covid-cxr including SHAP

train.py from covid-cxr converted to notebook format sudo -E /opt/tljh/user/bin/pip3 install dill sudo -E /opt/tljh/user/bin/pip3 install scikit-imageF sudo -E /opt/tljh/user/bin/pip3 install imblearn sudo -E /opt/tljh/user/bin/pip3 install opencv-python sudo -E /opt/tljh/user/bin/pip3 install tensorboard 11/21 source activate tensorflow2_latest_p37 pip3 install tensorflow --upgrade --force-reinstall (tensorflow2_latest_p37) ubuntu@ip-172-31-82-217:~/covid-cxr/src$ pip3 list | grep tensor tensorboard 2.4.0 tensorboard-plugin-wit 1.7.0 tensorflow 2.3.1 tensorflow-estimator 2.3.0 ubuntu@ip-172-31-82-217:~$ whereis activate tensorflow2_latest_p37 activate: /home/ubuntu/anaconda3/bin/activate export PATH=/opt/tljh/user/bin/:$PATH

In [1]:
```
import pandas as pd
import yaml
import os
import datetime
import random
import dill
import numpy as np
import matplotlib.pyplot as plt
import tensorflow.summary as tf_summary
from imblearn.over_sampling import RandomOverSampler
from math import ceil
from tensorflow.keras.metrics import BinaryAccuracy, CategoricalAccuracy, Precision, Recall, AUC
from tensorflow.keras.models import save_model
from tensorflow.keras.callbacks import EarlyStopping, TensorBoard, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorboard.plugins.hparams import api as hp
import sys
sys.path.append('src')
from models.models import *
from visualization.visualize import *
from custom.metrics import F1Score
from data.preprocess import remove_text
# ---- shap
import shap
import tensorflow.compat.v1.keras.backend as K
tf.compat.v1.disable_eager_execution()
import json
```

In [2]:
```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
import tensorflow as tf
tf.config.list_physical_devices('GPU')
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 8106617478863747651
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 8304264105208394324
physical_device_desc: "device: XLA_CPU device"
, name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 11148710307946168679
physical_device_desc: "device: XLA_GPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14648653952
locality {
  bus_id: 1
```

```
  links {
  }
}
incarnation: 12558268046275379528
physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:1e.0, compute capability: 7.5"
]
```

Out[2]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

In [3]:
```
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

In [4]:
```
def get_class_weights(histogram, class_multiplier=None):
    '''
    Computes weights for each class to be applied in the loss function during training.
    :param histogram: A list depicting the number of each item in different class
    :param class_multiplier: List of values to multiply the calculated class weights by. For further control
of class weighting.
    :return: A dictionary containing weights for each class
    '''
    weights = [None] * len(histogram)
    for i in range(len(histogram)):
        weights[i] = (1.0 / len(histogram)) * sum(histogram) / histogram[i]
    class_weight = {i: weights[i] for i in range(len(histogram))}
    if class_multiplier is not None:
        class_weight = [class_weight[i] * class_multiplier[i] for i in range(len(histogram))]
    print("Class weights: ", class_weight)
    #debug
    print("Class weights type:", type(class_weight))
    return class_weight
```

In [ ]:
```

```

In [5]:
```
def random_minority_oversample(train_set):
    '''
    Oversample the minority class using the specified algorithm
    :param train_set: Training set image file names and labels
    :return: A new training set containing oversampled examples
    '''
    X_train = shap.train_set[[x for x in train_set.columns if x != 'label']].to_numpy()
    if X_train.shape[1] == 1:
        X_train = np.expand_dims(X_train, axis=-1)
    Y_train = shap.train_set['label'].to_numpy()
    sampler = RandomOverSampler(random_state=np.random.randint(0, high=1000))
    X_resampled, Y_resampled = sampler.fit_resample(X_train, Y_train)
    filenames = X_resampled[:, 1]     # Filename is in second column
    label_strs = X_resampled[:, 2]    # Class name is in second column
    print("Train set shape before oversampling: ", X_train.shape, " Train set shape after resampling: ",
X_resampled.shape)
    train_set_resampled = pd.DataFrame({'filename': filenames, 'label': Y_resampled, 'label_str':
label_strs})
    return train_set_resampled
```

In [ ]:
```

```

In [6]:
```
def train_model(cfg, data, callbacks, verbose=1):
    '''
    Train a and evaluate model on given data.
```

```
    :param cfg: Project config (from config.yml)
    :param data: dict of partitioned dataset
    :param callbacks: list of callbacks for Keras model
    :param verbose: Verbosity mode to pass to model.fit_generator()
    :return: Trained model and associated performance metrics on the test set
    '''

    # If set in config file, oversample the minority class
    if cfg['TRAIN']['IMB_STRATEGY'] == 'random_oversample':
        data['TRAIN'] = random_minority_oversample(data['TRAIN'])

    # Create ImageDataGenerators
    train_img_gen = ImageDataGenerator(rotation_range=10, preprocessing_function=remove_text,
                                       samplewise_std_normalization=True, samplewise_center=True)
    val_img_gen = ImageDataGenerator(preprocessing_function=remove_text,
                                       samplewise_std_normalization=True, samplewise_center=True)
    test_img_gen = ImageDataGenerator(preprocessing_function=remove_text,
                                       samplewise_std_normalization=True, samplewise_center=True)

    # Create DataFrameIterators
    img_shape = tuple(cfg['DATA']['IMG_DIM'])
    print("********** target_size **********", img_shape)
    print("********** batch_size **********", cfg['TRAIN']['BATCH_SIZE'])
    y_col = 'label_str'
    class_mode = 'categorical'
    train_generator = train_img_gen.flow_from_dataframe(dataframe=data['TRAIN'], directory=cfg['PATHS']
['RAW_DATA'],
        x_col="filename", y_col=y_col, target_size=img_shape, batch_size=cfg['TRAIN']['BATCH_SIZE'],
        class_mode=class_mode, validate_filenames=False)
    val_generator = val_img_gen.flow_from_dataframe(dataframe=data['VAL'], directory=cfg['PATHS']
['RAW_DATA'],
        x_col="filename", y_col=y_col, target_size=img_shape, batch_size=cfg['TRAIN']['BATCH_SIZE'],
        class_mode=class_mode, validate_filenames=False)
    test_generator = test_img_gen.flow_from_dataframe(dataframe=data['TEST'], directory=cfg['PATHS']
['RAW_DATA'],
        x_col="filename", y_col=y_col, target_size=img_shape, batch_size=cfg['TRAIN']['BATCH_SIZE'],
        class_mode=class_mode, validate_filenames=False, shuffle=False)

