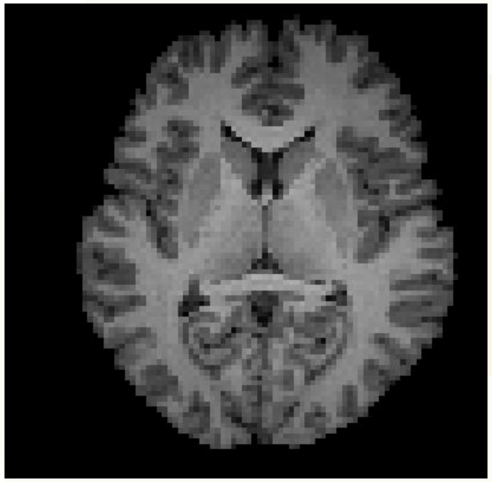


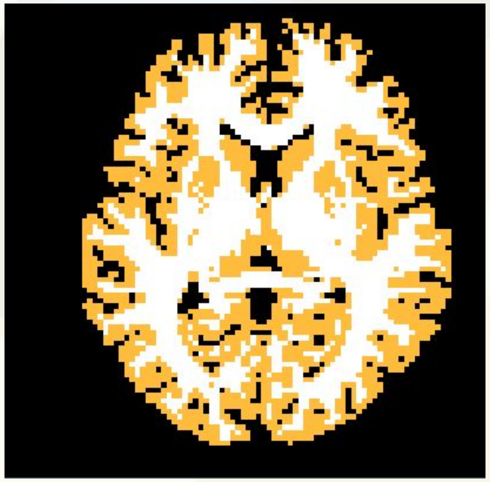
Outline

1. Brain imaging and neuroanatomy
2. Machine Learning
3. Deep Learning
4. Keras Example

1000 Functional Connectomes Project

- 261 MRI (skull stripped)
    - http://fcon_1000.projects.nitrc.org/fcpClassic/FcpTable.html
- Data stored in MINC (.mnc)
    - MINC is a medical imaging format based on HDF5
- GM/WM segmentation with FSL-5.0-fast



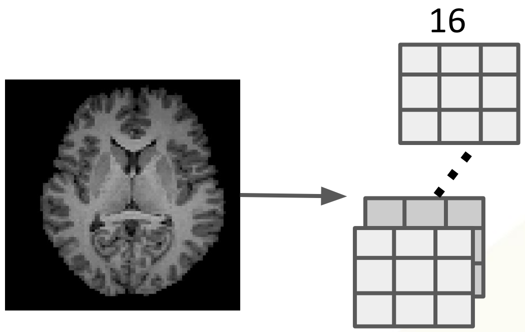


Build a Conv Net with Keras

Define inputs to your network

```
#Setup the input (IN) based on image dimensions <image_dim>
IN = Input(shape=(image_dim[1], image_dim[2],1))

#Create 4 convolutional layers with 3x3 kernels and 16, 32, 64, 128 kernels per layer
CONV1 = Conv2D(16,  kernel_size=[3,3],activation='relu',padding='same')(IN)
CONV2 = Conv2D(32,  kernel_size=[3,3],activation='relu',padding='same')(CONV1)
CONV3 = Conv2D(64,  kernel_size=[3,3],activation='relu',padding='same')(CONV2)
CONV4 = Conv2D(128, kernel_size=[3,3],activation='relu',padding='same')(CONV3)

#Setup output layer (OUT)
OUT = Conv2D(1, kernel_size=1,  padding='same', activation='softmax')(CONV4)

#Create model
model = keras.models.Model(inputs=[IN], outputs=OUT)
```

Build a Conv Net with Keras

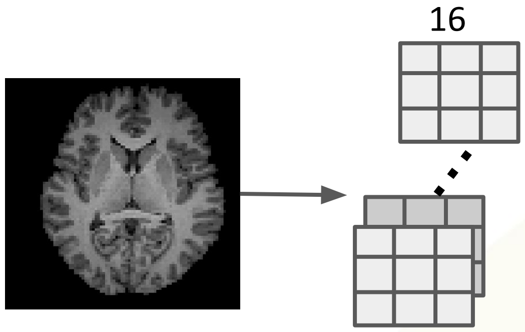


Create 1st layer of convolution kernels

```
#Setup the input (IN) based on image dimensions <image_dim>
IN = Input(shape=(image_dim[1], image_dim[2],1))

#Create 4 convolutional layers with 3x3 kernels and 16, 32, 64, 128 kernels per layer
CONV1 = Conv2D(16,
kernel_size=[3,3],activation= 'relu',padding='same')(IN)
CONV2 = Conv2D(32,  kernel_size=[3,3],activation='relu',padding='same')(CONV1)
CONV3 = Conv2D(64,  kernel_size=[3,3],activation='relu',padding='same')(CONV2)
CONV4 = Conv2D(128, kernel_size=[3,3],activation='relu',padding='same')(CONV3)

#Setup output layer (OUT)
OUT = Conv2D(3, kernel_size=1,  padding='same', activation='softmax')(CONV4)

#Create model
model = keras.models.Model(inputs=[IN], outputs=OUT)
```

