ITSP Final Documentation

*** This will be considered as your final documentation and will be made available for online usage.***

X-Ray Anomaly Detection using CNNs

Team PerXeptron

ITS20054

Keywords (Include 7 or more keywords which will help others find your documentation easily)

Transfer Learning, Model Ensemble, CNN, Deep Learning, CheXpert, CAM-Visualization, Django REST framework(DRF), Angular 9, Keras, TensorFlow, PyTorch

Table Of Contents:

Team Member Name	Roll Number	Email-Id
Prapti Kumar	190040075	praptijkumar@gmail.com
Ankit Kumar Misra	190050020	ankitkumarmisra2001@gmail.com
Harshit Varma	190100055	varmaharshit2@gmail.com
Richeek Das	190260036	richeekdas2001@gmail.com

Inspiration for Idea

Chest X-Ray interpretation and diagnosis is a highly time consuming and skill dependent task. If not done properly,it has severe consequences for the patient. The automation of this work with sufficiently accurate results would be a huge advancement in the field of healthcare.

In today's unprecedented times, with overburdened healthcare systems, we wanted to build an effective tool to help the society.

Link to our Abstract : PerXeptron: Abstract

Problem Statement

1. Use the recent boom in Deep Learning developmental resources to curb some of the problems in the medical world.
2. Radiology being a tedious task seems to be a great place to impact with automated intervention.

Problem statements proposed :

1. Create Deep Learning Models to quickly identify/diagnose common thoracic diseases from Chest XRays, and get as close to the current state-of-the-art as possible.
2. Develop a window such as an OPEN-API or an Interactive WebApp to expose the model endpoints.

Existing solutions in the Market

Automated Chest X-ray Interpretation – qXR v2.0: (Trained on **1.2M** X-Rays)
qXR detects abnormal chest X-rays, then identifies and localizes 15 common abnormalities. It also screens for tuberculosis, and is used in public health screening programs.

Diseases (Overlapping with our 5 diseases)	**AUC**
Consolidation	0.941 (0.934 - 0.948)
Cardiomegaly	0.950 (0.947 - 0.954)
Pleural Effusion	0.957 (0.953 - 0.960)

Reference: http://qure.ai/qxr.html

Diseases	**AUC (achieved by us)**
Consolidation	0.945
Cardiomegaly	0.880
Pleural Effusion	0.936

Apart from this marketed solution, it is mostly a research topic.
Most solutions for detecting X-Ray anomalies (especially Atelectasis, Cardiomegaly, Consolidation, Edema and Pleural Effusion) are submissions in the CheXpert competition.

Proposed Solution and Brief Description

Innovation :

To Build An Open-API :

- A REST API open to all developers, for posting Chest X-Rays anonymously and retrieving disease results and class activated heatmaps with high accuracy, using our **near state-of-the-art Deep Learning Ensemble**.

To Build An Interactive Web-App :

- A beautiful, light-weight and minimalistic web app to provide an easy interactive UI for the end-users.

Impact:

In a future where we envision precision medicine, automating Chest XRay diagnosis is a baby step but a bold one. It will make disease predictions faster and reduce the cases of False Negatives among radiologists. If the interface is minimal and the predictions are widely accepted among the radiology community, this can end up serving as a “Radiologist’s best friend” for sure.

Progress

Note : You can find detailed workload below in Contribution by each Team member.

WorkFlow Between Review Meets :

Review Meets	**Work Done**
Before Review Meet 1	● Mutually decided and finalized on the Dataset, Data Preprocessing and the Disease Classes which we were going to study. ● Decided on what our final product was going to be and completed the design elements, based on that, which had further changes later on.
Between Review Meet 1 and 2	● Created **home page, upload page,result page.** ● Trained DenseNet121, DenseNet169, DenseNet201, IRNv2, Xception to form a model ensemble. ● Completed the Django REST FRAMEWORK Backend.
After Review Meet 2	● Created **login, register and profile pages**.Enabled routing between components, ran tests and deployed the front-end. ● Completed the Authentication on the client side. ● Completed the model ensemble. ● Completed the **CAM Localization code**, to generate heatmaps of located diseases. ● Remodelled **Django Backend** code to integrate the **Heatmap Generator** and **Model Ensemble.** ● **Everything was successfully integrated and completed.**

Challenges:

- **Accuracy and AUC Saturation:**
  - After about 25% of the first epoch, the AUC value saturated to 0.70.
    - Solution: Designed a better classifier and reduced the learning rate to 0.05 times the default value, to avoid massive weight updates during the initial batches.

- **Featurewise Centering and Standard normalization:**
  - The above two preprocessing steps require the Keras ImageDataGenerator to be fit to the training data, so as to calculate the mean, standard deviation and other values required for centering and normalization. Our dataset has the Image paths stored in the dataframe, thus there was no direct way to fit the data generator to the training set, as the fit method expected a 4D image tensor as input.
    - Solutions proposed:
      1. Refactor the entire dataset to to be made up of flattened 4D image tensors instead of images. This is undesirable as the dataset is extremely large (~11GB) and we need to probably do this everytime we run the notebook.
      2. Create a randomly selected subset of the training data from the images and fit the data generator on this subset. Currently using this.

- **Google Colab and Drive Issues**:
    - Colab **runtime disconnect** after a period of inactivity.
        - Solution: JavaScript Injection to click the connect button after a fixed interval of time (~60s)
    - Colab's **RAM gets exhausted** on trying to fit the data generators to a large subset (i.e >0.3%) of the training data
        - Solution proposed: Creating the subset by appending small samples of fixed length, one at a time
    - Google Drive's **download limit exceeds** after some time.
        - Solution: Create a new google account, and copy the dataset from one account to another.
    - **12 hour limit on GPU** usage (Some models are very big and require about 1.5-2 hrs for a single epoch)
        - No solution found, all models were trained on <=5 epochs.

