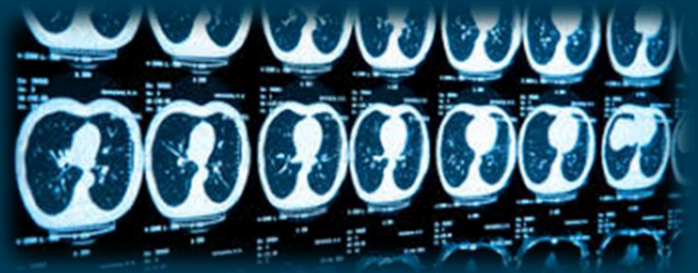
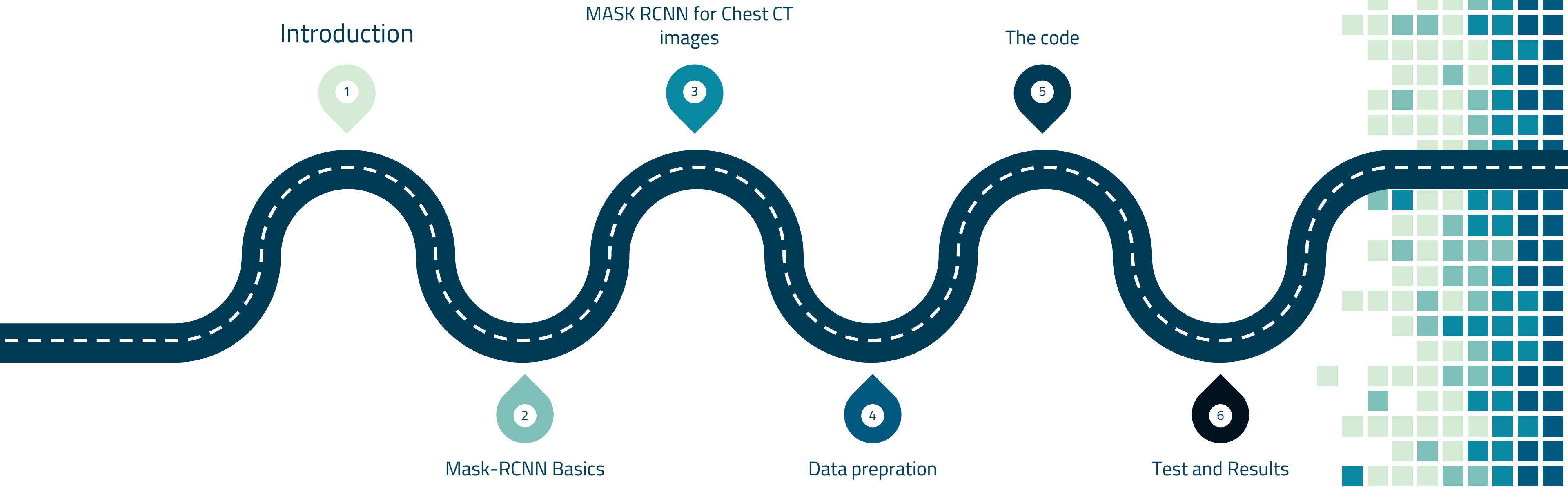
Power of Computer Vision: Image Classification Based on Regional Features of Chest CT-Scan Images