Previous layer is argument to new layer

Build a Conv Net with Keras

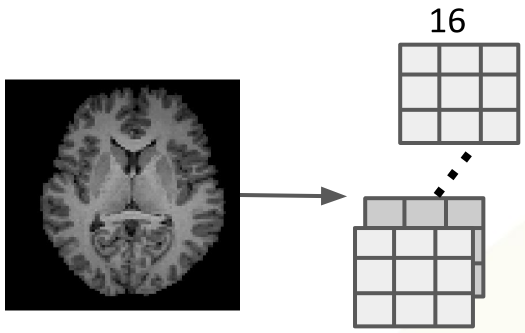


Create 1st layer of convolution kernels

```
#Setup the input (IN) based on image dimensions <image_dim>
IN = Input(shape=(image_dim[1], image_dim[2],1))

#Create 4 convolutional layers with 3x3 kernels and 16, 32, 64, 128 kernels per layer
CONV1 = Conv2D(16,
kernel_size=[3,3],activation= 'relu' ,padding='same')(IN)
CONV2 = Conv2D(32,  kernel_size=[3,3],activation='relu',padding='same')(CONV1)
CONV3 = Conv2D(64,  kernel_size=[3,3],activation='relu',padding='same')(CONV2)
CONV4 = Conv2D(128, kernel_size=[3,3],activation='relu',padding='same')(CONV3)

#Setup output layer (OUT)
OUT = Conv2D(3, kernel_size=1,  padding='same', activation='softmax')(CONV4)

#Create model
model = keras.models.Model(inputs=[IN], outputs=OUT)
```

Size of kernels

Build a Conv Net with Keras

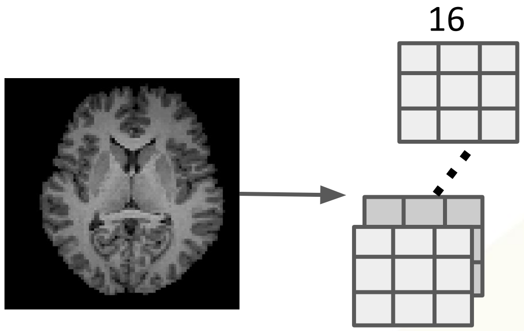


Create 1st layer of convolution kernels

```
#Setup the input (IN) based on image dimensions <image_dim>
IN = Input(shape=(image_dim[1], image_dim[2],1))

#Create 4 convolutional layers with 3x3 kernels and 16, 32, 64, 128 kernels per layer
CONV1 = Conv2D(16,
kernel_size=[3,3],activation= 'relu',padding='same')(IN)
CONV2 = Conv2D(32,  kernel_size=[3,3],activation='relu',padding='same')(CONV1)
CONV3 = Conv2D(64,  kernel_size=[3,3],activation='relu',padding='same')(CONV2)
CONV4 = Conv2D(128, kernel_size=[3,3],activation='relu',padding='same')(CONV3)

#Setup output layer (OUT)
OUT = Conv2D(3, kernel_size=1,  padding='same', activation='softmax')(CONV4)

#Create model
model = keras.models.Model(inputs=[IN], outputs=OUT)
```

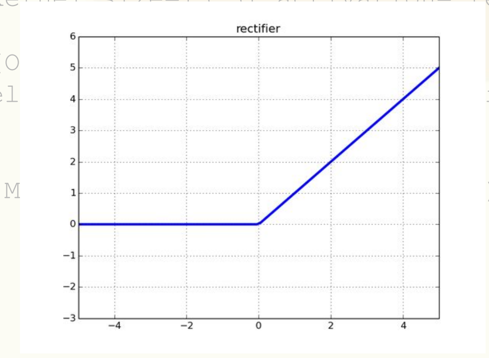
Number of kernels

Build a Conv Net with Keras



Create 1st layer of convolution kernels

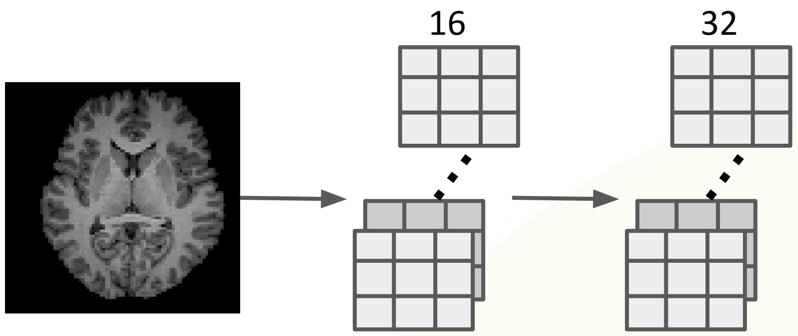
```
#Setup the input (IN) based on image dimensions <image_dim>
IN = Input(shape=(image_dim[1], image_dim[2],1))

#Create 4 convolutional layers with 3x3 kernels and 16, 32, 64, 128 kernels per layer
CONV1 = Conv2D(16,
kernel_size=[3,3],activation= 'relu',padding='same')(IN)
CONV2 = Conv2D(32,  kernel_size=[3,3],activation='relu',padding='same')(CONV1)
CONV3 = Conv2D(64,  kernel_size=[3,3],activation='relu',padding='same')(CONV2)
CONV4 = Conv2D(128, kernel_size=[3,3],activation='relu',padding='same')(CONV3)

#Setup output layer (O
OUT = Conv2D(3, kernel                                   ivation='softmax')(CONV4)

#Create model
model = keras.models.M                                   )
```