- **Running heavy ensemble models on local servers :**
    - Faced problems while running a **5 Keras** Model Ensemble alongside a **PyTorch** Model in the Backend.
        - Solution : Run Tensorflow and PyTorch on different threads with disconnected graphs. Also set a max memory limit on Tensorflow(which is by default set to consume all the memory available to it) using :

```
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0],
True)
tb._SYMBOLIC_SCOPE.value = True
```

Calculations:

- **Weighted Average while Ensembling:**
  Some models work great for predicting a particular disease, while others classify it poorly. Thus, taking a simple average of the final predictions won't be an ideal solution. Therefore, we have defined a weights matrix which contains the model-wise and disease-wise weights to be used while making the final prediction.
  We have found the weights empirically, and we normalize them disease-wise before taking the average:

```
WEIGHTS = {
    'xception' : [0.50, 0.30, 1.00, 1.10, 1.00],
    'irnv2'    : [0.40, 1.00, 0.70, 0.00, 0.75],
    'dn121'    : [1.50, 0.20, 1.00, 0.10, 0.40],
    'dn169'    : [0.80, 0.50, 0.90, 0.00, 0.75],
    'dn201'    : [0.80, 0.50, 0.60, 0.00, 0.10]
}
```

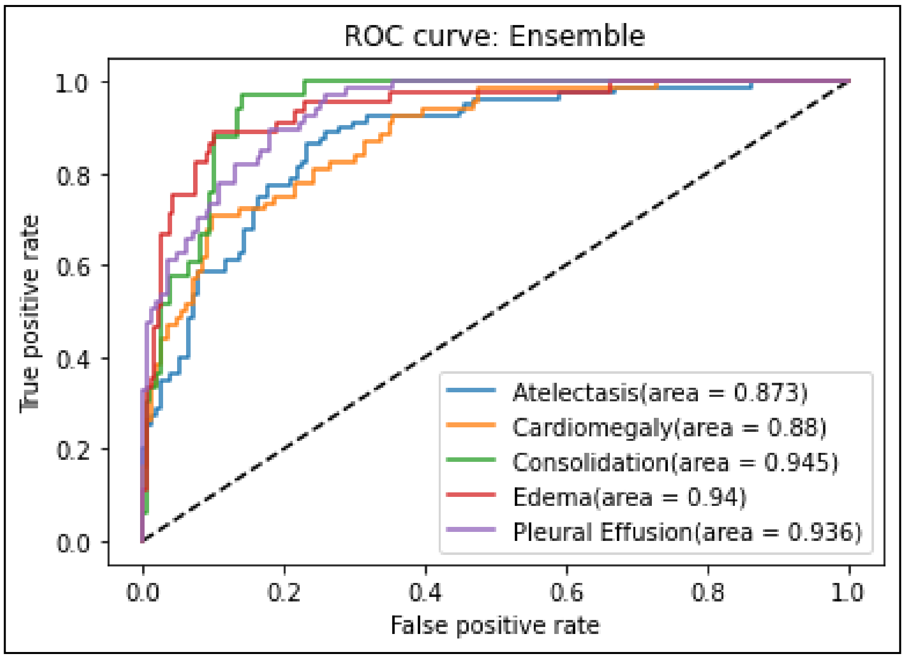
- **Optimal Threshold Selection**:
  We predict the required thresholds for prediction by maximizing the **Youden's Index**.
  Then, if the probability predicted by the model is more than this threshold value, we predict the 'presence' of the disease.

Results

Links to all the stuff we have created :

GitHub Organization:	PerXeptron
Models & Ensemble:	Model Creation, Ensemble and Experimentation
CAM-Visualization	CAM Localization
Backend:	Django REST FRAMEWORK Backend
Frontend:	Angular 9 Frontend
Repo for GitHub Pages	Compiled JS | GitHub Pages
Front end Deployed	DEMO FRONTEND
Team Video	Link to video demo
Final Presentation	Team PerXeptron: Final Presentation

Results achieved in the CNN Part:

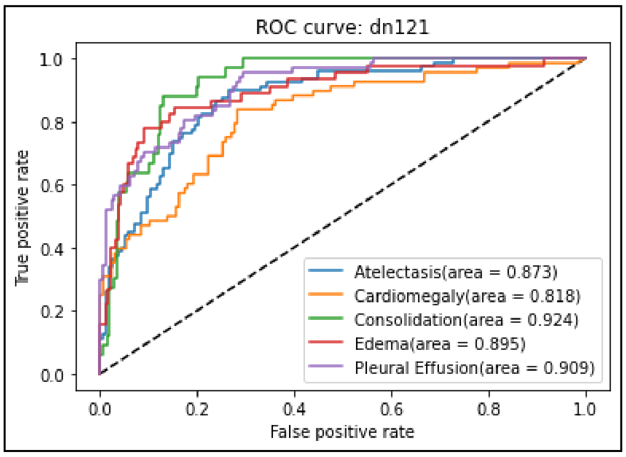


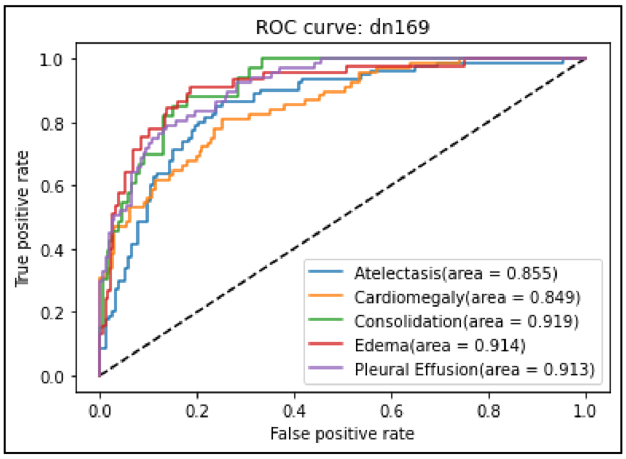
Final **Average AUC** (Area under the ROC Curve) achieved by the Ensemble: **0.915**
Final **Categorical Accuracy** achieved by the Ensemble: **51.7%**

