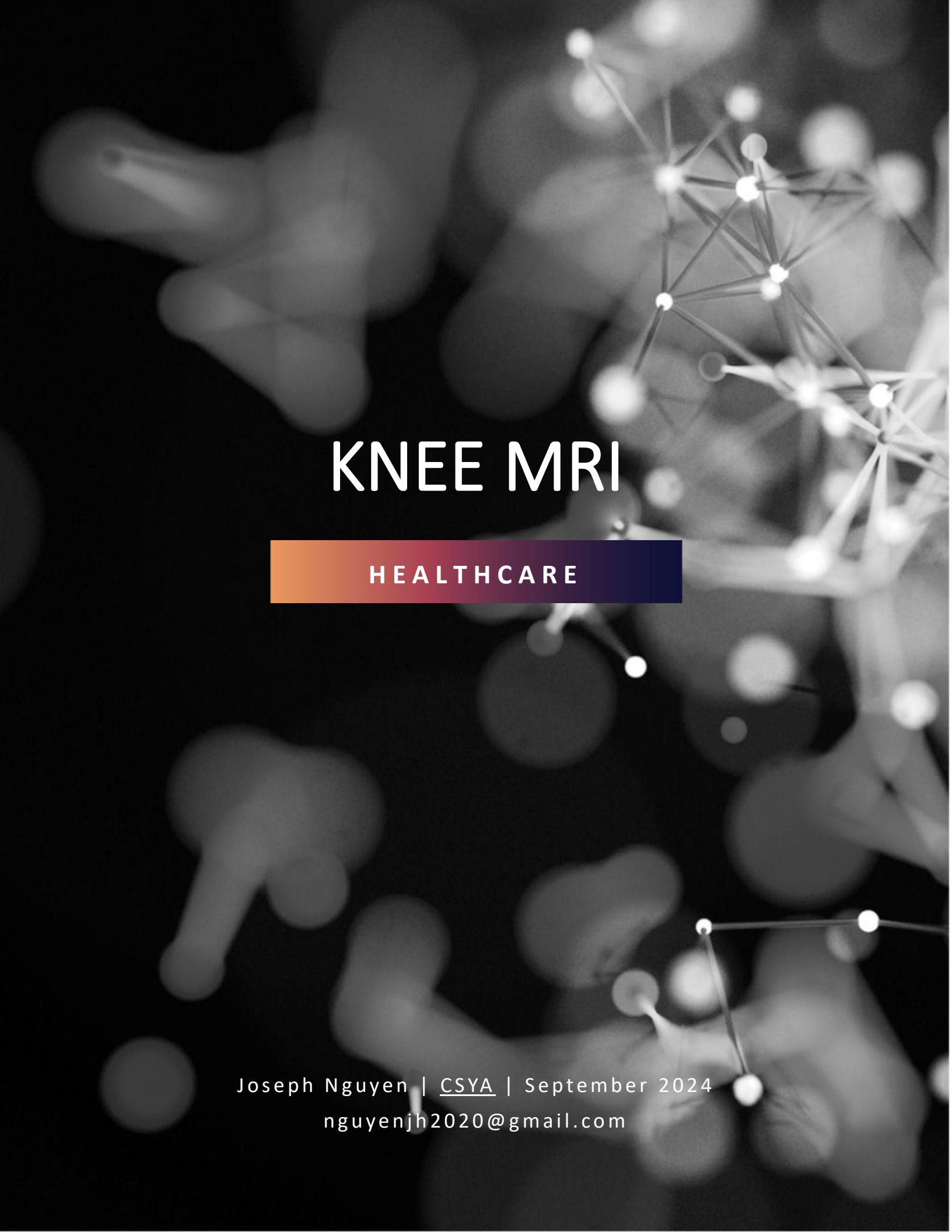


# KNEE MRI

HEALTHCARE

Joseph Nguyen | CSYA | September 2024
nguyenjh2020@gmail.com

# SUMMARY

**This project involves detecting ACL tears in knee MRI scans using a neural network algorithm. The project utilizes a Tensor dataset in CSV file format, where the data likely includes MRI images or their features, and the neural network is trained to identify patterns that indicate ACL injuries. This approach combines machine learning with healthcare imaging to potentially automate and improve diagnostic accuracy.**

# PROCESS

The process for the Python project to detect ACL tears using a neural network with a Tensor dataset in CSV file format involves several key steps.