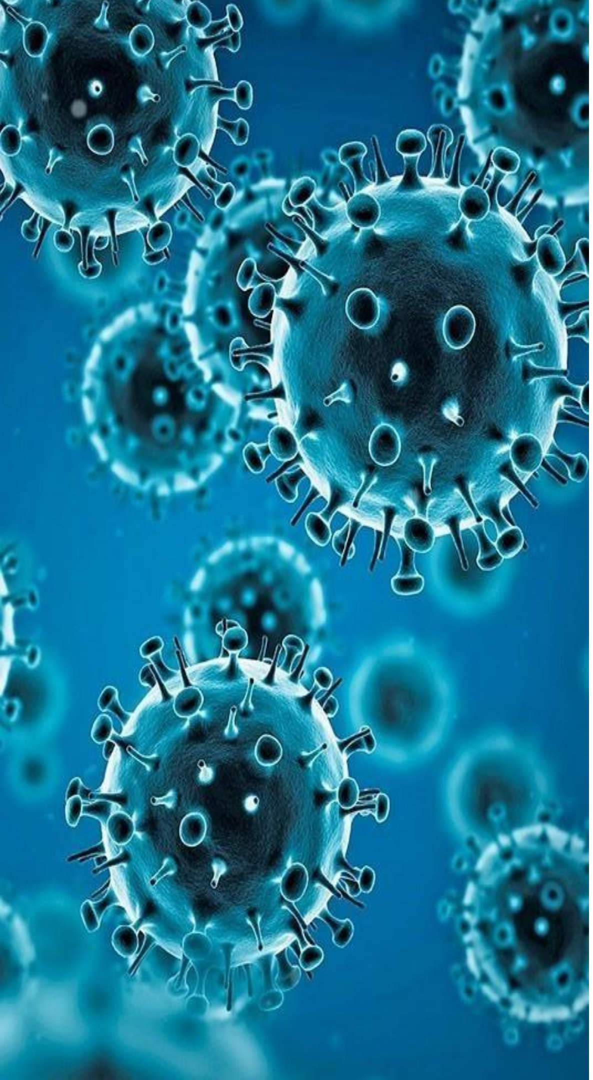


Hamidrea Rokhsati
Alireza Samadifardheris

ROADMAP



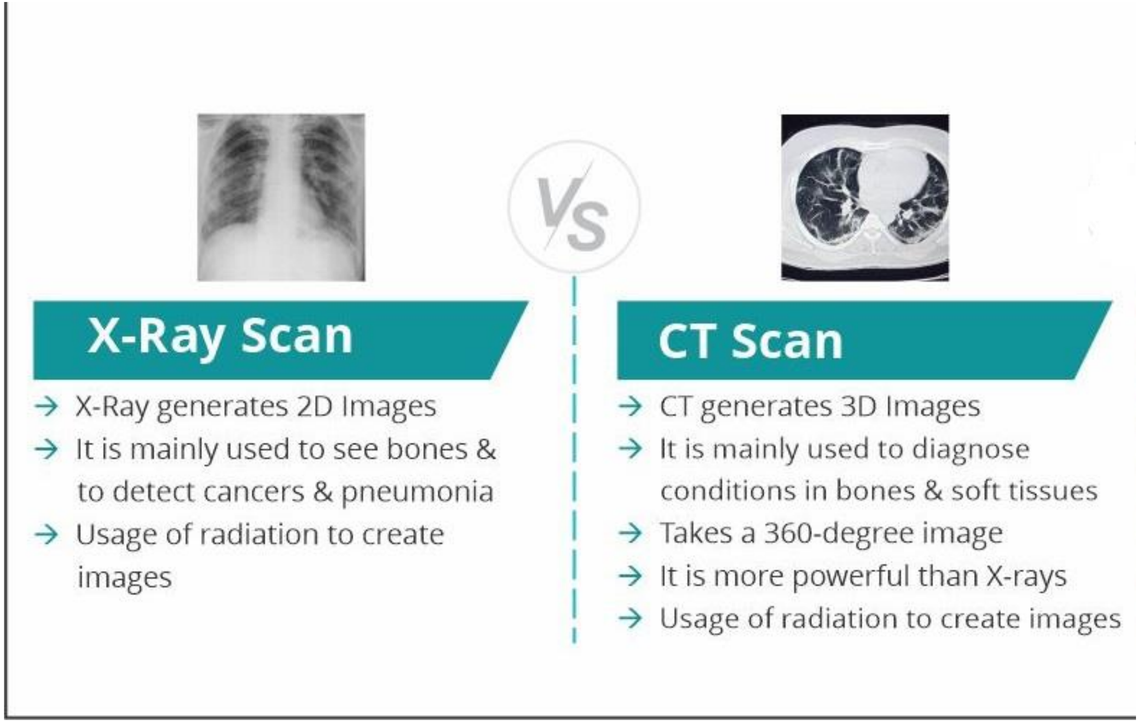
Introduction
1
2
Mask-RCNN Basics
MASK RCNN for Chest CT images
3
4
Data prepration
The code
5
6
Test and Results



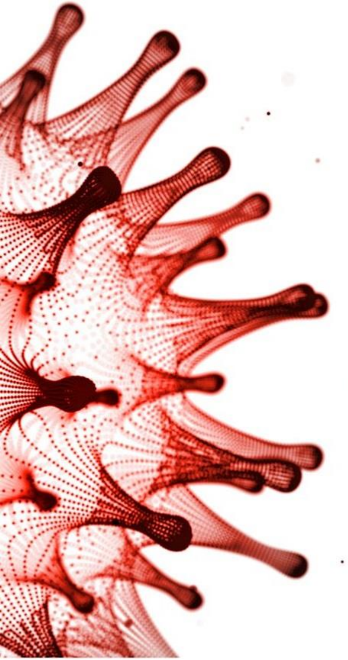
1. Introduction:

- Effective COVID-19 case screening is becoming increasingly crucial
- over 312,173,462 total covid-19 cases around the world and 5,501,000 deaths
- Our goal is to develop a COVID-19 prediction model based on detected regional features using chest CT scans.

1.1. Chest CT-Scan VS. Chest X-Ray



1.2. Chest CT-Scan VS. Chest X-Ray in pandemic



More precise than an X-ray and less time-consuming than RT-PCR

Automated COVID-19 prediction from chest CT scans:

- an approach to assist clinicians and radiologists

Chest CT scans show diverse symptoms of pneumonia:

including covid-19 (lesions)

1.3. Chest CT-Scan :

- What information can we extract ?

Normal

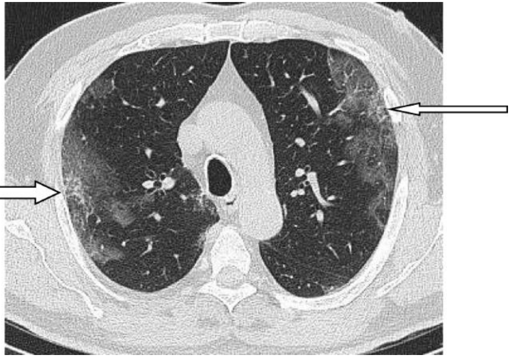
Common pneumonia

Covid-19

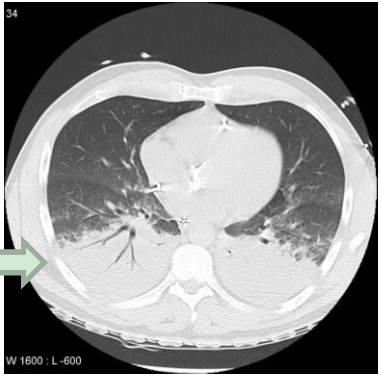
1.4. Critical factors:

- ✓ Associated critical factors of COVID-19 cases
- ✓ obvious differences between COVID-19 and other kinds of pneumonia(but rarely statistically significant.)
- ✓ Ground Glass Opacity (GGO) and Consolidation (C) are two forms of lesions that are linked to both diseases.
- ✓ The location (uni- vs bilateral), distribution (peripheral, diffuse), range, quantity, and attenuation of each lesion type are crucial in distinguishing between the two disorders.