    # Save model's ordering of class indices
    dill.dump(test_generator.class_indices, open(cfg['PATHS']['OUTPUT_CLASS_INDICES'], 'wb'))

    # Apply class imbalance strategy. We have many more X-rays negative for COVID-19 than positive.
    histogram = np.bincount(np.array(train_generator.labels).astype(int))  # Get class distribution
    class_weight = None
    if cfg['TRAIN']['IMB_STRATEGY'] == 'class_weight':
        class_multiplier = cfg['TRAIN']['CLASS_MULTIPLIER']
        class_multiplier = [class_multiplier[cfg['DATA']['CLASSES'].index(c)] for c in
test_generator.class_indices]
        class_weight = get_class_weights(histogram, class_multiplier)

    # Define metrics.
    covid_class_idx = test_generator.class_indices['COVID-19']   # Get index of COVID-19 class
    print("********** covid_class_idx **********", covid_class_idx)
    thresholds = 1.0 / len(cfg['DATA']['CLASSES'])      # Binary classification threshold for a class
    print("********** thresholds **********", thresholds)
    metrics = [CategoricalAccuracy(name='accuracy'),
```

```
                Precision(name='precision', thresholds=thresholds, class_id=covid_class_idx),
                Recall(name='recall', thresholds=thresholds, class_id=covid_class_idx),
                AUC(name='auc'),
                F1Score(name='f1score', thresholds=thresholds, class_id=covid_class_idx)]

    # Define the model.
    print('Training distribution: ', ['Class ' + list(test_generator.class_indices.keys())[i] + ': ' +
str(histogram[i]) + '. '
           for i in range(len(histogram))])
    input_shape = cfg['DATA']['IMG_DIM'] + [3]
    num_gpus = cfg['TRAIN']['NUM_GPUS']
    #debug
    print("******* GPU:", num_gpus)
    if cfg['TRAIN']['MODEL_DEF'] == 'dcnn_resnet':    #single_train
        model_def = dcnn_resnet
    elif cfg['TRAIN']['MODEL_DEF'] == 'resnet50v2':
        model_def = resnet50v2
    else:
        model_def = resnet101v2
    if cfg['TRAIN']['CLASS_MODE'] == 'binary':
        histogram = np.bincount(data['TRAIN']['label'].astype(int))
        output_bias = np.log([histogram[i] / (np.sum(histogram) - histogram[i]) for i in
range(histogram.shape[0])])
        model = model_def(cfg['NN']['DCNN_BINARY'], input_shape, metrics, 2, output_bias=output_bias,
gpus=num_gpus)
    else:
        n_classes = len(cfg['DATA']['CLASSES'])
        histogram = np.bincount(data['TRAIN']['label'].astype(int))
        output_bias = np.log([histogram[i] / (np.sum(histogram) - histogram[i]) for i in
range(histogram.shape[0])])
        model = model_def(cfg['NN']['DCNN_MULTICLASS'], input_shape, metrics, n_classes,
output_bias=output_bias,
                          gpus=num_gpus)
    #debug
    print("histogram type", type(histogram), histogram)

    # Train the model.
    steps_per_epoch = ceil(train_generator.n / train_generator.batch_size)
    val_steps = ceil(val_generator.n / val_generator.batch_size)
    # debug
    print("***** class weight", class_weight)
    class_weight = {0:26.589285714285715, 1:0.07643737166324435}

    history = model.fit(train_generator, steps_per_epoch=steps_per_epoch, epochs=cfg['TRAIN']['EPOCHS'],
                                validation_data=val_generator, validation_steps=val_steps,
callbacks=callbacks,
                                verbose=verbose, class_weight=class_weight)

    # Run the model on the test set and print the resulting performance metrics.
    print("********** test_generator ************** ", test_generator)
    test_results = model.evaluate(test_generator, verbose=1)
    test_metrics = {}
    test_summary_str = [['**Metric**', '**Value**']]
    for metric, value in zip(model.metrics_names, test_results):
        test_metrics[metric] = value
        print(metric, ' = ', value)
```

```
        test_summary_str.append([metric, str(value)])

    print("train_model function **************", model)
    return model, test_metrics, test_generator
```

In [7]:

```
#class_weight = {0:26.589285714285715, 1:0.07643737166324435}
#print(class_weight)
```

In [8]:

```
def multi_train(cfg, data, callbacks, base_log_dir):
    '''
    Trains a model a series of times and returns the model with the best test set metric (specified in cfg)
    :param cfg: Project config (from config.yml)
    :param data: Partitioned dataset
    :param callbacks: List of callbacks to pass to model.fit()
    :param base_log_dir: Base directory to write logs
    :return: The trained Keras model with best test set performance on the metric specified in cfg
    '''

    # Load order of metric preference
    metric_preference = cfg['TRAIN']['METRIC_PREFERENCE']
    best_metrics = dict.fromkeys(metric_preference, 0.0)
    if 'loss' in metric_preference:
        best_metrics['loss'] = 100000.0

    # Train NUM_RUNS models and return the best one according to the preferred metrics
    for i in range(cfg['TRAIN']['NUM_RUNS']):
        print("Training run ", i+1, " / ", cfg['TRAIN']['NUM_RUNS'])
        cur_callbacks = callbacks.copy()
        cur_date = datetime.datetime.now().strftime('%Y%m%d-%H%M%S')
        if base_log_dir is not None:
            log_dir = base_log_dir + cur_date
            cur_callbacks.append(TensorBoard(log_dir=log_dir, histogram_freq=1))

        # Train the model and evaluate performance on test set
        new_model, test_metrics, test_generator = train_model(cfg, data, cur_callbacks, verbose=1)

        # Log test set results and images
        if base_log_dir is not None:
            log_test_results(cfg, new_model, test_generator, test_metrics, log_dir)

        # If this model outperforms the previous ones based on the specified metric preferences, save this
one.
        for i in range(len(metric_preference)):
            if (((metric_preference[i] == 'loss') and (test_metrics[metric_preference[i]] <
best_metrics[metric_preference[i]]))
                    or ((metric_preference[i] != 'loss') and (test_metrics[metric_preference[i]] >
best_metrics[metric_preference[i]]))):
                best_model = new_model
                best_metrics = test_metrics
                best_generator = test_generator
                best_model_date = cur_date
                break
            elif (test_metrics[metric_preference[i]] == best_metrics[metric_preference[i]]):
                continue
            else:
```

```
                break

    print("Best model test metrics: ", best_metrics)
    return best_model, best_metrics, best_generator, best_model_date
```

In [ ]:

```
```

In [9]:

```
def random_hparam_search(cfg, data, callbacks, log_dir):
    '''
    Conduct a random hyperparameter search over the ranges given for the hyperparameters in config.yml and
log results
    in TensorBoard. Model is trained x times for y random combinations of hyperparameters.
    :param cfg: Project config
    :param data: Dict containing the partitioned datasets
    :param callbacks: List of callbacks for Keras model (excluding TensorBoard)
    :param log_dir: Base directory in which to store logs
    :return: (Last model trained, resultant test set metrics, test data generator)
    '''

    # Define HParam objects for each hyperparameter we wish to tune.
    hp_ranges = cfg['HP_SEARCH']['RANGES']
    HPARAMS = []
    HPARAMS.append(hp.HParam('KERNEL_SIZE', hp.Discrete(hp_ranges['KERNEL_SIZE'])))
    HPARAMS.append(hp.HParam('MAXPOOL_SIZE', hp.Discrete(hp_ranges['MAXPOOL_SIZE'])))
    HPARAMS.append(hp.HParam('INIT_FILTERS', hp.Discrete(hp_ranges['INIT_FILTERS'])))
    HPARAMS.append(hp.HParam('FILTER_EXP_BASE', hp.IntInterval(hp_ranges['FILTER_EXP_BASE'][0],
hp_ranges['FILTER_EXP_BASE'][1])))
    HPARAMS.append(hp.HParam('NODES_DENSE0', hp.Discrete(hp_ranges['NODES_DENSE0'])))
    HPARAMS.append(hp.HParam('CONV_BLOCKS', hp.IntInterval(hp_ranges['CONV_BLOCKS'][0],
hp_ranges['CONV_BLOCKS'][1])))
    HPARAMS.append(hp.HParam('DROPOUT', hp.Discrete(hp_ranges['DROPOUT'])))
    HPARAMS.append(hp.HParam('LR', hp.RealInterval(hp_ranges['LR'][0], hp_ranges['LR'][1])))
    HPARAMS.append(hp.HParam('OPTIMIZER', hp.Discrete(hp_ranges['OPTIMIZER'])))
    HPARAMS.append(hp.HParam('L2_LAMBDA', hp.Discrete(hp_ranges['L2_LAMBDA'])))
    HPARAMS.append(hp.HParam('BATCH_SIZE', hp.Discrete(hp_ranges['BATCH_SIZE'])))
    HPARAMS.append(hp.HParam('IMB_STRATEGY', hp.Discrete(hp_ranges['IMB_STRATEGY'])))

    # Define test set metrics that we wish to log to TensorBoard for each training run
    HP_METRICS = [hp.Metric(metric, display_name='Test ' + metric) for metric in cfg['HP_SEARCH']
['METRICS']]

    # Configure TensorBoard to log the results
    with tf.summary.create_file_writer(log_dir).as_default():
        hp.hparams_config(hparams=HPARAMS, metrics=HP_METRICS)

    # Complete a number of training runs at different hparam values and log the results.
    repeats_per_combo = cfg['HP_SEARCH']['REPEATS']   # Number of times to train the model per combination
of hparams
    num_combos = cfg['HP_SEARCH']['COMBINATIONS']     # Number of random combinations of hparams to attempt
    num_sessions = num_combos * repeats_per_combo       # Total number of runs in this experiment
    model_type = 'DCNN_BINARY' if cfg['TRAIN']['CLASS_MODE'] == 'binary' else 'DCNN_MULTICLASS'
    trial_id = 0
    for group_idx in range(num_combos):
        rand = random.Random()
        HPARAMS = {h: h.domain.sample_uniform(rand) for h in HPARAMS}
```

```
        hparams = {h.name: HPARAMS[h] for h in HPARAMS}  # To pass to model definition
        for repeat_idx in range(repeats_per_combo):
            trial_id += 1
            print("Running training session %d/%d" % (trial_id, num_sessions))
            print("Hparam values: ", {h.name: HPARAMS[h] for h in HPARAMS})
            trial_logdir = os.path.join(log_dir, str(trial_id))     # Need specific logdir for each trial
            callbacks_hp = callbacks + [TensorBoard(log_dir=trial_logdir, profile_batch=0,
write_graph=False)]

            # Set values of hyperparameters for this run in config file.
            for h in hparams:
                if h in ['LR', 'L2_LAMBDA']:
                    val = 10 ** hparams[h]      # These hyperparameters are sampled on the log scale.
                else:
                    val = hparams[h]
                cfg['NN'][model_type][h] = val

            # Set some hyperparameters that are not specified in model definition.
            cfg['TRAIN']['BATCH_SIZE'] = hparams['BATCH_SIZE']
            cfg['TRAIN']['IMB_STRATEGY'] = hparams['IMB_STRATEGY']

            # Run a training session and log the performance metrics on the test set to HParams dashboard in
TensorBoard
            with tf.summary.create_file_writer(trial_logdir).as_default():
                hp.hparams(HPARAMS, trial_id=str(trial_id))
                model, test_metrics, test_generator = train_model(cfg, data, callbacks_hp, verbose=0)
                for metric in HP_METRICS:
                    if metric._tag in test_metrics:
                        tf.summary.scalar(metric._tag, test_metrics[metric._tag], step=1)   # Log test
metric
    return
```

In [ ]:

```

```

In [10]:

```
def log_test_results(cfg, model, test_generator, test_metrics, log_dir):
    '''
    Visualize performance of a trained model on the test set. Optionally save the model.
    :param cfg: Project config
    :param model: A trained Keras model
    :param test_generator: A Keras generator for the test set
    :param test_metrics: Dict of test set performance metrics
    :param log_dir: Path to write TensorBoard logs
    '''

    # Visualization of test results
    test_predictions = model.predict(test_generator, verbose=0)
    test_labels = test_generator.labels
    covid_idx = test_generator.class_indices['COVID-19']
    plt = plot_roc("Test set", test_labels, test_predictions, class_id=covid_idx)
    roc_img = plot_to_tensor()
    plt = plot_confusion_matrix(test_labels, test_predictions, class_id=covid_idx)
    cm_img = plot_to_tensor()

    # Log test set results and plots in TensorBoard
    writer = tf_summary.create_file_writer(logdir=log_dir)
```

```
    # Create table of test set metrics
    test_summary_str = [['**Metric**','**Value**']]
    thresholds = cfg['TRAIN']['THRESHOLDS']  # Load classification thresholds
    for metric in test_metrics:
        if metric in ['precision', 'recall'] and isinstance(metric, list):
            metric_values = dict(zip(thresholds, test_metrics[metric]))
        else:
            metric_values = test_metrics[metric]
        test_summary_str.append([metric, str(metric_values)])