This process ensures that the Convolutional Neural Network (CNN) model can efficiently detect ACL tears from MRI scans, leveraging the power of deep learning for healthcare diagnostics.

- See the final project information in GitHub.

## DATA PREPARATION

- **Load CSV Dataset**: Read the Tensor dataset from the CSV file, where each row might represent an MRI scan or its extracted features. The dataset should include labels indicating whether an ACL tear is present or absent.
- **Data Cleaning and Preprocessing**: Handle missing or inconsistent data, normalize values (e.g., pixel intensities or extracted features), and split the data into training, validation, and test sets.
- **Data Reshaping**: If the dataset consists of MRI images, reshape the data to match the input format expected by the neural network. This might include converting 2D image data into arrays or vectors suitable for input layers.

```
import numpy as np
import pandas as pd
```

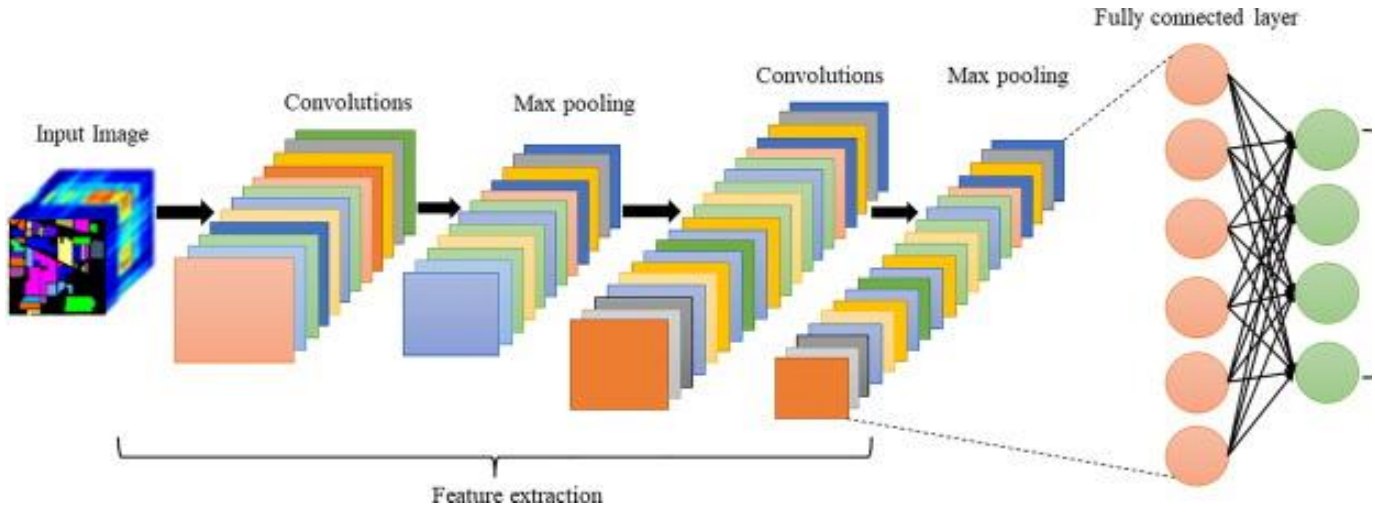
```
metadata_path = "/kaggle/input/kneemridataset/metadata.csv"
metadata = pd.read_csv(metadata_path)
metadata.head()
```

| | examId | seriesNo | aclDiagnosis | kneeLR | roiX | roiY | roiZ | roiHeight | roiWidth | roiDepth | volumeFilename |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 329637 | 8 | 0 | 1 | 139 | 184 | 14 | 74 | 72 | 3 | 329637-8.pck |
| **1** | 390116 | 9 | 0 | 0 | 113 | 105 | 10 | 83 | 98 | 6 | 390116-9.pck |
| **2** | 404663 | 8 | 1 | 1 | 120 | 117 | 15 | 101 | 115 | 2 | 404663-8.pck |
| **3** | 406320 | 9 | 0 | 0 | 117 | 124 | 12 | 91 | 80 | 3 | 406320-9.pck |
| **4** | 412857 | 8 | 0 | 1 | 122 | 105 | 14 | 83 | 98 | 4 | 412857-8.pck |

