National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Ankit Rungta
Student ID: 19241607

School of Computing
National College of Ireland

Supervisor: Dr. Rashmi Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ankit Rungta
Student ID:	19241607
Programme:	Data Analytics
Year:	2020-21
Module:	MSc Research Project
Supervisor:	Dr. Rashmi Gupta
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	952
Page Count:	27

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Ankit Rungta
Date:	13th August 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ankit Rungta
19241607

1 Introduction

This configuration manual summarizes the hardware and software requirements for reproducing the research. This manual will aid in comprehending the coding processes required to replicate this study, from environment setup to viewing the model findings. A step-by-step tutorial is included below, separated into several sections for convenience.

2 Hardware Requirement

The project was carried out on a Dell G3 15 laptop using the configuration shown in Figure 1 and Figure 2.

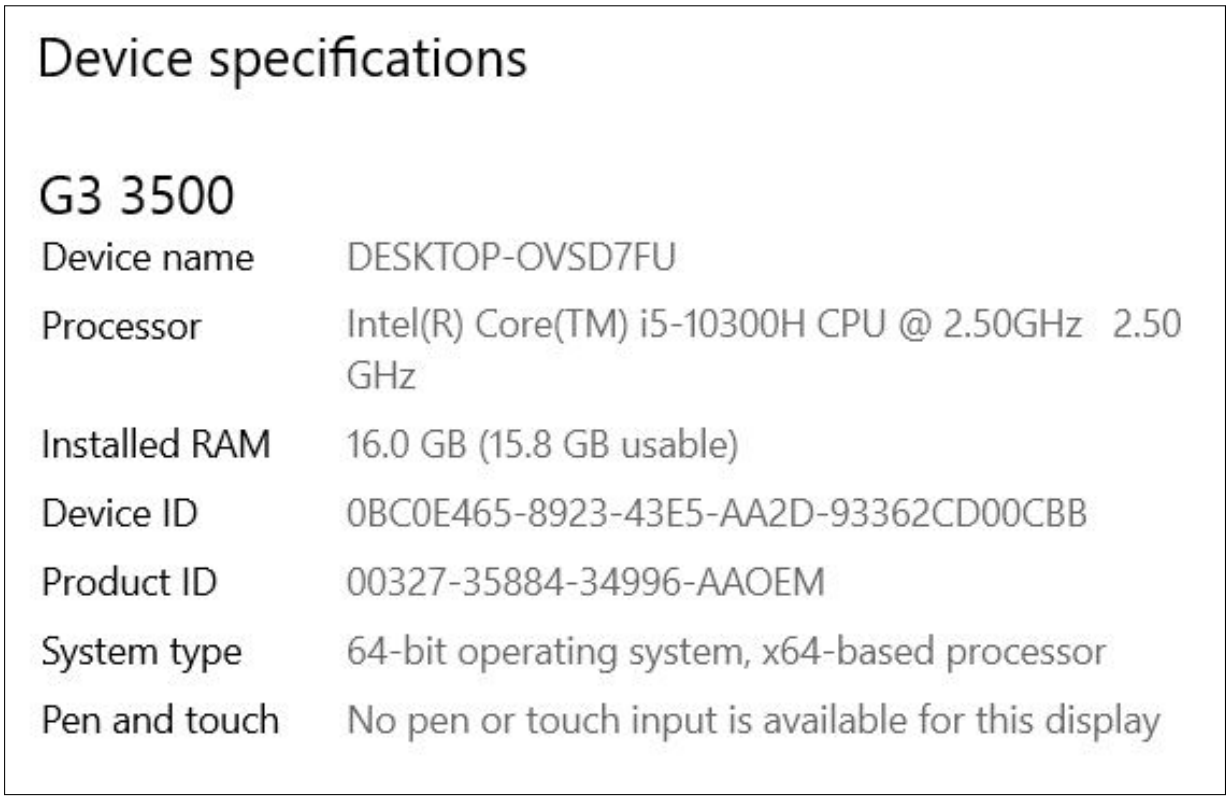

Figure 1: Device Specifications

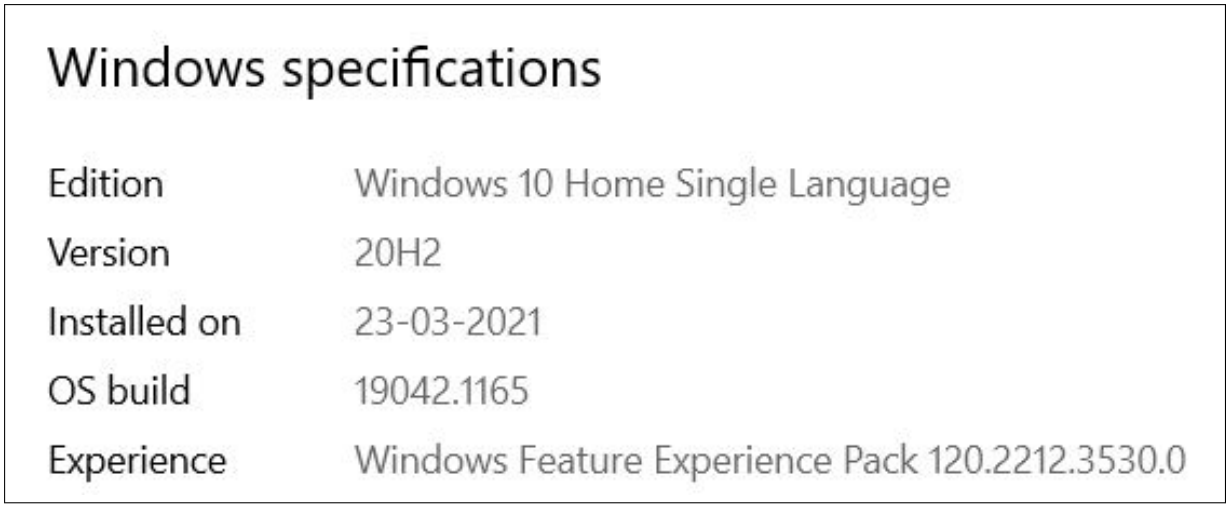

Figure 2: Windows Specifications

3 GPU Configuration

The research was conducted utilizing an Nvidia GeForce GTX 1650 configured as shown in Figure 3.

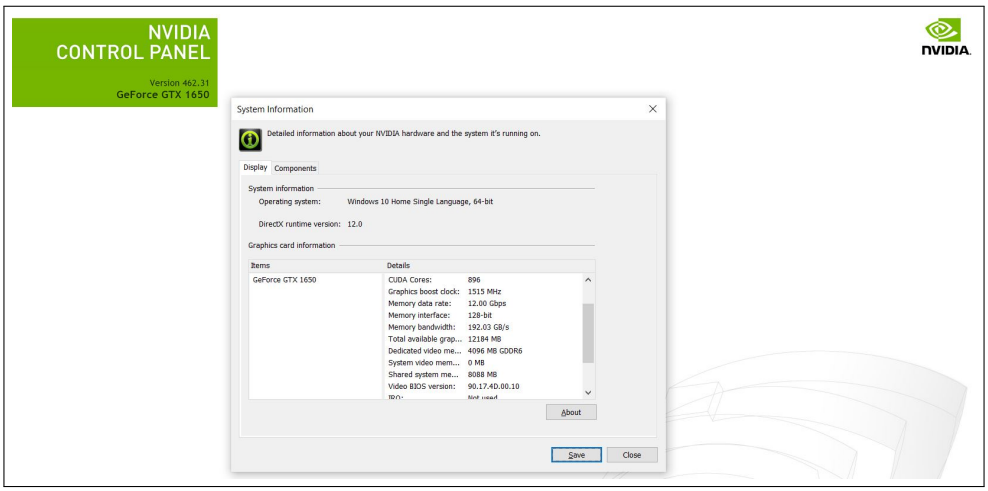

Figure 3: GPU Configuration

4 Software Requirement

The software programs listed in Table 1 were utilized to complete the project.

5 Software Environment Setup

The project was implemented using Python. This was accomplished through the use of the Anaconda development environment, whose dashboard is seen in Figure 4. Anaconda is a suite of programming, debugging, visualization, and data mining tools. In our project, we used Jupyter Notebook for code development and testing.

Name	Version
Anaconda Navigator	1.9.12
Jupyter Notebook	6.0.3
CMD.exe prompt	0.1.1
Python	3.9
Google Chrome	92.0
Overleaf	N/A
Google Colaboratory	N/A

Table 1: Software requirement for this project

5.1 Anaconda Installation

Anaconda was installed using the instructions found at `http://www.Anaconda.com/downloads`. Following successful installation, the dashboard will appear as illustrated in Figure 4.