https://qph.fs.quoracdn.net/main-qimg-f3918735cc25a283752cb33f9ed77d48

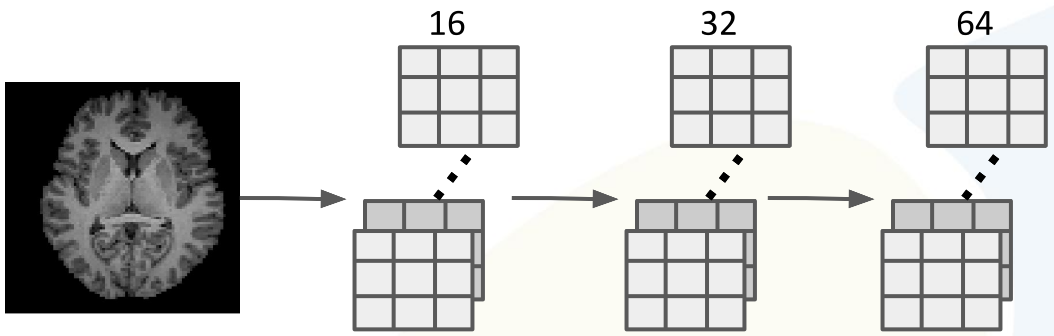
Activation function = rectified linear

Build a Conv Net with Keras



```
#Setup the input (IN) based on image dimensions <image_dim>
IN = Input(shape=(image_dim[1], image_dim[2],1))

#Create 4 convolutional layers with 3x3 kernels and 16, 32, 64, 128 kernels per layer
CONV1 = Conv2D(16,  kernel_size=[3,3],activation='relu',padding='same')(IN)
CONV2 = Conv2D(32, 
kernel_size=[3,3],activation= 'relu',padding='same')(CONV1)
CONV3 = Conv2D(64,  kernel_size=[3,3],activation='relu',padding='same')(CONV2)
CONV4 = Conv2D(128, kernel_size=[3,3],activation='relu',padding='same')(CONV3)

#Setup output layer (OUT)
OUT = Conv2D(3, kernel_size=1,  padding='same', activation='softmax')(CONV4)

#Create model
model = keras.models.Model(inputs=[IN], outputs=OUT)
```

