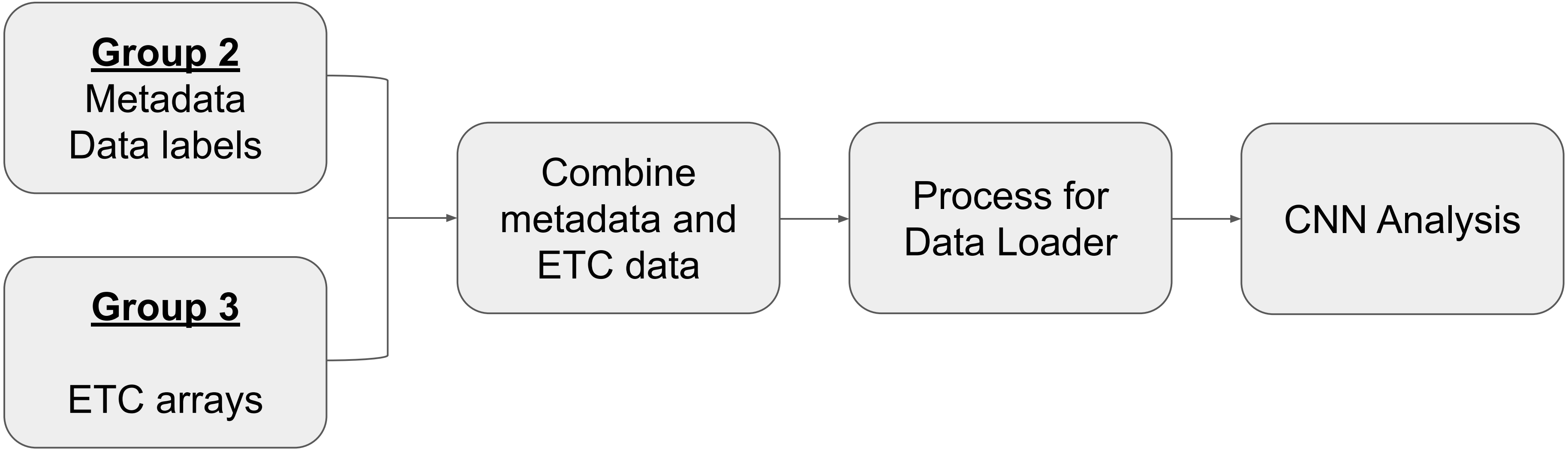
Group 4 - Deep Learning

Joshua Temple, Jill Check, Julia Zheng, Joy Li, Carlos Pimentel, Paige Durant, Dan Dick, Sarah McGuire