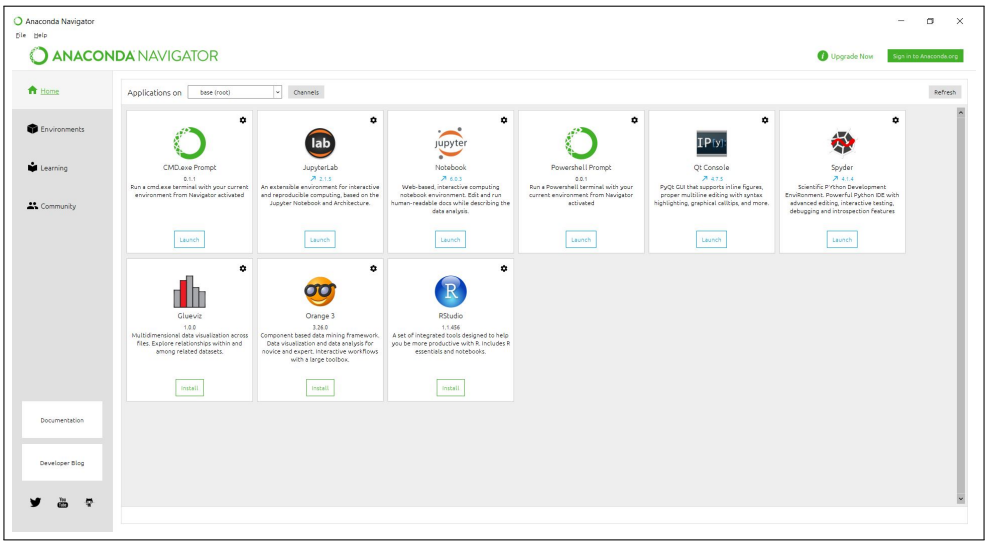

Figure 4: Anaconda Dashboard

5.2 Juypter Installation

Launch the "Jupyter" notebook from the anaconda dashboard, as highlighted in Figure 5.

5.3 Tensorflow Installation

In Anaconda, perform the following: Install the tensorflow environment by utilizing the CMD.EXE prompt in the Navigator window by using the instructions from the Anaconda tensorflow website[1].

[1] `https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/`

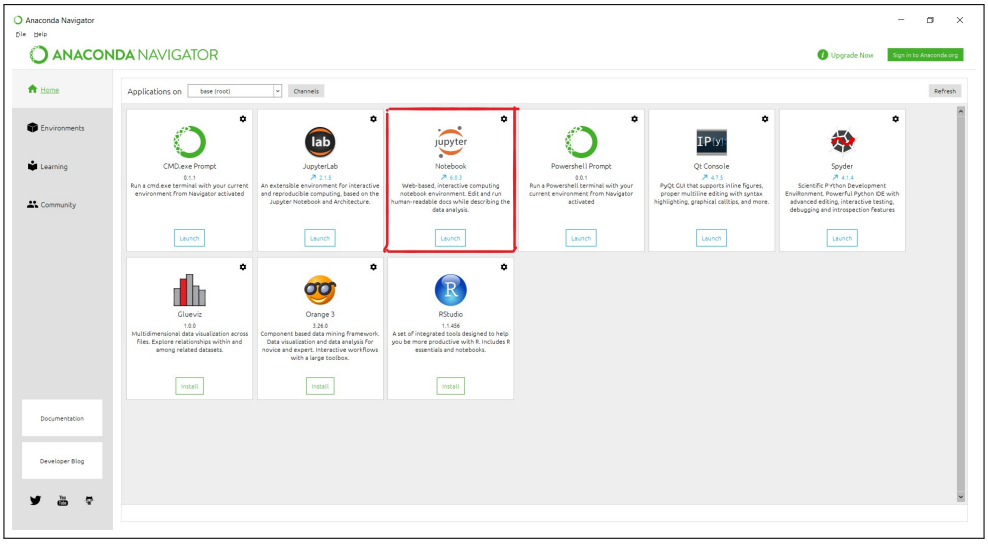

Figure 5: Jupyter Notebook in Anaconda Dashboard

5.4 Google Colab

The novel model is trained using Google colab. It is a completely free cloud service. Colab, like a jupyter notebook, requires an initial drive connection through a folder directory. The runtime may be changed by choosing between a GPU and a TPU.

6 Dataset Acquisition

The dataset is available for download from the Kaggle public repository [2]. The training dataset has 30.1K chest radiograph images, whereas the test dataset contains 3582. Along with the images, a CSV file is included that contains binary labels, image IDs, and patient IDs. This CSV file contains information on the normal, abnormal, and borderline tubes and lines on chest radiograph. Additionally, a CSV file with data annotations for the lines and tubes on chest radiographs is included to assist anyone outside the medical sector in tracing them on chest radiographs.

7 Base Configuration and Data Preparation

The following code listing illustrates the research project's base configurations.

```
1
2  class BaseConfig:
3
4      seed = 44
5      num_classes = 11
6      class_list = [0,1,2,3,4,5,6,7,8,9,10]
7      batch_size = 32
8      n_epochs = 15
9      drop_rate = 0.4
10     scheduler_params = {
11                         'ReduceLROnPlateau':
12                             {
```

[2] https://www.kaggle.com/c/ranzcr-clip-catheter-line-classification/data

```
                            'monitor':'val_loss',
                            'mode':'max',
                            'factor':0.5,
                            'patience':2,
                            'threshold':0.0001,
                            'threshold_mode':'rel',
                            'cooldown':0,
                            'min_lr':1e-5,
                            'eps':1e-08,
                            'verbose':True
                        },
                }

    # optimizer
    optimizer = tf.keras.optimizers.Adam(lr = 1e-4)

    # criterion
    loss = 'binary_crossentropy'

    target_size_dim = 300

    metrics = tf.keras.metrics.AUC(multi_label=True)

    reduceLROnPlat = ReduceLROnPlateau(monitor='val_loss', factor=0.8,
                                       patience=2, verbose=1,
                                       mode='auto',
                                       epsilon=0.0001,
                                       cooldown=5, min_lr=0.00001)

    checkpoint = ModelCheckpoint('best_model.hdf5',
                            monitor= 'val_loss',
                            verbose=1,
                            save_best_only=True,
                            mode= 'min',
                            save_weights_only = False)

    checkpoint_last = ModelCheckpoint('last_model.hdf5',
                            monitor= 'val_loss',
                            verbose=1,
                            save_best_only=False,
                            mode= 'min',
                            save_weights_only = False)


    early = EarlyStopping(monitor= 'val_loss',
                      mode= 'min',
                      patience=5)

    labels = [
                'ETT - Abnormal',
                'ETT - Borderline',
                'ETT - Normal',
                'NGT - Abnormal',
                'NGT - Borderline',
                'NGT - Incompletely Imaged',
                'NGT - Normal',
                'CVC - Abnormal',
                'CVC - Borderline',
```

```
                'CVC - Normal',
                'Swan Ganz Catheter Present'
            ]

    paths = {
                'train_path':'D:/Data Analytics/Semester 3/Research/
    Research_Data/Train-256',
                'test_path': 'D:/Data Analytics/Semester 3/Research/
    Research_Data/test',
                'csv_path': 'D:/Data Analytics/Semester 3/Research/
    Research_Data/train.csv',
                'model_weight_path_folder': 'D:/Data Analytics/Semester
     3/Research/Research_Data/tfkerasefficientnetimagenetnotop',
                'train_annotations':'D:/Data Analytics/Semester 3/
    Research/Research_Data/train_annotations.csv'
            }
```

Listing 1: Base Configuration

The CSV file having the binary labels, patient IDs and Image Id was loaded. Figure 6 shows the created dataframe from the CSV file.

```
df = pd.read_csv(config.paths['csv_path'])
df.head()
```

Listing 2: Data Preparation

	StudyInstanceUID	ETT - Abnormal	ETT - Borderline	ETT - Normal	NGT - Abnormal	NGT - Borderline	NGT - Incompletely Imaged	NGT - Normal	CVC - Abnormal	CVC - Borderline	CVC - Normal	Swan Ganz Catheter Present	PatientID
43.8.498.26697628953273228189...		0	0	0	0	0	0	1	0	0	0	0	ec89415d1
43.8.498.46302891597398758759...		0	0	1	0	0	1	0	0	0	1	0	bf4c6da3c
43.8.498.23819260719748494858...		0	0	0	0	0	0	0	0	1	0	0	3fc1c97e5
43.8.498.68286643202323212801...		0	0	0	0	0	0	0	1	0	0	0	c31019814
43.8.498.10050203009225938259...		0	0	0	0	0	0	0	0	0	1	0	207685cd1

Figure 6: CSV file having binary Labels

8 Data Loader

The dataframe was divided into sections for training and validation.

tf.data.Dataset.from_tensor_slices creates a dataset with items that are slices of the tensors.