# BUILDING THE MODEL

## USING THE NEURAL NETWORK

- **Import Libraries**: Use libraries such as TensorFlow, Keras, or PyTorch for neural network implementation.
- **Define Model Architecture**: Create the neural network layers. For MRI image classification, you could use a Convolutional Neural Network (CNN) with layers like:
- **Input Layer**: Accept the reshaped MRI scan data.
- **Convolutional Layers**: Extract features from the MRI images by applying filters.
- **Pooling Layers**: Reduce dimensionality to focus on important features.
- **Fully Connected Layers**: Classify the extracted features to determine the likelihood of an ACL tear.
- **Output Layer**: A sigmoid or softmax activation function to predict the presence or absence of an ACL tear.



# TRAINING THE MODEL

## COMPILE THE MODEL

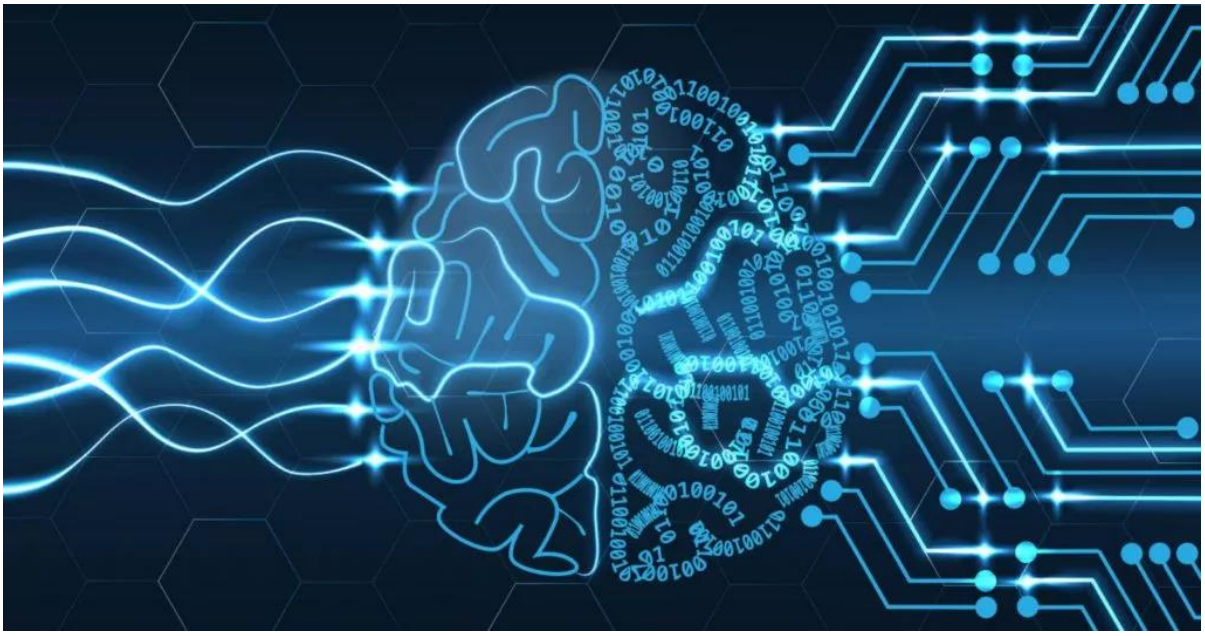
Set the loss function (e.g., binary cross-entropy for classification), optimizer (e.g., Adam or SGD), and evaluation metrics (e.g., accuracy, precision, recall).

## TRAIN THE NEURAL NETWORK

Fit the model on the training data, using the validation set to monitor performance and adjust parameters. This process may involve adjusting hyperparameters like learning rate, batch size, and number of epochs.

## DATA AUGMENTATION (OPTIONAL)

If you are using MRI image data, apply transformations like rotations or flips to increase the dataset's diversity and avoid overfitting.