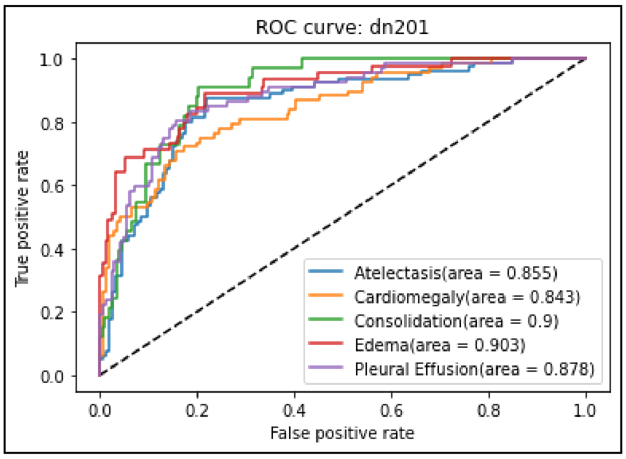
Results achieved by the **individual models**:

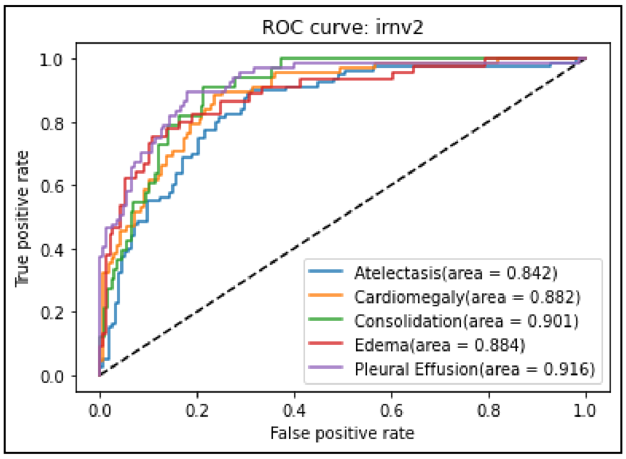
Model Name	Average AUC	Categorical Accuracy
Xception	0.894	46.6%
Inception-ResNet-v2	0.885	44.9%
DenseNet-121	0.884	41.0%
DenseNet-169	0.890	43.2%
DenseNet-201	0.876	46.2%

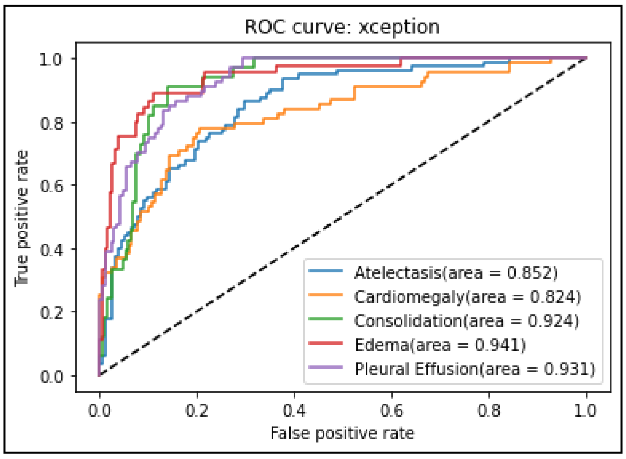
Individual ROC Curves





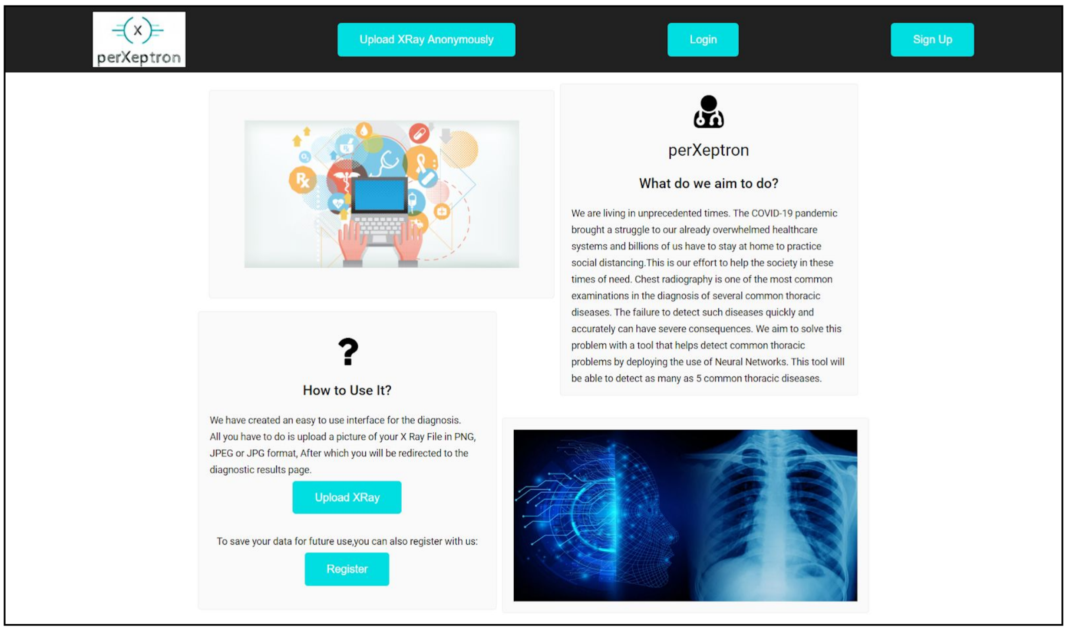






Interactive web-app:

Homepage:



perXeptron

What do we aim to do?