Slicing the provided tensors along their first dimension. This procedure maintains the structure of the input tensors by removing the first dimension from each tensor and replacing it with the dataset dimension. All input tensors must have the same first-dimensional size. Figure 7 shows the generated data.

```
df['path'] = 'D:/Data Analytics/Semester 3/Research/Research_Data/train
    /' + df['StudyInstanceUID']+'.jpg'

X_train, X_valid = train_test_split(df, test_size = 0.1, random_state=
    config.seed, shuffle=True)

Train_df = tf.data.Dataset.from_tensor_slices((X_train.path.values,
    X_train[config.labels].values))
```

```
6
7 Valid_df = tf.data.Dataset.from_tensor_slices((X_valid.path.values,
    X_valid[config.labels].values))
```

Listing 3: Data Loader

```
Path: b'D:/Data Analytics/Semester 3/Research/Research_Data/train/1.2.826.0.1.3680043.8.498.774509870240338813923802440920931277
658.jpg', Label: [0 0 0 0 0 0 0 0 1 0 0]
Path: b'D:/Data Analytics/Semester 3/Research/Research_Data/train/1.2.826.0.1.3680043.8.498.926576837605271565087491015778290766
177.jpg', Label: [0 0 0 0 0 0 0 0 1 0 0]
Path: b'D:/Data Analytics/Semester 3/Research/Research_Data/train/1.2.826.0.1.3680043.8.498.934860616571746689230972147382309744
627.jpg', Label: [0 0 0 0 0 0 0 0 0 1 0]
Path: b'D:/Data Analytics/Semester 3/Research/Research_Data/train/1.2.826.0.1.3680043.8.498.977925946576357731409676565886436833
153.jpg', Label: [0 0 0 0 0 0 0 0 1 0 0]
Path: b'D:/Data Analytics/Semester 3/Research/Research_Data/train/1.2.826.0.1.3680043.8.498.102246018216959470154972847943630057
67.jpg', Label: [0 0 0 0 0 0 0 0 0 1 0]
```

Figure 7: Loaded Data with the path to the training Image

9 Data Generator

The training and validation data was generated using the below code snippet. Figure 8 shows the generated data.

```
1  def process_data_train(image_path, label):
2
3      img = tf.io.read_file(image_path)
4      img = tf.image.decode_jpeg(img, channels=3)
5      img = tf.image.random_brightness(img, 0.3)
6      img = tf.image.random_flip_left_right(img)
7      img = tf.image.resize(img, [config.target_size_dim, config.
       target_size_dim])
8      return img, label
9
10 def process_data_valid(image_path, label):
11     img = tf.io.read_file(image_path)
12     img = tf.image.decode_jpeg(img, channels=3)
13     img = tf.image.resize(img, [config.target_size_dim, config.
       target_size_dim])
14     return img, label
15
16 Train_df = Train_df.map(process_data_train, num_parallel_calls=tf.data.
       experimental.AUTOTUNE)
17 Valid_df = Valid_df.map(process_data_valid, num_parallel_calls=tf.data.
       experimental.AUTOTUNE)
18
19 for image, label in Train_df.take(1):
20     plt.imshow(image.numpy().astype('uint8'))
21     plt.show()
22     print("Image shape: ", image.numpy().shape)
23     print("Label: ", config.labels[np.argmax(label.numpy())])
```

Listing 4: Data Generator

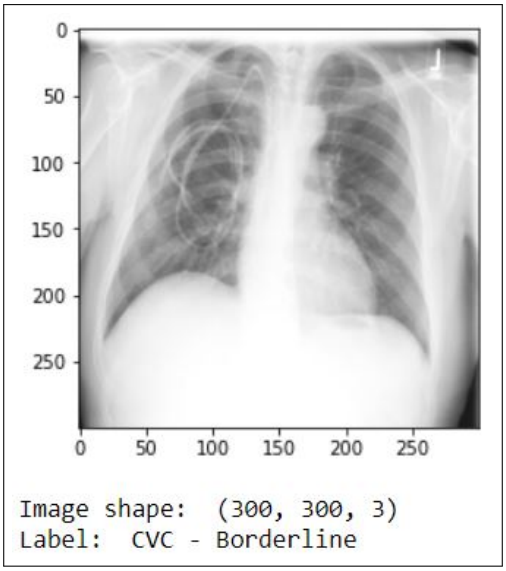

Figure 8: Generated Data

10 Exploratory Data Analysis

After loading the data, few EDA (Exploratory Data Analysis) were performed to better understand the data. Figure 9 shows the distribution of labels.

```
df_EDA = pd.read_csv(config.paths['csv_path'], index_col=0)
df_EDA.head()

df_tmp = df_EDA.iloc[:, :-1].sum()
df_tmp = df_tmp.sort_values(ascending=True)

fig = px.bar(x=df_tmp.values, y=df_tmp.index)
fig.update_layout(
        title = {"text": "Distribution of Labels", "font_size" :18, "x"
    : 0.5},
        xaxis_title="Count",
        yaxis_title="Label",
    )
fig.update_traces(
    marker_color=px.colors.qualitative.Prism
)
fig.show()
```

Listing 5: Distribution of labels

Figure 10 shows the count of each label present in the dataset.

```
def show_values_on_bars(axs):
    def _show_on_single_plot(ax):
        for p in ax.patches:
            _x = p.get_x() + p.get_width() / 2
            _y = p.get_y() + p.get_height()
            value = '{:.0f}'.format(p.get_height())
            ax.text(_x, _y, value, ha="center")

    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_single_plot(ax)
```

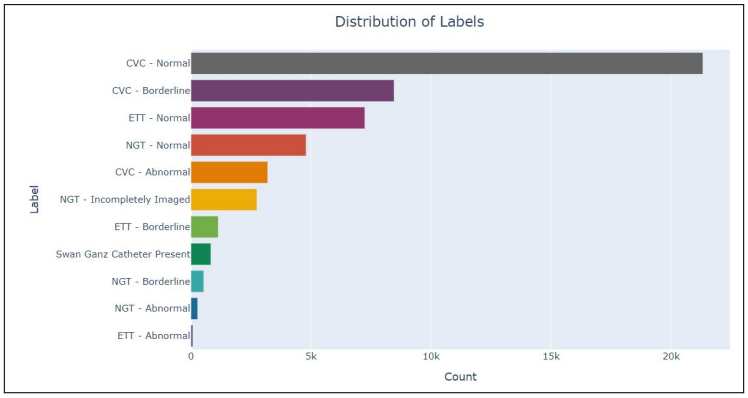

Figure 9: Distribution of Labels

```
    else:
        _show_on_single_plot(axs)

f, axes = plt.subplots(3,3,figsize=(14,14))

_palette = "tab10"

# CVC

sns.countplot(df_EDA['CVC - Normal'], ax = axes[0,0], palette=_palette)
axes[0,0].set_xlabel('CVC - Normal', fontsize=14)
axes[0,0].set_ylabel('Count', fontsize=14)
axes[0,0].yaxis.tick_left()

sns.countplot(df_EDA['CVC - Borderline'], ax = axes[0,1], palette=
    _palette)
axes[0,1].set_xlabel('CVC - Borderline', fontsize=14)
axes[0,1].set_ylabel('', fontsize=14)
axes[0,1].yaxis.set_label_position("right")
axes[0,1].yaxis.tick_left()

sns.countplot(df_EDA['CVC - Abnormal'], ax = axes[0,2], palette=
    _palette)
axes[0,2].set_xlabel('CVC - Abnormal', fontsize=14)
axes[0,2].set_ylabel('', fontsize=14)
axes[0,2].yaxis.set_label_position("right")
axes[0,2].yaxis.tick_left()

# ETT

sns.countplot(df_EDA['ETT - Normal'], ax = axes[1,0], palette=_palette)
axes[1,0].set_xlabel('ETT - Normal', fontsize=14)
axes[1,0].set_ylabel('Count', fontsize=14)
axes[1,0].yaxis.tick_left()

sns.countplot(df_EDA['ETT - Borderline'], ax = axes[1,1], palette=
    _palette)
axes[1,1].set_xlabel('ETT - Borderline', fontsize=14)
axes[1,1].set_ylabel('', fontsize=14)
axes[1,1].yaxis.set_label_position("right")
axes[1,1].yaxis.tick_left()