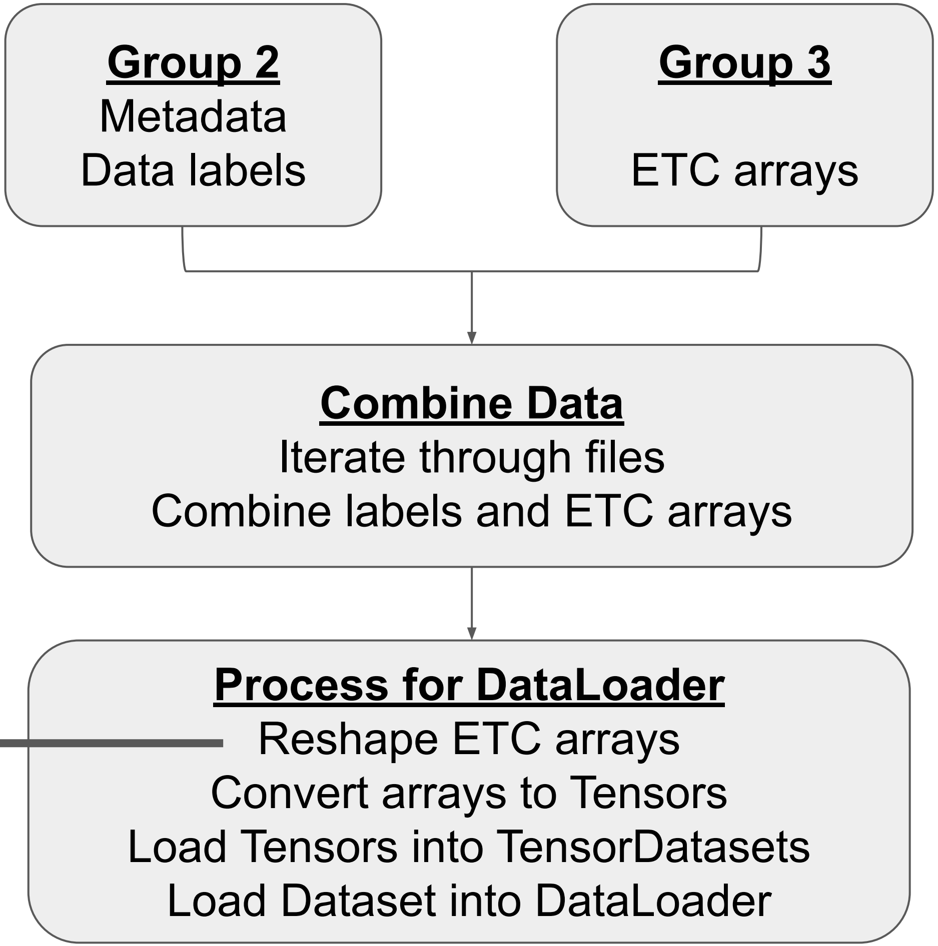
HRT 841
December 8th, 2022

Workflow

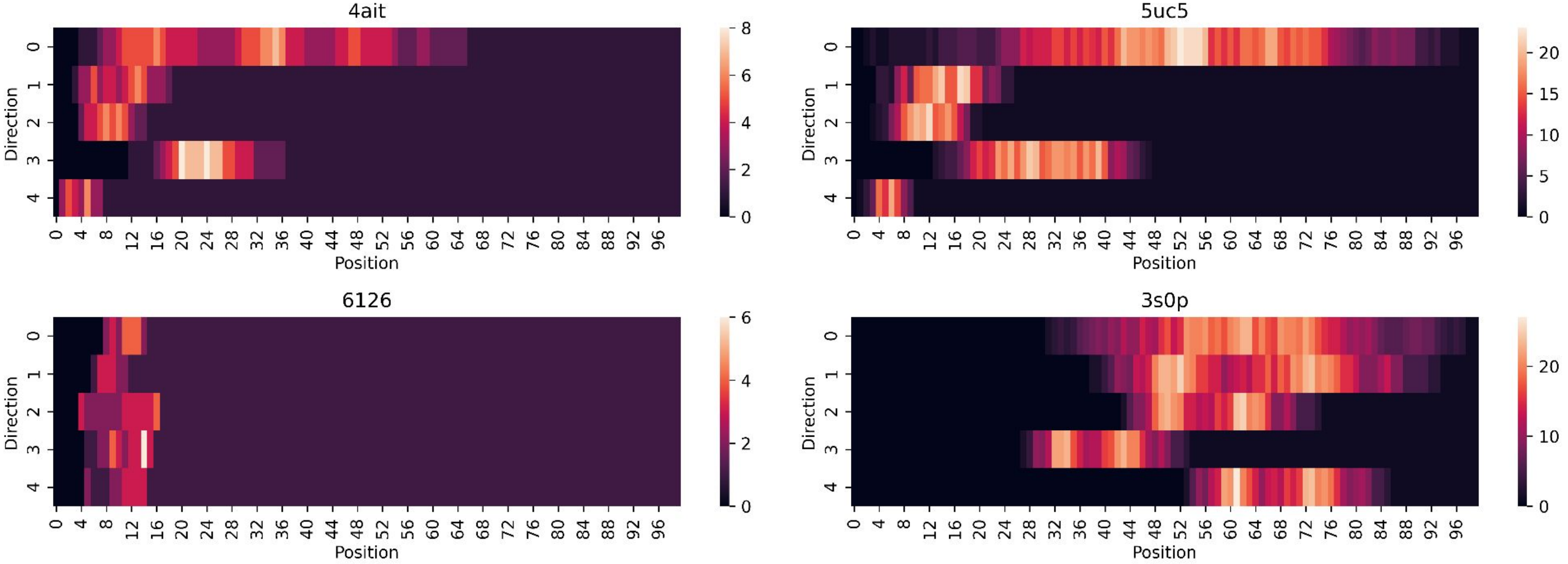

Group 2
Metadata
Data labels
Group 3
ETC arrays
Combine metadata and ETC data
Process for Data Loader
CNN Analysis