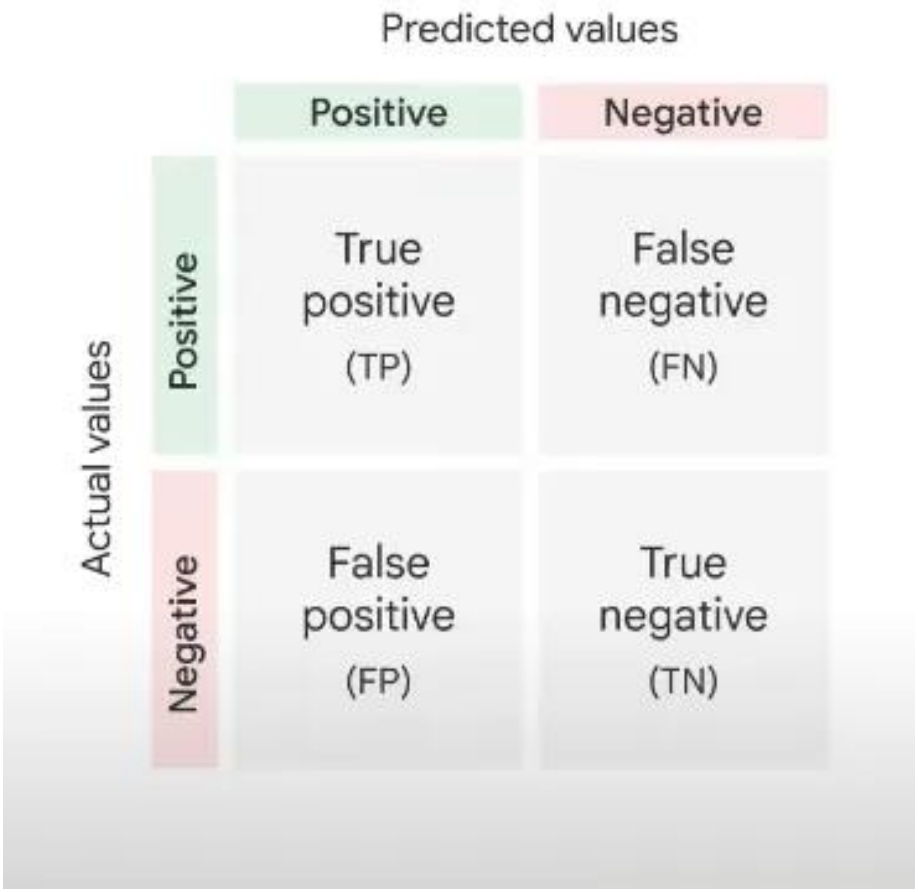
# EVALUATING THE MODEL

## TEST THE MODEL

Evaluate its performance on the test dataset to measure accuracy, precision, recall, F1 score, etc. Use confusion matrices to visualize classification performance.

## TUNING THE MODEL

Based on performance, adjust hyperparameters or modify the network architecture to improve results.



