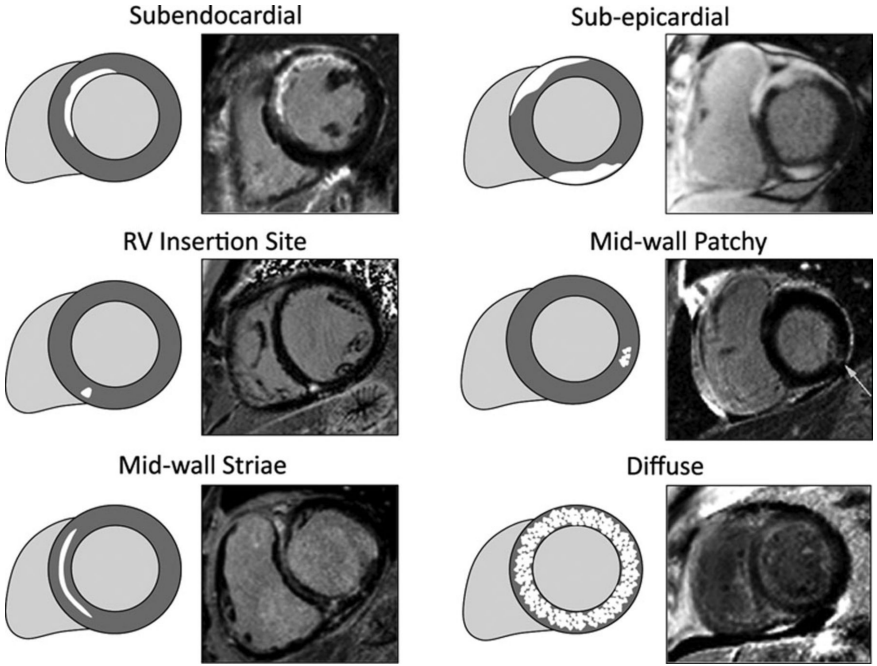
Heart Nuclear Magnetic Resonance Imaging Analysis

Alessandro Casalino
Michele M. Crudele
Sabrina Giorgetti
Daniele Mellino

Università degli Studi di Padova

Analysis of MRI images



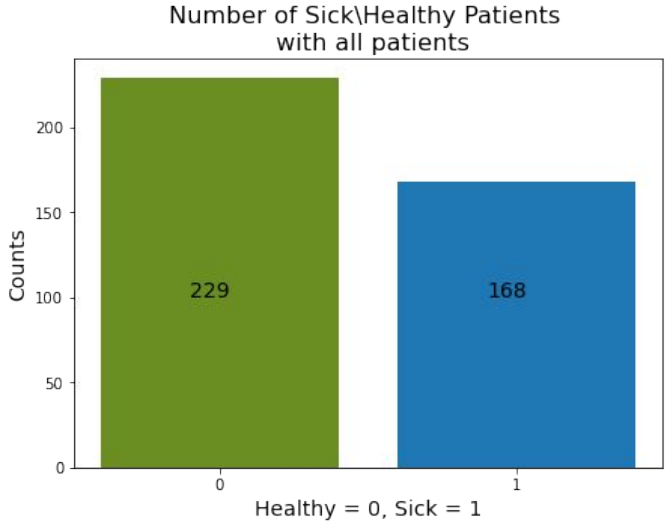
The purpose of the project is to design a CNN in order to classify a patient as healthy or sick based on the images of an MRI scan of the heart.

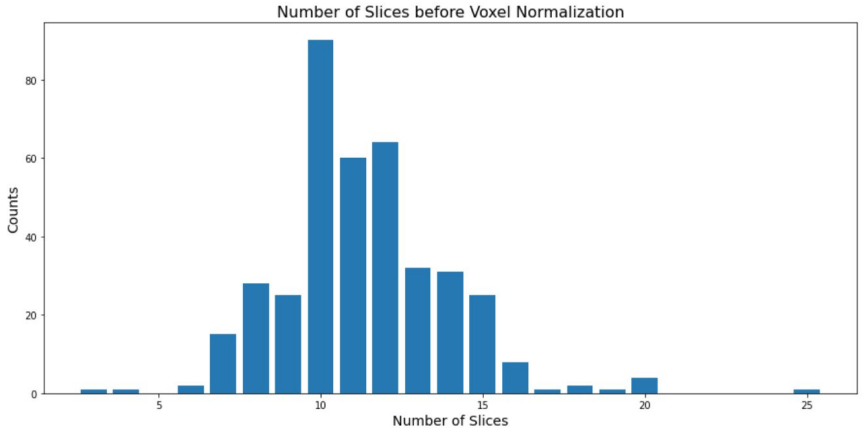
The dataset was provided by the Padova Cardiology Department and Hospital (real case of study/patients).

Dataset

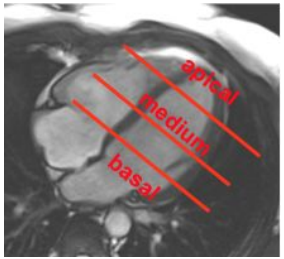
On the initial dataset of 500 patients a first filtering has been performed, obtaining 397 patients.

For each patient we know the related label:

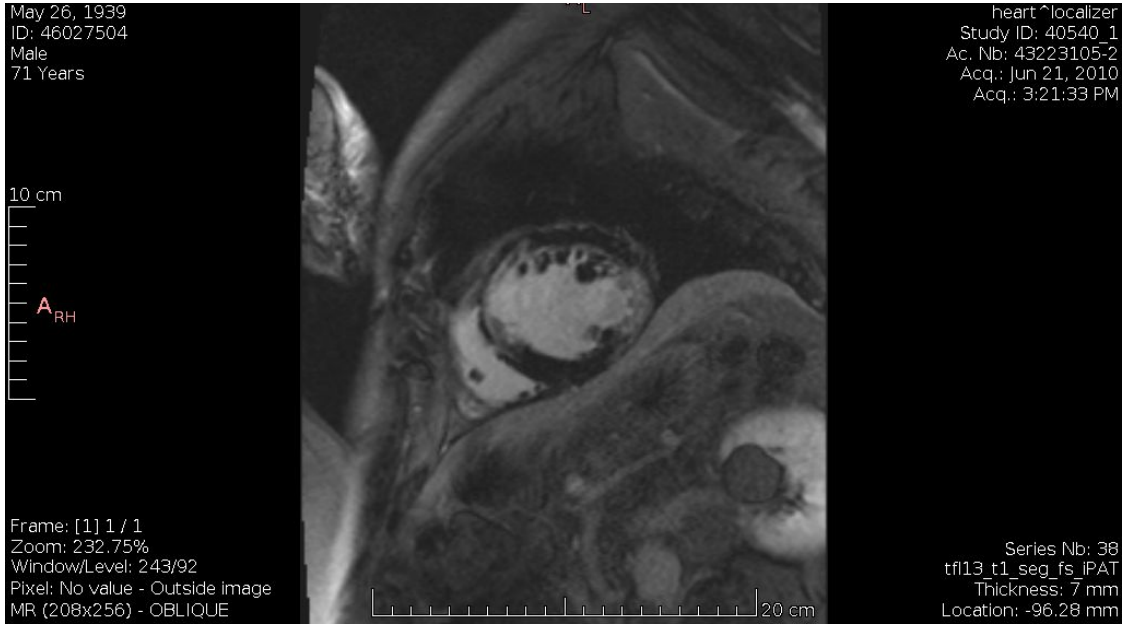




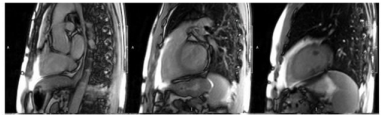
For each patient we have multiple images from the MRI, called *slice*.



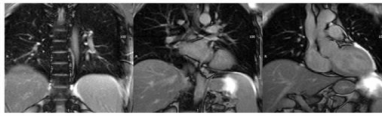
DICOM files



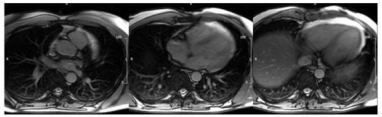
Sagittal plane



Coronal Plane



Axial plane



Non-uniform dataset

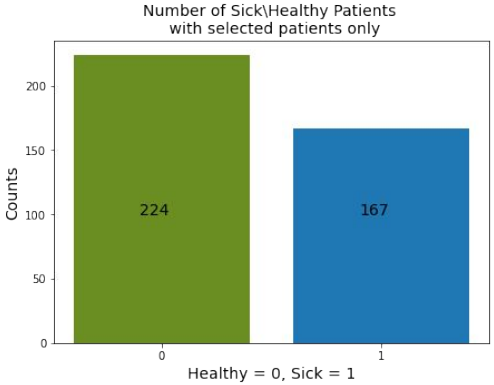
Each *slice* is a DICOM file, which contains various information, many of which are important for the analysis.

The data have been taken over the years with different machines, settings: the slices between patients present very different characteristics.

Dataset Loading and Patient Class

```python
patients_directory = "../../dicom"
all_filenames = os.listdir(patients_directory)
patients_folders = [f for f in all_filenames if os.path.isdir(join(patients_directory,f))]
del(all_filenames)
patients_folders = sorted(patients_folders)
axial_patients = ["GIROTTO_STEFANO_043Y", "ONGETTA_FRANCESCO_029Y", "LONZU_SALVATORE_035Y"]
coronal_patients = ["ZAMBON_ALESSANDRO_052Y"]
other_patients = ["BELLINI_DANIELE_058Y", "ZAGO_GIULIANO_082Y", "BORTOLETTO_PAOLO_062Y"]
patients_excluded = axial_patients + coronal_patients + other_patients

df_patients = pd.DataFrame(pd.read_csv('../../dicom/names.csv'))
print(df_patients)
# This patient has two folders but only one entry in csv
index_majorana = np.where(df_patients["Name"] == "MAJORANA CARMELA")[0][0]
all_labels = df_patients["LGE_si_no"].values
all_labels = all_labels.tolist()
all_labels.insert(index_majorana, 1)
all_labels = np.array(all_labels)
labels = []
for p_i, p in enumerate(patients_folders):
    if not p in patients_excluded:
        labels.append(all_labels[p_i])
patients_folders = [p for p in patients_folders if not p in patients_excluded]
```

	Name	dicomID	LGE_si_no
0	AGACHI PAUL	047Y	1
1	AGOSTINELLO DENIS	026Y	0
2	ALTOE' ROBERTO	054Y	0
3	ALVARO VINCENZO	056Y	0
4	AMBROSI IVANO	038Y	0
...	...	...	...
392	ZINATO CLAUDIA	048Y	0
393	ZORZI GIORGIO	061Y	0
394	ZUCCHERI ISABELLA	019Y	0
395	ZUIN GIANLUCA	035Y	0
396	ZUIN MARIA	077Y	0



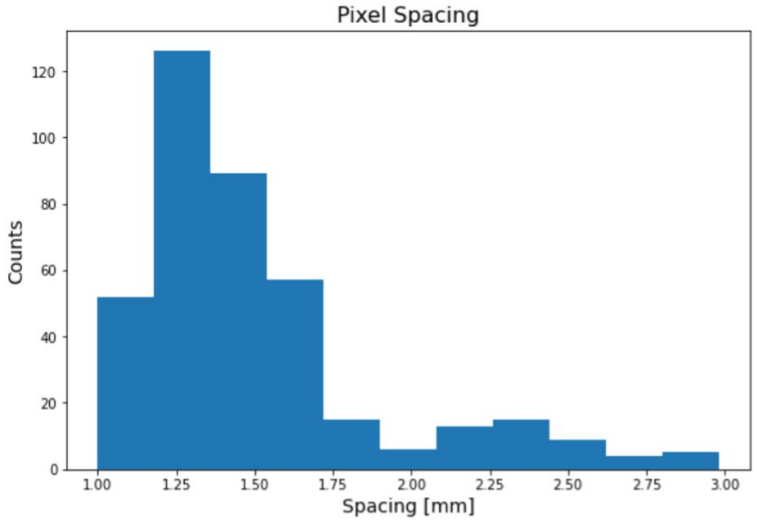

```python
class Patient(object):
    def __init__(self, patient_folder, PRINTING=False):
        self.patient_folder = patient_folder
        self.filenames = self.__get_filenames__(PRINTING)
        self.dcms = self.__load_dcms__()
        self.arrays, self.ims = self.__load_images__()
        # Sorting according to Slice Location
        indices = sorted(range(len(self.dcms)), key=lambda k: self.dcms[k].SliceLocation)
        temp_filenames = []
        temp_dcms = []
        temp_arrays = []
        for idx in indices:
            temp_filenames.append(self.filenames[idx])
            temp_dcms.append(self.dcms[idx])
            temp_arrays.append(self.arrays[idx])
        self.filenames = temp_filenames
        self.dcms = temp_dcms
        self.arrays = temp_arrays

    def __get_filenames__(self, PRINTING=False):
        filenames = os.listdir(join(patients_directory, self.patient_folder))
        filenames = [f for f in filenames if f[-4:] == ".dcm"]
        filenames = sorted(filenames)
        if PRINTING:
            print(filenames)
            print(f"Number of images: {len(filenames)}")
        return [join(patients_directory, self.patient_folder, f) for f in filenames]

    def __load_dcms__(self):
        return [dcmread(f) for f in self.filenames]

    def __load_images__(self, crop=True, size=(192, 192)):
        arrays = []
        ims = []
        for i in range(len(self.dcms)):
            array = self.dcms[i].pixel_array
            # get PIL image
            mode = get_PIL_mode(self.dcms[i])
            # convert to RGB for yolo detection
            im = Image.fromarray(array, mode=mode).convert("P").convert("RGB")
            # crop pil image
            if crop:
                im    = center_crop(im, *size)
                array = np.asarray(im)
                if len(array.shape) == 3:
                    array = array[:,:,0]
            arrays.append(array)
            ims.append(im)
        return arrays, ims
```