    # Create table of model and train config values
    hparam_summary_str = [['**Variable**', '**Value**']]
    for key in cfg['TRAIN']:
        hparam_summary_str.append([key, str(cfg['TRAIN'][key])])
    if cfg['TRAIN']['CLASS_MODE'] == 'binary':
        for key in cfg['NN']['DCNN_BINARY']:
            hparam_summary_str.append([key, str(cfg['NN']['DCNN_BINARY'][key])])
    else:
        for key in cfg['NN']['DCNN_BINARY']:
            hparam_summary_str.append([key, str(cfg['NN']['DCNN_BINARY'][key])])

    # Write to TensorBoard logs
    with writer.as_default():
        tf_summary.text(name='Test set metrics', data=tf.convert_to_tensor(test_summary_str), step=0)
        tf_summary.text(name='Run hyperparameters', data=tf.convert_to_tensor(hparam_summary_str), step=0)
        tf_summary.image(name='ROC Curve (Test Set)', data=roc_img, step=0)
        tf_summary.image(name='Confusion Matrix (Test Set)', data=cm_img, step=0)
    return
```

In [ ]:

```
```

In [11]:

```
def train_experiment(cfg=None, experiment='single_train', save_weights=True, write_logs=True):
    '''
    Defines and trains HIFIS-v2 model. Prints and logs relevant metrics.
    :param experiment: The type of training experiment. Choices are {'single_train'}
    :param save_weights: A flag indicating whether to save the model weights
    :param write_logs: A flag indicating whether to write TensorBoard logs
    :return: A dictionary of metrics on the test set
    '''

    # Load project config data
    if cfg is None:
        cfg = yaml.full_load(open(os.getcwd() + "/config.yml", 'r'))

    # Set logs directory
    cur_date = datetime.datetime.now().strftime('%Y%m%d-%H%M%S')
    print(cfg['PATHS']['LOGS'])
    log_dir = cfg['PATHS']['LOGS'] + "training/" + cur_date if write_logs else None
    if not os.path.exists(cfg['PATHS']['LOGS'] + "training\\"):
        os.makedirs(cfg['PATHS']['LOGS'] + "training\\")

    # Load dataset file paths and labels
    data = {}
    data['TRAIN'] = pd.read_csv(cfg['PATHS']['TRAIN_SET'])
```

```
    data['VAL'] = pd.read_csv(cfg['PATHS']['VAL_SET'])
    data['TEST'] = pd.read_csv(cfg['PATHS']['TEST_SET'])

    # Set callbacks.
    early_stopping = EarlyStopping(monitor='val_loss', verbose=1, patience=cfg['TRAIN']['PATIENCE'],
mode='min', restore_best_weights=True)
    callbacks = [early_stopping]

    # Conduct the desired train experiment
    if experiment == 'hparam_search':
        log_dir = cfg['PATHS']['LOGS'] + "hparam_search\\" + cur_date
        random_hparam_search(cfg, data, callbacks, log_dir)
    else:
        if experiment == 'multi_train':
            base_log_dir = cfg['PATHS']['LOGS'] + "training\\" if write_logs else None
            model, test_metrics, test_generator, cur_date = multi_train(cfg, data, callbacks, base_log_dir)
        else:
            if write_logs:
                tensorboard = TensorBoard(log_dir=log_dir, histogram_freq=1)
                callbacks.append(tensorboard)
            #
            '''
            print(cfg)
            print(data)
            print(callbacks)
            '''
            # print("********** single data ********** ", data) # data labels and image file names
            model, test_metrics, test_generator = train_model(cfg, data, callbacks)
            print("********** single ********** ", model)
            if write_logs:
                log_test_results(cfg, model, test_generator, test_metrics, log_dir)
        if save_weights:
            model_path = cfg['PATHS']['MODEL_WEIGHTS'] + 'model' + cur_date + '.h5'
            save_model(model, model_path)  # Save the model's weights
    return (model,test_metrics,test_generator)
```

In [12]:

```
#epoch comes from config.yml
cfg = yaml.full_load(open("/home/ubuntu/covid-cxr/config.yml", 'r'))
# print("1",cfg)
cfg['TRAIN']['EXPERIMENT_TYPE'] #single train