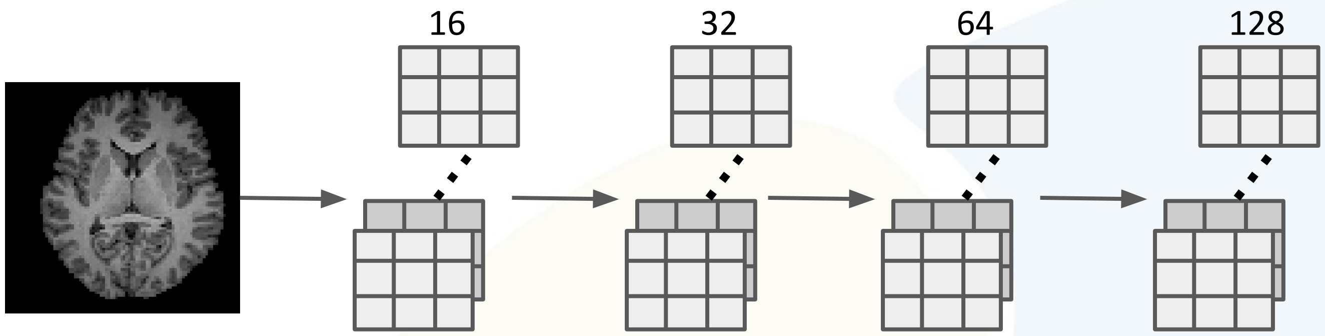
Build a Conv Net with Keras



Create 3rd layer of convolution kernels

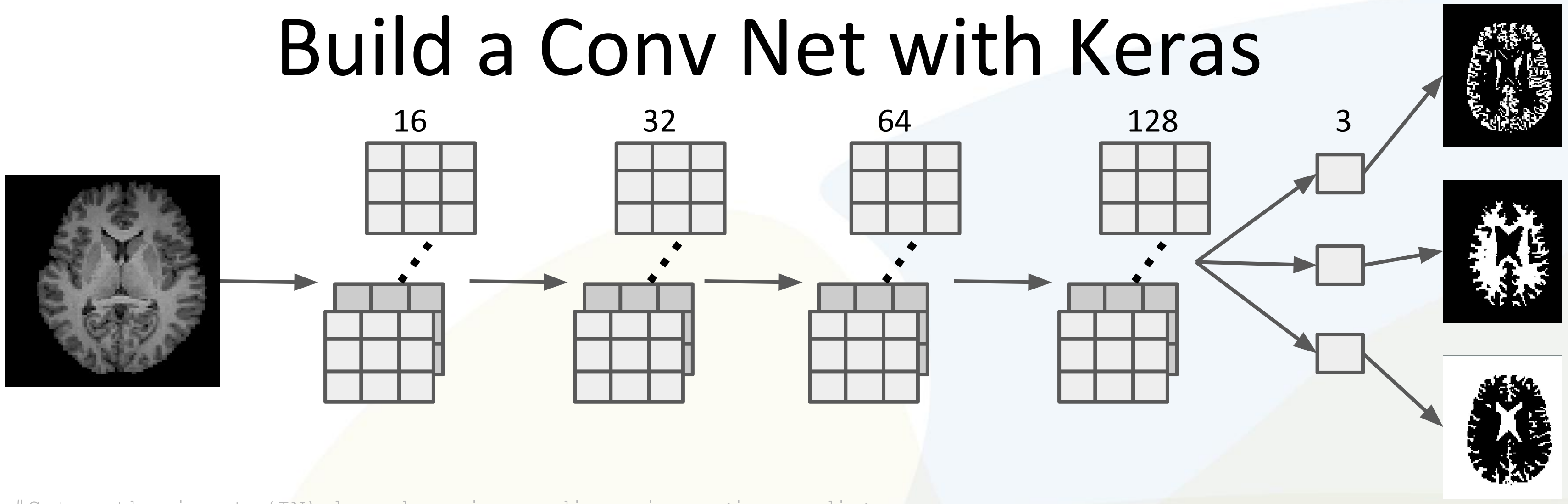
```
#Setup the input (IN) based on image dimensions <image_dim>
IN = Input(shape=(image_dim[1], image_dim[2],1))

#Create 4 convolutional layers with 3x3 kernels and 16, 32, 64, 128 kernels per layer
CONV1 = Conv2D(16,  kernel_size=[3,3],activation='relu',padding='same')(IN)
CONV2 = Conv2D(32,  kernel_size=[3,3],activation='relu',padding='same')(CONV1)
CONV3 = Conv2D(64, kernel_size=[3,3],activation= 'relu',padding='same')(CONV2)
CONV4 = Conv2D(128, kernel_size=[3,3],activation='relu',padding='same')(CONV3)

#Setup output layer (OUT)
OUT = Conv2D(3, kernel_size=1,  padding='same', activation='softmax')(CONV4)

#Create model
model = keras.models.Model(inputs=[IN], outputs=OUT)
```

Build a Conv Net with Keras



```
#Setup the input (IN) based on image dimensions <image_dim>
IN = Input(shape=(image_dim[1], image_dim[2],1))

#Create 4 convolutional layers with 3x3 kernels and 16, 32, 64, 128 kernels per layer
CONV1 = Conv2D(16,  kernel_size=[3,3],activation='relu',padding='same')(IN)
CONV2 = Conv2D(32,  kernel_size=[3,3],activation='relu',padding='same')(CONV1)
CONV3 = Conv2D(64,  kernel_size=[3,3],activation='relu',padding='same')(CONV2)
CONV4 = Conv2D(128,
kernel_size=[3,3],activation= 'relu',padding='same')(CONV3)

#Setup output layer (OUT)
OUT = Conv2D(3, kernel_size=1,  padding='same', activation='softmax')(CONV4)

#Create model
model = keras.models.Model(inputs=[IN], outputs=OUT)
```

Build a Conv Net with Keras



```
#Setup the input (IN) based on image dimensions <image_dim>
IN = Input(shape=(image_dim[1], image_dim[2],1))

#Create 4 convolutional layers with 3x3 kernels and 16, 32, 64, 128 kernels per layer
CONV1 = Conv2D(16,  kernel_size=[3,3],activation='relu',padding='same')(IN)
CONV2 = Conv2D(32,  kernel_size=[3,3],activation='relu',padding='same')(CONV1)
CONV3 = Conv2D(64,  kernel_size=[3,3],activation='relu',padding='same')(CONV2)
CONV4 = Conv2D(128, kernel_size=[3,3],activation='relu',padding='same')(CONV3)

#Setup output layer (OUT)
OUT = Conv2D(3, kernel_size=1,  padding='same',
activation='softmax')(CONV4)

#Create model
model = keras.models.Model(inputs=[IN], outputs=OUT)
```

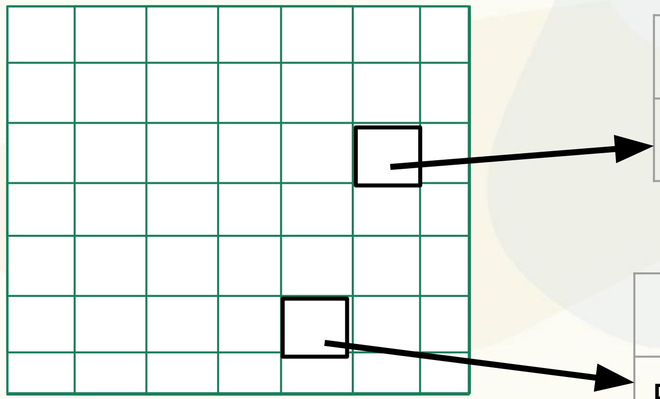
Create final output layer

Activation function = softmax

Softmax Output of Network

<u>Softmax</u>

- Creates pseudo-probability distribution
  - for each category values between (0,1)
  - sum of values from each category = 1
- Each pixel = [P(GM) P(WM), P(BG)]
- 3D output array = (Width, Height, 3)



	GM	WM	CSF
Probability	0.1	0.85	0.05

	GM	WM	CSF
Probability	0.7	0.2	0.1

- Transform to 2D image by finding class with max probability at each pixel

Setup & Fit Model

make_and_run_model.py

```
def compile_and_run(X_train, Y_train, X_validate, Y_validate, epochs, model):
     #setup optimizer
     ada = keras.optimizers.Adam(0.0001)

     #setup define loss function
     loss function = 'categorical_crossentropy’

     #compile the model
     model.compile(loss = loss_function, optimizer=ada,metrics=[categorical_accuracy] )

     #fit model
     model.fit([X_train],Y_train,batch_size,validation_data=([X_validate],Y_validate),epochs=epo
     chs])

     #save model
     model.save(model_name)
```