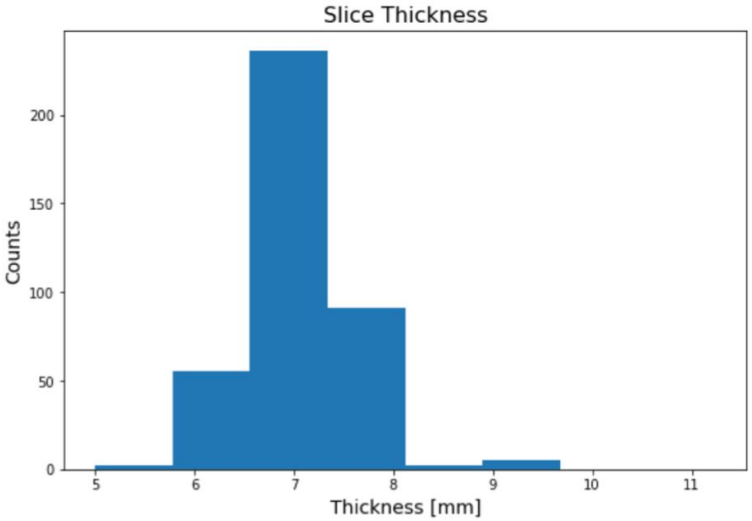
Pre-processing

In this phase we want to filter and process the dicom files to extract the images of the hearts, in order to have an optimal input for the CNN.

The first step is to find the most frequent values for :

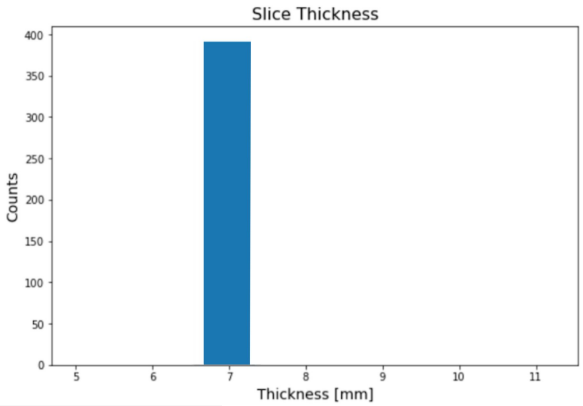
- Pixel Spacing : 1.25 mm
- Slice Thickness: 7 mm

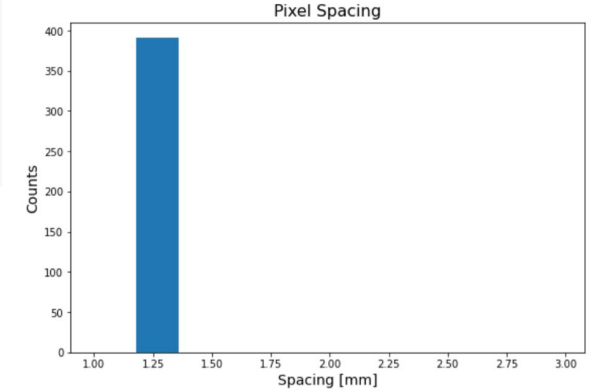




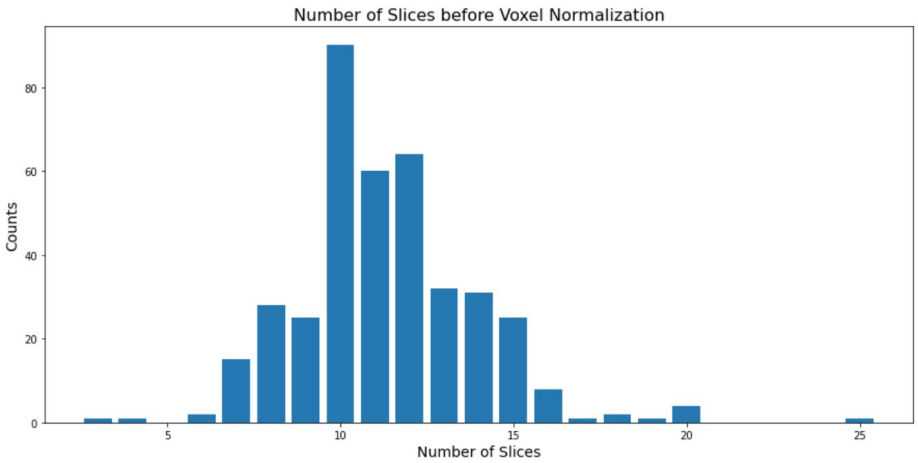
Voxel Normalization

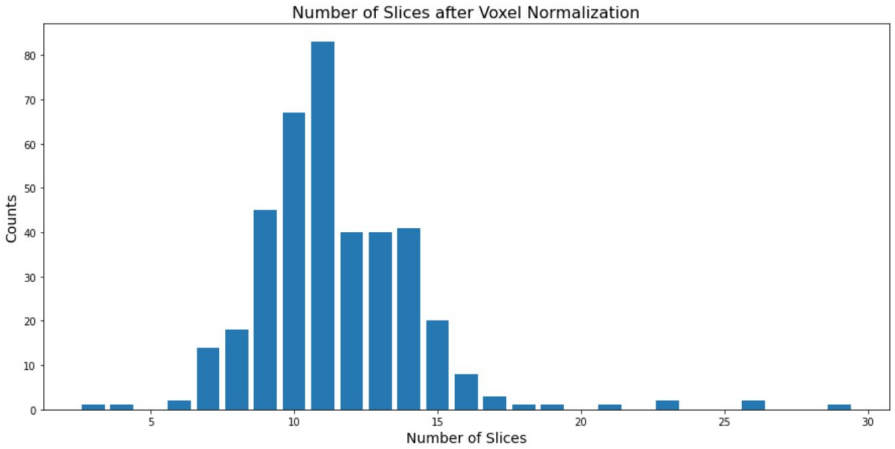
```python
def resample(patient, new_spacing=[1.25, 1.25, 7]):
    """
    new_spacing = [x, y, z]
    """
    spacing = list(patient.dcms[0].PixelSpacing) + [patient.dcms[0].SliceThickness]
    spacing = np.array(spacing).astype(float)
    image = np.stack(patient.arrays, 2)

    resize_factor = spacing / new_spacing
    new_real_shape = image.shape * resize_factor
    new_shape = np.round(new_real_shape)
    real_resize_factor = new_shape / image.shape
    new_spacing = spacing / real_resize_factor

    image = scipy.ndimage.interpolation.zoom(image, real_resize_factor)

    return image, new_spacing
```





Number of slices after normalization






```
for p_name_i, p_name in enumerate(patients_folders):
    p = Patient(p_name)
    image, new_scale = resample(p)
    x, y, z = image.shape
    if z - len(p.dcms) > 3:
        print(f"N slices before: {len(p.dcms)}, after: {z}, i = {p_name_i}")
```

N slices before: 20, after: 26, i = 79
N slices before: 20, after: 26, i = 112
N slices before: 25, after: 29, i = 317

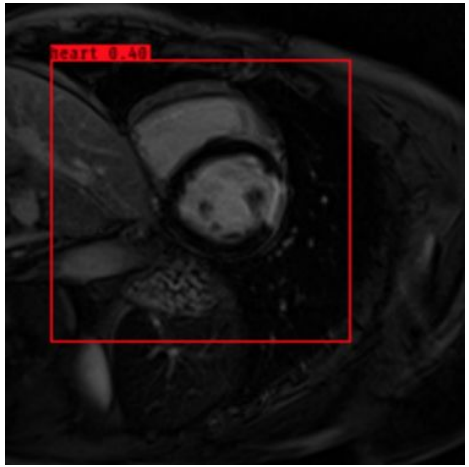
Yolo model

```
params = {
    "model_path": "logs_first_stage/best_model.h5",
    "classes_path": "model_data/heart_classes.txt",
    "anchors_path": 'model_data/yolo_anchors.txt',
    "model_image_size": (192, 192)
}
yolo = YOLO(**params)
```

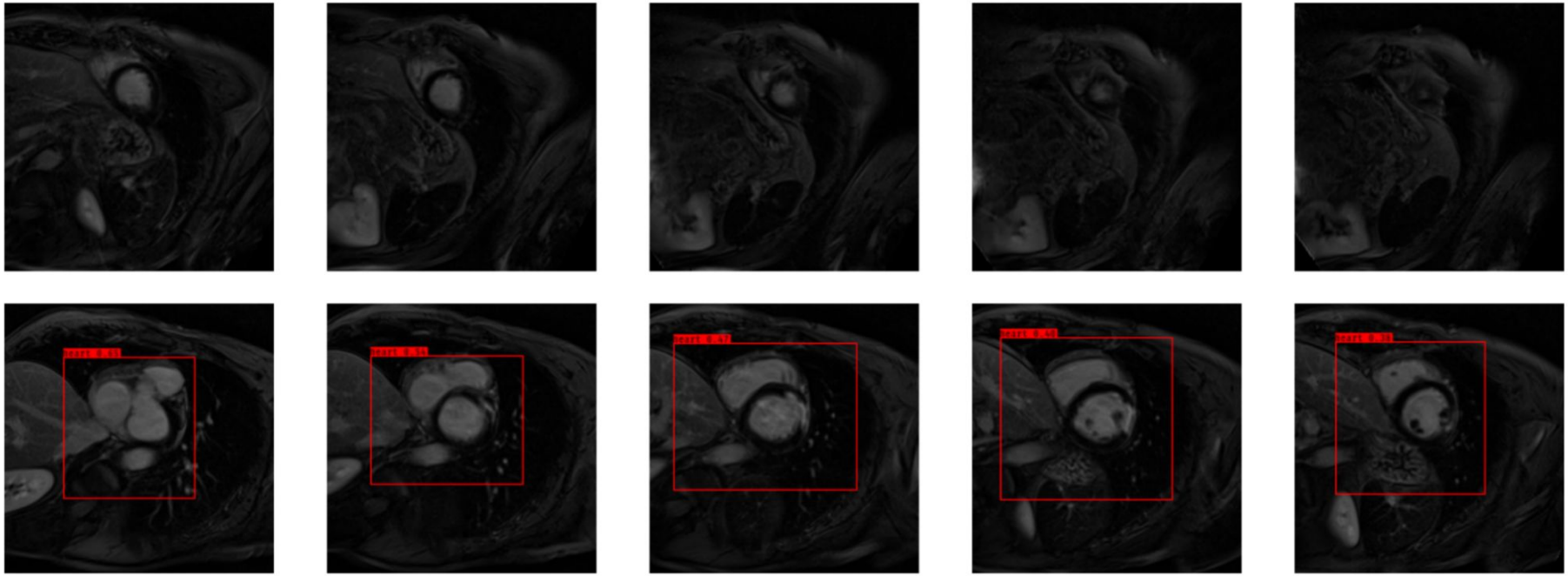
Example on one patient

```
r_image, bbox, scores, classes = yolo.detect_image(patient0.ims[9].copy())
print(type(r_image))
r_image
```

```
(192, 192, 3)
Found 1 boxes for img
heart 0.40 (20, 24) (142, 139)
4.344549025176093
<class 'PIL.Image.Image'>
```