```

```
sns.countplot(df_EDA['ETT - Abnormal'], ax = axes[1,2], palette=
    _palette)
axes[1,2].set_xlabel('ETT - Abnormal', fontsize=14)
axes[1,2].set_ylabel('', fontsize=14)
axes[1,2].yaxis.set_label_position("right")
axes[1,2].yaxis.tick_left()

show_values_on_bars(axes)

# NGT

sns.countplot(df_EDA['NGT - Normal'], ax = axes[2,0], palette=_palette)
axes[2,0].set_xlabel('NGT - Normal', fontsize=14)
axes[2,0].set_ylabel('Count', fontsize=14)
axes[2,0].yaxis.tick_left()

sns.countplot(df_EDA['NGT - Borderline'], ax = axes[2,1], palette=
    _palette)
axes[2,1].set_xlabel('NGT - Borderline', fontsize=14)
axes[2,1].set_ylabel('', fontsize=14)
axes[2,1].yaxis.set_label_position("right")
axes[2,1].yaxis.tick_left()

sns.countplot(df_EDA['NGT - Abnormal'], ax = axes[2,2], palette=
    _palette)
axes[2,2].set_xlabel('NGT - Abnormal', fontsize=14)
axes[2,2].set_ylabel('', fontsize=14)
axes[2,2].yaxis.set_label_position("right")
axes[2,2].yaxis.tick_left()

show_values_on_bars(axes)

plt.show()
```

Listing 6: Count of labels

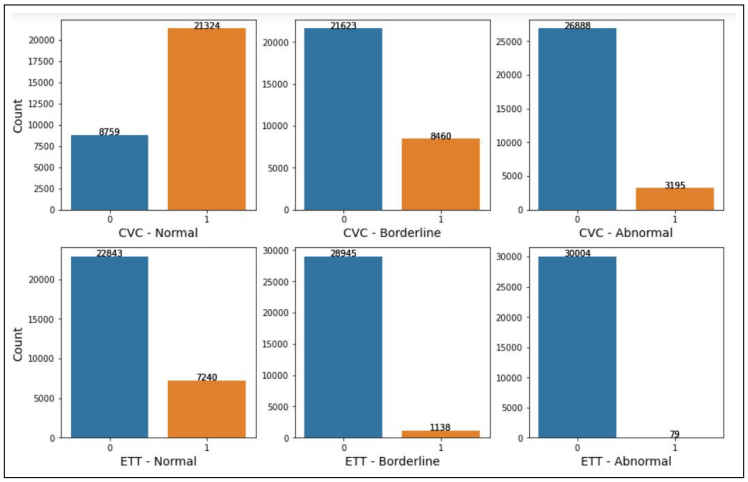

Figure 10: Count of Labels

Figure 11 shows the annotated data points to help non-medical people to trace the tubes and lines on chest radiographs.

```
def plot_image_alongwith_annotations(row_ind):
```

```
    row = df_annotations.iloc[row_ind]
    image_path = 'D:/Data Analytics/Semester 3/Research/Research_Data/train/' +row['StudyInstanceUID'] +'.jpg'

    label = row["label"]
    data = np.array(ast.literal_eval(row["data"]))

    plt.figure(figsize=(16, 5))
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.subplot(1, 2, 2)
    plt.imshow(image)
    plt.scatter(data[:, 0], data[:, 1])
    plt.suptitle(label, fontsize=15)
```

Listing 7: Data Annotation

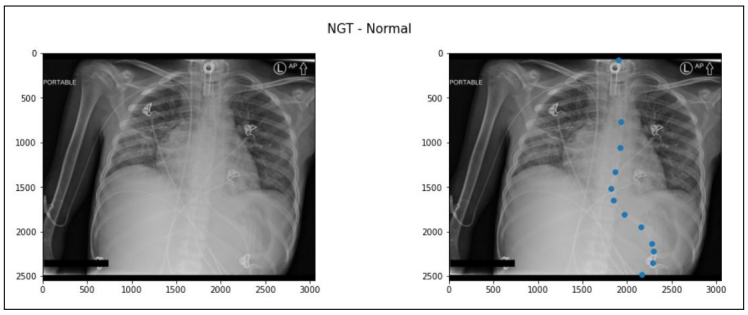

Figure 11: Annotated Data on chest Radiograph

Figure 12 shows the distribution of CVC normal label.

```
def visualize_chest_image_batch(image_ids):
    plt.figure(figsize=(16, 10))

    for ind, image_id in enumerate(image_ids):
        plt.subplot(2, 3, ind + 1)
        image = cv2.imread('D:/Data Analytics/Semester 3/Research/Research_Data/train/'+ f"{image_id}.jpg")
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        plt.imshow(image)
        plt.axis("off")

    plt.show()

def plot_statistics(df, col):
    plt.figure(figsize=(16, 2))
    sns.countplot(y=df[col])

    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.xlabel("Number of observations", fontsize=15)
    plt.ylabel(col, fontsize=15)
    plt.title(f"Distribution of {col}", fontsize=16);

    plt.show()
```

```
25
26 def process_class(col_name):
27     plot_statistics(df_EDA, col_name)
28     tmp_df = df_EDA[df_EDA[col_name] == 1]
29     visualize_chest_image_batch(random.sample(tmp_df.index.tolist(), 6)
       )
30
31 process_class("CVC - Normal")
```

Listing 8: Distribution of labels

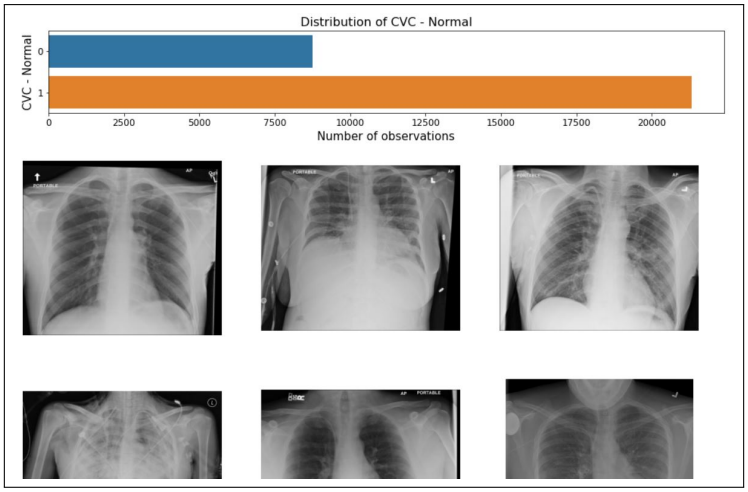

Figure 12: Distribution of CVC normal label

11 Data Pre-Processing

Data augmentation is done using the albumentation library and CLAHE is used to increase the contrast of the image.

11.1 Data Augmentation

The data is augmented to make the model more accurate. Figure 13 shows the augmented images.

```
1 def configure_for_performance(ds, batch_size = config.batch_size):
2
3     ds = ds.cache('D:/Data Analytics/Semester 3/Research/Research_Data/
      dump.tfcache')
4     ds = ds.repeat()
5     ds = ds.shuffle(buffer_size=1024)
6     ds = ds.batch(batch_size)
7     ds = ds.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
8
9     return ds
10
11 train_ds_batch = configure_for_performance(Train_df)
12 valid_ds_batch = Valid_df.batch(config.batch_size*2)
13
14 image_batch, label_batch = next(iter(train_ds_batch))
```

```
15
16 data_augmentation = keras.Sequential(
17     [
18         tf.keras.layers.experimental.preprocessing.RandomRotation(0.05,
       interpolation='nearest'),
19     ]
20 )
21
22 plt.figure(figsize=(10, 10))
23
24 for i in range(16):
25
26     augmented_images = data_augmentation(image_batch)
27     ax = plt.subplot(4, 4, i + 1)
28     plt.imshow(augmented_images[i].numpy().astype("uint8"))
29     label = config.labels[np.argmax(label_batch[i].numpy())]
30     plt.title(label)
31     plt.axis("off")
```