Base Model

models/neurotech_models.py

```
def base_model(image_dim, nK, kernel_size, drop_out):
     # nK = number of kernels per layer
     # kernel_size = size of the kernels
     # dropout = dropout rate for each layer

     #Setup the input (IN) and output (OUT) layers based on image dimensions
     IN = OUT = Input(shape=(image_dim[1], image_dim[2],1))

     n_layers=int(len(nK)) # number of layer
     kDim=[kernel_size] * n_layers #list of kernel size equal in length to n_layers

     for i in range(n_layers): # for each layer...
          OUT=Conv2D(nK[i],kernel_size=[kDim[i],kDim[i]],activation='relu',padding='same')(OUT)
          OUT = Dropout(drop_out)(OUT)

     OUT = Conv2D(1, kernel_size=1,  padding='same', activation='sigmoid')(OUT)
     model = keras.models.Model(inputs=[IN], outputs=OUT)
     return(model)
```

model_0_0

models/neurotech_models.py

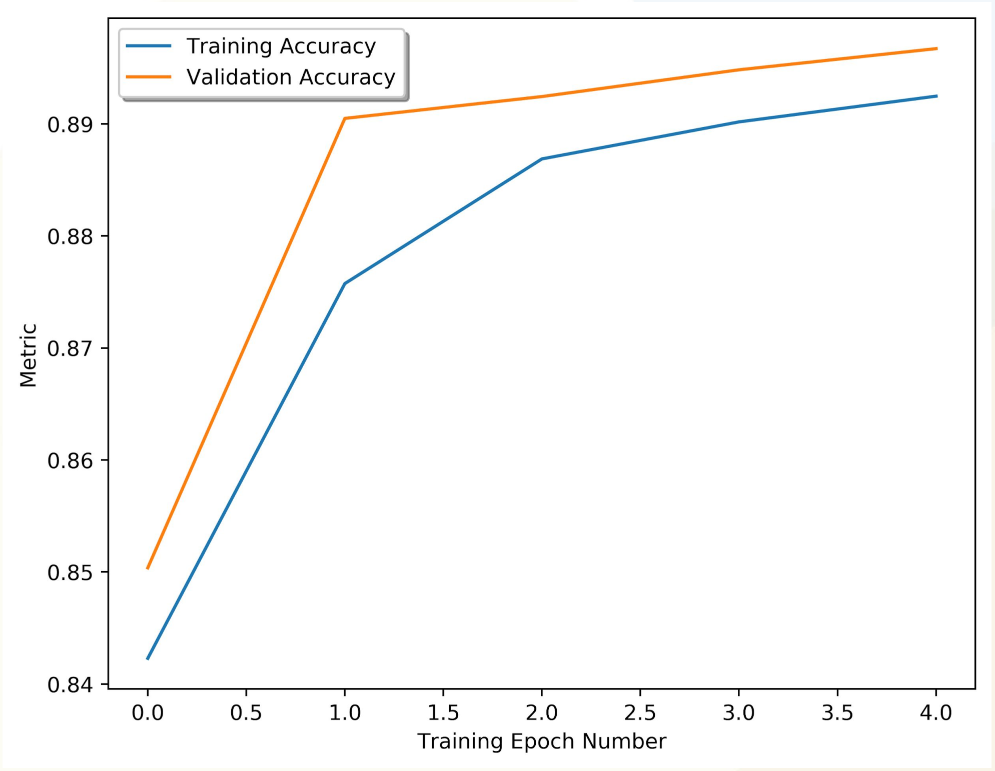
```
def model_0_0(image_dim):
    # Create a convolutional network with
    # [3,3] x 16
    # [3,3] x 16
    # [3,3] x 16
    nK=[16,16,16]
    kernel_size = 3
    drop_out=0
    return base_model( image_dim, nK, kernel_size, drop_out)
```