DataLoader

```
array([[[0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 3, 5, ..., 1, 1, 1]],

       [[0, 0, 1, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 1, 3, ..., 1, 1, 1]],

       [[0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1]],

       [[0, 0, 0, ..., 3, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1]]])
```



Visualization of Reshaped ETC Arrays



Convolutional Neural Network (CNN) examples

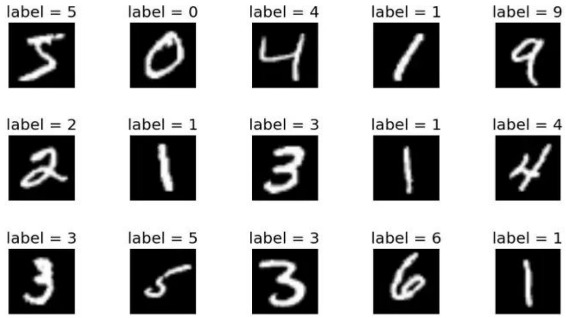
Import MNIST dataset → **Split test and train data** → **Define CNN parameters**

Split test and train data:

- Training data: [60000, 28, 28]
- Testing data: [10000, 28, 28]

Define CNN parameters:

- 2 fully convolutional layers
- ReLu activation function
- Entropy loss function
- Learning rate = 0.01
- Max pooling

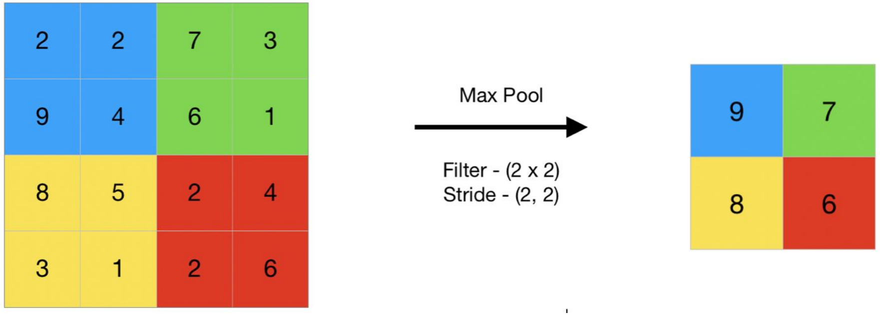


Akash Panchal, Towards Data Science

Train the model → **Evaluate the model using test data**

Train the model: Create a train function and pass through model parameters, data and number of epochs

Evaluate the model using test data: Output: percentage of correctly classified test data set images



https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

CNN for binomial classification

1. Setup and loading data

1.1 Install dependencies and setup

```
[5] pip install tensorflow tensorflow-gpu opencv-python matplotlib
```

```
import tensorflow as tf
import os
import cv2
import imghdr
from matplotlib import pyplot as plt
import numpy as np

# Deep Learning Model

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout

# Model Evaluation

from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy

# Saving the model

from tensorflow.keras.models import load_model
```

1.2 Load Data

```
#Known data directory, this directory has to have all data sorted. Class 1
# images on a directory, class 2 images in another directory and so on.
data_dir = 'path_to_data_directory'
```

```
[ ] # Code line to know which classes 'data_dir' has inside
os.listdir(data_dir)
```

```
[ ] # Code line to automatically indicate the code the data directory and how big
# the batches of images will be
# tf.keras.utils.image_dataset_from_directory?? can help to check other dataset
# parameters

data = tf.keras.utils.image_dataset_from_directory('path_to_data_directory',
                                                   batch_size = 15)
```

```
[ ] # Visual representation of the data set
data_iterator = data.as_numpy_iterator()
batch = data_iterator.next()
```

```
[ ] # Visual representation of the data set
fig, ax = plt.subplots(ncols=8, figsize = (20,20))
for idx, img in enumerate(batch[0][:8]):
  ax[idx].imshow(img.astype(int))
  ax[idx].title.set_text(batch[1][idx])
```