Listing 9: Data Augmentation

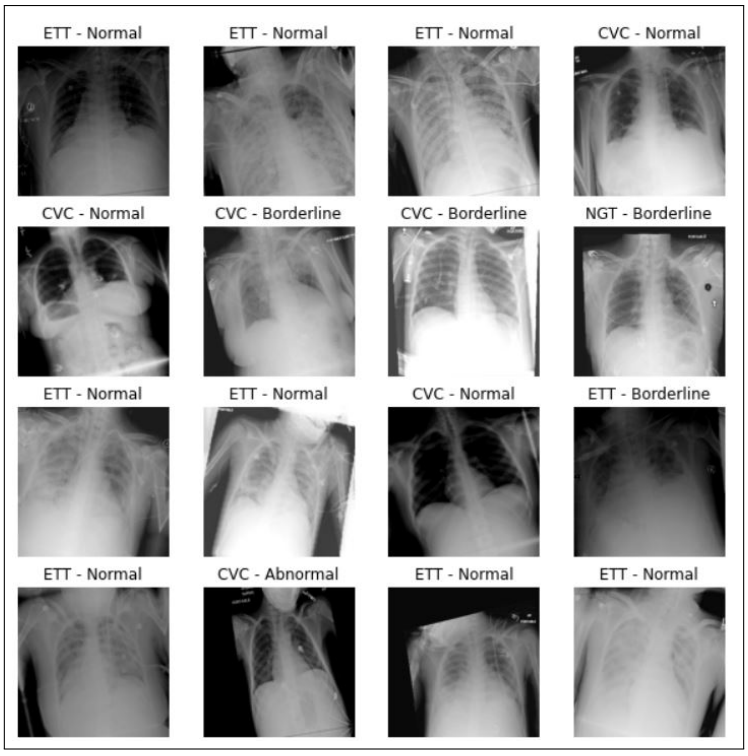

Figure 13: Augmented Data

11.2 CLAHE

The data is went through CLAHE to increase the contrast of the image. Figure 14 shows the augmented images.

```
1 class CLAHE_DATASET(Dataset):
2     def __init__(self, files_folder_path, df, num_channels , transfroms
      = None ):
3         self.files_folder_path = files_folder_path
```

```
        self.df = df
        self.transforms = transfroms
        self.num_channels = num_channels

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):

        image_id = self.df.StudyInstanceUID.values[idx]
        image = cv2.imread(os.path.join(self.files_folder_path, image_id + ".jpg" ))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


        if self.transforms:
            image = self.transforms(image=image)['image']
        labels = self.df[self.df.StudyInstanceUID == image_id].values.tolist()[0][1:-1]
        labels = torch.tensor(labels,dtype= torch.float32) #.view(1,-1)

        return image, labels

def Show_Xrays(augmentation):
    num_channels = 3

    trainset = CLAHE_DATASET('D:/Data Analytics/Semester 3/Research/Research_Data/train/',  df_CLAHE, num_channels , augmentation )

    trainloader = DataLoader(trainset, batch_size = 9 , num_workers = 3 , shuffle = False)

    fig = plt.figure()
    fig.set_size_inches(25, 25)
    #fig.savefig('test2png.png', dpi=100)

    for batch_i, (data, target) in tqdm(enumerate(trainloader)):
        #print(data.shape)
        if batch_i == 1:
            break
        for i in range(data.shape[0]):
            ax = plt.subplot(3,3, i+1)
            plt.tight_layout()
            ax.axis('off')
            plt.imshow(data[i])

#Taking a lot of time to run.
train_augs = albumentations.Compose([   albumentations.RandomResizedCrop(img_size, img_size, scale=(0.9, 1), p=1),
                                        albumentations.CLAHE(clip_limit=(1,4), p= 1),
                                    ]a)
Show_Xrays(train_augs)
```

Listing 10: CLAHE

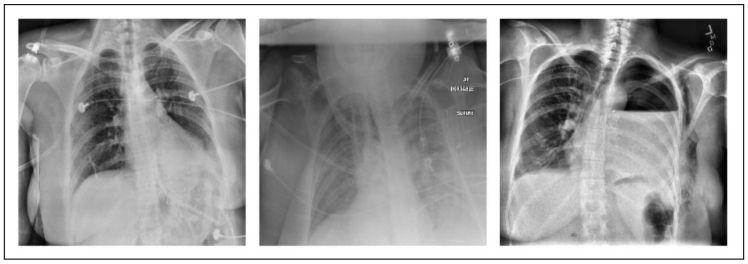

Figure 14: CLAHE

12 Model Building

Machine learning models (Baseline and Novel) are built to determine whether or not the tubes and lines on the chest radiographs are accurately found. The baseline technique use EfficientNet, a transfer learning model, whereas the novel approach utilizes EfficientNet in conjunction with CBAM (Convolutional Block Attention Module) (Woo et al.; 2018) to divide the features across channel and spatial axes[3].

12.1 Baseline Approach- EfficientNet

EfficientNet is used as the baseline approach to create a machine learning model to detect the malpositioned catheters and tubes on chest radiographs. Figure 15 shows the baseline model summary.

```
def load_pretrained_model(weights_path, drop_connect, target_size_dim,
    layers_to_unfreeze=5):

    model = EfficientNetB3(
            weights=None,
            include_top=False,
            drop_connect_rate=0.4
        )

    model.load_weights(weights_path)

    model.trainable = True

    return model

def build_my_model(base_model, optimizer, metrics, loss):

    inputs = tf.keras.layers.Input(shape=(config.target_size_dim,
    config.target_size_dim, 3))
    x = data_augmentation(inputs)
    outputs_eff = base_model(x)
    global_avg_pooling = GlobalAveragePooling2D()(outputs_eff)
    dense_1= Dense(256)(global_avg_pooling)
    bn_1 = BatchNormalization()(dense_1)
    activation = Activation('relu')(bn_1)
    dropout = Dropout(0.3)(activation)
    dense_2 = Dense(len(config.labels), activation='sigmoid')(dropout)

    my_model = tf.keras.Model(inputs, dense_2)
```

[3]https://gist.github.com/innat/99888fa8065ecbf3ae2b297e5c10db70

```
28
29      my_model.compile(
30          optimizer=config.optimizer,
31          loss=config.loss,
32          metrics=config.metrics
33      )
34
35      return my_model
36
37  model_weights_path = 'D:/Data Analytics/Semester 3/Research/
        Research_Data/tfkerasefficientnetimagenetnotop/efficientnetb3_notop.
        h5'
38
39  base_model = load_pretrained_model(model_weights_path, config.drop_rate
        , config.target_size_dim)
40
41  my_model = build_my_model(base_model, config.optimizer, metrics = [
        config.metrics], loss=config.loss)
42
43  my_model.summary()
44
45  #Model Training
46  steps_per_epoch = len(X_train) // config.batch_size
47
48  history = my_model.fit(
49                              train_ds_batch,
50                              validation_data = valid_ds_batch,
51                              epochs = config.n_epochs,
52                              callbacks = callbacks_list,
53                              steps_per_epoch = steps_per_epoch
54                          )
```

Listing 11: Baseline Model (EfficientNet)

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 300, 300, 3)]     0
_________________________________________________________________
sequential (Sequential)      (None, 300, 300, 3)       0
_________________________________________________________________
efficientnetb3 (Functional)  (None, None, None, 1536)  10783535
_________________________________________________________________
global_average_pooling2d (Gl (None, 1536)              0
_________________________________________________________________
dense (Dense)                (None, 256)               393472
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 11)                2827
=================================================================
Total params: 11,180,858
Trainable params: 11,093,043
Non-trainable params: 87,815
```

Figure 15: Baseline Model Summary

12.2 Novel Approach- EfficientNet + CBAM

EfficientNet along with CBAM is used as the novel approach to create a machine learning model to detect the malpositioned catheters and tubes on chest radiographs. A class is created from which various parameters for various folds can be managed and modified. The data_Train_Config class allows for the configuration of image_size, nets per fold, and so on. I've utilized the EfficientNet model in this case. Figure 16 shows the novel model layers.