How to run a model with *minc_keras.py*

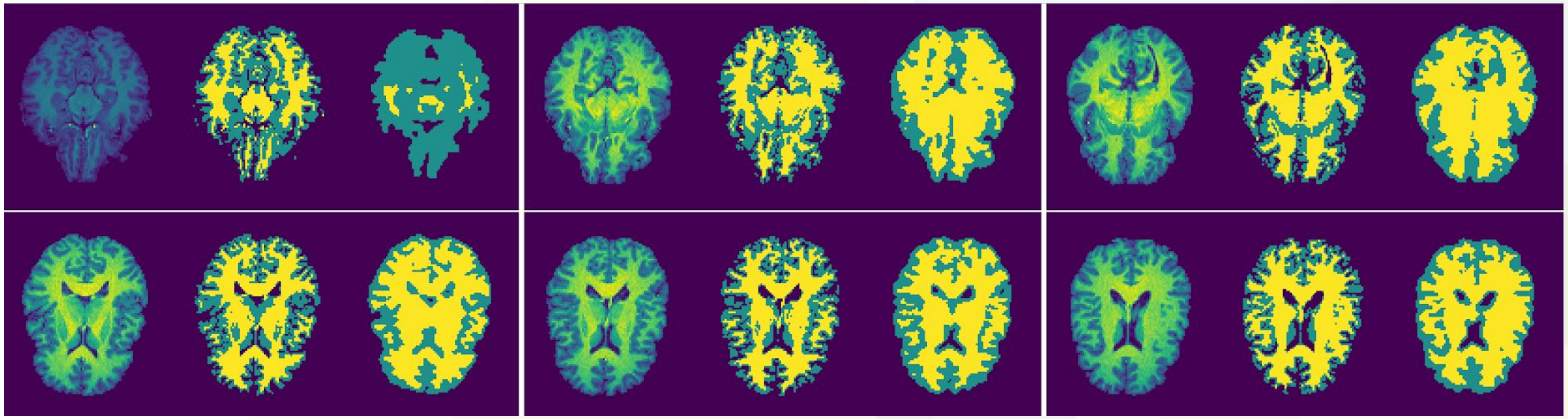
```
python3 minc_keras/minc_keras.py
          --source output/                # source dir where input data is stored
          --target .                      # output dir where results will be placed
          --epochs 5                      # number of epochs to use to fit model
          --model-type "model_0_0”        # name of model you want to use
          --input-str "*T1w_anat*"        # string that identifies the input MRI images
          --label-str "*seg*"             # string that identifies the labeled GM/WM images
          --predict 1                     # comma-separated list of subjects for prediction
          --ratios 0.2 0.15               # ratio to use in training/validation/test splits
          --activation-hidden "relu"      # activation function for hidden layer
          --activation-output "softmax" # activation function for output layer
```

Results for model_0_0

- Parameters = 4,851
- Test Accuracy= 0.896



Results for model_0_0



model_1_0

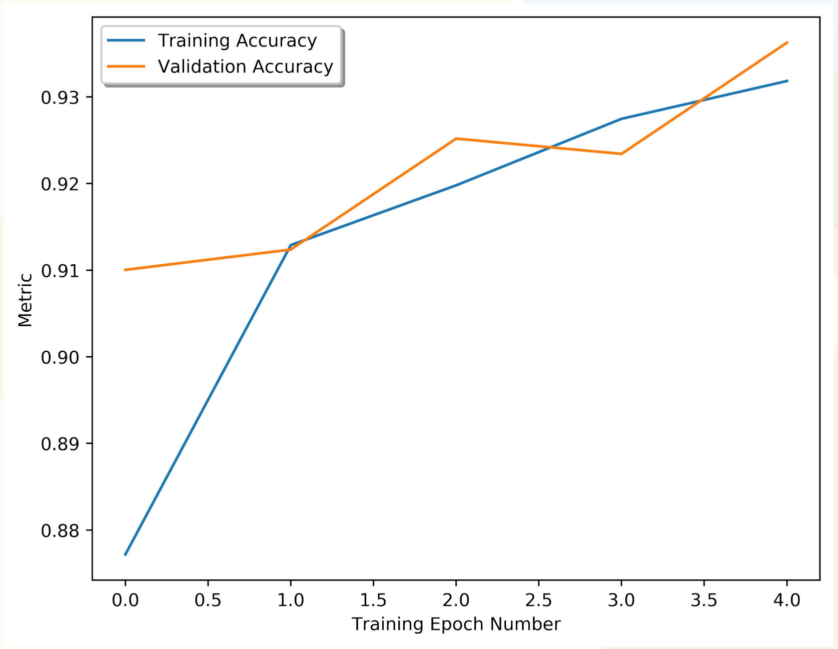
models/neurotech_models.py

```
def model_1_0(image_dim):
     # Create a convolutional network with
     # [3,3] x 16
     # [3,3] x 16
     # [3,3] x 32
     # [3,3] x 32
     # [3,3] x 32
     # [3,3] x 64
     # [3,3] x 64
     # [3,3] x 64
     # [3,3] x 128
     nK=[16,16,32,32,32,64,64,64,128]
     kernel_size = 3
     drop_out=0
     return base_model( image_dim, nK, kernel_size, drop_out)
```