Yolo problem



Yolo detection

Extracting bounding boxes

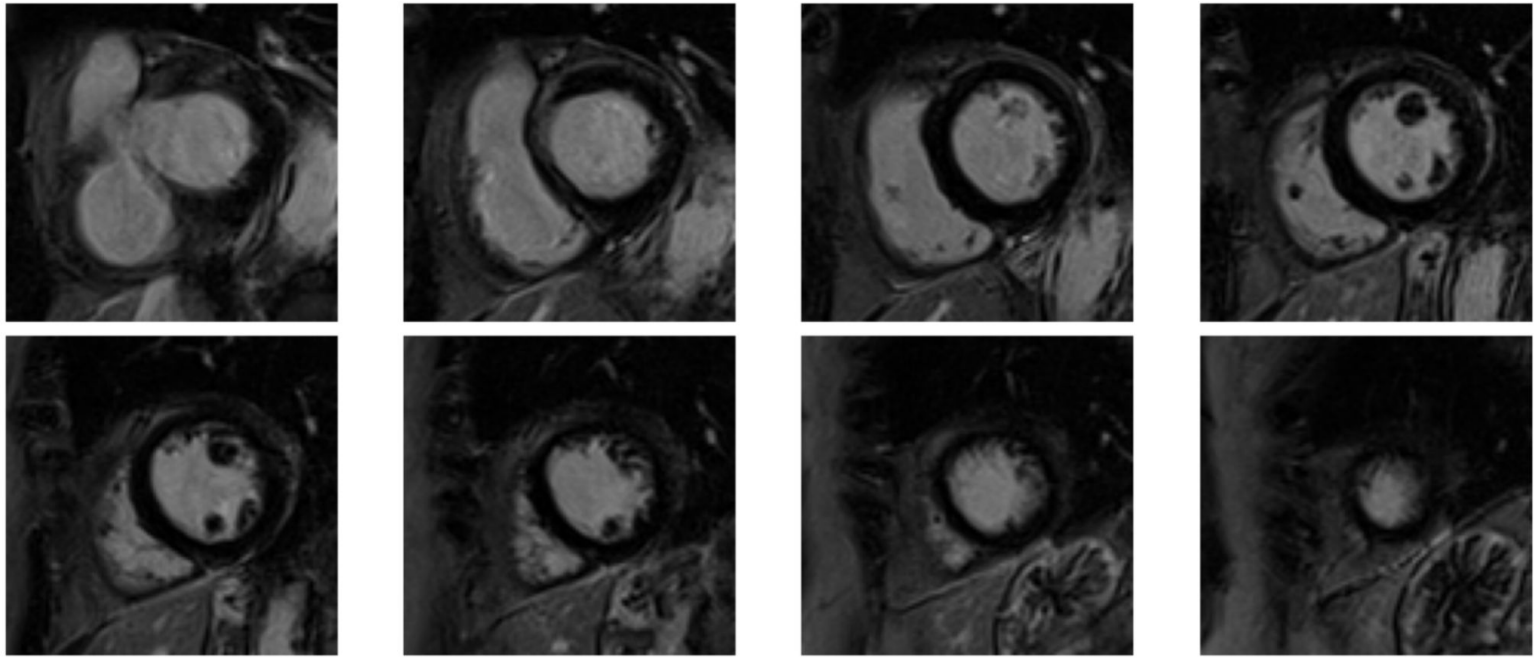
```
for name in [patients_folders[0]]:
    p = Patient(name)
    # Voxel normalization
    array3D, _ = resample(p)
    # PIL images, needed for yolo
    images = []
    for i in range(array3D.shape[2]):
        images.append(array_to_image(array3D[:,:,i]))
    # List of bounding boxes
    bboxes = []
    for i in range(len(images)):
        _, bbox, scores, _ = yolo.detect_image(images[i])
        ##################
        print(bbox)
        ## Checks if no bounding boxes are detected
        if bbox.shape == (0, 4):
            bboxes.append(None)
        else:  # keep only the bbox with the highest score
            best_score_i = np.argmax(scores)
            best_bbox = bbox[best_score_i]
            ###########
            print(best_bbox)
            top, left, bottom, right = best_bbox
            top  = max(0, np.floor(top + 0.5).astype('int32'))
            left = max(0, np.floor(left + 0.5).astype('int32'))
            bottom = min(array3D.shape[1], np.floor(bottom + 0.5).astype('int32'))
            right  = min(array3D.shape[0], np.floor(right + 0.5).astype('int32'))
            best_bbox = np.array([top, left, bottom, right])
            print("==============", best_bbox)
            bboxes.append(best_bbox)

    # Saving bboxes
    path = f"../data_saved/patients_bboxes/{name}"
    if not os.path.isdir(path):
        os.mkdir(path)
    with open(f"../data_saved/patients_bboxes/{name}/bboxes.pkl", "wb") as f:
        pickle.dump(bboxes, f)
```

Finding the common box for each patient

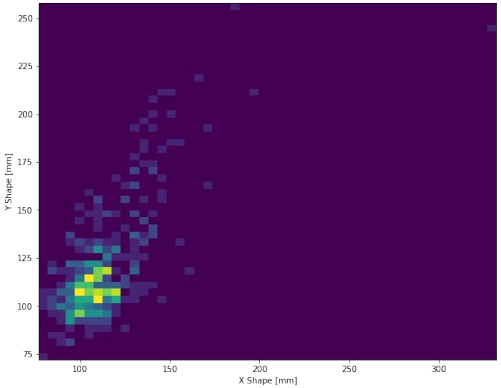
```
max_bboxes = []

for name in patients_folders:
    path = f"../data_saved/patients_bboxes/{name}"
    with open(f"{path}/bboxes.pkl", "rb") as f:
        bboxes = pickle.load(f)
    # [top, left, bottom, right]
    max_values = np.array([1000, 1000, 0, 0])
    for bbox in bboxes:
        if type(bbox) == type(None):
            continue
        max_values[0] = min(max_values[0], bbox[0])
        max_values[1] = min(max_values[1], bbox[1])
        max_values[2] = max(max_values[2], bbox[2])
        max_values[3] = max(max_values[3], bbox[3])
    max_bboxes.append(max_values)
```

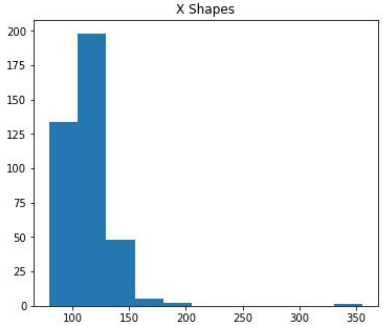
Hearts : input of the CNN

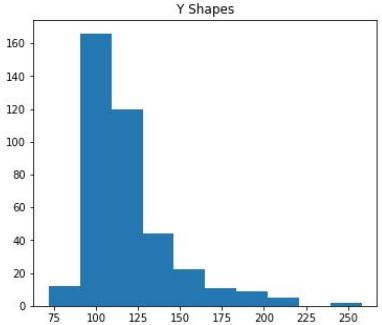
```python
for name_i, name in enumerate(patients_folders):
    p = Patient(name)
    array3D, _ = resample(p)
    bb = max_bboxes[name_i]
    print(bb, array3D.shape)
    heart = array3D[bb[0]:bb[2], bb[1]:bb[3], :]
    with open(f"../data_saved/patients_hearts/{name}/heart.npy", "wb") as f:
        np.save(f, heart)
```



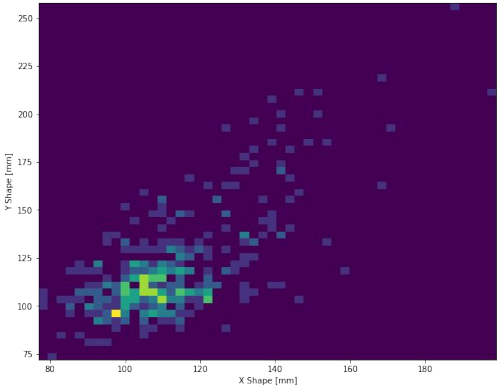
Problematic Patient

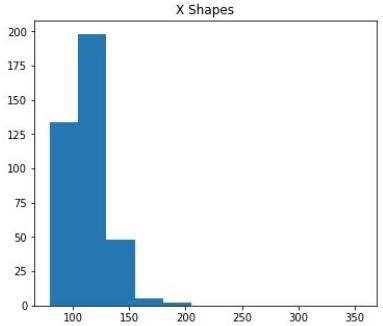


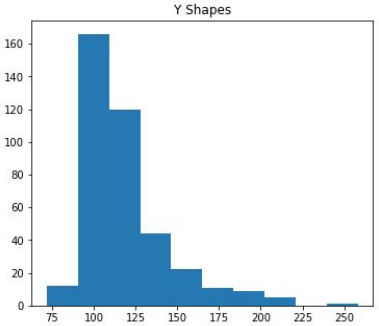




Shapes with problematic patient





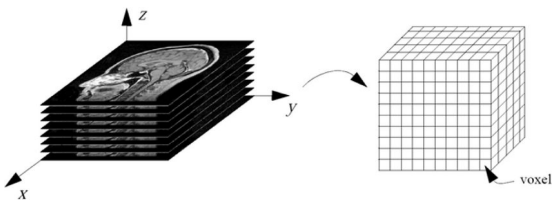


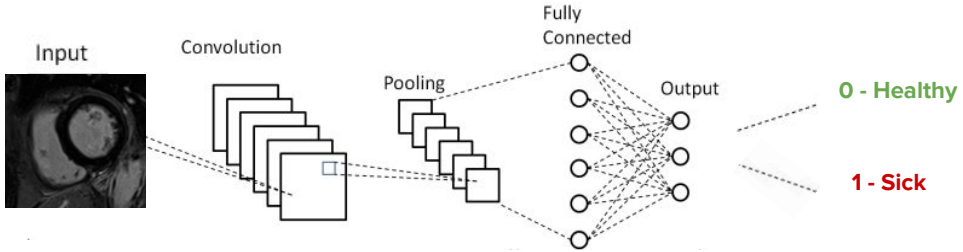
Shapes without problematic patient

Convolutional Neural Network (CNN)

CNN 2D

- **Pretrained Densenet121** : *all slices*
- **Pretrained Densenet121** : all slices and *feature extraction*
- **Pretrained Densenet121** : *9 slices* and *feature extraction*
- **Multislice 2D concatenation** : *9 slices*
- **Multislice 2D independent branches**: 9 slices
- **Multislice 2D Densenet121**: *9 slices* and *feature extraction*



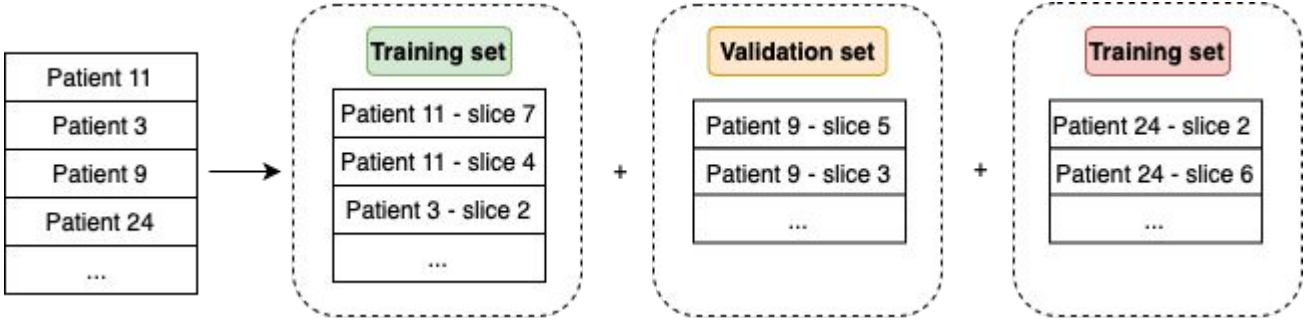


CNN 3D

- **3D convolution**
- **3D multislice**

Training validation and test set

To define the training, validation and test sets we separate our dataset based on the patients: this allows not to introduce any bias in the network.



Slice processing

The label for each slide is taken from the label of the patient: this can introduce an error .

$\Longrightarrow$

A slice can have as label 1 even if the the characteristic features of the disease are not visible.

All slices

Take into consideration all the slices, the number of slices per patients is then variable.

9 slices

Eliminate patients with less then nine slices and select only nine for those with more.

For the selection we adopt different criteria.

Multislices

Multiple input given by 3 slices one for each region: basal, medium and apical one.

Slice processing

The label for each slide is taken from the label of the patient: this can introduce an error .

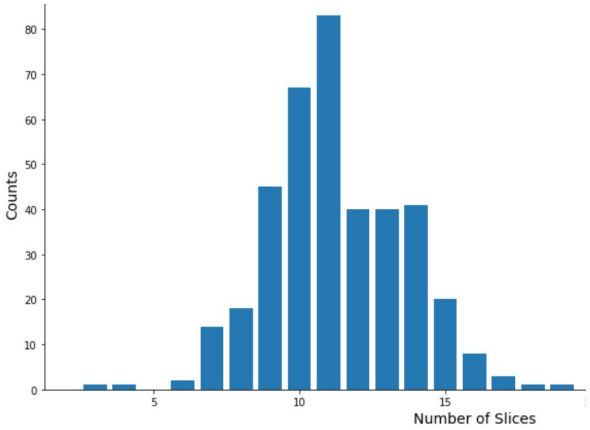
$\Longrightarrow$

A slice can have as label 1 even if the the characteristic features of the disease are not visible.