We are living in unprecedented times. The COVID-19 pandemic brought a struggle to our already overwhelmed healthcare systems and billions of us have to stay at home to practice social distancing.This is our effort to help the society in these times of need. Chest radiography is one of the most common examinations in the diagnosis of several common thoracic diseases. The failure to detect such diseases quickly and accurately can have severe consequences. We aim to solve this problem with a tool that helps detect common thoracic problems by deploying the use of Neural Networks. This tool will be able to detect as many as 5 common thoracic diseases.

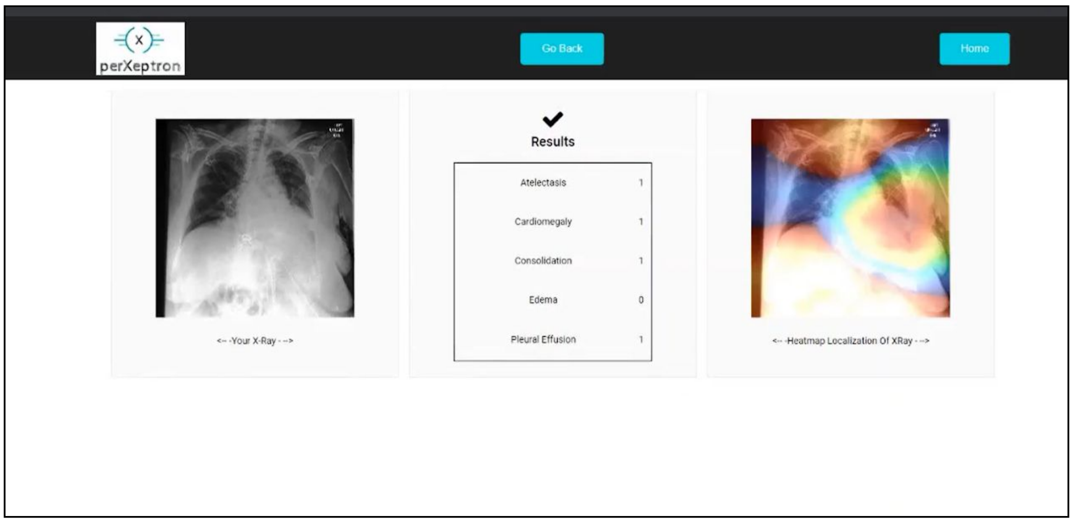
?

How to Use It?

We have created an easy to use interface for the diagnosis. All you have to do is upload a picture of your X Ray File in PNG, JPEG or JPG format, After which you will be redirected to the diagnostic results page.

To save your data for future use,you can also register with us:

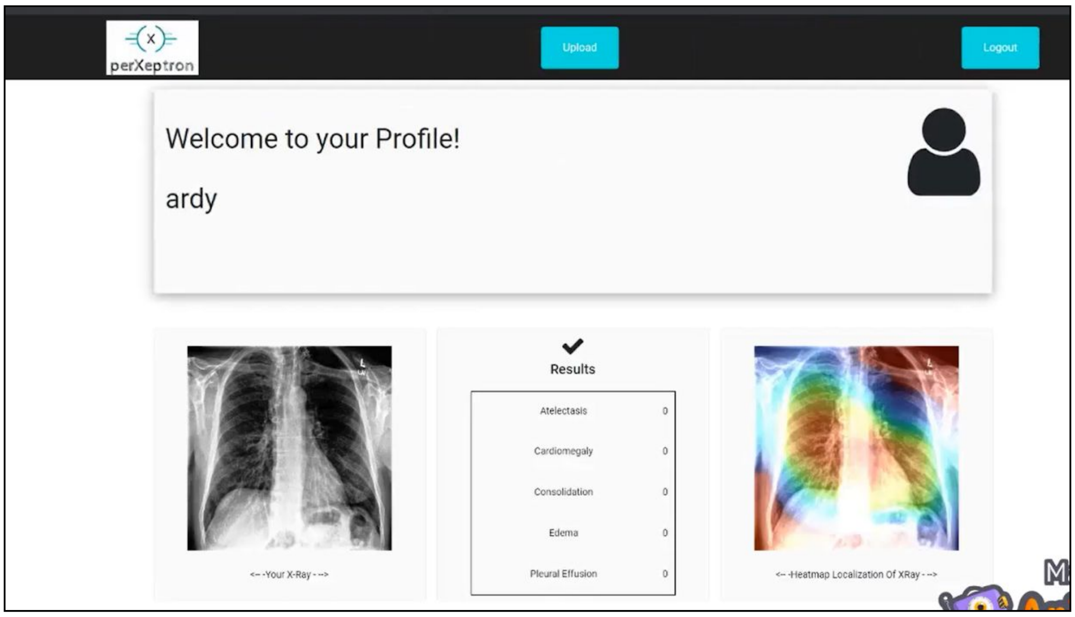
Results page:



Results

Atelectasis	1
Cardiomegaly	1
Consolidation	1
Edema	0
Pleural Effusion	1

Profile page:



Welcome to your Profile!