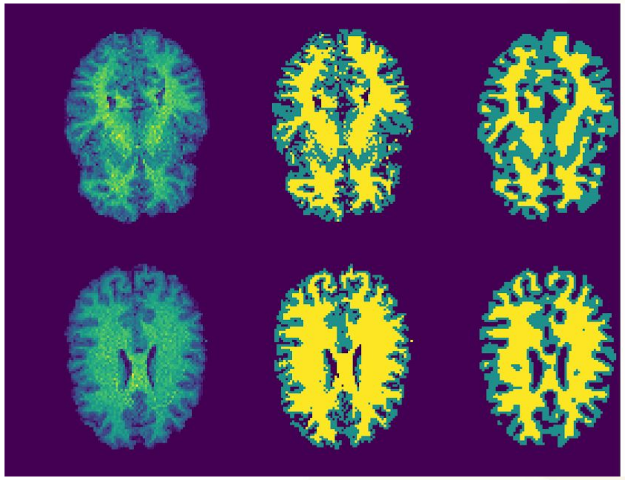
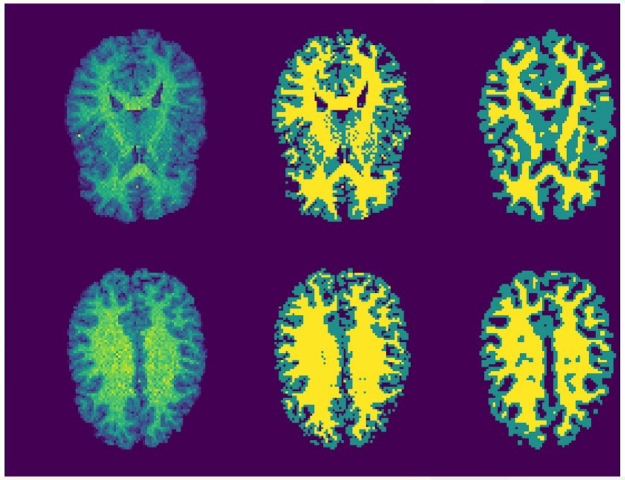
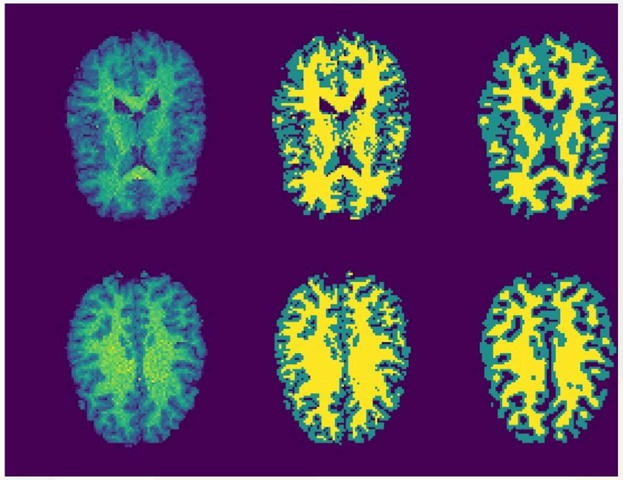
```
python3 minc_keras/minc_keras.py --source output/ --target . --epochs 5 --model-type "model_1_0"
--input-str "*T1w_anat*" --label-str "*seg*" --predict 1 --ratios .2 .15
```

Results for model_1_0

- Parameters = 192,211
    - model_0_0: 4,851
- Test Accuracy = 0.93
    - model_0_0: 0.89



Results for model_1_0

model_4_0

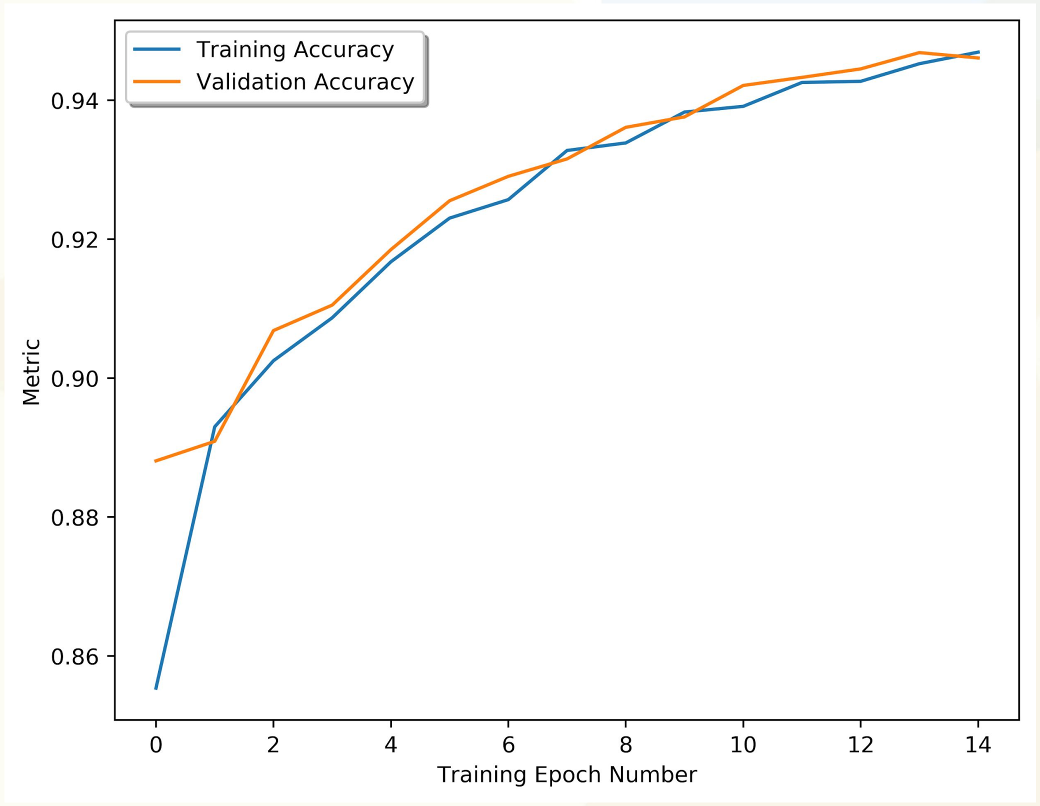
models/neurotech_models.py

```
def model_4_0(image_dim):
      # Create a convolutional network with
      # [5,5] x 32
      # [5,5] x 32
      # [5,5] x 32
      # [5,5] x 32
      # [5,5] x 64
      # [5,5] x 64
      # [5,5] x 64
      # [5,5] x 128
      # [5,5] x 128
      # [5,5] x 128
      nK=[32,32,32,32,64,64,64,128,128,128]
      kernel_size = 5
      drop_out=0
      return base_model( image_dim, nK, kernel_size, drop_out)
```