All slices

Take into consideration all the slices, the number of slices per patients is then variable.

9 slices



Multislices

Multiple input given by 3 slices one for each region: basal, medium and apical one.

Patient Accuracy

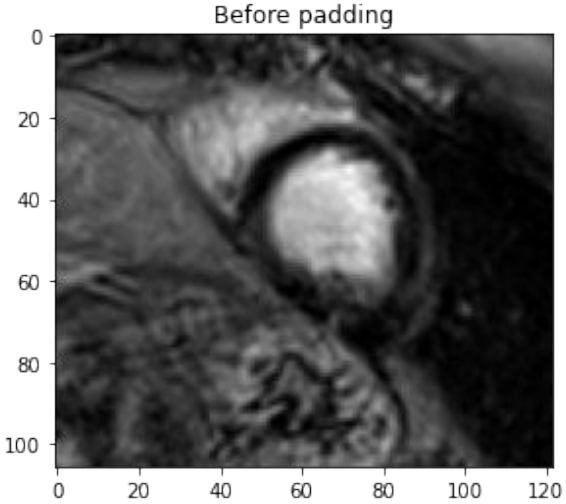
The accuracy of the test set has been evaluated assigning the label 1 to a patient if just one slice was labeled 1.

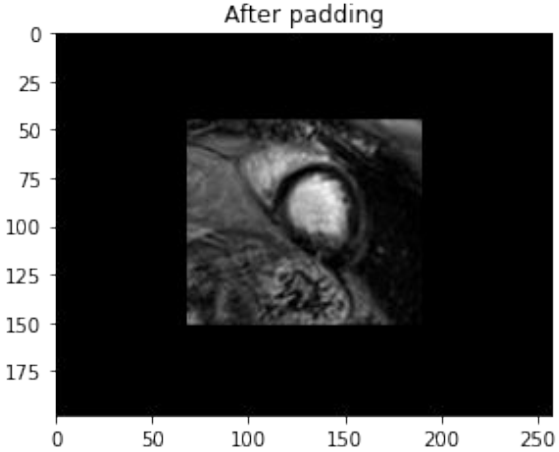
```python
def predict_patient(model, dataset, names, device):
    model.eval()

    labels = []
    pred = []

    n = len(dataset)
    with torch.no_grad():
        for i in range(n):
            x, y = dataset[i]
            x = x[None,:,:,:]
            x = x.to(device)
            y_pred = model(x)

            # Get label with highest score
            y_prob = F.softmax(y_pred, dim = -1)
            top_pred = y_prob.argmax(1, keepdim=True)

            labels.append(y)
            pred.append(top_pred.cpu())
```

```python
# Organizing according to patients names
pat_labels = dict()
pat_pred = dict()
for i in range(n):
    name = names[i]
    # The idea is to change label/prediction only if patient is considered negative until now
    # Patient is negative if no yet labeled/predicted
    if not name in pat_labels:
        current_label = 0
    if not name in pat_pred:
        current_pred = 0
    # Checks if patient has been considered negative until now (label/prediction)
    if name in pat_labels:
        if pat_labels[name] == 0:
            current_label = 0
    if name in pat_pred:
        if pat_pred[name] == 0:
            current_pred = 0
    # If patient is negative, we substitute with a new label/prediction
    if current_label == 0:
        pat_labels[name] = labels[i]
    if current_pred == 0:
        pat_pred[name] = int(pred[i][0,0])


labels, pred = [], []
names_unique = pat_labels.keys()
names_unique2 = pat_pred.keys()
assert(names_unique == names_unique2)
for name in names_unique:
    labels.append(pat_labels[name])
    pred.append(pat_pred[name])

return labels, pred, list(names_unique)
```

Padding

As the heart slices have different shapes and we want as inputs the same dimension, we search for maximum values of the linear dimensions of the shapes and then we apply a *zero padding* to all the slices.





First experiments: **Densenet121**

Layer (type)	Output Shape	Param #	Tr. Param #
Conv2d-1	[1, 64, 40, 40]	9,408	0
BatchNorm2d-2	[1, 64, 40, 40]	128	0
ReLU-3	[1, 64, 40, 40]	0	0
MaxPool2d-4	[1, 64, 20, 20]	0	0
_DenseBlock-5	[1, 256, 20, 20]	335,040	0
BatchNorm2d-6	[1, 256, 20, 20]	512	0
ReLU-7	[1, 256, 20, 20]	0	0
Conv2d-8	[1, 128, 20, 20]	32,768	0
AvgPool2d-9	[1, 128, 10, 10]	0	0
_DenseBlock-10	[1, 512, 10, 10]	919,680	0
BatchNorm2d-11	[1, 512, 10, 10]	1,024	0
ReLU-12	[1, 512, 10, 10]	0	0
Conv2d-13	[1, 256, 10, 10]	131,072	0
AvgPool2d-14	[1, 256, 5, 5]	0	0
_DenseBlock-15	[1, 1024, 5, 5]	2,837,760	0
BatchNorm2d-16	[1, 1024, 5, 5]	2,048	0
ReLU-17	[1, 1024, 5, 5]	0	0
Conv2d-18	[1, 512, 5, 5]	524,288	0
AvgPool2d-19	[1, 512, 2, 2]	0	0
_DenseBlock-20	[1, 1024, 2, 2]	2,158,080	0
BatchNorm2d-21	[1, 1024, 2, 2]	2,048	0
Linear-22	[1, 2]	2,050	2,050

Total params: 6,955,906
Trainable params: 2,050
Non-trainable params: 6,953,856

Input transformations

We need to adapt our input to the pretrained model

```
mean_imagenet = [0.485, 0.456, 0.406]
std_imagenet = [0.229, 0.224, 0.225]

train_transforms = torchvision.transforms.Compose([
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(), # values are normalized between 0 and 1
    torchvision.transforms.Lambda(lambda x: x.repeat(3, 1, 1) ),
    torchvision.transforms.Normalize(mean_imagenet, std_imagenet)
])

test_transforms = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Lambda(lambda x: x.repeat(3, 1, 1) ),
    torchvision.transforms.Normalize(mean_imagenet, std_imagenet)
])
```

Fine tuning

We need lot of data to train a CNN

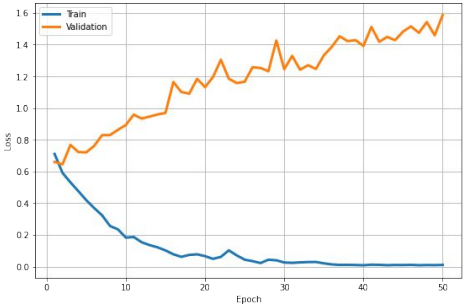
The first layers of a CNN are more “generic” and can be pretrained on a large dataset to be fixed in future application

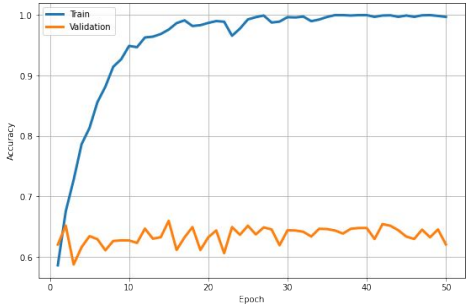
So, in the end, we can train all or only the last few layers (or only the very last) and, if the dataset we’re studying is enough similar to the one used for pretraining, we should even obtain better than training all the layers. We also have to change the output of last layer, because in this case we want binay classification.

Problem: we must hope that our dataset can be considered “similar” to the one used for pretraining.

Pretrained **densenet121** - Training phase

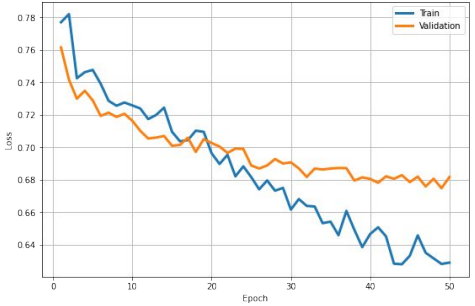
Learning rate: 10^{-3}

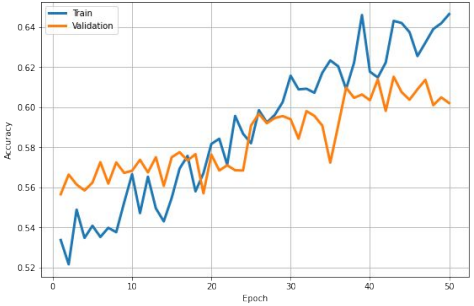




lr 10^{-3}	**Predictions**	
Labels	0	1
0	188	360
1	127	261

Learning rate: 10^{-5}

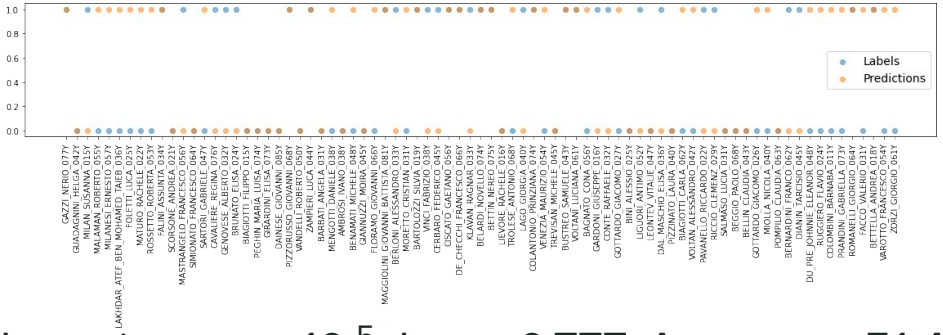




lr 10^{-5}	**Predictions**	
Labels	0	1
0	331	195
1	252	158

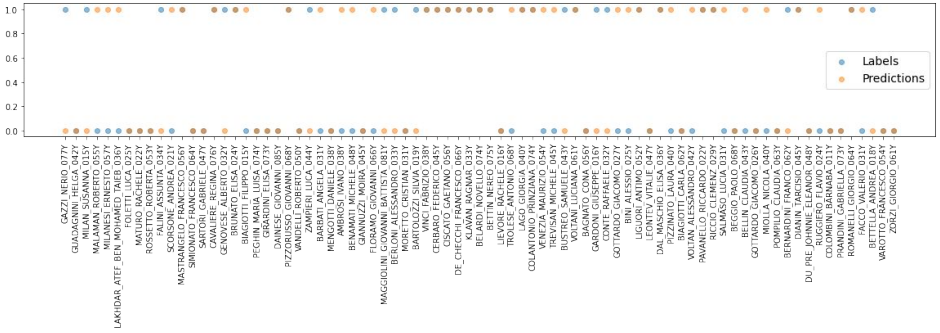
Pretrained **densenet121** - Test phase

Learning rate: 10^{-3}, Loss: 0.835, Accuracy: 50.31%, Patient Accuracy: 44.30%



lr 10^{-3}	**Predictions**	
Labels	0	1
0	20	26
1	18	15

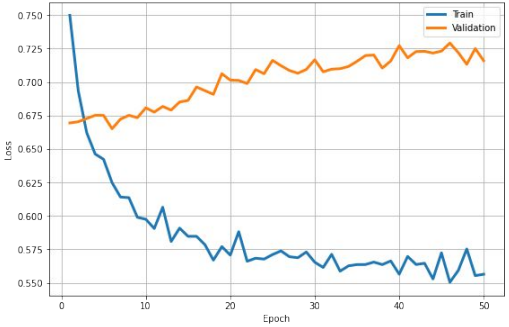
Learning rate: 10^{-5}, Loss: 0.777, Accuracy: 51.41%, Patient Accuracy: 56.96%