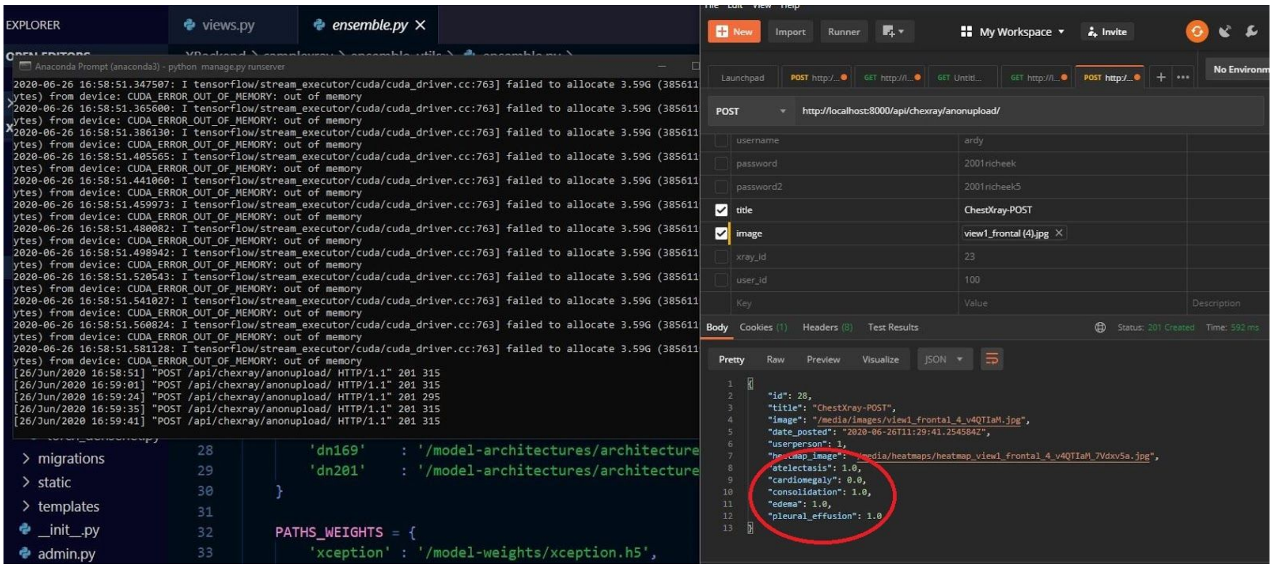
ardy

Results

Atelectasis	0
Cardiomegaly	0
Consolidation	0
Edema	0
Pleural Effusion	0

OPEN-API Example :

You can query the backend server and get predictions:



```
Anaconda Prompt (anaconda3) - python  manage.py runserver
2020-06-26 16:58:51.347507: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.365600: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.386130: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.405565: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.441060: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.459973: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.480082: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.498942: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.520543: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.541027: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.560824: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.581128: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
[26/Jun/2020 16:58:51] "POST /api/chexray/anonupload/ HTTP/1.1" 201 315
[26/Jun/2020 16:59:01] "POST /api/chexray/anonupload/ HTTP/1.1" 201 315
[26/Jun/2020 16:59:24] "POST /api/chexray/anonupload/ HTTP/1.1" 201 295
[26/Jun/2020 16:59:35] "POST /api/chexray/anonupload/ HTTP/1.1" 201 315
[26/Jun/2020 16:59:41] "POST /api/chexray/anonupload/ HTTP/1.1" 201 315
```

```python
        'dn169'     : '/model-architectures/architecture
        'dn201'     : '/model-architectures/architecture
    }

    PATHS_WEIGHTS = {
        'xception' : '/model-weights/xception.h5',
```

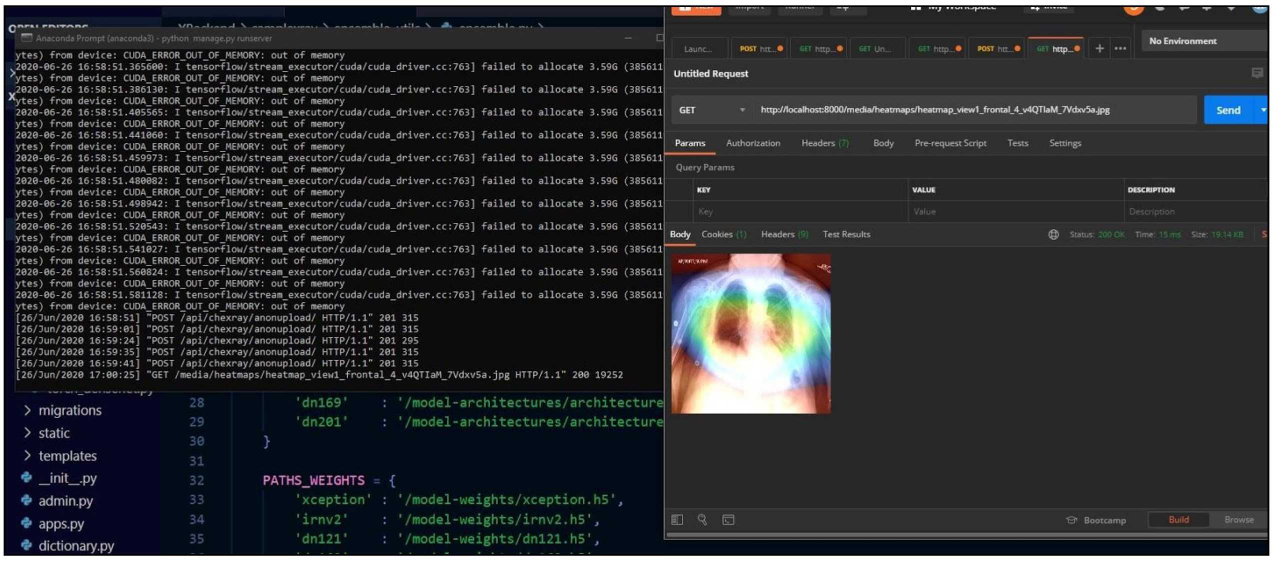
POST http://localhost:8000/api/chexray/anonupload/