CNN for binomial classification

2. Image Preprocessing

2.1 Scale Images

```
[ ] # Scaling the image means that the pixels that had a numerical value between
    # 0 and 255, now will have a value between 0 and 1 in order to facilitate the
    # calculus.

    data = data.map(lambda x,y: (x/255, y))
```

```
[ ] # Proof tha the max value is 1
    batch[0].max()
```

```
[ ] # Proof tha the min value is 0
    batch[0].min()
```

```
[ ] # Visual representation of the scaled data
    scaled_iterator = data.as_numpy_iterator()
    batch = scaled_iterator.next()
```

```
[ ] fig, ax = plt.subplots(ncols=8, figsize = (20,20))
    for idx, img in enumerate(batch[0][:8]):
      ax[idx].imshow(img)
      ax[idx].title.set_text(batch[1][idx])
```

2.2 Split Data

```
[ ] # This will let us know how many batches we have
    # A batch can be seen as an array that contains info of given quantity of images
    # (this quantity was given in the 3rd code line of '1.2 Load Data')
    len(data)
```

```
[ ] # Define the # of batches that will be used for training (~70% of data)
    train_size = int(len(data)*.7)
    train_size
```

```
[ ] # Define the # of batches that will be used for validation (~20% of data)
    val_size = int(len(data)*.2)+1
    val_size
```

```
[ ] # Define the # of batches that will be used for testing (~10% of data)
    test_size = int(len(data)*.1)+1
    test_size
```

```
[ ] # Sorting data
    train = data.take(train_size)
    val = data.skip(train_size).take(val_size)
    test = data.skip(train_size + val_size).take(test_size)
```

CNN for binomial classification

3.1 Deep learning model (architecture design)

```
[ ] # Define the type of model
    # Sequential means that there is just one kind of input and only one output
    model = Sequential()
```

```
[ ] # tf.keras.layers?? Manual to the different kind of layer
```

```
[ ] # Architecture design

    model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape = (256,256,3)))
    model.add(MaxPooling2D())
    # 16 is the number of filters
    # (3,3) size of the filter
    # 1 means that the convolution will go trhoug every single part of the image

    model.add(Conv2D(32, (3,3), 1, activation = 'relu'))
    model.add(MaxPooling2D())

    model.add(Conv2D(16, (3,3), 1, activation = 'relu'))
    model.add(MaxPooling2D())

    model.add(Flatten())

    model.add(Dense(256, activation = 'relu'))
    model.add(Dense(1, activation = 'sigmoid'))
    # I used sigmoid because it is a binomial CNN so the sigmoid activation function
    # works well for binary classification because it just has two outputs, 0 or 1
```

```
# This code line allows to compile the CNN but it is intended to work for binary
# classification
model.compile('adam', loss = tf.losses.BinaryCrossentropy(),
              metrics = ['accuracy'])
```

CNN for binomial classification

Training and validation

3.2 Training the CNN

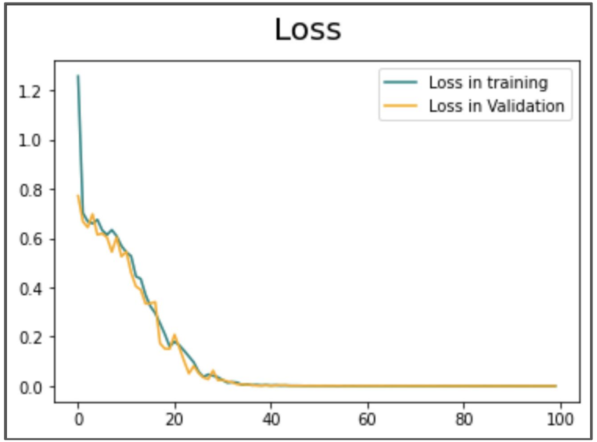
```
# Establish directory for callbacks
logdir = 'path_to_logs_directory'
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

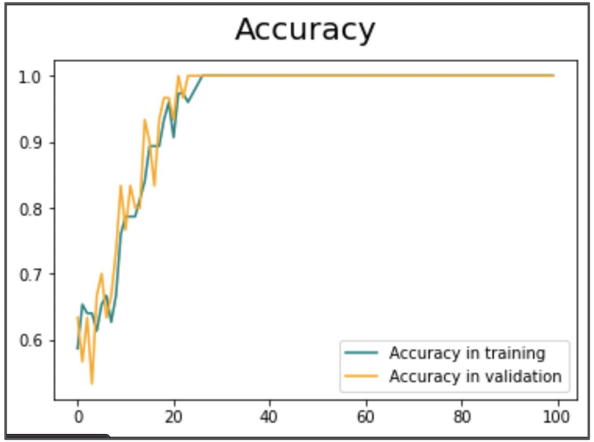
```
# Fit the model
# Set # of epochs
hist = model.fit(train, epochs = 100, validation_data = val,
                 callbacks = [tensorboard_callback])
```

3.3 Training and validation Performance