GGO



C



Two types of Lesion Detection using Deep Learning

1. Straightforward (feature extractor + class logits) using ResNet, ResNeXt, DenseNet or a tailored solution (e.g. COVNet, COVNet-CT).

2. Feature extractor + semantic segmentation: predicted masks are concatenated with the feature maps to predict the image class.

To be concrete we are using:

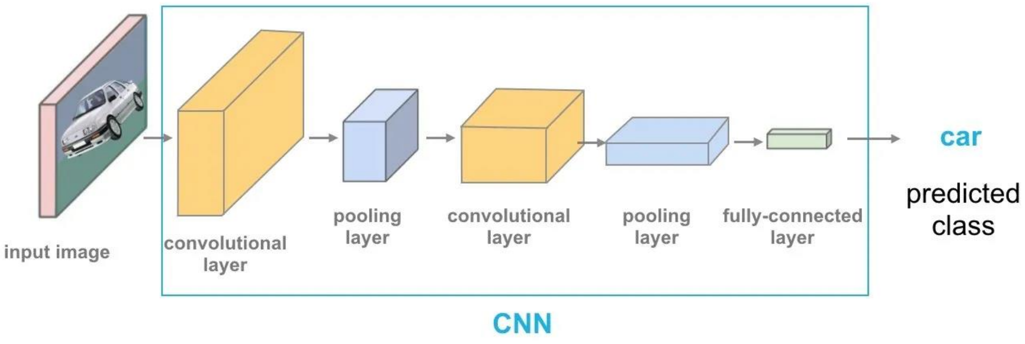
MASK-RCNN

To mask lesions and predict the class of scans

COVID-CT-Mask-Net:
Prediction of COVID-19 from CT Scans Using Regional Features
Aram Ter-Sarkisov

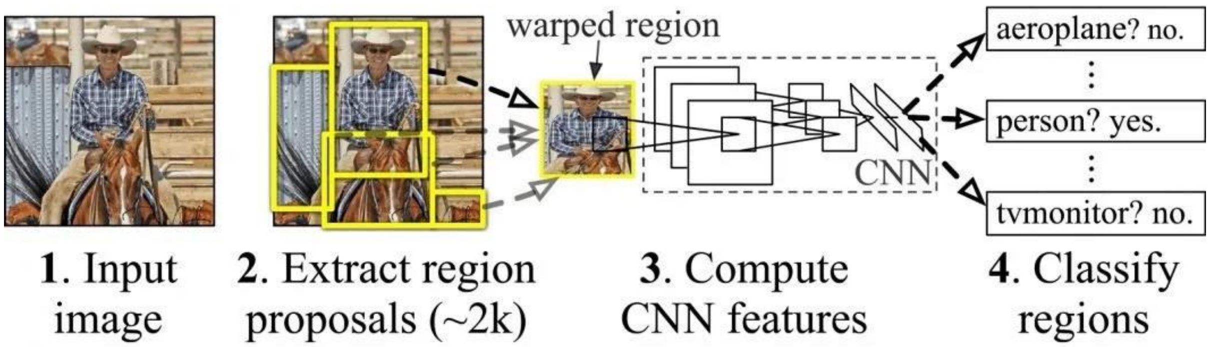
2.1. Mask RCNN:

- Is a Convolutional Neural Network



2.2. Mask RCNN:

- Is a Region-Based Convolutional Neural Network (RCNN) based on FASTER-RCNN



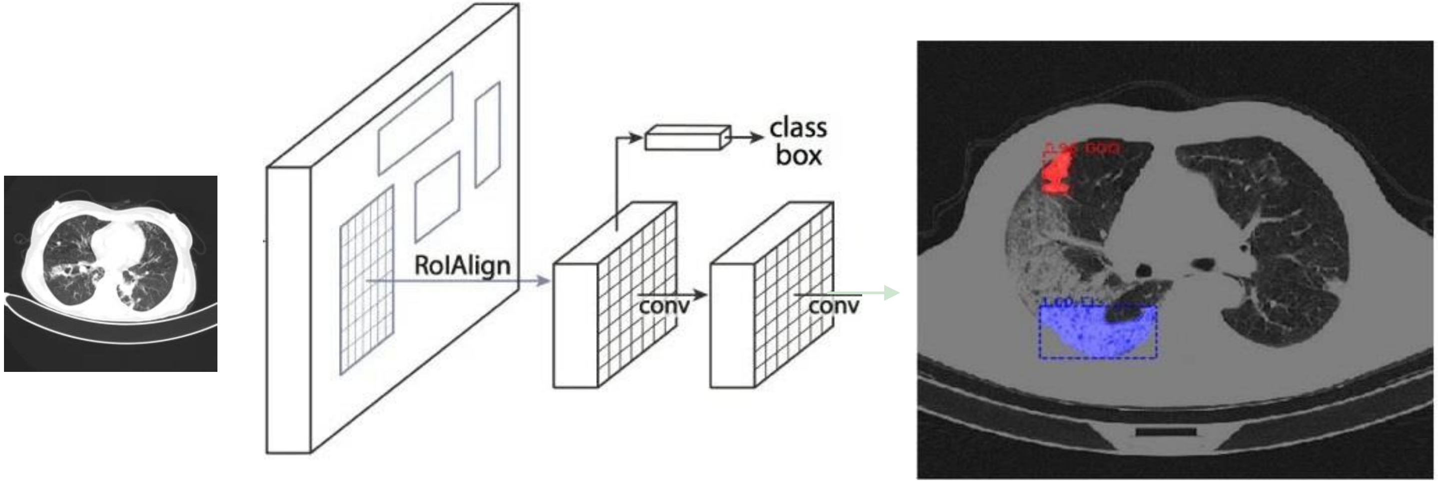
2.2.1. FASTER-RCNN:

- Is an improved version of R-CNN architectures with two stages:
  - **Region Proposal Network (RPN):** is simply a Neural Network that proposes multiple objects that are available within a particular image.
  - **Fast R-CNN.** This extracts features using RoIPool (Region of Interest Pooling) from each candidate box and performs classification and bounding-box regression. RoIPool is an operation for extracting a small feature map from each RoI in detection.
- Faster R-CNN advances this stream by learning the attention mechanism with a Region Proposal Network and Fast R-CNN architecture.
- The reason why "Fast R-CNN" is faster than R-CNN is that you don't have to feed 2'000 region proposals to the convolutional neural network every time. Instead, the convolution operation is done only once per image, and a feature map is generated from it.
- Furthermore, Faster R-CNN is an optimized form of R-CNN because it is built to enhance computation speed (run R-CNN much faster).
- The main difference between Fast and Faster RCNN is that that Fast R-CNN uses selective search for generating Regions of Interest, while Faster R-CNN uses a "Region Proposal Network" (RPN)

2.3. Mask RCNN:

- Is augmented on Faster-RCNN
- While Faster R-CNN has 2 outputs for each candidate object, a class label, and a bounding-box offset, Mask R-CNN is the addition of a third branch that outputs the object mask. The additional mask output is distinct from the class and box outputs, requiring the extraction of a much finer spatial layout of an object.
- The key element of Mask R-CNN is the pixel-to-pixel alignment, which is the main missing piece of Fast/Faster R-CNN.

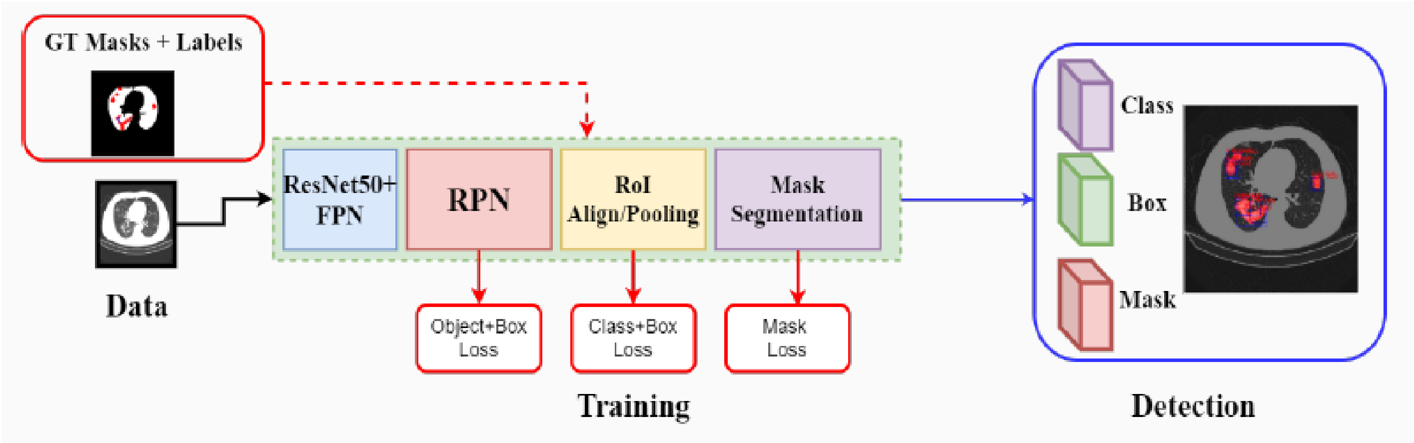
2.4. What do we expect?



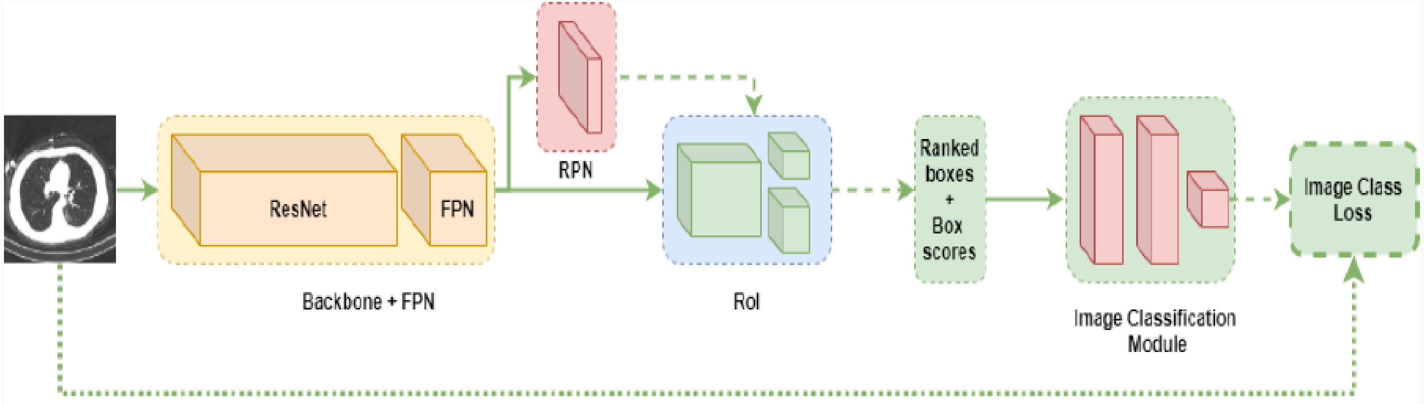
2.5. Why Mask RCNN:

- **Simplicity:** Mask R-CNN is simple to train.
- **Performance:** Mask R-CNN outperforms all existing, single-model entries on every task.
- **Efficiency:** The method is very efficient and adds only a small overhead to Faster R-CNN.
- Faster R-CNN and Mask R-CNN solve the problem of detecting (predicting the bounding box + class) and segmenting each object independently, i.e., the model understands objects at the instance level rather than the image or pixel level.
- This contrasts with image classification techniques such as ResNet and semantic (pixel-level) segmentation techniques such as UNet. Mask R-CNN can handle partial occlusion and disconnected objects instead of nameless feature maps and image-wide score maps.
- As a result, Mask R-CNN performs extremely accurate instance segmentation and prediction.

3.1. Training Mask RCNN:



3.2. Classification using Mask RCNN:



3.3. Algorithm

Set E:total number of epochs, α: learning rate, λ: weight regularization parameter.
Initialize COVID-CT-Mask-Net with the weights and anchors from the segmentation model.
for 1 **to** E **do**
  Input :Batch of CT images, sparse label vector **L** with C classes
  Extract backbone features from the images in the batch
  RPN: predict bounding boxes containing objects and their scores
  RoI: extract N box coordinates predictions and their scores
  Predict N masks (ignored in our implementation)
  Regions Of Interest Output :Batch of N encoded boxes and their confidence scores (tensor $N \times 5$)
  Classifier Module **S** : accept the ranked boxes and scores, convert batch to feature vector, extract global features
  COVID-CT-Mask-Net Output:Vector of image class predictions $\hat{\mathbf{s}}$
  Binary per-class cross-entropy loss: $\mathcal{L}(\hat{\mathbf{s}}, \mathrm{L}) = -\sum_{k=1}^{C} L_k \times \log \sigma(\hat{s}_k)$
end
Return the best model

DISCLAIMER

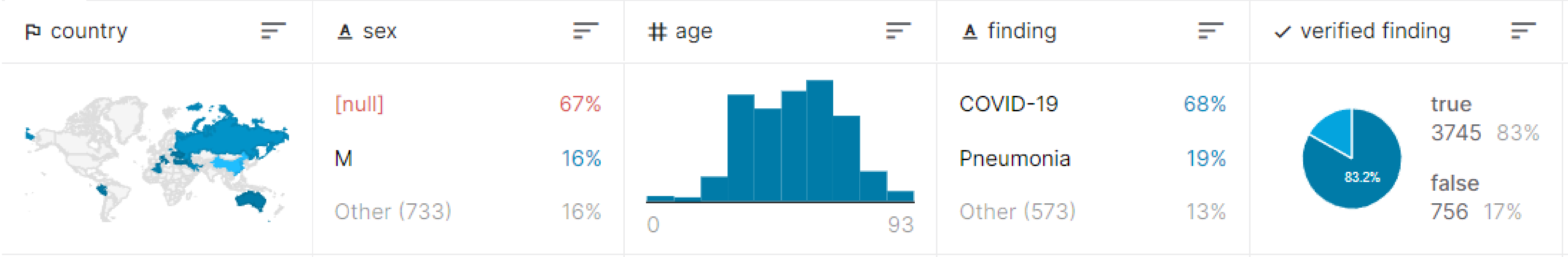
This presentation is for the Computer vision project only

DO NOT USE THIS CODE FOR SELF DIAGNOSING AND CONTACT HEALTH AUTHORITIES IF YOU HAVE ISSUES

Let’s dive into the code now!

4.1. DATA

- Where do we get data from?
- China Consortium of Chest CT Image Investigation were used to create a dataset of CT images and metadata
- All CT images are divided into three categories: new coronavirus pneumonia (NCP) related to SARS-CoV-2 virus infection, common pneumonia, and normal controls.
- A summary of data: comprising 194,922 CT slices from 3,745 patients