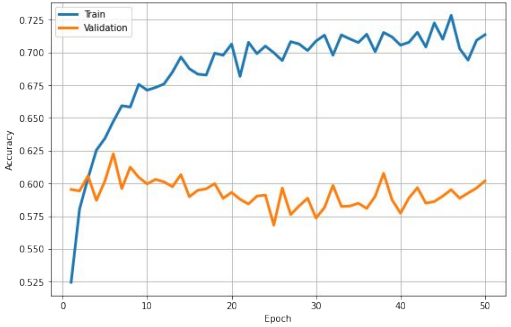


lr 10^{-5}	**Predictions**	
Labels	0	1
0	25	21
1	13	20

Pretrained **densenet121** - feature extraction - Training phase

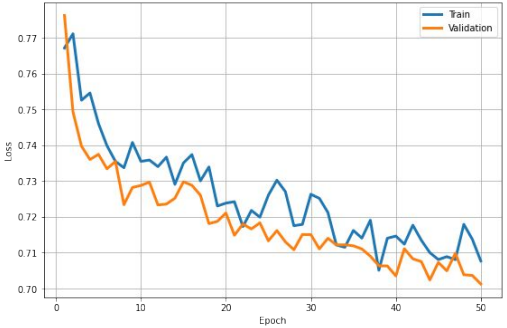
Learning rate: 10^{-3}

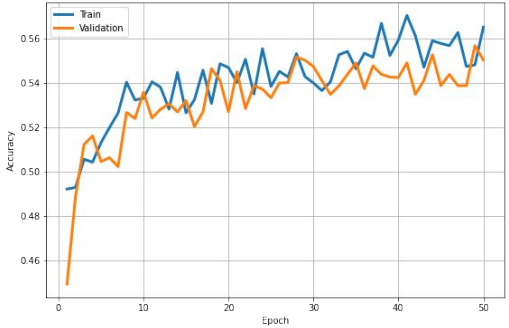




lr 10^{-3}	**Predictions**	
Labels	0	1
0	373	153
1	281	129

Learning rate: 10^{-5}

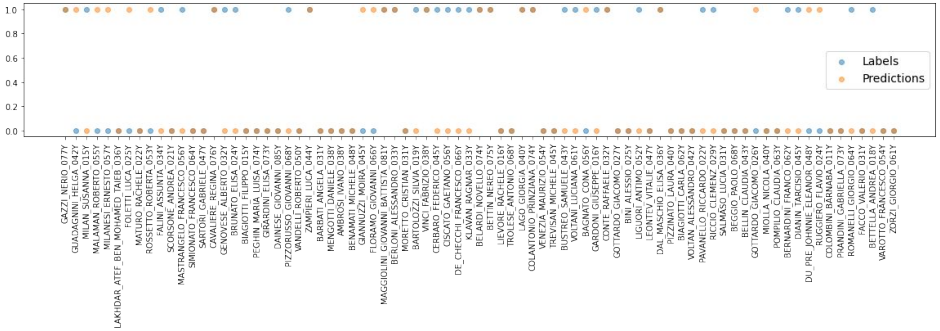




lr 10^{-5}	**Predictions**	
Labels	0	1
0	314	212
1	236	174

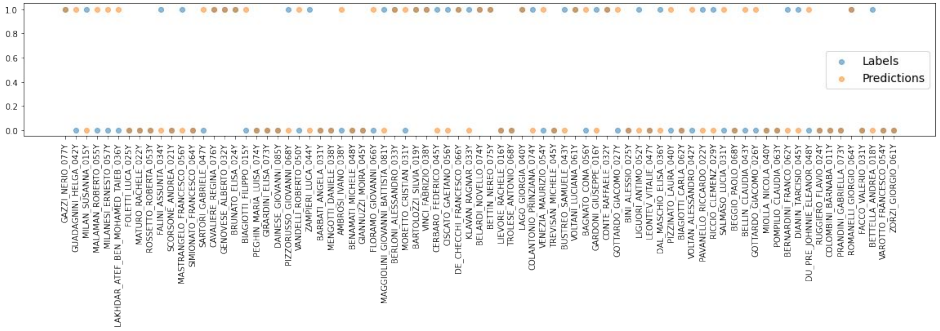
Pretrained **densenet121** - feature extraction - Test phase

Learning rate: 10^{-3}, Loss: 0.765, Accuracy: 53.54%, Patient Accuracy: 59.49%



	Predictions	
Labels	0	1
0	35	11
1	21	12

Learning rate: 10^{-5}, Loss: 0.741, Accuracy: 51.95%, Patient Accuracy: 51.90%



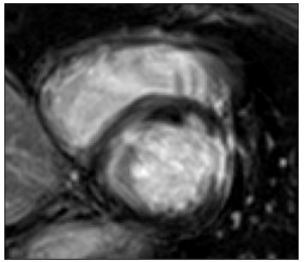
	Predictions	
Labels	0	1
0	27	19
1	19	14

Pretrained **densenet121** - 9 slices

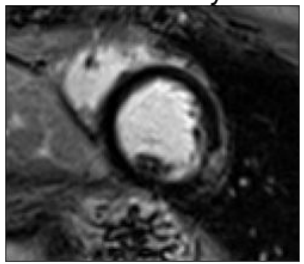
In this case to extract the 9 slices, we equally subdivide the 3D image along the slices to get the 3 sections, each corresponding to the **basal**, **mid-cavity** or **apical** region. Then, in each section, we take the 3 central slices.

```python
less_9_slices_patients = []
for name in patients_folders:
    try:
        path = f"../data_saved/patients_hearts/{name}/heart.npy"
        with open(path, "rb") as f:
            heart = np.load(f)
        if heart.shape[2] < 9:
            less_9_slices_patients.append(name)
    except:
        continue
print("Patients with less than 9 slices:")
print(less_9_slices_patients)
patients_excluded = axial_patients + coronal_patients + other_patients + less_9_slices_patients
```

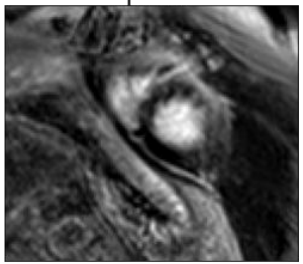
Basal



Mid-cavity



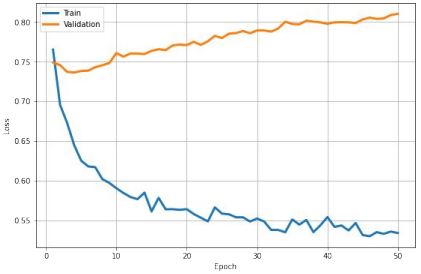
Apical

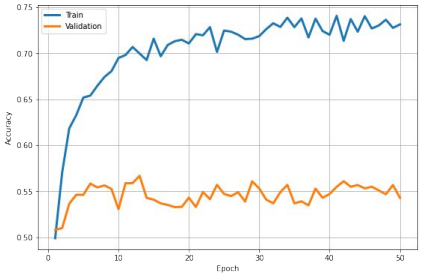


```python
hearts_9 = dict()

for name in patients_folders:
    heart = load_heart(name)
    n_slices = heart.shape[2]
    # If (N slices == 9) we don't have to do anything
    if n_slices == 9:
        hearts_9[name] = heart
        continue
    bp1 = round(n_slices/3)
    bp2 = round(n_slices/3*2)
    # Subdividing into 3 groups
    group1 =  heart[0:bp1]
    group2 = heart[bp1:bp2]
    group3 = heart[bp2:]
    # Taking the 3 central slices of each group
    c1 = bp1 // 2
    c2 = (bp1 + bp2) // 2
    c3 = (bp2 + n_slices) // 2
    indices = [c1-1, c1, c1+1, c2-1, c2, c2+1, c3-1, c3, c3+1]
    h9 = heart[:,:,indices]
    hearts_9[name] = h9
```

Pretrained **densenet121** - 9 slices - Training phase

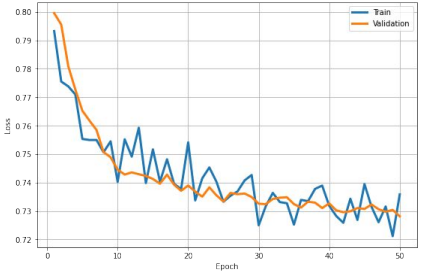
Learning rate: 10^{-3}

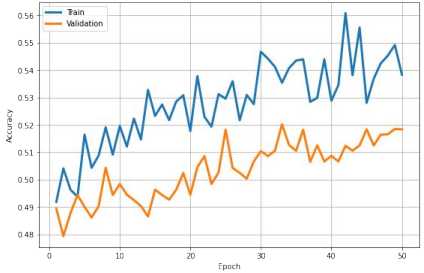




lr 10^{-3}	**Predictions**	
Labels	0	1
0	254	115
1	180	99

Learning rate: 10^{-5}

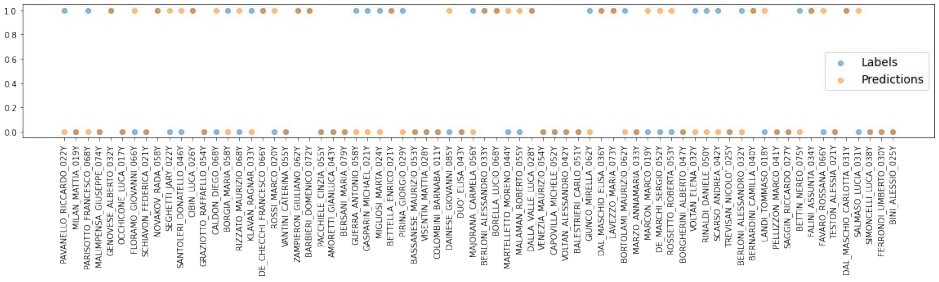




lr 10^{-5}	**Predictions**	
Labels	0	1
0	243	126
1	151	128

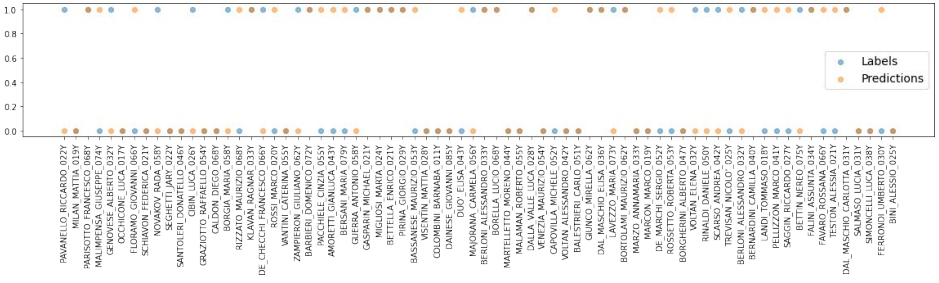
Pretrained **densenet121** - 9 slices - Test phase

Learning Rate: 10^{-3}, Loss: 0.664, Accuracy: 61.59%, Patient Accuracy: 56.94%



lr 10^{-3}	**Predictions**	
Labels	0	1
0	26	15
1	16	15

Learning Rate: 10^{-5}, Loss: 0.662, Accuracy: 61.98%, Patient Accuracy: 52.78%



lr 10^{-5}	**Predictions**	
Labels	0	1
0	22	19
1	15	16

Pretrained **densenet121** - further training

Seeing the last results, we decide to further train the model, obtaining, with learning rate: 10^{-3}

<u>Considering all the slices:</u>

Loss: 0.773, Accuracy: 53.63%, Patient Accuracy: 55.70%

<u>Considering 9 slices per patient:</u>

Loss: 0.658, Accuracy: 63.54%, Patient Accuracy: 54.17%

We can notice how results are worse than previous according to Patient Accuracy. This surely because of overfitting.

all slices	**Predictions**	
Labels	0	1
0	29	17
1	18	15

9 slices	**Predictions**	
Labels	0	1
0	26	15
1	18	13

Pretrained **densenet121** - further training

learning rate: 10^{-5}

<u>Considering all the slices:</u>

Loss: 0.743, Accuracy: 51.45%, Patient Accuracy: 48.10%

<u>Considering 9 slices per patient:</u>

Loss: 0.654, Accuracy: 62.24%, Patient Accuracy: 52.78%

Results are again worse than previous according to Patient Accuracy.

all slices	**Predictions**	
Labels	0	1
0	24	22
1	19	14

9 slices	**Predictions**	
Labels	0	1
0	23	18
1	16	15

Observations

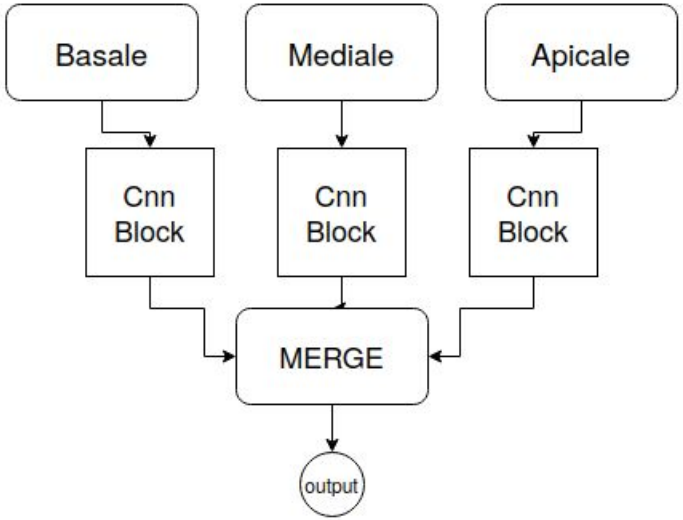
- While there are better results with fine tuning, we're very close to **random guess**
- Two likely explanations: **too few patients** and **not suitable** training phase.
- From training/validation loss, we can assume that the model learns not to find structure to represent the pathology, but **similarity between slices**.
- The best results are obtained using feature extraction with all the slices randomly shuffled for training. Still, the results vary a lot if we change the seed and we'd prefer a more stable and more interpretable model.