```

#RANZCR-CliP Dataloader
def decoder_Path(with_labels=True, target_size=(300, 300), ext='jpg'):
    def decode(path):
        file_bytes = tf.io.read_file(path)
        if ext == 'png':
            img = tf.image.decode_png(file_bytes, channels=3)
        elif ext in ['jpg', 'jpeg']:
            img = tf.image.decode_jpeg(file_bytes, channels=3)
        else:
            raise ValueError("Image extension not supported")

        img = tf.cast(img, tf.float32) / 255.0
        img = tf.image.resize(img, target_size)

        return img

    def decode_with_image_labels(path, label):
        return decode(path), label

    return decode_with_image_labels if with_labels else decode


def image_augment(with_labels=True):
    def augment(img):
        img = tf.image.random_flip_left_right(img)
        img = tf.image.random_flip_up_down(img)
        return img

    def augment_with_image_labels(img, label):
        return augment(img), label

    return augment_with_image_labels if with_labels else augment


def dataset_creation(paths, labels=None, bsize=32, cache=True,
                     decode_fn=None, augment_fn=None,
                     augment=True, repeat=True, shuffle=1024,
                     cache_dir=""):
    if cache_dir != "" and cache is True:
        os.makedirs(cache_dir, exist_ok=True)

    if decode_fn is None:
        decode_fn = decoder_Path(labels is not None)

    if augment_fn is None:
        augment_fn = image_augment(labels is not None)

    AUTO = tf.data.experimental.AUTOTUNE
```

```
    slices = paths if labels is None else (paths, labels)

    dset = tf.data.Dataset.from_tensor_slices(slices)
    dset = dset.map(decode_fn, num_parallel_calls=AUTO)
    dset = dset.cache(cache_dir) if cache else dset
    dset = dset.map(augment_fn, num_parallel_calls=AUTO) if augment
   else dset
    dset = dset.repeat() if repeat else dset
    dset = dset.shuffle(shuffle) if shuffle else dset
    dset = dset.batch(bsize).prefetch(AUTO)

    return dset

#Training Parameters
class data_Train_Config(BaseConfig):
    EPOCH        = 6
    FOLDS        = 5
    TTA          = 5
    VERBOSITY    = 0
    WORKERS      = 2
    LABEL_SMOOTH = 0.0
    MULTIPROCESS = False  # Didn't work as expected

    # Learning Rate for each fold
    LR_RATE = {
        '0' : 1e-3,  # learning rate in fold 0
        '1' : 1e-3,  # learning rate in fold 1
        '2' : 1e-4,  # learning rate in fold 2
        '3' : 1e-4,  # learning rate in fold 3
        '4' : 1e-4   # learning rate in fold 4
    }

    # Batch Size for each fold
    BATCH_SIZE = {
        '0' : 32,  # batch size in fold 0
        '1' : 128,  # batch size in fold 1
        '2' : 86,  # batch size in fold 2
        '3' : 128,  # batch size in fold 3
        '4' : 128   # batch size in fold 4
    }

    IMG_SIZE = {
        '0': 850, # size of the image in fold 0
        '1': 240, # size of the image in fold 1
        '2': 260, # size of the image in fold 2
        '3': 224, # size of the image in fold 3
        '4': 240  # size of the image in fold 4
    }


    BASE_NETS = {
        '0' : [efn.EfficientNetB0,
               'noisy-student'],
        '1' : [efn.EfficientNetB1,
               'noisy-student'],
        '2' : [efn.EfficientNetB2,
               'imagenet'],
        '3' : [efn.EfficientNetB0,
```

```
                'imagenet'],
        '4' : [efn.EfficientNetB1,
                'imagenet'],
    }

#Channel and Spatial Attention Classes
class Spatial_Attention(tf.keras.layers.Layer):
    def __init__(self, kernel_size=3):
        '''
        paper: https://arxiv.org/abs/1807.06521
        code: https://gist.github.com/innat/99888
    fa8065ecbf3ae2b297e5c10db70
        '''
        super(Spatial_Attention, self).__init__()
        self.conv1 = tf.keras.layers.Conv2D(64, kernel_size=kernel_size
    ,
                                            use_bias=False,
                                            kernel_initializer='
    he_normal',
                                            strides=1, padding='same',
                                            activation=tf.nn.relu6)
        self.conv2 = tf.keras.layers.Conv2D(32, kernel_size=kernel_size
    ,
                                            use_bias=False,
                                            kernel_initializer='
    he_normal',
                                            strides=1, padding='same',
                                            activation=tf.nn.relu6)
        self.conv3 = tf.keras.layers.Conv2D(16, kernel_size=kernel_size
    ,
                                            use_bias=False,
                                            kernel_initializer='
    he_normal',
                                            strides=1, padding='same',
                                            activation=tf.nn.relu6)
        self.conv4 = tf.keras.layers.Conv2D(1, kernel_size=kernel_size,
                                            use_bias=False,
                                            kernel_initializer='
    he_normal',
                                            strides=1, padding='same',
                                            activation=tf.math.sigmoid)

    def call(self, inputs):
        avg_out = tf.reduce_mean(inputs, axis=3)
        max_out = tf.reduce_max(inputs,  axis=3)
        x = tf.stack([avg_out, max_out], axis=3)
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        return self.conv4(x)

# A custom layer of channel attention in place of mixing both the
    channel and spatial attention module
class Channel_Attention(tf.keras.layers.Layer):
    def __init__(self, ratio=8):
        '''
        paper: https://arxiv.org/abs/1807.06521
```

```
155         code: https://gist.github.com/innat/99888
    fa8065ecbf3ae2b297e5c10db70
156         '''
157         super(Channel_Attention, self).__init__()
158         self.ratio = ratio
159         self.gapavg = tf.keras.layers.GlobalAveragePooling2D()
160         self.gmpmax = tf.keras.layers.GlobalMaxPooling2D()
161 
162     def build(self, input_shape):
163         self.conv1 = tf.keras.layers.Conv2D(input_shape[-1]//self.ratio
    ,
164                                             kernel_size=1,
165                                             strides=1, padding='same',
166                                             use_bias=True, activation=
    tf.nn.relu)
167 
168         self.conv2 = tf.keras.layers.Conv2D(input_shape[-1],
169                                             kernel_size=1,
170                                             strides=1, padding='same',
171                                             use_bias=True, activation=
    tf.nn.relu)
172         super(Channel_Attention, self).build(input_shape)
173 
174     def call(self, inputs):
175         # compute gap and gmp pooling
176         gapavg = self.gapavg(inputs)
177         gmpmax = self.gmpmax(inputs)
178         gapavg = tf.keras.layers.Reshape((1, 1, gapavg.shape[1]))(
    gapavg)
179         gmpmax = tf.keras.layers.Reshape((1, 1, gmpmax.shape[1]))(
    gmpmax)
180         # forward passing to the respected layers
181         gapavg_out = self.conv2(self.conv1(gapavg))
182         gmpmax_out = self.conv2(self.conv1(gmpmax))
183         return tf.math.sigmoid(gapavg_out + gmpmax_out)
184 
185     def get_output_shape_for(self, input_shape):
186         return self.compute_output_shape(input_shape)
187 
188     def compute_output_shape(self, input_shape):
189         output_len = input_shape[3]
190         return (input_shape[0], output_len)
191 
192 #AttentionWeighted Class creation (GWAP)
193 class AttentionWeightedAverage2D(tf.keras.layers.Layer):
194     def __init__(self, **kwargs):
195         self.init = tf.keras.initializers.get('uniform')
196         super(AttentionWeightedAverage2D, self).__init__(** kwargs)
197 
198     def build(self, input_shape):
199         self.input_spec = [tf.keras.layers.InputSpec(ndim=4)]
200         assert len(input_shape) == 4
201         self.W = self.add_weight(shape=(input_shape[3], 1),
202                                  name='{}_W'.format(self.name),
203                                  initializer=self.init)
204         self._trainable_weights = [self.W]
205         super(AttentionWeightedAverage2D, self).build(input_shape)
206 
```

```
    def call(self, x):
        # computes a probability distribution over the timesteps
        # uses 'max trick' for numerical stability
        # reshape is done to avoid issue with Tensorflow
        # and 2-dimensional weights
        logits  = K.dot(x, self.W)
        x_shape = K.shape(x)
        logits  = K.reshape(logits, (x_shape[0], x_shape[1], x_shape
    [2]))
        ai      = K.exp(logits - K.max(logits, axis=[1,2], keepdims=
    True))

        att_weights    = ai / (K.sum(ai, axis=[1,2], keepdims=True) + K
    .epsilon())
        weighted_input = x * K.expand_dims(att_weights)
        result         = K.sum(weighted_input, axis=[1,2])
        return result

    def get_output_shape_for(self, input_shape):
        return self.compute_output_shape(input_shape)

    def compute_output_shape(self, input_shape):
        output_len = input_shape[3]
        return (input_shape[0], output_len)

#Model Layers Creation
class Chest_Data_Classifier(tf.keras.Model):
    def __init__(self, dim):
        super(Chest_Data_Classifier, self).__init__()
        # Defining All Layers in __init__
        # Layer of Block
        self.Base  = efn.EfficientNetB5(
            input_shape=(data_Train_Config.IMG_SIZE['0'],
                         data_Train_Config.IMG_SIZE['0'], 3),
            weights=None,
            include_top=False)
        # Keras Built-in
        self.GAP1 = tf.keras.layers.GlobalAveragePooling2D()
        self.GAP2 = tf.keras.layers.GlobalAveragePooling2D()
        self.BAT  = tf.keras.layers.BatchNormalization()
        self.ADD  = tf.keras.layers.Add()
        self.AVG  = tf.keras.layers.Average()
        self.DROP = tf.keras.layers.Dropout(rate=0.5)
        # Customs
        self.CAN  = Channel_Attention()
        self.SPN1 = Spatial_Attention()
        self.SPN2 = Spatial_Attention()
        self.AWG  = AttentionWeightedAverage2D()
        # Tail
        self.DENS = tf.keras.layers.Dense(512, activation=tf.nn.relu)
        self.OUT  = tf.keras.layers.Dense(len(target_cols),
                                          activation='sigmoid',
                                          dtype=tf.float32)

    def call(self, input_tensor, training=False):
        # Base Inputs
        x      = self.Base(input_tensor)
        # Attention Modules 1
```