model,test_metrics,test_generator = train_experiment(cfg=cfg, experiment=cfg['TRAIN']['EXPERIMENT_TYPE'],
save_weights=True, write_logs=True)
print("This is model: ",model)
print("This is test_metrics: ",test_metrics)
print("This is test_generator: ",test_generator)
```

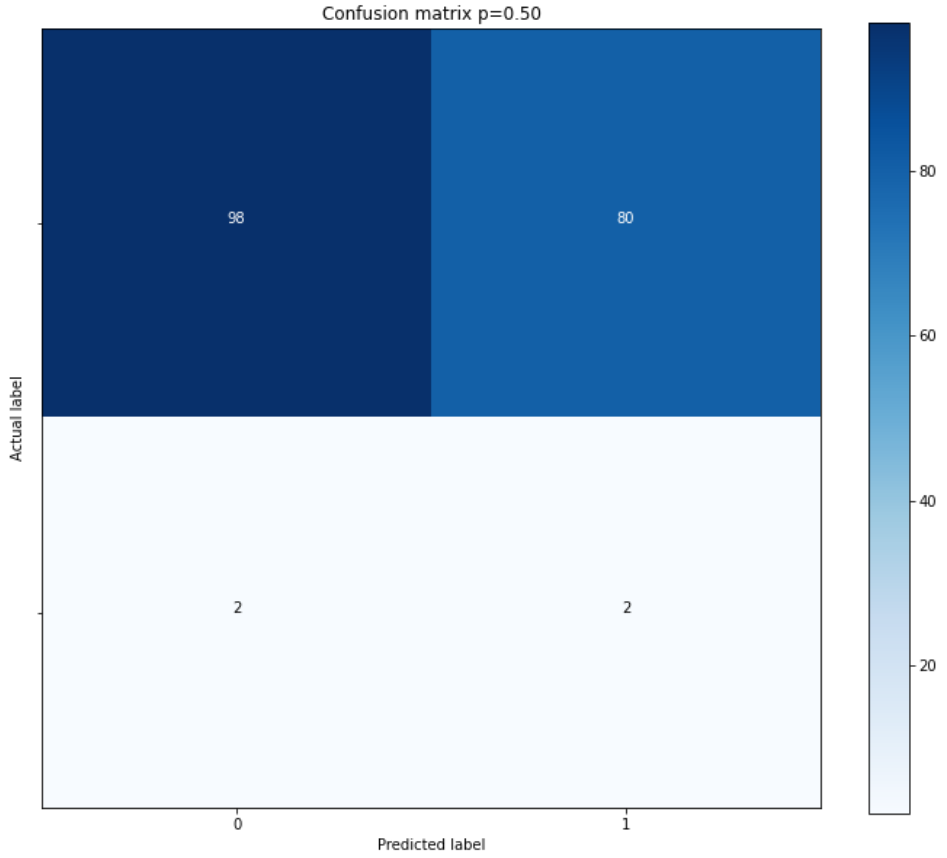
```
/home/ubuntu/covid-cxr/results/logs/
********** target_size ********** (224, 224)
********** batch_size ********** 32
Found 1489 non-validated image filenames belonging to 2 classes.
Found 146 non-validated image filenames belonging to 2 classes.
Found 182 non-validated image filenames belonging to 2 classes.
Class weights:  [26.589285714285715, 0.07643737166324435]
Class weights type: <class 'list'>
********** covid_class_idx ********** 0
********** thresholds ********** 0.5
Training distribution:  ['Class COVID-19: 28. ', 'Class non-COVID-19: 1461. ']
******* GPU: 1
MODEL CONFIG:  {'KERNEL_SIZE': '(3,3)', 'STRIDES': '(1,1)', 'INIT_FILTERS': 16, 'FILTER_EXP_BASE': 3, 'MAXPOOL_SIZE': '(2,
2)', 'CONV_BLOCKS': 3, 'NODES_DENSE0': 128, 'LR': 1e-05, 'OPTIMIZER': 'adam', 'DROPOUT': 0.4, 'L2_LAMBDA': 0.0001}
```

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
conv0_0 (Conv2D)                (None, 224, 224, 16) 448         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 224, 224, 16) 64          conv0_0[0][0]
__________________________________________________________________________________________________
leaky_re_lu (LeakyReLU)         (None, 224, 224, 16) 0           batch_normalization[0][0]
__________________________________________________________________________________________________
conv0_1 (Conv2D)                (None, 224, 224, 16) 2320        leaky_re_lu[0][0]
__________________________________________________________________________________________________
concat0 (Concatenate)           (None, 224, 224, 19) 0           conv0_1[0][0]
                                                                 input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 224, 224, 19) 76          concat0[0][0]
__________________________________________________________________________________________________
leaky_re_lu_1 (LeakyReLU)       (None, 224, 224, 19) 0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 112, 112, 19) 0           leaky_re_lu_1[0][0]
__________________________________________________________________________________________________
conv1_0 (Conv2D)                (None, 112, 112, 48) 8256        max_pooling2d[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 112, 112, 48) 192         conv1_0[0][0]
__________________________________________________________________________________________________
leaky_re_lu_2 (LeakyReLU)       (None, 112, 112, 48) 0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
conv1_1 (Conv2D)                (None, 112, 112, 48) 20784       leaky_re_lu_2[0][0]
__________________________________________________________________________________________________
concat1 (Concatenate)           (None, 112, 112, 67) 0           conv1_1[0][0]
                                                                 max_pooling2d[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 112, 112, 67) 268         concat1[0][0]
__________________________________________________________________________________________________
leaky_re_lu_3 (LeakyReLU)       (None, 112, 112, 67) 0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 56, 56, 67)   0           leaky_re_lu_3[0][0]
__________________________________________________________________________________________________
conv2_0 (Conv2D)                (None, 56, 56, 144)  86976       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 56, 56, 144)  576         conv2_0[0][0]
__________________________________________________________________________________________________
leaky_re_lu_4 (LeakyReLU)       (None, 56, 56, 144)  0           batch_normalization_4[0][0]
__________________________________________________________________________________________________
conv2_1 (Conv2D)                (None, 56, 56, 144)  186768      leaky_re_lu_4[0][0]
__________________________________________________________________________________________________
concat2 (Concatenate)           (None, 56, 56, 211)  0           conv2_1[0][0]
                                                                 max_pooling2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 56, 56, 211)  844         concat2[0][0]
__________________________________________________________________________________________________
leaky_re_lu_5 (LeakyReLU)       (None, 56, 56, 211)  0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 28, 28, 211)  0           leaky_re_lu_5[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 165424)       0           max_pooling2d_2[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 165424)       0           flatten[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          21174400    dropout[0][0]
__________________________________________________________________________________________________
leaky_re_lu_6 (LeakyReLU)       (None, 128)          0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 2)            258         leaky_re_lu_6[0][0]
__________________________________________________________________________________________________
output (Activation)             (None, 2)            0           dense_1[0][0]
==================================================================================================
Total params: 21,482,230
Trainable params: 21,481,220
Non-trainable params: 1,010
__________________________________________________________________________________________________
histogram type <class 'numpy.ndarray'> [1461   28]
***** class weight [26.589285714285715, 0.07643737166324435]
WARNING:tensorflow:From /opt/tljh/user/lib/python3.7/site-packages/tensorflow/python/keras/engine/training_v1.py:2048: Mode