Images are labeled as [Class-string]_[PatientID]_[ScanNum]_[SliceNum].png

4.2. Libraries:

OpenCV

python™

PyTorch

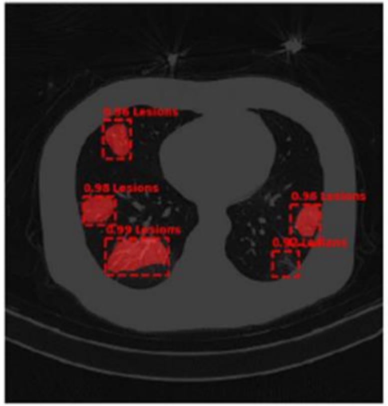
torchvision

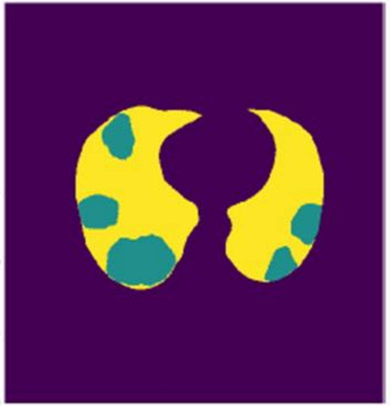
MASK-RCNN

etc

5.1. Segmentation

- Training stage : inputs are CT images and their masked versions





We used a trained model to segment our images :

Mask GGO AND C

With their BB and probability

1_Instance_segmentation.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text Connect Editing

```python
maskrcnn_args = {'num_classes': None, 'min_size': 512, 'max_size': 1024, 'box_detections_per_img': 100,
                 'box_nms_thresh': roi_nms, 'box_score_thresh': confidence_threshold, 'rpn_nms_thresh': rpn_nms,
                 'box_head': box_head, 'rpn_anchor_generator': anchor_generator, 'mask_head':None,
                 'mask_predictor': mask_predictor, 'box_predictor': box_predictor}
```

1

```python
# Instantiate the segmentation model
maskrcnn_model = mask_net.maskrcnn_resnet_fpn(backbone_name, truncation, pretrained_backbone=False, **maskrcnn_args)
# Load weights
maskrcnn_model.load_state_dict(ckpt['model_weights'])
# Set to evaluation mode
print(maskrcnn_model)
maskrcnn_model.eval().to(device)
```

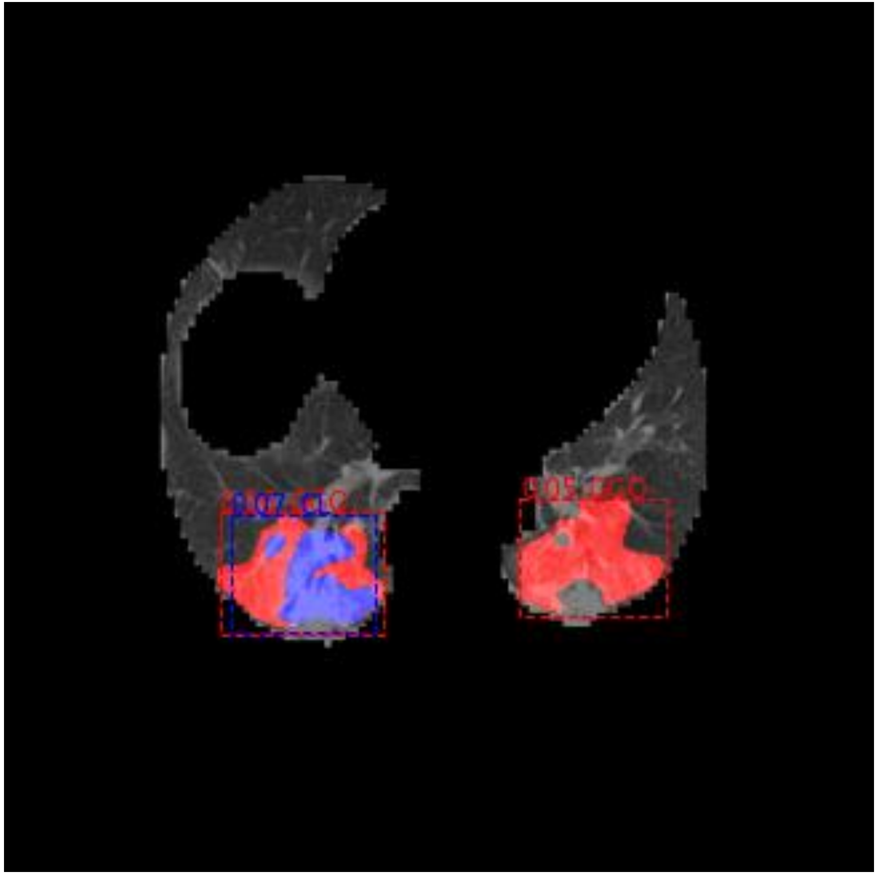
2

```python
start_time = time.time()
# get the correct masks and mask colors
if mask_type == "ggo":
   ct_classes = {0: '__bgr', 1: 'GGO'}
   ct_colors = {1: 'red', 'mask_cols': np.array([[255, 0, 0]])}
elif mask_type == "merge":
   ct_classes = {0: '__bgr', 1: 'Lesion'}
   ct_colors = {1: 'red', 'mask_cols': np.array([[255, 0, 0]])}
elif mask_type == "both":
   ct_classes = {0: '__bgr', 1: 'GGO', 2: 'CL'}
   ct_colors = {1: 'red', 2: 'blue', 'mask_cols': np.array([[255, 0, 0], [0, 0, 255]])}

# run the inference with provided hyperparameters for images in the provided directory
test_ims = os.listdir(os.path.join(data_dir, img_dir))

for j, ims in enumerate(tqdm(test_ims)):
    test_step(os.path.join(os.path.join(data_dir, img_dir), ims), device, maskrcnn_model,confidence_threshold, mask_threshold, save_dir, ct_classes, ct_colors, j)
    if True: #REMOVE THIS AND BREAK LINE TO DO IT ON ALL IMAGES IN DIR
      break
end_time = time.time()
print("Inference took {0:.1f} seconds".format(end_time - start_time))
```