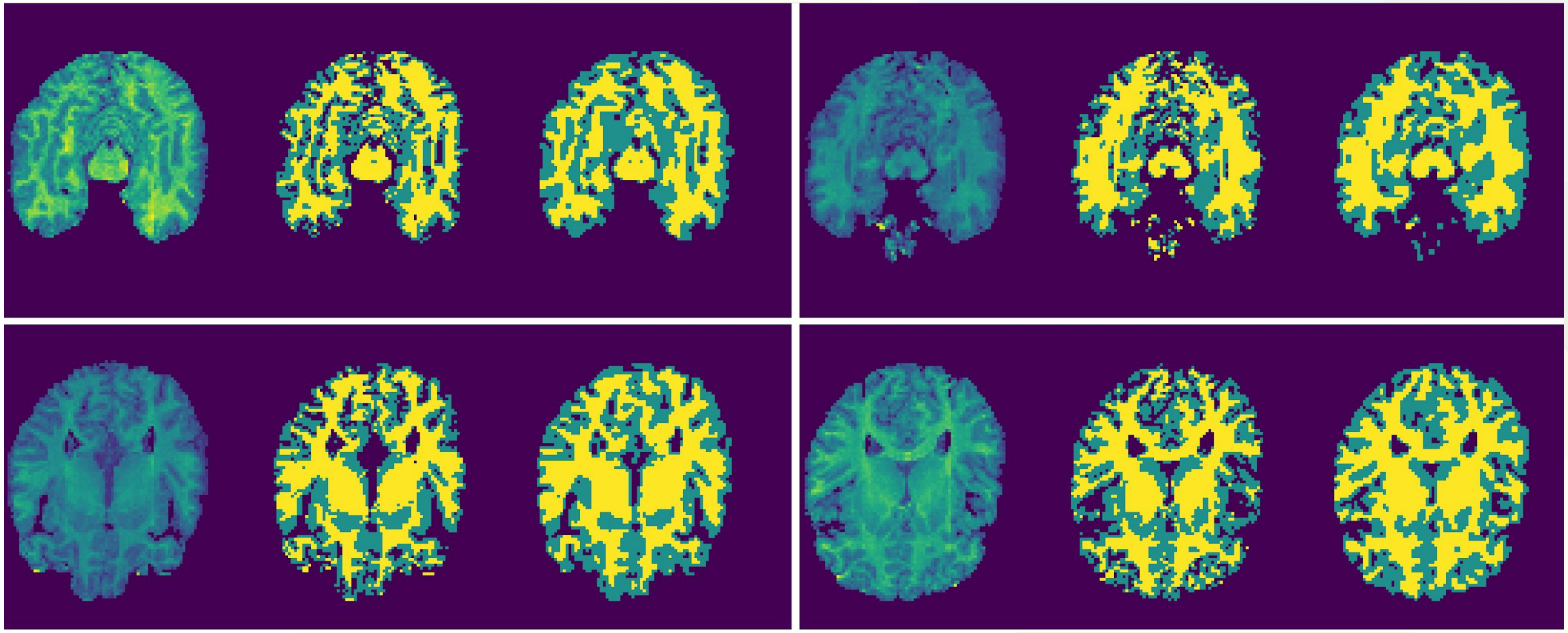
```
python3 minc_keras/minc_keras.py --source output/ --target . --epochs 15 --model-type"model_4_0”
--input-str "*T1w_anat*” --label-str "*seg*" --predict 1 --ratios .1 .15
```

Results for model_4_0

- Parameters = 1,358,691
  - model_1_0: 192,211
- Test Accuracy: 0.946
  - model_1_0: 0.93



Results for model_4_0



Take home message

- Metrics should be compared to baseline performance
  - if model predicts all 0 and has 70% accuracy, then 80% accuracy is not very good
- Beware of overfitting!
  - regularization can help
  - use an appropriate method of cross-validation
  - good practice to keep data aside for final validation
  - get more data
- More parameters doesn't always produce a better model
  - stacking more layers may not be best way to improve performance
  - more sophisticated networks have better feature representation

Support provided by

neuro

Institut et hôpital neurologiques de Montréal
Montreal Neurological Institute and Hospital

NEUROTECHX

CENTRE LUDMER CENTER

NEUROINFORMATIQUE & SANTÉ MENTALE | NEUROINFORMATICS & MENTAL HEALTH

MCIN MCGILL CENTRE for INTEGRATIVE NEUROSCIENCE

McGill | Centre universitaire de santé McGill McGill University Health Centre

Free coding

1. Modifying the code
    a. Beginner
        i. --nk "16,107,9" → *sets number of kernels per layer*
        ii. --kernel-size 3 → *set size of kernels*
        iii. --dropout 0.25 → *set drop out rate*
        iv. --model-type "custom"
    b. Advanced
        i. Create/Login to your Github
        ii. Fork minc_keras
        iii. git clone onto your computer / edit code / push back to your repo
        iv. clone from your forked minc_keras on Google Collab
2. Can you find a model that does better than 95% without overfitting?
    a. What happens when you change the activation functions?
3. You can change the training/validation/test ratio to increase samples for training
    a. e.g., --ratios 0.3 0.3 (30% training, 30% validation, 60% split)
4. Ask volunteers and organizers for help!