l.state_updates (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
Epoch 1/50
 2/47 [>.............................] - ETA: 17s - batch: 0.5000 - size: 32.0000 - loss: 606.6207 - accuracy: 0.2500 - prec
ision: 0.0208 - recall: 0.5000 - auc: 0.1973 - f1score: 0.0400   WARNING:tensorflow:Callbacks method `on_train_batch_begin`
is slow compared to the batch time (batch time: 0.2268s vs `on_train_batch_begin` time: 0.4296s). Check your callbacks.
47/47 [==============================] - 61s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 582.7913 - accuracy: 0.2055 -
precision: 0.0175 - recall: 0.7500 - auc: 0.1426 - f1score: 0.0343 - val_loss: 608.5675 - val_accuracy: 0.0205 - val_precisi
on: 0.0205 - val_recall: 1.0000 - val_auc: 0.0208 - val_f1score: 0.0403
Epoch 2/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 547.4379 - accuracy: 0.2062 -
precision: 0.0224 - recall: 0.9643 - auc: 0.1298 - f1score: 0.0437 - val_loss: 464.0191 - val_accuracy: 0.3630 - val_precisi
```

```
on: 0.0312 - val_recall: 1.0000 - val_auc: 0.3106 - val_f1score: 0.0606
Epoch 3/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 517.8561 - accuracy: 0.4627 -
precision: 0.0304 - recall: 0.8929 - auc: 0.4757 - f1score: 0.0588 - val_loss: 425.9707 - val_accuracy: 0.1301 - val_precisi
on: 0.0231 - val_recall: 1.0000 - val_auc: 0.0830 - val_f1score: 0.0451
Epoch 4/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 493.7674 - accuracy: 0.3244 -
precision: 0.0252 - recall: 0.9286 - auc: 0.2843 - f1score: 0.0491 - val_loss: 410.6108 - val_accuracy: 0.4452 - val_precisi
on: 0.0357 - val_recall: 1.0000 - val_auc: 0.4697 - val_f1score: 0.0690
Epoch 5/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 472.6975 - accuracy: 0.4876 -
precision: 0.0342 - recall: 0.9643 - auc: 0.4977 - f1score: 0.0661 - val_loss: 407.3293 - val_accuracy: 0.3836 - val_precisi
on: 0.0323 - val_recall: 1.0000 - val_auc: 0.3604 - val_f1score: 0.0625
Epoch 6/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 454.4735 - accuracy: 0.4103 -
precision: 0.0288 - recall: 0.9286 - auc: 0.3639 - f1score: 0.0559 - val_loss: 408.1635 - val_accuracy: 0.1644 - val_precisi
on: 0.0240 - val_recall: 1.0000 - val_auc: 0.1490 - val_f1score: 0.0469
Epoch 7/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 438.0636 - accuracy: 0.4708 -
precision: 0.0320 - recall: 0.9286 - auc: 0.4599 - f1score: 0.0619 - val_loss: 406.6501 - val_accuracy: 0.4247 - val_precisi
on: 0.0345 - val_recall: 1.0000 - val_auc: 0.3670 - val_f1score: 0.0667
Epoch 8/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 423.7054 - accuracy: 0.5756 -
precision: 0.0424 - recall: 1.0000 - auc: 0.6035 - f1score: 0.0814 - val_loss: 405.3163 - val_accuracy: 0.3425 - val_precisi
on: 0.0303 - val_recall: 1.0000 - val_auc: 0.3065 - val_f1score: 0.0588
Epoch 9/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 410.6014 - accuracy: 0.5487 -
precision: 0.0387 - recall: 0.9643 - auc: 0.5541 - f1score: 0.0744 - val_loss: 400.0344 - val_accuracy: 0.6438 - val_precisi
on: 0.0545 - val_recall: 1.0000 - val_auc: 0.7096 - val_f1score: 0.1034
Epoch 10/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 398.9574 - accuracy: 0.5353 -
precision: 0.0363 - recall: 0.9286 - auc: 0.5465 - f1score: 0.0699 - val_loss: 395.4248 - val_accuracy: 0.5959 - val_precisi
on: 0.0484 - val_recall: 1.0000 - val_auc: 0.6735 - val_f1score: 0.0923
Epoch 11/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 388.1432 - accuracy: 0.5433 -
precision: 0.0382 - recall: 0.9643 - auc: 0.5593 - f1score: 0.0736 - val_loss: 388.7398 - val_accuracy: 0.3493 - val_precisi
on: 0.0306 - val_recall: 1.0000 - val_auc: 0.2983 - val_f1score: 0.0594
Epoch 12/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 379.0371 - accuracy: 0.1793 -
precision: 0.0209 - recall: 0.9286 - auc: 0.1293 - f1score: 0.0408 - val_loss: 388.7238 - val_accuracy: 0.0753 - val_precisi
on: 0.0217 - val_recall: 1.0000 - val_auc: 0.0577 - val_f1score: 0.0426
Epoch 13/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 369.1328 - accuracy: 0.5326 -
precision: 0.0374 - recall: 0.9643 - auc: 0.5351 - f1score: 0.0720 - val_loss: 373.1083 - val_accuracy: 0.6918 - val_precisi
on: 0.0435 - val_recall: 0.6667 - val_auc: 0.7525 - val_f1score: 0.0816
Epoch 14/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 360.3953 - accuracy: 0.7132 -
precision: 0.0596 - recall: 0.9643 - auc: 0.7631 - f1score: 0.1123 - val_loss: 366.1200 - val_accuracy: 0.7055 - val_precisi
on: 0.0455 - val_recall: 0.6667 - val_auc: 0.7675 - val_f1score: 0.0851
Epoch 15/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 352.6794 - accuracy: 0.6555 -
precision: 0.0484 - recall: 0.9286 - auc: 0.6971 - f1score: 0.0920 - val_loss: 362.0435 - val_accuracy: 0.4589 - val_precisi
on: 0.0366 - val_recall: 1.0000 - val_auc: 0.4226 - val_f1score: 0.0706
Epoch 16/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 345.4769 - accuracy: 0.4251 -
precision: 0.0285 - recall: 0.8929 - auc: 0.4001 - f1score: 0.0552 - val_loss: 356.7981 - val_accuracy: 0.5342 - val_precisi
on: 0.0423 - val_recall: 1.0000 - val_auc: 0.5198 - val_f1score: 0.0811
Epoch 17/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 338.2371 - accuracy: 0.7374 -
precision: 0.0647 - recall: 0.9643 - auc: 0.7933 - f1score: 0.1213 - val_loss: 352.7303 - val_accuracy: 0.3356 - val_precisi
on: 0.0300 - val_recall: 1.0000 - val_auc: 0.3103 - val_f1score: 0.0583
Epoch 18/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 331.6985 - accuracy: 0.5131 -
precision: 0.0360 - recall: 0.9643 - auc: 0.5384 - f1score: 0.0693 - val_loss: 343.9278 - val_accuracy: 0.6986 - val_precisi
on: 0.0638 - val_recall: 1.0000 - val_auc: 0.7823 - val_f1score: 0.1200
Epoch 19/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 325.5620 - accuracy: 0.7206 -
precision: 0.0591 - recall: 0.9286 - auc: 0.7829 - f1score: 0.1111 - val_loss: 345.0051 - val_accuracy: 0.1712 - val_precisi