	Key	Value	Description
☐	username	ardy	
☐	password	2001richeek	
☐	password2	2001richeek5	
☑	title	ChestXray-POST	
☑	image	view1_frontal (4).jpg	
☐	xray_id	23	
☐	user_id	100	

Status: 201 Created Time: 592 ms

```json
{
    "id": 28,
    "title": "ChestXray-POST",
    "image": "/media/images/view1_frontal_4_v4QTIaM.jpg",
    "date_posted": "2020-06-26T11:29:41.254584Z",
    "userperson": 1,
    "heatmap_image": "/media/heatmaps/heatmap_view1_frontal_4_v4QTIaM_7Vdxv5a.jpg",
    "atelectasis": 1.0,
    "cardiomegaly": 0.0,
    "consolidation": 1.0,
    "edema": 1.0,
    "pleural_effusion": 1.0
}
```

And you also get the CAM Localized Heatmap:



```
Anaconda Prompt (anaconda3) - python  manage.py runserver
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.365600: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.386130: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.405565: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.441060: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.459973: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.480082: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.498942: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.520543: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.541027: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.560824: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
2020-06-26 16:58:51.581128: I tensorflow/stream_executor/cuda/cuda_driver.cc:763] failed to allocate 3.59G (385611
ytes) from device: CUDA_ERROR_OUT_OF_MEMORY: out of memory
[26/Jun/2020 16:58:51] "POST /api/chexray/anonupload/ HTTP/1.1" 201 315
[26/Jun/2020 16:59:01] "POST /api/chexray/anonupload/ HTTP/1.1" 201 315
[26/Jun/2020 16:59:24] "POST /api/chexray/anonupload/ HTTP/1.1" 201 295
[26/Jun/2020 16:59:35] "POST /api/chexray/anonupload/ HTTP/1.1" 201 315
[26/Jun/2020 16:59:41] "POST /api/chexray/anonupload/ HTTP/1.1" 201 315
[26/Jun/2020 17:00:25] "GET /media/heatmaps/heatmap_view1_frontal_4_v4QTIaM_7Vdxv5a.jpg HTTP/1.1" 200 19252
```

```python
        'dn169'     : '/model-architectures/architecture
        'dn201'     : '/model-architectures/architecture
    }

    PATHS_WEIGHTS = {
        'xception' : '/model-weights/xception.h5',
        'irnv2'    : '/model-weights/irnv2.h5',
        'dn121'    : '/model-weights/dn121.h5',
```

Untitled Request

GET http://localhost:8000/media/heatmaps/heatmap_view1_frontal_4_v4QTlaM_7Vdxv5a.jpg

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 15 ms Size: 19.14 KB

Project Video: Team PerXeptron: Video

Final Presentation: Team PerXeptron: Final Presentation

Learning Value

- **General:**
  - Fundamentals of Deep Learning and Computer Vision.
  - Fundamentals of Web development.
  - Best practices while writing code.
  - Experience of building deep learning models on large(~11GB) datasets with limited computational resources.
  - Experience of using Google Colab and Git
  - Continuous Integration and Deployment

- **Transfer Learning:**
  The fundamentals of transfer learning, like when to fine-tune the entire model (Large dataset, but different from the pre-trained model's dataset), when to fine-tune a fixed number of layers (Large dataset and similar to the pre-trained model's dataset), or when to keep the entire base model frozen (Small dataset, but similar to the pre-trained model's dataset).
  Reference: Transfer learning from pre-trained models

- **Keras:**
  Gained familiarity with the different functionalities in Keras like:
  - Callbacks: EarlyStopping, ModelCheckpoint, CyclicLR (https://github.com/bckenstler/CLR/blob/master/clr_callback.py)
  - Different Layers: GlobalAveragePooling2D (https://arxiv.org/pdf/1312.4400.pdf), BatchNormalization (https://arxiv.org/abs/1502.03167), Dropout (https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf)
  - ImageDataGenerators (Multi-label image classification Tutorial with Keras ImageDataGenerator)

- **Image Pre-processing**:
  Gained exposure to a few image pre-processing techniques:
  - **Histogram Equalization:** Increasing the global contrast of an image by "spreading out" the histogram of the intensity values (Histogram Equalization) & (Deep Learning for abnormality detection in Chest X-Ray images).
  - **Center Cropping:** An alternative to resizing images, can be used when most information is contained in the central region of the image and resizing the image may result in distortion of information contained.

- **Model Ensembling:**
  The process of combining (via a **weighted average**) the predictions from different models (In our case: Xception, Inception-ResNet-v2, DenseNet-121, DenseNet-169, DenseNet-201), and thus making the overall prediction more robust. The individual models have an AUC of about **88**, while ensembling increases it to **91.5**, the categorical accuracies of the individual models is about **45%**, while ensembling increases it to **51.7%**.

- **Authentication**
  This project utilises **DRF Token and Session Authentication** for authenticating client side hashes. So, we got to learn a lot about **session** and **local storage** and user retention to deal with passwords and sensitive information on the client side.