```
        # Channel Attention + Spatial Attention
        canx    = self.CAN(x)*x
        spnx    = self.SPN1(canx)*canx
        # Global Weighted Average Poolin
        gapx    = self.GAP1(spnx)
        wvgx    = self.GAP2(self.SPN2(canx))
        gapavg = self.AVG([gapx, wvgx])
        # Attention Modules 2
        # Attention Weighted Average (AWG)
        awgavg = self.AWG(x)
        # Summation of Attentions
        x = self.ADD([gapavg, awgavg])
        # Tails
        x = self.BAT(x)
        x = self.DENS(x)
        x  = self.DROP(x, training=training)
        return self.OUT(x)

    #The most convenient method to print model.summary() in suclassed model
    def build_graph(self):
        x = tf.keras.layers.Input(shape=(data_Train_Config.IMG_SIZE['0'],
                                         data_Train_Config.IMG_SIZE['0'],3))
        return tf.keras.Model(inputs=[x], outputs=self.call(x))

# plot
model = Chest_Data_Classifier((data_Train_Config.IMG_SIZE['0'],
                           data_Train_Config.IMG_SIZE['0'], 3))

model.save_weights("D:/Data Analytics/Semester 3/Research/model.hd5")
tf.keras.utils.plot_model(
    model.build_graph(), show_shapes=True,
    show_layer_names=True, expand_nested=False
)
```

Listing 12: Novel Approach- EfficientNet + CBAM

13 Model Training

The built model was trained to predict the classification label of the test dataset. It was validated on the validation dataset and early stopping was also used to stop the epochs when there is no improvement in the validation loss (Singh et al.; 2019).

13.1 Baseline Approach- EfficientNet

Figure 17 shows the baseline training.

```
steps_per_epoch = len(X_train) // config.batch_size

history = my_model.fit(
                           train_ds_batch,
                           validation_data = valid_ds_batch,
                           epochs = config.n_epochs,
                           callbacks = callbacks_list,
```

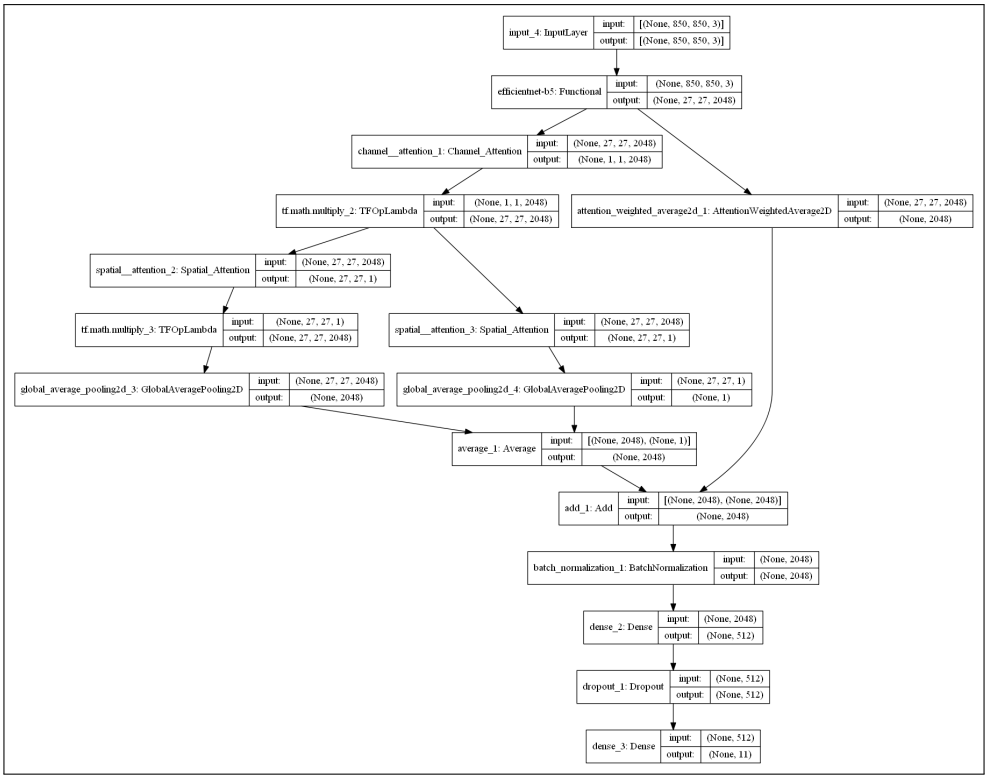

Figure 16: Novel Model

```
                steps_per_epoch = steps_per_epoch
            )
```

Listing 13: Baseline Model training

```
Epoch 00008: saving model to last_model.hdf5
846/846 [==============================] - 659s 779ms/step - loss: 0.1666 - auc: 0.8985 - val_loss: 0.1683 - val_auc: 0.8935
Epoch 9/15
846/846 [==============================] - ETA: 0s - loss: 0.1579 - auc: 0.9125
Epoch 00009: val_loss improved from 0.16832 to 0.16614, saving model to best_model.hdf5

Epoch 00009: saving model to last_model.hdf5
846/846 [==============================] - 661s 781ms/step - loss: 0.1579 - auc: 0.9125 - val_loss: 0.1661 - val_auc: 0.8984
Epoch 10/15
846/846 [==============================] - ETA: 0s - loss: 0.1491 - auc: 0.9234
Epoch 00010: val_loss did not improve from 0.16614

Epoch 00010: saving model to last_model.hdf5
846/846 [==============================] - 660s 780ms/step - loss: 0.1491 - auc: 0.9234 - val_loss: 0.1702 - val_auc: 0.8924
Epoch 11/15
846/846 [==============================] - ETA: 0s - loss: 0.1411 - auc: 0.9305
Epoch 00011: val_loss did not improve from 0.16614

Epoch 00011: saving model to last_model.hdf5

Epoch 00011: ReduceLROnPlateau reducing learning rate to 7.999999797903001e-05.
846/846 [==============================] - 660s 780ms/step - loss: 0.1411 - auc: 0.9305 - val_loss: 0.1793 - val_auc: 0.8813
Epoch 12/15
846/846 [==============================] - ETA: 0s - loss: 0.1303 - auc: 0.9479
Epoch 00012: val_loss did not improve from 0.16614

Epoch 00012: saving model to last_model.hdf5
846/846 [==============================] - 660s 780ms/step - loss: 0.1303 - auc: 0.9479 - val_loss: 0.1762 - val_auc: 0.8802
Epoch 13/15
846/846 [==============================] - ETA: 0s - loss: 0.1211 - auc: 0.9538
Epoch 00013: val_loss did not improve from 0.16614
```

Figure 17: Baseline Training

13.2 Novel Approach- EfficientNet + CBAM

Novel appraoch was trained using 5-fold cross-validation. Figure 18 shows the novel model training.