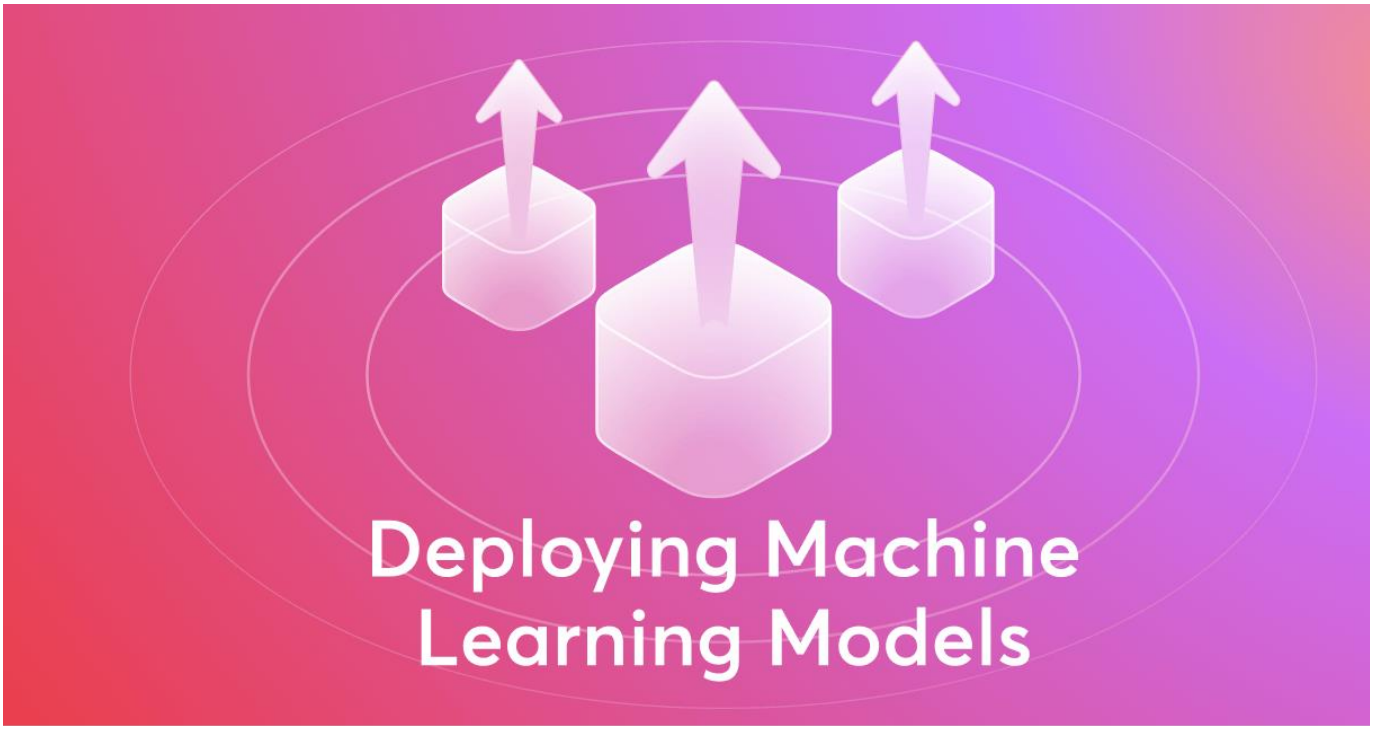
# PREDICTION AND DEPLOYMENT

## PREDICT ACL TEARS

Use the trained model to make predictions on new MRI scans. The model will output a probability or binary classification indicating whether an ACL tear is present.

## SAVE THE MODEL

Save the trained model for future use or deployment in a real-world diagnostic tool, where clinicians could use it to assist in diagnosis.



# VISUALIZATION AND REPORTING

## PLOT TRAINING METRICS

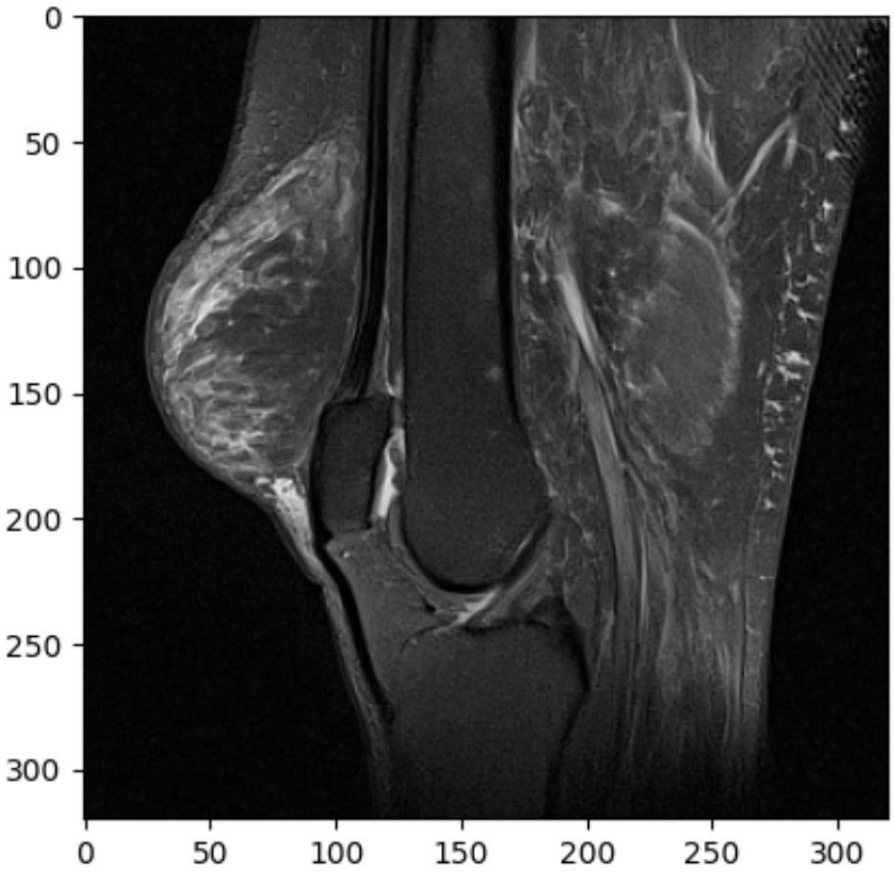
Visualize the loss and accuracy curves over training epochs to understand the learning process.