on: 0.0242 - val_recall: 1.0000 - val_auc: 0.1283 - val_f1score: 0.0472
Epoch 20/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 320.0400 - accuracy: 0.3808 -
precision: 0.0285 - recall: 0.9643 - auc: 0.3562 - f1score: 0.0553 - val_loss: 343.1973 - val_accuracy: 0.0753 - val_precisi
on: 0.0217 - val_recall: 1.0000 - val_auc: 0.0507 - val_f1score: 0.0426
Epoch 21/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 314.0853 - accuracy: 0.6971 -
precision: 0.0566 - recall: 0.9643 - auc: 0.7448 - f1score: 0.1069 - val_loss: 337.8356 - val_accuracy: 0.1575 - val_precisi
on: 0.0238 - val_recall: 1.0000 - val_auc: 0.1265 - val_f1score: 0.0465
Epoch 22/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 308.9892 - accuracy: 0.4056 -
precision: 0.0296 - recall: 0.9643 - auc: 0.3928 - f1score: 0.0575 - val_loss: 327.9206 - val_accuracy: 0.3219 - val_precisi
on: 0.0294 - val_recall: 1.0000 - val_auc: 0.2576 - val_f1score: 0.0571
Epoch 23/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 303.7563 - accuracy: 0.5944 -
precision: 0.0429 - recall: 0.9643 - auc: 0.6236 - f1score: 0.0821 - val_loss: 319.3743 - val_accuracy: 0.6781 - val_precisi
on: 0.0600 - val_recall: 1.0000 - val_auc: 0.7445 - val_f1score: 0.1132
Epoch 24/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 298.8835 - accuracy: 0.7314 -
precision: 0.0613 - recall: 0.9286 - auc: 0.7904 - f1score: 0.1150 - val_loss: 320.7784 - val_accuracy: 0.1986 - val_precisi
on: 0.0250 - val_recall: 1.0000 - val_auc: 0.1454 - val_f1score: 0.0488
```

```
Epoch 25/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 294.1756 - accuracy: 0.5487 -
precision: 0.0387 - recall: 0.9643 - auc: 0.5678 - f1score: 0.0744 - val_loss: 313.4618 - val_accuracy: 0.3767 - val_precisi
on: 0.0319 - val_recall: 1.0000 - val_auc: 0.3538 - val_f1score: 0.0619
Epoch 26/50
47/47 [==============================] - 59s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 289.6183 - accuracy: 0.6232 -
precision: 0.0460 - recall: 0.9643 - auc: 0.6602 - f1score: 0.0878 - val_loss: 306.6709 - val_accuracy: 0.5959 - val_precisi
on: 0.0484 - val_recall: 1.0000 - val_auc: 0.6418 - val_f1score: 0.0923
Epoch 27/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 285.2711 - accuracy: 0.6300 -
precision: 0.0468 - recall: 0.9643 - auc: 0.6516 - f1score: 0.0893 - val_loss: 305.0718 - val_accuracy: 0.8288 - val_precisi
on: 0.0769 - val_recall: 0.6667 - val_auc: 0.8800 - val_f1score: 0.1379
Epoch 28/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 281.1745 - accuracy: 0.5554 -
precision: 0.0379 - recall: 0.9286 - auc: 0.5673 - f1score: 0.0728 - val_loss: 302.9978 - val_accuracy: 0.3630 - val_precisi
on: 0.0312 - val_recall: 1.0000 - val_auc: 0.3106 - val_f1score: 0.0606
Epoch 29/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 277.0509 - accuracy: 0.6971 -
precision: 0.0566 - recall: 0.9643 - auc: 0.7214 - f1score: 0.1069 - val_loss: 306.4081 - val_accuracy: 0.0753 - val_precisi
on: 0.0217 - val_recall: 1.0000 - val_auc: 0.0351 - val_f1score: 0.0426
Epoch 30/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 273.0998 - accuracy: 0.4547 -
precision: 0.0322 - recall: 0.9643 - auc: 0.4415 - f1score: 0.0624 - val_loss: 297.6060 - val_accuracy: 0.2603 - val_precisi
on: 0.0270 - val_recall: 1.0000 - val_auc: 0.2306 - val_f1score: 0.0526
Epoch 31/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 268.8303 - accuracy: 0.7435 -
precision: 0.0683 - recall: 1.0000 - auc: 0.7893 - f1score: 0.1279 - val_loss: 289.9391 - val_accuracy: 0.7671 - val_precisi
on: 0.0571 - val_recall: 0.6667 - val_auc: 0.8238 - val_f1score: 0.1053
Epoch 32/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 265.1663 - accuracy: 0.7495 -
precision: 0.0698 - recall: 1.0000 - auc: 0.8047 - f1score: 0.1305 - val_loss: 289.6119 - val_accuracy: 0.3562 - val_precisi
on: 0.0309 - val_recall: 1.0000 - val_auc: 0.3302 - val_f1score: 0.0600
Epoch 33/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 261.4362 - accuracy: 0.7428 -
precision: 0.0660 - recall: 0.9643 - auc: 0.8033 - f1score: 0.1236 - val_loss: 281.4286 - val_accuracy: 0.5616 - val_precisi
on: 0.0448 - val_recall: 1.0000 - val_auc: 0.6100 - val_f1score: 0.0857
Epoch 34/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 258.0620 - accuracy: 0.5695 -
precision: 0.0405 - recall: 0.9643 - auc: 0.5694 - f1score: 0.0777 - val_loss: 287.5359 - val_accuracy: 0.1233 - val_precisi
on: 0.0229 - val_recall: 1.0000 - val_auc: 0.0949 - val_f1score: 0.0448
Epoch 35/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 254.3706 - accuracy: 0.6125 -
precision: 0.0448 - recall: 0.9643 - auc: 0.6382 - f1score: 0.0856 - val_loss: 279.5475 - val_accuracy: 0.3630 - val_precisi
on: 0.0312 - val_recall: 1.0000 - val_auc: 0.3481 - val_f1score: 0.0606
Epoch 36/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 251.0329 - accuracy: 0.5467 -
precision: 0.0385 - recall: 0.9643 - auc: 0.5707 - f1score: 0.0741 - val_loss: 271.9737 - val_accuracy: 0.5616 - val_precisi
on: 0.0448 - val_recall: 1.0000 - val_auc: 0.6400 - val_f1score: 0.0857
Epoch 37/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 247.3493 - accuracy: 0.7213 -
precision: 0.0632 - recall: 1.0000 - auc: 0.7852 - f1score: 0.1189 - val_loss: 267.1379 - val_accuracy: 0.8082 - val_precisi
on: 0.0690 - val_recall: 0.6667 - val_auc: 0.8712 - val_f1score: 0.1250
Epoch 38/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 244.0961 - accuracy: 0.8355 -
precision: 0.0996 - recall: 0.9643 - auc: 0.8922 - f1score: 0.1806 - val_loss: 266.0392 - val_accuracy: 0.5548 - val_precisi
on: 0.0441 - val_recall: 1.0000 - val_auc: 0.6183 - val_f1score: 0.0845
Epoch 39/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 240.8325 - accuracy: 0.8254 -
precision: 0.0944 - recall: 0.9643 - auc: 0.8820 - f1score: 0.1720 - val_loss: 264.3241 - val_accuracy: 0.4384 - val_precisi
on: 0.0353 - val_recall: 1.0000 - val_auc: 0.4398 - val_f1score: 0.0682