Software/Hardware used

Software:

- Python 3.7.6
- Keras
- Django 3.0.7
- Django REST Framework 3.11.0
- Angular 9
- Angular Material
- Google Colab
- Git & GitHub
- OpenCV
- PyTorch

Hardware:

Running the **Backend** Server on :

OS	Windows
GPU	GTX 1060 6GB
CPU	i7 6700HQ ~2.60GHz
RAM	16GB

Suggestions for others

Suggestions regarding the Deep Learning aspect of this project:

- A basic knowledge of machine learning concepts will help you gain better intuitions about various deep learning approaches.
- Before trying on the CheXpert dataset, which is huge, it is recommended that you try building a smaller model on a similar, but smaller dataset (Like this Chest X-Ray Images (Pneumonia)) to gain some understanding about working with CXR images. If you are completely new to Deep Learning for Computer Vision, you should probably first try on smaller, 'Hello world' like problems, like MNIST Digit Classification, or Dogs vs Cats classifier.

Suggestions regarding the Web app aspect of this project:

- Spend enough time to learn the basics of web development like **HTML, CSS** and **JS,** that's where you should start.
- Start with Basic **Django** and **REST Framework projects** and get yourself familiarised with **TypeScript** too (Angular can be pretty confusing).
- Be careful while developing your web app so as to make it responsive on smaller screens- this is needed in today's world where mobiles and tablets are used most frequently.

Contribution by each Team Member

*Team **Leader***: **Richeek Das**
Contributions:

- **Enabled GitHub Collaboration :**
  Created GitHub Organization PerXeptron to enable seamless collaboration between team members, in well-separated different repositories. It was surely a hands-on experience of working in, as well as leading a team efficiently and it **can be taken as one of the most important learning values of this project.**

- **DataSet Preprocessing :**
  Researched a lot of papers and kaggle notebooks to find a robust way of handling the huge amount of **NaN** values in the training dataframe. I also set up the initial Colab Notebooks and devised a quick way to get the datasets into working inside the Colab environment.

- **DenseNet-121 Training :**
  Made modifications in Harshit's notebook to train a DenseNet121 transfer learning model

- **Django Backend :**
  - **Built the entire Django Backend from scratch.**
  - **Built a secure and authenticated Django REST API on top of the Django Backend to expose the Model Ensemble.**
  - **Successfully Integrated the Backend with the HeatMap Generator and 5 Keras Model Ensemble.**
  - **Find the complete API documentation here.**

- **Angular 9 Frontend :**
  - Wrote the entire **TypeScript** code base for the different **Angular Page Components.**
  - Used **Angular Services** to :
    - Manage **Tokens**, in **local storage** of browser.(Will upgrade to **cookies** later)
    - Do **User** and **XRay** functionalities using **GET** and **POST** requests to **Backend REST API.**

- **CAM Localization :**
  - Experimented with a lot of **Class Activation Map** algorithms.
  - Finally chose the basic, tried and tested **CAM Localization** Algorithm in this project.
  - Wrote fast **PyTorch code** for this, to prevent speed compromises on the heatmap generation utility.

Team Member: **Harshit Varma**
Contributions:

- **Model Ensembling**:
  Ensembled the final 5 models. (Notebook link)
  (Namely: Xception, Inception-ResNet-v2, DenseNet-121, DenseNet-169, DenseNet-201).
  Used a **weighted average** for computing the probabilities of presence/absence of diseases. The weights were found empirically. Using the weighted average to make the predictions, the overall average AUC (Area under the ROC curve) value improved to **91.5**, while the current State-Of-The-Art is **94** (https://arxiv.org/pdf/1911.06475.pdf).

  The effect of ensembling:
  Model: Xception, Avg AUC: 0.894, Categorical Accuracy : 0.466
  Model: Inception-ResNet-v2, Avg AUC: 0.885, Categorical Accuracy : 0.449
  Model: DenseNet-121, Avg AUC: 0.884, Categorical Accuracy : 0.41
  Model: DenseNet-169, Avg AUC: 0.89, Categorical Accuracy : 0.432
  Model: DenseNet-201, Avg AUC: 0.876, Categorical Accuracy : 0.462
  Ensemble, Avg AUC: **0.915,** Categorical Accuracy **: 0.517**

- **Getting Disease-wise Optimal Thresholds:**
  For further improving the results, we need to select the thresholds for the model based on the TPR(True Positive Rate) and FPR (False Positive Rate). The thresholds can be optimized by maximizing the **Youden's Index.**
  Reference: Youden's J Statistic for Threshold Determination

- **Image Pre-processing:**
  Tried out image pre-processing techniques to improve the results:

  - **Feature-wise Standard Normalization and Feature-wise Centering:**
    Feature-wise Standard Normalization and Feature-wise Centering are important to ensure that the optimization algorithms converge quickly to the local minimum.
    Given that the training data is very big (~11GB), it is not possible to fit the ImageDataGenerators on this, as Colab's RAM cannot handle all these images at once. Thus, instead of using the entire dataset for calculating the mean and standard deviation, I **randomly selected a sample of images** (0.3% of the training data) and fit the ImageDataGeneretors on this. Since the training set is very big (>200,000 Images) we can safely assume that this sample contains **most of the variations in the data** required to calculate the data required for performing Feature-wise Standard Normalization and Feature-wise Centering.