## VISUALIZE PREDICTIONS

If using images, plot MRI scans with the model's predicted labels for ACL tears.

## GENERATE REPORTS

Provide a summary of the model's performance and the potential use cases for medical professionals.



# JUPYTER NOTEBOOK (CODE)

In [2]:
```
import numpy as np
import pandas as pd
```

In [3]:
```
metadata_path = "/kaggle/input/kneemridataset/metadata.csv"
metadata = pd.read_csv(metadata_path)
metadata.head()
```

Out[3]:

| | examId | seriesNo | aclDiagnosis | kneeLR | roiX | roiY | roiZ | roiHeight | roiWidth | roiDepth | volumeFilename |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 329637 | 8 | 0 | 1 | 139 | 184 | 14 | 74 | 72 | 3 | 329637-8.pck |
| **1** | 390116 | 9 | 0 | 0 | 113 | 105 | 10 | 83 | 98 | 6 | 390116-9.pck |
| **2** | 404663 | 8 | 1 | 1 | 120 | 117 | 15 | 101 | 115 | 2 | 404663-8.pck |
| **3** | 406320 | 9 | 0 | 0 | 117 | 124 | 12 | 91 | 80 | 3 | 406320-9.pck |
| **4** | 412857 | 8 | 0 | 1 | 122 | 105 | 14 | 83 | 98 | 4 | 412857-8.pck |

In [4]:
```
from tqdm import tqdm
import os
import pickle

all_slice_counts = []

counter = 1

patient_count = 0

for index, row in tqdm(metadata.iterrows(), total = len(metadata)):

    temp_file_path = "/kaggle/input/kneemridataset/vol0" + str(counter) + '/' + row['volumeFilename']
    if os.path.exists(temp_file_path) == False:
        counter += 1
        if counter > 8:
            break
        temp_file_path = "/kaggle/input/kneemridataset/vol0" + str(counter) + '/' + row['volumeFilename']

    with open(temp_file_path, "rb") as file:
        temp_data = pickle.load(file)

    if temp_data.shape[1] == temp_data.shape[2] == 320:
        patient_count +=1

    all_slice_counts.append(temp_data.shape[0])
```

```
80%|████████  | 736/917 [00:53<00:13, 13.78it/s]
```

In [5]:
```
print (patient_count)

#used to be 737
```

```
731
```

In [7]:
```
minimum_slice_count = min(all_slice_counts)
print(minimum_slice_count)
```

```
21
```

In [9]:
```
#number of patients (approx 80% of this data available), depth, image width, image height
x_size = (patient_count,  minimum_slice_count, 320, 320) # for each patient we have a corresponding 3d image of si
y_size = (patient_count) #
x = np.zeros(x_size)
y = np.zeros(y_size)
```