```

####### CROSS-VALIDATION LOOP
# timer
```

```
cv_start = time.time()

# clear memory
gc.collect()

# placeholders
oof = None

# cross-validation
for fold in range(CFG['num_folds']):

    # feedback
    print('-' * 55)
    print('FOLD {:d}/{:d}'.format(fold + 1, CFG['num_folds']))
    print('-' * 55)

    # prepare model
    model = get_model(CFG, device)
    model_path = ['D:/Data Analytics/Semester 3/Research/v11/weights_fold0.pth',
                  'D:/Data Analytics/Semester 3/Research/v11/weights_fold1.pth',
                  'D:/Data Analytics/Semester 3/Research/v11/weights_fold2.pth',
                  'D:/Data Analytics/Semester 3/Research/v11/weights_fold3.pth',
                  'D:/Data Analytics/Semester 3/Research/v11/weights_fold4.pth']
    model.load_state_dict(torch.load(model_path[fold], map_location = device))

    # prepare data
    df_trn, df_val = get_data(df, fold, CFG)


    # export OOF predictions
    val_preds_df = pd.DataFrame(val_preds_best, columns = ['pred ' + c for c in CFG['targets']])
    val_preds_df = pd.concat([df_val, val_preds_df], axis = 1)
    oof          = pd.concat([oof,    val_preds_df], axis = 0).reset_index(drop = True)
    oof.to_csv(CFG['out_path'] + 'oof.csv', index = False)

    # feedback
    print('-' * 55)
    print('Best: auc = {:.4f} (epoch {}), loss = {:.4f} (epoch {})'.format(
        np.max(val_metrics), np.argmax(val_metrics) + 1, np.min(val_losses), np.argmin(val_losses) + 1))
    print('-' * 55)

    # plot loss dynamics
    plot_results(trn_losses, val_losses, val_metrics, fold, CFG)

# feedback
print('')
```

```
print('Finished in {:.2f} minutes'.format((time.time() - cv_start) /
    60))
```

Listing 14: Novel Approach- EfficientNet + CBAM

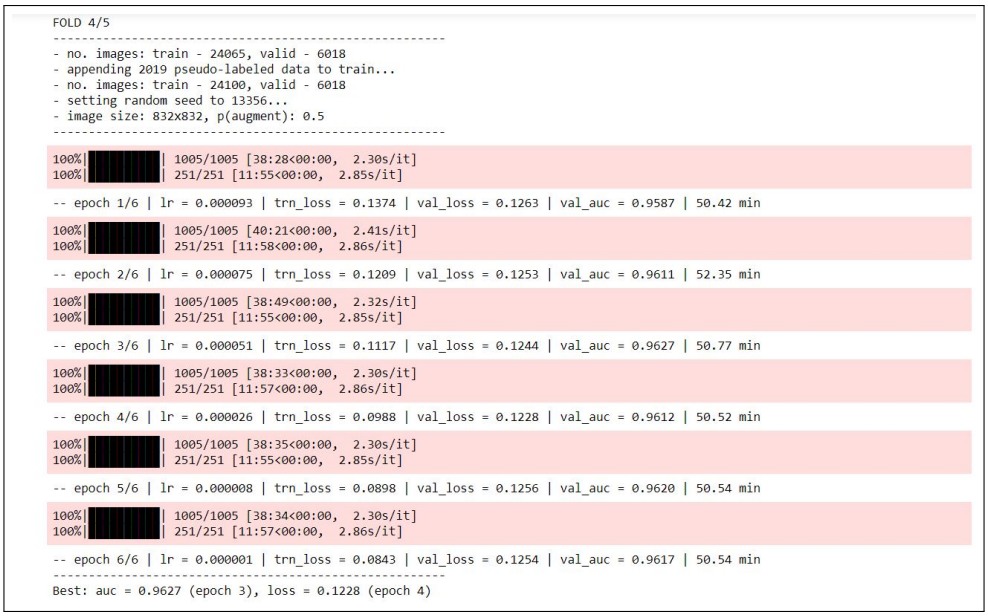

Figure 18: Novel Model Training

14 Test Predictions

Trained model was used to predict the data on test dataset and a CSV having the probability percentage of being the particular label (for ex.- ETT normal, abnormal, borderline) was created.

14.1 Baseline Approach- EfficientNet

Figure 19 shows the baseline final CSV.

```
test_images = glob.glob('D:/Data Analytics/Semester 3/Research/
    Research_Data/test/*.jpg')

df_test = pd.DataFrame(np.array(test_images), columns=['Path'])
df_test.head()

test_ds = tf.data.Dataset.from_tensor_slices((df_test.Path.values))


def process_test(image_path):

    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, [config.target_size_dim, config.
    target_size_dim])

    return img

```

```
test_ds = test_ds.map(process_test, num_parallel_calls=tf.data.
    experimental.AUTOTUNE).batch(config.batch_size*2)

pred_y = my_model.predict(test_ds, workers=4, verbose=1)

df_ss = pd.DataFrame(pred_y, columns = config.labels)
df_test['image_id'] = df_test.Path.str.split('/').str[-1].str[:-4]
df_ss['StudyInstanceUID'] = df_test['image_id']
df_ss.head()

cols_reordered = ['StudyInstanceUID', 'ETT - Abnormal', 'ETT -
    Borderline', 'ETT - Normal', 'NGT - Abnormal',
        'NGT - Borderline', 'NGT - Incompletely Imaged', 'NGT - Normal',
        'CVC - Abnormal', 'CVC - Borderline', 'CVC - Normal',
        'Swan Ganz Catheter Present']

df_order = df_ss[cols_reordered]
df_order.to_csv('submission.csv', index=False)
df_order.head()
```

Listing 15: Baseline Test Predictions

StudyInstanceUID	ETT - Abnormal	ETT - Borderline	ETT - Normal	NGT - Abnormal	NGT - Borderline	NGT - Incompletely Imaged	NGT - Normal	CVC - Abnormal	CVC - Borderline	CVC - Normal	Swan Ganz Catheter Present
1.3680043.8.498.24641136930096467169...	0.000040	0.000086	0.000028	0.000163	0.000263	5.792791e-05	0.000194	0.004411	0.440072	0.686257	0.000056
1.3680043.8.498.12690617441924311870...	0.000054	0.000032	0.000011	0.000050	0.000048	3.019369e-05	0.000067	0.070709	0.218832	0.814413	0.000137
1.3680043.8.498.12475334287210977140...	0.000001	0.000003	0.000002	0.000009	0.000005	3.140399e-07	0.000004	0.004553	0.006865	0.983964	0.000013
1.3680043.8.498.55782720675326550262...	0.000352	0.002103	0.000833	0.001191	0.000931	7.462741e-03	0.002320	0.013834	0.077376	0.920613	0.000217
1.3680043.8.498.31365479801404007311...	0.023695	0.567942	0.556721	0.019073	0.121555	1.584163e-01	0.348152	0.063127	0.386769	0.342052	0.000546

Figure 19: Baseline Final CSV

14.2 Novel Approach- EfficientNet + CBAM

Highlighted label in the Figure 20 shows the of the model is very accurate as it is the only label having high probability percentage. Figure 20 shows the novel model final CSV.

```
test_paths = BaseConfig.TEST_IMG_PATH + submit['StudyInstanceUID'] + '.
    jpg'
test_decoder = build_decoder(with_labels=False,
                             target_size=(TrainConfig.IMG_SIZE['0'],
                                          TrainConfig.IMG_SIZE['0']))
dtest = build_dataset(
    test_paths, bsize=TrainConfig.BATCH_SIZE['0'],
    repeat=False, shuffle=False, augment=False,
    cache=False, decode_fn=test_decoder
)

submit[target_cols] = model.predict(dtest, verbose=1)
submit.to_csv('submission.csv', index=False)
submit.head()
```

Listing 16: Novel Test Predictions

StudyInstanceUID	ETT - Abnormal	ETT - Borderline	ETT - Normal	NGT - Abnormal	NGT - Borderline	NGT - Incompletely Imaged	NGT - Normal	CVC - Abnormal	CVC - Borderline	CVC - Normal	Swan Ganz Catheter Present
43.8.498.46923145579096002617...	2.208163e-05	2.539416e-02	9.708167e-01	1.479139e-05	7.429431e-04	3.008313e-03	9.940944e-01	0.006447	0.059842	0.968643	9.999943e-01
43.8.498.84006870182611080091...	1.069400e-09	3.707888e-08	3.857427e-07	4.198821e-07	8.645618e-08	6.542579e-09	8.789124e-08	0.049204	0.024251	0.956644	3.082640e-06
43.8.498.12219033294413119947...	1.110414e-10	2.721529e-08	2.237614e-08	2.845800e-07	2.130279e-06	1.291258e-08	9.623458e-08	0.002942	0.253688	0.812886	3.176127e-07
43.8.498.84994474380235968109...	5.419382e-04	1.187538e-04	2.124890e-05	9.107623e-03	7.470000e-04	9.905593e-01	2.024026e-03	0.048668	0.316277	0.732969	3.136972e-04
43.8.498.35798987793805669662...	2.853687e-09	3.941081e-07	2.147806e-06	3.483655e-07	1.494529e-06	2.056559e-09	1.747906e-05	0.002341	0.026494	0.986989	3.982171e-07

Figure 20: Novel Model CSV

References

Singh, V., Danda, V., Gorniak, R., Flanders, A. and Lakhani, P. (2019). Assessment of critical feeding tube malpositions on radiographs using deep learning, *Journal of digital imaging* **32**(4): 651–655.

Woo, S., Park, J., Lee, J.-Y. and Kweon, I. S. (2018). Cbam: Convolutional block attention module, *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19.