```
#Visual representation of the training and validation loss performance
fig = plt.figure()
plt.plot(hist.history['loss'], color = 'teal', label = 'loss in training')
plt.plot(hist.history['val_loss'], color = 'orange', label = 'loss in validation')
fig.suptitle('loss', fontsize = 20)
plt.legend(loc = "upper left")
plt.show()
```

```
#Visual representation of the training and validation accuracy performance
fig = plt.figure()
plt.plot(hist.history['accuracy'], color = 'teal', label = 'Accuracy in training')
plt.plot(hist.history['val_accuracy'], color = 'orange', label = 'Accuracy in validation')
fig.suptitle('Accuracy', fontsize = 20)
plt.legend(loc = "lower right")
plt.show()
```





CNN for binomial classification

4. Making predictions

4 Making predictions

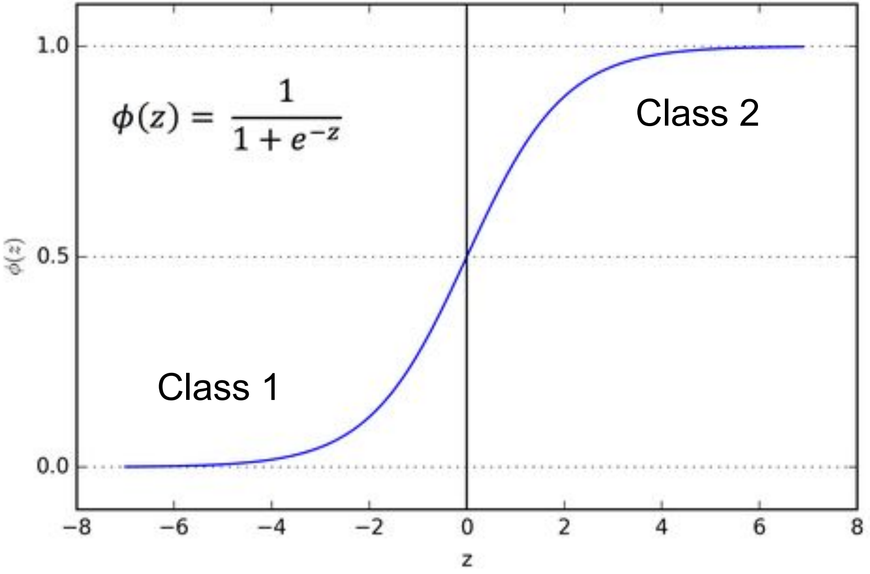
```
# I suggest that for this part, we can use images that haven't been part of any
# of the datasets (training, validation and test) they could be new images.

# Load the new image and visualize it
img = cv2.imread('path_to_image_which_is_going_to_be_predicted')
plt.imshow(img)
plt.show()
```

```
# Resize the new image
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```

```
# Scale and pass the image trhoug the CNN
yhat = model.predict(np.expand_dims(resize/255, 0))
```

```
# print the final value (this value is the one requiered for the classification)
yhat
```

```
# I used a sigmoidal function so all the values below 0.5 will become 0 (class 1)
# and all the values higher than 0.5 will become 1 (class 2)
if yhat > 0.5:
  print(f'It is predicted to be class 2')
else:
  print(f'It is predicted to be class 1')
```



Next Steps for the CNN

- Build a CNN that can handle 3D data (hyper-cylinder or hyper-sphere)
- Decide how many convolutional and pooling layers to use
    - Also consider: activation functions, kernel size, stride
- Test model with a sample of random proteins
- Train and optimize the CNN
- May need to tweak ECT data depending on model accuracy and compute time

Reflections on the Project

- Would combine groups 3 and 4 because deep learning requires data from group 3 before CNNs can be constructed.
- Learned general process of CNN–no time to optimize one.
- Practiced working on collaborative programming (w/ GitHub).