In [10]:
```
import pickle
import os
from tqdm import tqdm

counter = 1
custom_index = 0

for index, row in tqdm(metadata.iterrows(), total = patient_count+6):
    temp_file_path = "/kaggle/input/kneemridataset/vol0" + str(counter) + "/" + row['volumeFilename']
    if os.path.exists(temp_file_path) == False:
        counter += 1
        if counter > 8:
            break
        temp_file_path = "/kaggle/input/kneemridataset/vol0" + str(counter) + "/" + row['volumeFilename']

    with open(temp_file_path, "rb") as file:
        temp_data = pickle.load(file)

    if temp_data.shape[1] == temp_data.shape[2] == 320:
        d = temp_data.shape[0]

        if d % 2 == 1:
            start_index = int(d/2) - int(minimum_slice_count-1)/2
            end_index = int(d/2) + int(minimum_slice_count-1)/2 + 1
        else:
            start_index = int((d-1)/2) - int(minimum_slice_count-1)/2
            end_index = int((d-1)/2) + int((minimum_slice_count-1)/2) + 1

        cropped_patient_data = temp_data[int(start_index):int(end_index), :, :] #21x320x320

        x[custom_index] = cropped_patient_data
        y[custom_index] = row["aclDiagnosis"]

        custom_index += 1
```

```
100%|██████████| 736/737 [00:24<00:00, 30.02it/s]
```

In [11]:
```
print(x.shape)
print(y.shape)
```

```
(731, 21, 320, 320)
(731,)
```

In [15]:
```
list_y = list(y)
print("Healthy Patient Count:" , list_y.count(0))
print("Partially Torn Patient Count:", list_y.count(1))
print("Fully Torn Patient Count:", list_y.count(2))
```

```
Healthy Patient Count: 543
Partially Torn Patient Count: 143
Fully Torn Patient Count: 45
```

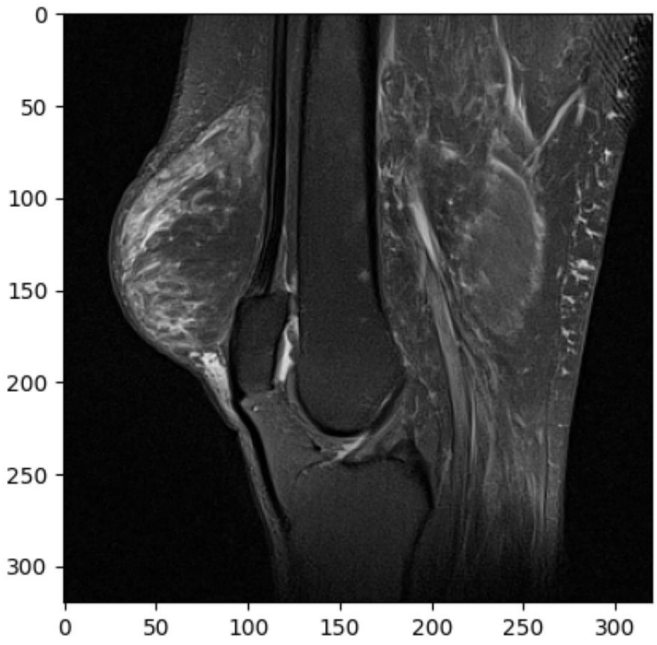
In [10]:
```
import matplotlib.pyplot as plt

print(y[0])

plt.imshow(x[0][10], cmap="gray")
```

```
0.0
```

Out[10]: <matplotlib.image.AxesImage at 0x7d79974eabf0>




In [11]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
```

In [12]:
```
#converting our numpy arrays to pytorch tensor for compatible machine learning processing
x_tensor = torch.tensor(x, dtype = torch.float32)
y_tensor = torch.tensor(y, dtype = torch.float32)
```

In [13]:
```
print (x_tensor.shape)
print (y_tensor.shape)
```

```
torch.Size([731, 21, 320, 320])
torch.Size([731])
```

In [14]:
```
x_tensor_flatenned = x_tensor.view(patient_count, -1)
print (x_tensor_flatenned.shape)
```

```
torch.Size([731, 2150400])
```

In [15]:
```
print (x_tensor_flatenned.shape)
print (y_tensor.shape)
```