Next step is to train a multislice CNN.

Multi-slices CNN 2D (idea)

- Take into account the fact that the disease can be present only in one region (basal-medial-apical)

- We build an architecture that takes in input one slice per region, analyze them and then merge the estimations.



Multi-slices CNN 2D (Input- random approach)

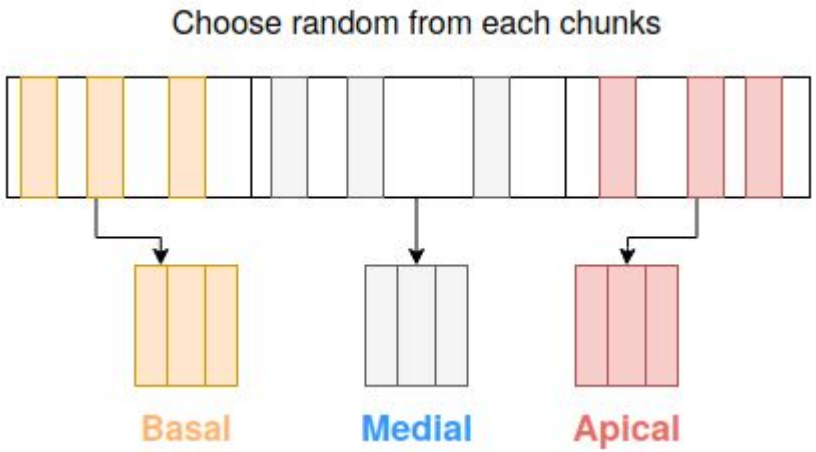
After the preprocessing, the number of slices per patient is in the range (9,29).

We divide the number of slices in each slices in 3 chunks.

In the first approach we select randomly 3 slices from each chunk.

```
# example random sampling
s=13 #number of slices
sl=np.arange(s)
chunk = s//3
add = s%3
print(sl[0:chunk],sl[chunk:chunk*2 + add],sl[chunk*2 + add :])
#division on chunks
chosen = random.sample(list(sl[0:chunk]), 3)
chosen1 = random.sample(list(sl[chunk:chunk*2 + add]), 3)
chosen2= random.sample(list(sl[chunk*2 + add :]), 3)

print(np.sort(chosen),np.sort(chosen1),np.sort(chosen2))
```

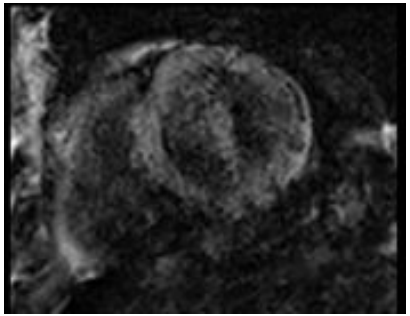
```
[0 1 2 3] [4 5 6 7 8] [ 9 10 11 12]
[0 2 3] [6 7 8] [ 9 10 11]
```

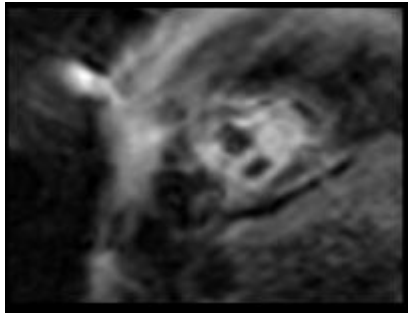


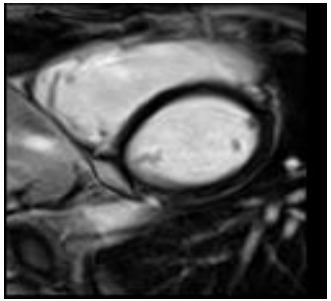
Multi-slices CNN 2D (problem-random approach)

However, this approach is not optimal for the following reason.

- It happens that the first and last slices represents the very edges of the heart, they are more likely to be corrupted by noise.
- We would like to catch slices far from each others, if we catch them randomly it may be not the case.





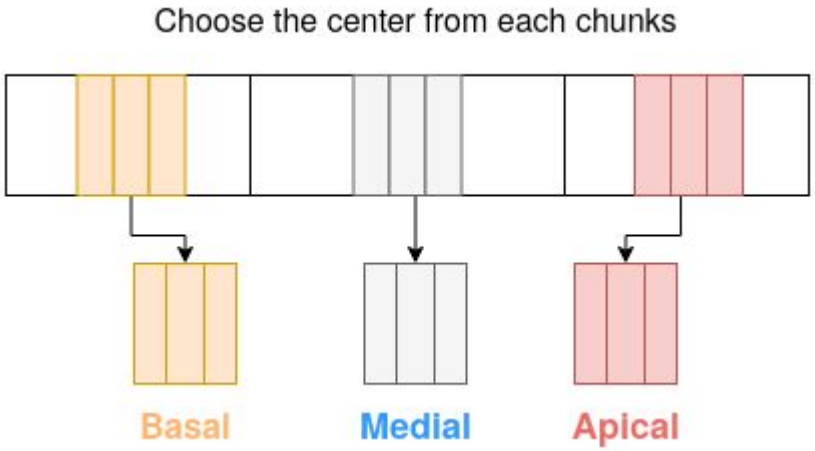


Multi-slices CNN 2D (Input-central approach)

In the central approach we select the 3 centered slices from each chunk.

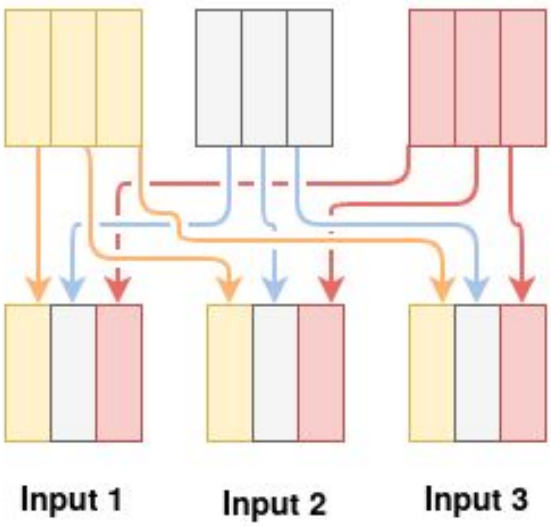
```
#  example operation on index
def get_middle(arr):
    n = arr.shape[0] / 2.0
    n_int = round(n)
    if int(n) % 2 == 1:
        return arr[n_int-2:n_int + 1]
    else:
        return arr[n_int-1:n_int + 2]

s=15; sl=np.arange(s)
chunk = s//3;add = s%3
print(sl[0:chunk],sl[chunk:chunk*2 + add],sl[chunk*2 + add :])
chosen = get_middle(sl[0:chunk])
chosen1 =  get_middle(sl[chunk:chunk*2 + add])
chosen2=  get_middle(sl[chunk*2 + add :])
print(chosen,chosen1,chosen2)
```

```
[0 1 2 3 4] [5 6 7 8 9] [10 11 12 13 14]
[1 2 3] [6 7 8] [11 12 13]
```



Multi-slices CNN 2D (Input)

- The final inputs of the Multi-slices CNNs consist of three triplets for each input patient.




```
#training
basale_tn=np.concatenate((basale_t[:,:,:,0],basale_t[:,:,:,1],basale_t[:,:,:,2]), axis=0)
mediale_tn=np.concatenate((mediale_t[:,:,:,0],mediale_t[:,:,:,1],mediale_t[:,:,:,2]), axis=0)
apicale_tn=np.concatenate((apicale_t[:,:,:,0],apicale_t[:,:,:,1],apicale_t[:,:,:,2]), axis=0)

label2_tn=np.concatenate((label2_t,label2_t,label2_t))
```

Multi-slices CNN 2D (patient accuracy)

- We consider a patient ill if at least one of the 3 inputs triplets of the model predict 1.
- Viceversa id the model predicts 0 for each triplets will be heal.

```
def patient_accuracy(model,weightsmodel=None,use_w=False):
    if use_w== True : model.load_weights(weightsmodel)
    counts = 0
    a=len(basale_tes) #number of patients
    for p in range( a ):
        ii=[p,p+a, p+(a*2)]; tot=0
        for i in ii:
            tot += np.round(model.predict_step(\
                [basale_tesn[ii,:,:,np.newaxis],\
                 mediale_tesn[ii,:,:,np.newaxis],\
                 apicale_tesn[ii,:,:,np.newaxis]]) )
        tot=np.sum(tot)
        if int(tot)==0: tot=0
        else : tot = 1
        if tot==label2_tesn[p]: counts+=1
    print("Patient accuracy is :",counts/a)
```

Multi-slices CNN 2D-Concatenation

The first architecture is sketched on the right.

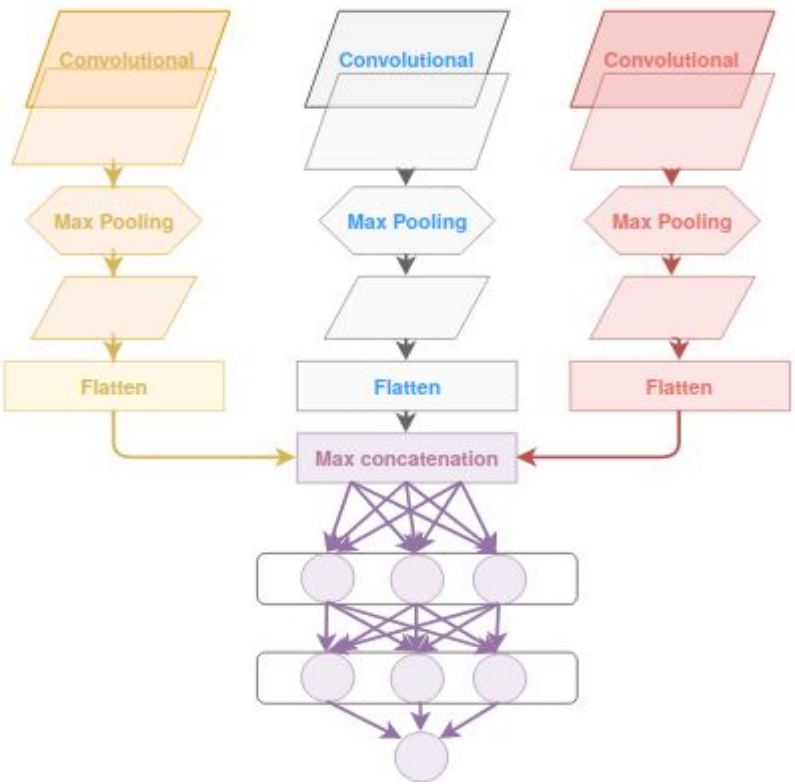
The convolutional layer in each branch from top to bottom has:

- filters: 9,6,3 ;
- kernel size: 5x5 each
- max pooling size: (4x4)

The FC layers has dimension 300, 30, 1. Dropouts to 50% is set in the first FC layer.

The activation functions is **relu** for all the layers except the last two FC in which it's **sigmoid**.

The loss used is the **binary cross entropy.**



Multi-slices 2D-Concatenation (Results)

Here we show the result obtained using **Adam** optimizer with different learning rate.