  - **Histogram Equalization:**
    - As outlined in the below paper, histogram equalization increases the contrast between the 'bone' region (Not important) and the 'lung' region (Important).
    - Reference: Deep Learning for abnormality detection in Chest X-Ray images

- **Center Cropping:**
  There are 2 major reasons why center cropping can help:
  - By default, chest x-ray images contain a lot of seemingly **useless information**, like surrounding bone-structure, etc. These may "misguide" the neural network and affect the results.
  - We resize the original 390x320 images to 224x224 images before feeding it into the network. This essentially "squeezes" the images, which may **distort the affected region**, which in turn may affect the network's predictions. Center cropping won't change the image's aspect ratio, thus won't "distort" useful information.
  - Reference: A Transfer Learning Method for Pneumonia Classification and Visualization

Unfortunately, the last 2 (Center Cropping and Histogram Equalization) did not consistently work across all of the 5 models, thus were not used for obtaining the final results.
Histogram Equalization slightly improved the results for the Xception and the Inception-ResNet-v2 models, but worsened the predictions of the DenseNet models.
Center cropping had very little effect on the results, although nothing can be said conclusively since we are training on <=5 epochs.

- **Model Training and Testing**:
  - Trained and tested the **Xception** and the **Inception-ResNet-v2** models (Pre-trained on the ImageNet weights and fully fine-tuned on the CheXpert Dataset) .
    Tried the **NASNet-Large** model too, but couldn't proceed with training as there is a bug in keras which only accepts input of the shape (331x331), and Colab's RAM gets exhausted in handling images this big.
  - Came up with the **Classifier Architecture** which is used on top of **all the base models used**. This architecture was obtained after experimenting with a lot of different architectures. The following article helped a lot in deciding this architecture.
    Reference: Transfer learning from pre-trained models
  - **Tuned** the hyperparameters (Mainly the **learning rate**) empirically to improve the speed and accuracies.
  - Tried **Cyclic Learning Rates** (https://github.com/bckenstler/CLR/blob/master/clr_callback.py) to prevent loss plateaus during training, although this resulted in negligible change as the models are only getting trained for 5 epochs.

Team Member: **Ankit Kumar Misra**
Contributions:

- **Model Training and Testing:**
  - Trained and tested the **DenseNet-169** and **DenseNet-201** based convolutional neural networks, pretrained on ImageNet weights and all layers fine-tuned using the CheXpert dataset.
  - Tried out a couple of ways of handling missing values (NaNs) and uncertainty labels in the data, and verified that the method outlined in the state-of-the-art paper on this subject does produce the best results.
  - Experimented with image preprocessing, including feature-wise mean and normalization of images.
  - Tried applying cyclic learning rates and histogram equalization on the **DenseNet-169** and **DenseNet-201**, to try to deal with local minimas and plateaus in the loss function, and to improve performance, and learned that they did not work out very well with my models.

- **Other:**
  - Designed **diagrams of model pipelines** and the **final model ensemble** for graphic illustration of our prediction method.
  - Tried implementing Gradient-Weighted Class Activation Mappings (Grad-CAM) in Keras, for localizing areas of the chest X-ray image that suggest the presence of disease, using this GitHub repository for help, but was unable implement some functions of class GradCAM in Keras, so we decided to go with simpler Class Activation Mappings.
  - Worked on the **project description video**, explained a brief summary of the project and also major takeaways for our team.

Team Member: **Prapti Kumar**
Contributions:

- **Easy-to-use Web app:**
  - Used **Angular 9** to create a smooth,interactive web-app for the project.
  - Made the site mobile-**responsive** by using CSS Media-Query and Bootstrap.
  - Designed and Created neat and user-friendly home page, login page,upload page, results page and profile page.
  - Enabled routing between all the components.
  - Tested the web app using Karma test-runner.
  - Tried implementing CircleCI **continuous integrations** on the web app to ensure proper workflow.
  - **Deployed** the web app using GitHub Pages.
- **Other:**
  - Worked on the **project description video**, explained the motivation of the project, created animations and compiled the contents of the video.

References and Citations

Research Papers:

- Pham, H., Le, T., Tran, D., Ngo, D., &; Nguyen, H. (2020, June 12).
  Interpreting chest X-rays via CNNs that exploit hierarchical disease dependencies and uncertainty labels.
  https://arxiv.org/abs/1911.06475

- Tataru, C. (2017).
  Deep Learning for abnormality detection in Chest X-Ray images
  http://cs231n.stanford.edu/reports/2017/pdfs/527.pdf

- Luján-García JE, Yáñez-Márquez C, Villuendas-Rey Y, Camacho-Nieto O.
  A Transfer Learning Method for Pneumonia Classification and Visualization.
  Applied Sciences. 2020; 10(8):2908.
  https://www.mdpi.com/2076-3417/10/8/2908

- B. Zhou, A. Khosla, A. Lapedriza, A. Oliva and A. Torralba
  Learning Deep Features for Discriminative Localization
  2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 2921-2929, doi: 10.1109/CVPR.2016.319.
  https://ieeexplore.ieee.org/document/7780688

GitHub Repositories:

- Code for the Cyclic Learning Rate Class was taken from:
  https://github.com/bckenstler/CLR
- This repository helped a lot with the CAM Localization implementation:
  https://github.com/zoogzog/chexnet

Kaggle Notebooks:

- Used as a reference: https://www.kaggle.com/estebvac/chexpert-keras-base/
- Used as a reference: Youden's J Statistic for Threshold Determination

Medium Articles:

- Image Recognition using Pre Trained Xception Model in 5 steps
- Multi-label image classification Tutorial with Keras ImageDataGenerator
- Transfer learning from pre-trained models
- DenseNet Architectures

Documentations:

- Keras
- TensorFlow
- OpenCV
- Angular
- Django
- Django REST FRAMEWORK

Others:

- Reference Book used: Deep Learning with Python by François Chollet
- Stack Overflow

Disclaimer

Licenses