```
torch.Size([731, 2150400])
torch.Size([731])
```

In [16]:
```
dataset = TensorDataset(x_tensor_flatenned, y_tensor)
dataloader = DataLoader(dataset, batch_size = 16, shuffle = True)
```

In [17]:
```
class simple_neural_network(nn.Module):
    def __init__(self):
        super(simple_neural_network, self).__init__()
        self.input = nn.Linear(32 *320 *320, 2048)
        self.hidden = nn.Linear(2048, 512)
        self.output = nn.Linear(521, 3)

    def forward(self, x):
        #x comes in as a 2 million length array
        x= torch.relu(self.input(x)) #now x is 2048 length
        x= torch.relu(self.hidden(x)) #now x is 512 length
        x= self.hidden(x) #now x is 3 length
        return x
```

In [ ]:
```
model = simple_neural_network()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), Ir = 0.001)
```

In [ ]:
```
#training loop (this loop will run a bunch of times and keep improving our model)

epochs
```

# ABOUT KNEE-MRI DATASET

## LICENSING



## DESCRIPTION

KneeMRI dataset was gathered retrospectively from exam records made on a Siemens Avanto 1.5T MR scanner and obtained by proton density-weighted fat suppression technique at the Clinical Hospital Centre Rijeka, Croatia, from 2006 until 2014. The dataset consists of 917 12-bit grayscale volumes of either left or right knees. Each volume record was assigned a diagnosis concerning the condition of the anterior cruciate ligament in a double-blind fashion, i.e. each volume record was labelled according to the ligament condition: (1) healthy, (2) partially injured, or (3) completely ruptured. A wider rectangular region of interest (ROI) was manually extracted from the original volumes and is also annotated. For more details regarding the dataset, the reader is referred to the paper stated under the "acknowledging source" section of this webpage.

This dataset was built with the intention of providing scientists, involved with machine vision and/or machine learning, an easy way of working with the data.

## MAGNETIC RESONANCE IMAGING

- Magnetic Resonance Imaging (MRI) is a medical imaging technique used in radiology to form a picture of the anatomy and the physiological processes of the body.
- MRI is used to diagnose how well you responded to treatment as well as detecting tears and structural problems such as heart attacks, brain injury, blood vessel damage, etc.

## 🤔 SOME CONSIDERATIONS ABOUT THE DATA

- The slices are significantly different from a plane to another: this is the first thing I noticed as a non-specialist.
- Within a given plane, the slices may substantially differ as well. In fact, and we'll see it later, some slices can better highlight an ACL tear.
- In the next post, we'll build an MRI tear classification per plane. We'll see next that the combination of these three models outperforms individual models.
- An MRI scan taken according to a given plane can be considered as a volume of stacked slices. As we previously said that cases don't necessarily share the same of slices, MRIs cannot then be put in batches. We'll see how to handle this efficiently.

## FILE ATTRIBUTES

**READ.ME** - file contains some basic information regarding this archive.
**example.py** - Python script used to demonstrate how to access the files.
**metadata.csv** - comma-delimited (csv) data containing descriptions of distinct volumes (header attribute information included).
**volumetric_data** - directory containing all knee MR volumes, archived using 7-zip lossless file compression.

|-***example.pck*** - an example Python .pck file (just to inspect whether one wants to be bothered downloading the archive)
|-***vol01.7z*** - compressed independent archive (1/10), containing 92 cases
|-***vol02.7z*** - compressed independent archive (2/10), containing 92 cases
|...
|-***vol10.7z*** - compressed independent archive (10/10), containing the remaining 89 cases

## COLUMNS DESCRIPTION

1. **aclDiagnosis:** The Lachman test is the most accurate test for detecting an ACL tear. Magnetic resonance imaging is the primary study used to diagnose ACL injury in the United States. It can also identify concomitant meniscal injury, collateral ligament tear, and bone contusions.

2. **KneeLR:** Means if its left or right.

## ACKNOWLEDGING SOURCES

If you are using this dataset in your work, please acknowledge the source (Clinical Hospital Centre Rijeka, Croatia) and reference this paper (preprint pdf):

I. Štajduhar, M. Mamula, D. Miletić, G. Unal, Semi-automated detection of anterior cruciate ligament injury from MRI, Computer Methods and Programs in Biomedicine, Volume 140, 2017, Pages 151–164.