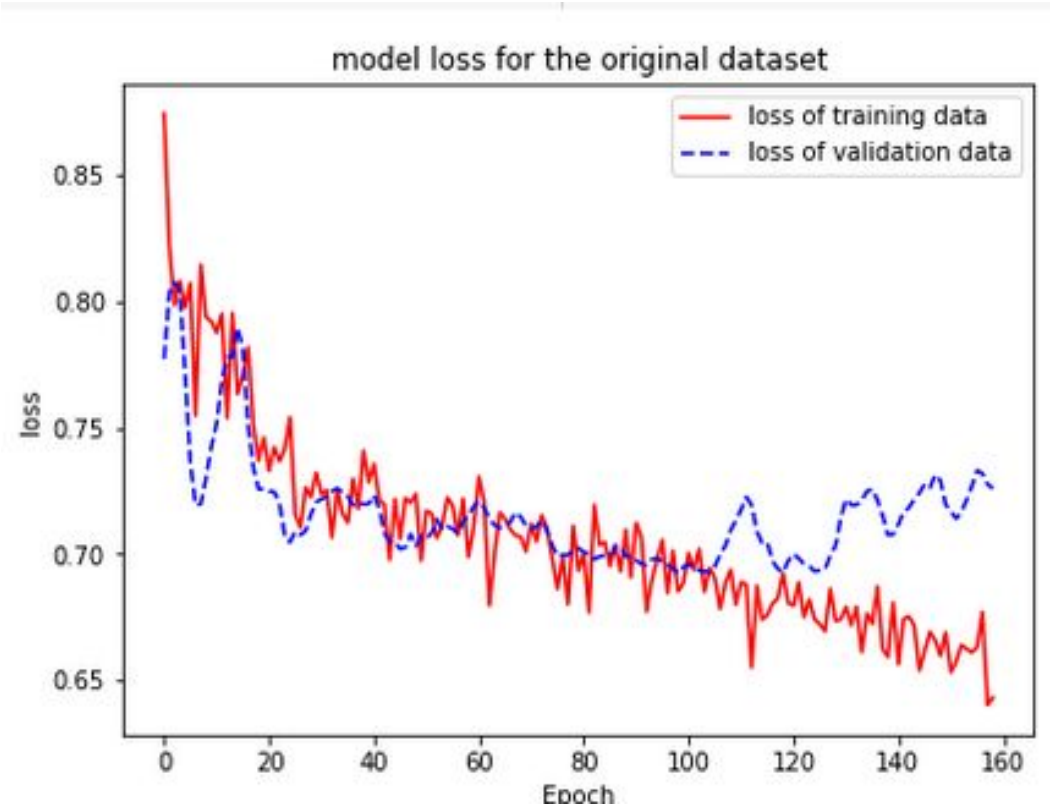
The result is obtained using **early stopping** with patience = 60 monitoring the **validation loss**. The maximum number of **epochs** is 200

	Learning Rate	Patient Accuracy	Dataset
1	1e-5	50%	R
2	3e-5	53%	R
3	5e-5	**61%**	R
4	1e-4	55%	R

	Learning Rate	Patient Accuracy	Dataset
1	1e-5	52%	C
2	3e-5	55%	C
3	5e-5	58%	C
4	1e-4	53%	C

Multi-slices 2D-Concatenation (Best Result and consideration)

- As we can see from the plot the model **over-fit** after some epoch.
- In general concatenate the result in this way *does not* seem to *improve* the performances of the model.
- We consider **different kind of concatenation**, through our trials Average and Max concatenation seem to perform better.



Multi-slices 2D - Indipendent branches

The first architecture is sketched on the right.

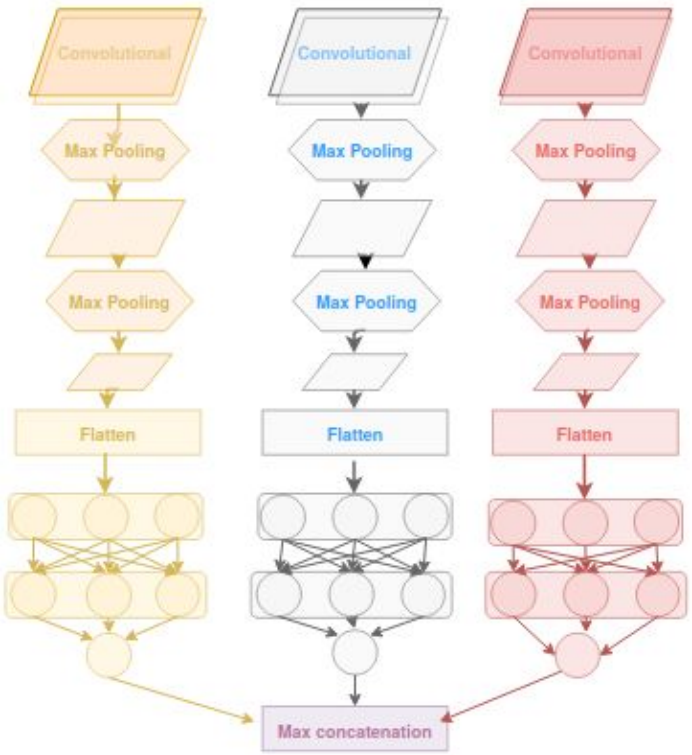
The convolutional layer in each branch from top to bottom has:

- filters: 9, 6, 3, 1 ;
- kernel size: 5, 5, 3, 1
- max pooling size: (2x2) each

The FC layers has dimension 150, 100, 1. Dropouts to 50% is set in the first FC layer.

Given a branch the activation functions is **relu** for all the layers except the last FC in which it's **sigmoid**.

The loss used is the **binary cross entropy.**



Multi-slices 2D - Indipendent branches (Results)

Here we show the result obtained using **Adam** optimizer with different learning rate.

The result is obtained using **early stopping** with patience = 60 monitoring the **validation loss**. The maximum number of **epochs** is 200

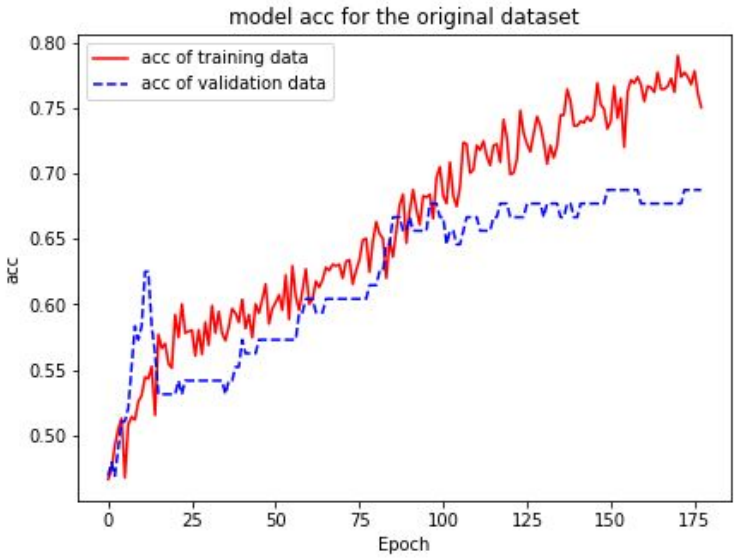
	Learning Rate	Patient Accuracy	Dataset
1	1e-5	55%	R
2	3e-5	53%	R
3	5e-5	58%	R
4	1e-4	50%	R

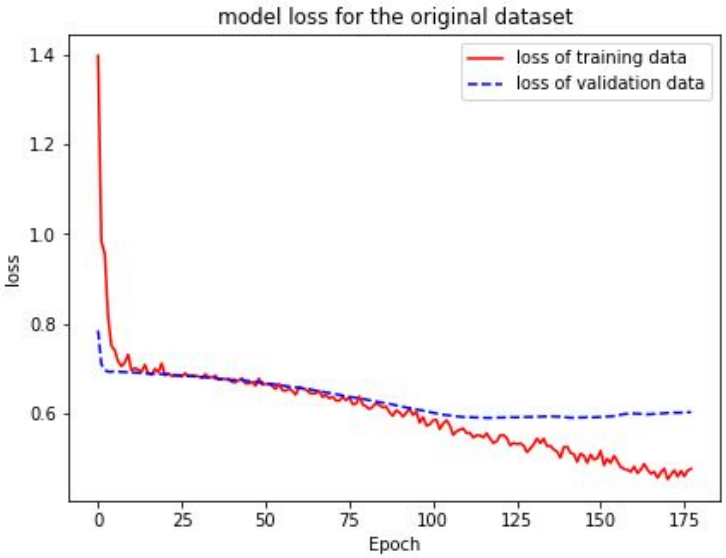
	Learning Rate	Patient Accuracy	Dataset
1	1e-5	55%	C
2	3e-5	56%	C
3	5e-5	61%	C
4	1e-4	**64%**	C

Multi-slices 2D- I.B. (Best Result and consideration)

From this architecture we can observe that :

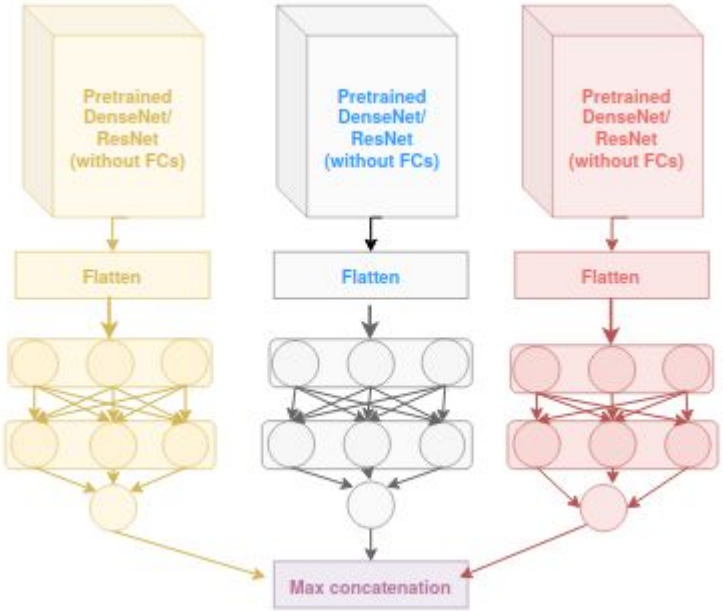
- Consider one CNN for each zone increase the performances of our model
- After a while the model tends to overfit

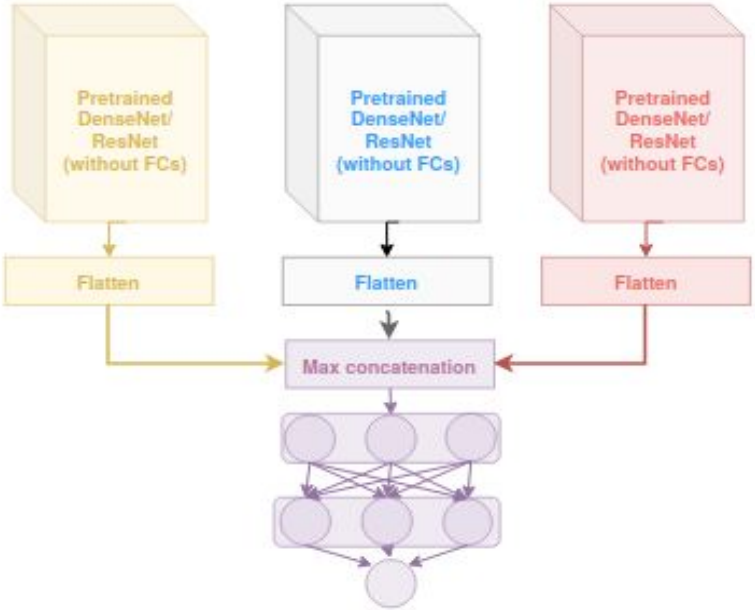




Multi-slices 2D: pretrained models.

We did try to insert use a freezed pre-trained model for each branch of the CNN. However their performance are not relevant.



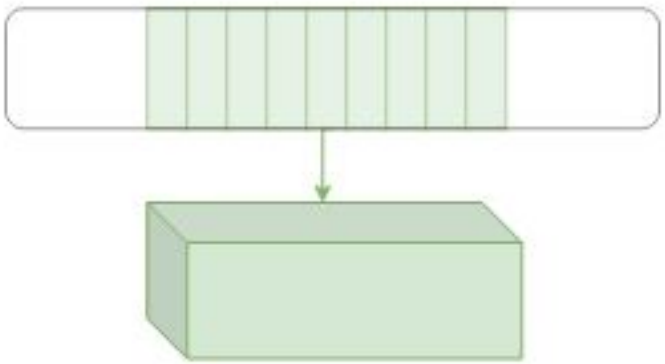


3D CNN (Input)

In 3D convolution, the kernel moves in three dimensions. They are spread in medical clinical imaging in which we have 3D image.

They allow us to consider the volumetric information inside the images.

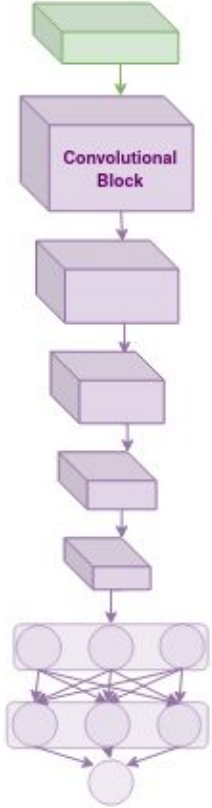
The input consists in the 9 **centered slices** considered as a volume.