3



5.2. Classification

- First stage: = Training based on a pre-trained model:
- Aim: keep training on our data to decrease loss
- In only 10 epochs: with 3000 train and 1000 validation images:
- We reached to Train loss and Validation loss respectively less than 0.01 and 0.02
- How long ? 1 hour on Colab GPU

2_train_classifier.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

Connect Editing

```
dataset_covid_pars_train_cl = {'stage': 'train', 'data': train_data_dir, 'img_size': (512,512)}
datapoint_covid_train_cl = dataset.COVID_CT_DATA(**dataset_covid_pars_train_cl)
#
dataset_covid_pars_eval_cl = {'stage': 'eval', 'data': val_data_dir, 'img_size': (512,512)}
datapoint_covid_eval_cl = dataset.COVID_CT_DATA(**dataset_covid_pars_eval_cl)
#
dataloader_covid_pars_train_cl = {'shuffle': True, 'batch_size': batch_size}
dataloader_covid_train_cl = data.DataLoader(datapoint_covid_train_cl, **dataloader_covid_pars_train_cl)
#
dataloader_covid_pars_eval_cl = {'shuffle': True, 'batch_size': batch_size}
dataloader_covid_eval_cl = data.DataLoader(datapoint_covid_eval_cl, **dataloader_covid_pars_eval_cl)
#
ckpt = torch.load(pretrained_model, map_location=device)
#
covid_mask_net_args = {'num_classes': None, 'min_size': 512, 'max_size': 1024, 'box_detections_per_img': roi_batch_size, 'box_nms_thresh': roi_nms, 'box_score_thresh': -0.01, 'rpn_nms_thresh': rpn_nms}

# copy the anchor generator parameters, create a new one to avoid implementations' clash
sizes = ckpt['anchor_generator'].sizes
aspect_ratios = ckpt['anchor_generator'].aspect_ratios
anchor_generator = AnchorGenerator(sizes, aspect_ratios)
# out_channels:256, FPN
# num_classes:3 (1+2)
box_head = TwoMLPHead(in_channels=256*7*7, representation_size=128)
box_predictor = FastRCNNPredictor(in_channels=128, num_classes=n_c)

covid_mask_net_args['rpn_anchor_generator'] = anchor_generator
covid_mask_net_args['box_predictor'] = box_predictor
covid_mask_net_args['box_head'] = box_head
covid_mask_net_args['s_representation_size'] = s_features
# Instantiate the model
covid_mask_net_model = mask_net.fasterrcnn_resnet_fpn(backbone_name, truncation, **covid_mask_net_args)
# which parameters to train?
trained_pars = []
# if the weights are loaded from the segmentation model:
if pretrained_classifier is None:
```

1

2

3

4

2_train_classifier.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

Connect Editing

```python
def step(stage, e, dataloader, optimizer, device, model, save_every, lrate, model_name, anchors, save_dir):
    epoch_loss = 0
    for id, b in enumerate(tqdm(dataloader)):
        optimizer.zero_grad()
        X, y = b
        if device == torch.device('cuda'):
            X, y = X.to(device), y.to(device)
        # some batches are less than batch_size
        batch_s = X.size()[0]
        batch_scores = []
        # input all images in the batch into COVID-Mask-Net to get B scores
        for id in range(batch_s):
            image = [X[id]]  # remove the batch dimension
            predict_scores = model(image)
            batch_scores.append(predict_scores[0]['final_scores'])
        # batchify scores/image and compute binary cross-entropy loss
        batch_scores = torch.stack(batch_scores)
        batch_loss = F.binary_cross_entropy_with_logits(batch_scores, y)
        if stage == "train":
            batch_loss.backward()
            optimizer.step()
        else:
            pass
        epoch_loss += batch_loss.clone().detach().cpu().numpy()
    epoch_loss = epoch_loss / len(dataloader)
    if not (e+1) % save_every and stage == "eval":
        model.eval()
        state = {'epoch': str(e+1), 'model_weights': model.state_dict(),'optimizer_state': optimizer.state_dict(), 'lrate': lrate, 'anchor_generator': anchors,
                 'model_name': model_name}
        if model_name is None:
            torch.save(state, os.path.join(save_dir, "covid_ct_mask_net_ckpt_" + str(e+1) + ".pth"))
        else:
            torch.save(state, os.path.join(save_dir, model_name + "_ckpt_" + str(e+1) + ".pth"))
        utils.set_to_train_mode(model)
```

6.1. Test

- On trained model
- On 1000 test images(not used in train)

3_test_classifier.ipynb

File Edit View Insert Runtime Tools Help Last edited on January 14

+ Code + Text

```
classes=("0: Control", "1: Common Pneumonia", " 2: COVID-19")
normalize=True
plot_confusion_matrix(cm,classes,normalize,'confusion_matrix',cmap=plt.cm.Blues)
print("Evaluation took {0:.1f} seconds".format(end_time - start_time))
```

the only point is to compare predicted label with actual class extracted from labels

```
def test_step(im_input, model, source_dir, device, c_matrix):
    correct_class = int(im_input.split('/')[-1].split('_')[0])
    im = PILImage.open(os.path.join(source_dir, im_input))
    if im.mode != 'RGB':
        im = im.convert(mode='RGB')
    # get rid of alpha channel
    img = np.array(im)
    # print(img)
    if img.shape[2] > 3:
        img = img[:, :, :3]
    t_ = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize(512),
        transforms.ToTensor()])
    img = t_(img)
    if device == torch.device('cuda'):
        img = img.to(device)
    out = model([img])
    pred_class = out[0]['final_scores'].argmax().item()
    # get confusion matrix
    c_matrix[correct_class, pred_class] += 1
```