3D CNN (Architecture)

input_7 (InputLayer)	[(None, 199, 258, 9, 1)]	0
cropping3d_6 (Cropping3D)	(None, 198, 198, 9, 1)	0
conv3d_30 (Conv3D)	(None, 198, 198, 9, 9)	585
batch_normalization_30 (Batc	(None, 198, 198, 9, 9)	36
conv3d_31 (Conv3D)	(None, 198, 198, 9, 18)	10386
max_pooling3d_24 (MaxPooling	(None, 66, 66, 4, 18)	0
batch_normalization_31 (Batc	(None, 66, 66, 4, 18)	72
conv3d_32 (Conv3D)	(None, 66, 66, 4, 27)	31131
max_pooling3d_25 (MaxPooling	(None, 22, 22, 4, 27)	0
batch_normalization_32 (Batc	(None, 22, 22, 4, 27)	108
conv3d_33 (Conv3D)	(None, 22, 22, 4, 54)	93366
max_pooling3d_26 (MaxPooling	(None, 11, 11, 2, 54)	0
batch_normalization_33 (Batc	(None, 11, 11, 2, 54)	216

conv3d_34 (Conv3D)	(None, 11, 11, 2, 102)	352614
max_pooling3d_27 (MaxPooling	(None, 5, 5, 2, 102)	0
batch_normalization_34 (Batc	(None, 5, 5, 2, 102)	408
global_max_pooling3d (Global	(None, 102)	0
dropout_18 (Dropout)	(None, 102)	0
dense_18 (Dense)	(None, 150)	15450
dropout_19 (Dropout)	(None, 150)	0
dense_19 (Dense)	(None, 100)	15100
dropout_20 (Dropout)	(None, 100)	0
dense_20 (Dense)	(None, 1)	101

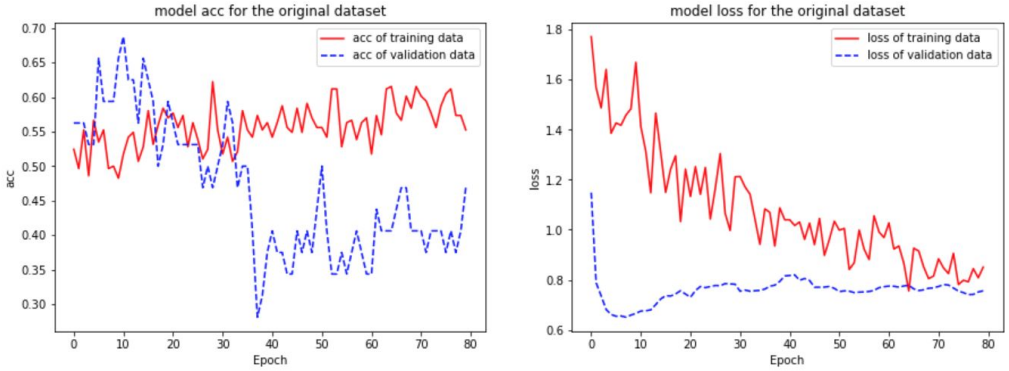


We tried different architectures as in the previous cases.

Issue and criticality are similar, therefore we show one.

3D CNN (Results)

The result has been obtained using **Adam** with learning rate **lr=5e-5**, we save the model that achieve the best validation accuracy.



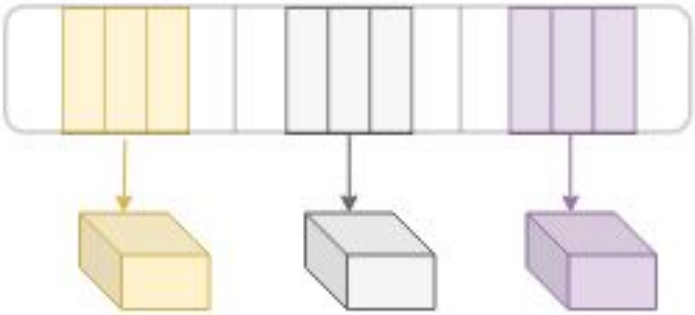

```python
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate(test_3d, label2_tes)
print("test loss, test acc:", results)
```

```
Evaluate on test data
2/2 [==============================] - 2s 268ms/step - loss: 0.7090 - acc: 0.6111
test loss, test acc: [0.708975076675415, 0.6111111044883728]
```

3D Multislices-CNN (Input)

- For patients with an hight number of slices the 9 middle one consider only the medial region.

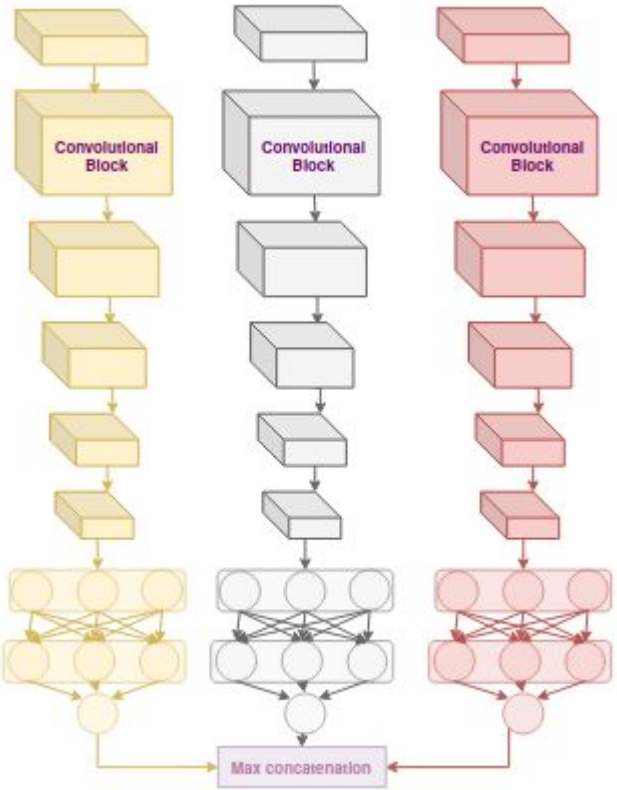
- Therefore we also tried a Multislices CNN with 3D convolution.



3D Multislices-CNN (Architecture)

The architecture consist of 3 CNN 3D, one for each zone similar to the previous one.

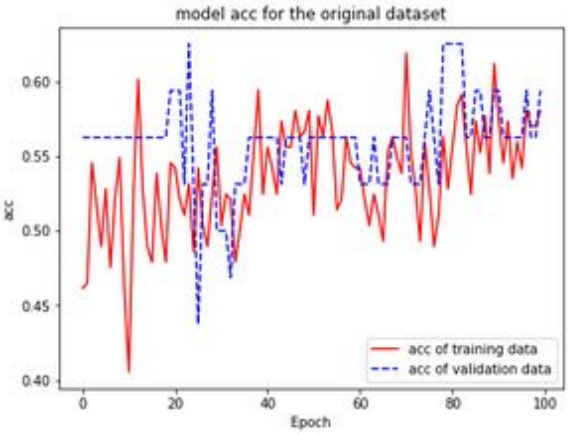
The 3 CNNs are then concatenated in the last layer.

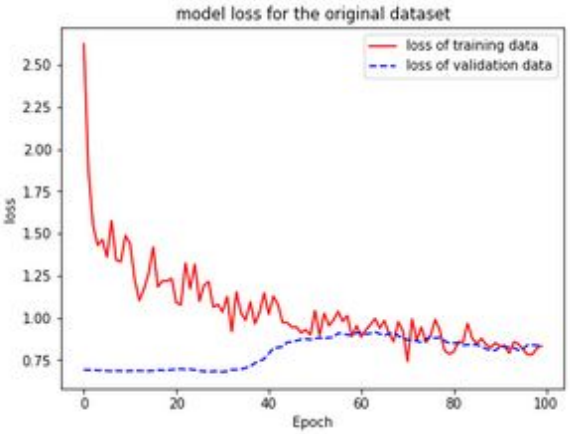


3D Multislices-CNN (Result)

The best result is obtained through :

- Adam optimizer with lr=5e-5;
- batch size = 20;
- 100 epochs.






```
model.load_weights("../content/drive/MyDrive/LOCPBheart/BEST3d_67.h5")
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model.evaluate([basale_tes[:,:,:,:,np.newaxis], mediale_tes[:,:,:,:,np.newaxis], apicale_tes[:,:,:,:,np.newaxis]], label2_tes)
print("test loss, test acc:", results)
```

```
Evaluate on test data
2/2 [==============================] - 2s 362ms/step - loss: 0.6166 - acc: 0.6667
test loss, test acc: [0.6166435480117798, 0.6666666865348816]
```

Consideration

- Multislice CNNs generally perform better than other models.
- In the 3D CNNs we reach a higher score but more real slices per patients would be needed.
- Division in three zone works out.
- Data augmentation does not increase the performances.
- Trials without interpolation made no further improvement.

Summary

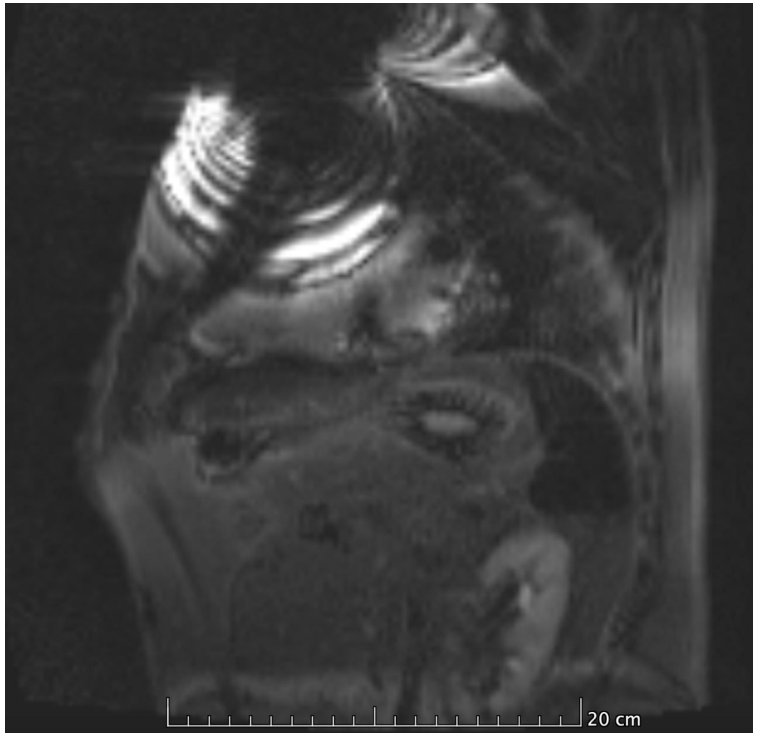
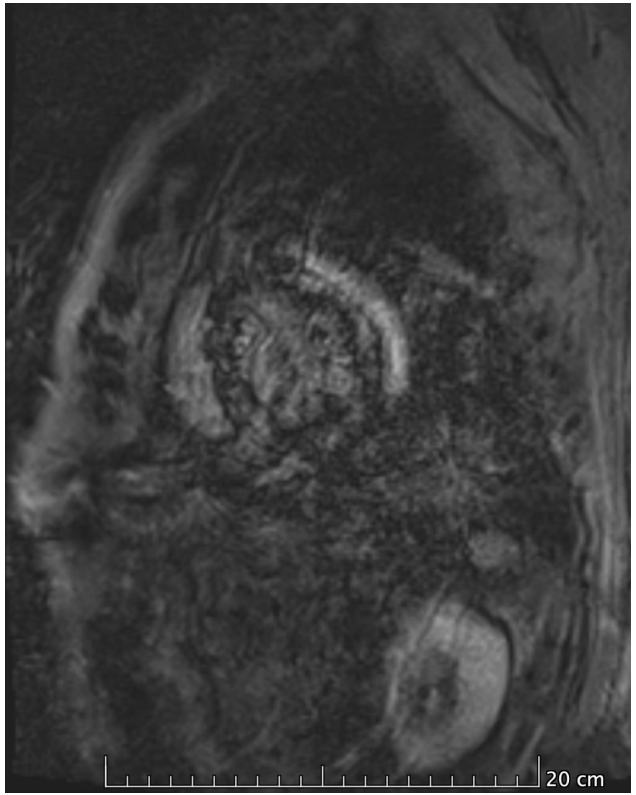
In the different CNNs implemented the ***maximum score*** reached on the test is no more than ***67%.***
The following assumption can be made to explain such results:

- Dataset too small: CNNs require a large number of image as input
- Bias introduced labeling all the slice with the patient label
- Noise introduced with normalization

Possible developments

- Introducing a model that redirect each slices in the relative CNN chunk
- Build a dataset with the the proper label for each slide (require medical knowledge)
- Fine tuning of the parameters in models with the best score so far
- Implementation of a new model that uses heart segmentation

Thanks for the attention!

BACKUP SLIDES

DICOM files of removed patients