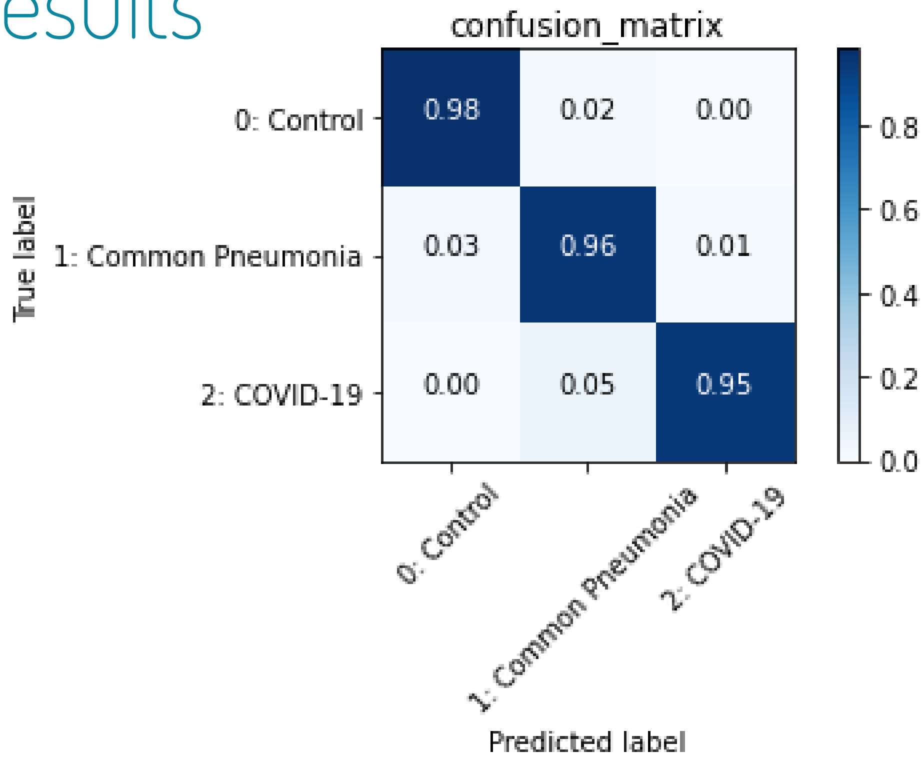
1

2

of previous stages we have confusion matrix but lets visulize it in this code you can uncomment some lines to have TP,TN, etc

```
def plot_confusion_matrix(cm, classes, normalize, title='Confusion matrix', cmap=plt.cm.Blues):
    cm=cm.cpu().data.numpy()
```

6.2. Test Results



3_test_classifier.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

RAM Disk Editing

```
Confusion Matrix for 3-class problem:
0: Control, 1: Normal Pneumonia, 2: COVID
tensor([[423,   8,   0],
        [ 10, 336,   5],
        [  0,  11, 207]], device='cuda:0', dtype=torch.int32)
------------------------------------------
------------------------------------------
Class Sensitivity:
tensor([[0.9814, 0.0186, 0.0000],
        [0.0285, 0.9573, 0.0142],
        [0.0000, 0.0505, 0.9495]], device='cuda:0')
------------------------------------------
Overall accuracy:
tensor(0.9660, device='cuda:0')
                                  0         1         2  micro-average
accuracy                   0.982000  0.966000  0.984000       0.977333
f1                         0.979167  0.951841  0.962791       0.966000
false_discovery_rate       0.023095  0.053521  0.023585       0.034000
false_negative_rate        0.018561  0.042735  0.050459       0.034000
false_positive_rate        0.017575  0.029276  0.006394       0.017000
negative_predictive_value  0.985891  0.976744  0.986041       0.983000
positive_predictive_value  0.976905  0.946479  0.976415       0.966000
precision                  0.976905  0.946479  0.976415       0.966000
recall                     0.981439  0.957265  0.949541       0.966000
sensitivity                0.981439  0.957265  0.949541       0.966000
specificity                0.982425  0.970724  0.993606       0.983000
true_negative_rate         0.982425  0.970724  0.993606       0.983000
true_positive_rate         0.981439  0.957265  0.949541       0.966000
Normalized confusion matrix
[[0.98143852 0.01856148 0.        ]
 [0.02849003 0.95726496 0.01424501]
 [0.         0.05045871 0.94954127]]
Evaluation took 53.3 seconds
```

53s completed at 6:17 PM

THANKS!

Any questions?

You can find us at:

Alireza Samadifardheris:

Samadifardheris.1961823@studenti.uniroma1.it

Hamidreza Rokhsati:

Rokhsati.1960699@studeni.uniroma1.it

CREDITS

- https://www.kaggle.com/hgunraj/covidxctAlex git hub
- COVID-CT-Mask-Net: Prediction of COVID-19 from CT Scans Using Regional Features Aram Ter-Sarkisov
- EDL-COVID: Ensemble Deep Learning for COVID-19 Case Detection From Chest X-Ray Images Shanjiang Tang et al
- www.worldmeter.com
- https://viso.ai/