Epoch 40/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 237.5602 - accuracy: 0.8032 -
precision: 0.0872 - recall: 1.0000 - auc: 0.8709 - f1score: 0.1605 - val_loss: 258.8141 - val_accuracy: 0.6438 - val_precisi
on: 0.0545 - val_recall: 1.0000 - val_auc: 0.7072 - val_f1score: 0.1034
Epoch 41/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 234.3241 - accuracy: 0.8972 -
precision: 0.1547 - recall: 1.0000 - auc: 0.9530 - f1score: 0.2679 - val_loss: 254.8574 - val_accuracy: 0.7329 - val_precisi
on: 0.0714 - val_recall: 1.0000 - val_auc: 0.7995 - val_f1score: 0.1333
Epoch 42/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 232.5647 - accuracy: 0.0819 -
precision: 0.0194 - recall: 0.9643 - auc: 0.0632 - f1score: 0.0380 - val_loss: 265.8920 - val_accuracy: 0.0274 - val_precisi
on: 0.0207 - val_recall: 1.0000 - val_auc: 0.0209 - val_f1score: 0.0405
Epoch 43/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 228.7223 - accuracy: 0.5594 -
precision: 0.0409 - recall: 1.0000 - auc: 0.5523 - f1score: 0.0787 - val_loss: 248.0632 - val_accuracy: 0.8493 - val_precisi
on: 0.0870 - val_recall: 0.6667 - val_auc: 0.9122 - val_f1score: 0.1538
Epoch 44/50
47/47 [==============================] - 58s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 225.5939 - accuracy: 0.7522 -
precision: 0.0684 - recall: 0.9643 - auc: 0.8036 - f1score: 0.1277 - val_loss: 246.2094 - val_accuracy: 0.6918 - val_precisi
on: 0.0625 - val_recall: 1.0000 - val_auc: 0.7777 - val_f1score: 0.1176
Epoch 45/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 222.5717 - accuracy: 0.7085 -
precision: 0.0606 - recall: 1.0000 - auc: 0.7702 - f1score: 0.1143 - val_loss: 243.6819 - val_accuracy: 0.5479 - val_precisi
on: 0.0435 - val_recall: 1.0000 - val_auc: 0.5839 - val_f1score: 0.0833
Epoch 46/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 219.7763 - accuracy: 0.8059 -
precision: 0.0857 - recall: 0.9643 - auc: 0.8709 - f1score: 0.1574 - val_loss: 243.2209 - val_accuracy: 0.4315 - val_precisi
on: 0.0349 - val_recall: 1.0000 - val_auc: 0.3987 - val_f1score: 0.0674
Epoch 47/50
```

```
47/47 [==============================] - 56s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 216.8608 - accuracy: 0.8261 -
precision: 0.0947 - recall: 0.9643 - auc: 0.8740 - f1score: 0.1725 - val_loss: 240.8487 - val_accuracy: 0.4384 - val_precisi
on: 0.0353 - val_recall: 1.0000 - val_auc: 0.4201 - val_f1score: 0.0682
Epoch 48/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 214.2628 - accuracy: 0.6911 -
precision: 0.0556 - recall: 0.9643 - auc: 0.7416 - f1score: 0.1051 - val_loss: 233.9505 - val_accuracy: 0.8425 - val_precisi
on: 0.1154 - val_recall: 1.0000 - val_auc: 0.8969 - val_f1score: 0.2069
Epoch 49/50
47/47 [==============================] - 56s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 211.5602 - accuracy: 0.7992 -
precision: 0.0831 - recall: 0.9643 - auc: 0.8410 - f1score: 0.1530 - val_loss: 244.4279 - val_accuracy: 0.0548 - val_precisi
on: 0.0213 - val_recall: 1.0000 - val_auc: 0.0371 - val_f1score: 0.0417
Epoch 50/50
47/47 [==============================] - 57s 1s/step - batch: 23.0000 - size: 31.6809 - loss: 209.3789 - accuracy: 0.4782 -
precision: 0.0336 - recall: 0.9643 - auc: 0.4689 - f1score: 0.0650 - val_loss: 232.0964 - val_accuracy: 0.5685 - val_precisi
on: 0.0455 - val_recall: 1.0000 - val_auc: 0.5836 - val_f1score: 0.0870
********** test_generator **********  <tensorflow.python.keras.preprocessing.image.DataFrameIterator object at 0x7f35185
d9150>
loss  =  239.81982930501303
accuracy  =  0.5494506
precision  =  0.024390243
recall  =  0.5
auc  =  0.58940643
f1score  =  0.046511628
train_model function ********** <tensorflow.python.keras.engine.functional.Functional object at 0x7f35185e7650>
********** single **********  <tensorflow.python.keras.engine.functional.Functional object at 0x7f35185e7650>
True (-)ves:  98
False (+)ves:  80
False (-)ves:  2
True (+)ves:  2
ERROR:tensorflow:==================================
Object was never used (type <class 'tensorflow.python.framework.ops.Operation'>):
<tf.Operation 'ROCCurveTestSet/write_summary/assert_non_negative/assert_less_equal/Assert/Assert' type=Assert>
If you want to mark it as used call its "mark_used()" method.
It was originally created here:
  File "/opt/tljh/user/lib/python3.7/site-packages/tensorflow/python/util/dispatch.py", line 201, in wrapper
    return target(*args, **kwargs)  File "/opt/tljh/user/lib/python3.7/site-packages/tensorflow/python/ops/check_ops.py", li
ne 947, in assert_less_equal
    np.less_equal, x, y, data, summarize, message, name)  File "/opt/tljh/user/lib/python3.7/site-packages/tensorflow/pytho
n/ops/check_ops.py", line 373, in _binary_assert
    return control_flow_ops.Assert(condition, data, summarize=summarize)  File "/opt/tljh/user/lib/python3.7/site-packages/t
ensorflow/python/util/dispatch.py", line 201, in wrapper
    return target(*args, **kwargs)  File "/opt/tljh/user/lib/python3.7/site-packages/tensorflow/python/util/tf_should_use.p
y", line 249, in wrapped
    error_in_function=error_in_function)
==================================
WARNING:tensorflow:From /opt/tljh/user/lib/python3.7/site-packages/tensorflow/python/util/deprecation.py:574: calling map_fn
_v2 (from tensorflow.python.ops.map_fn) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Use fn_output_signature instead
ERROR:tensorflow:==================================
Object was never used (type <class 'tensorflow.python.framework.ops.Operation'>):
<tf.Operation 'ConfusionMatrixTestSet/write_summary/assert_non_negative/assert_less_equal/Assert/Assert' type=Assert>
If you want to mark it as used call its "mark_used()" method.
It was originally created here:
  File "/opt/tljh/user/lib/python3.7/site-packages/tensorflow/python/util/dispatch.py", line 201, in wrapper
    return target(*args, **kwargs)  File "/opt/tljh/user/lib/python3.7/site-packages/tensorflow/python/ops/check_ops.py", li
ne 947, in assert_less_equal
    np.less_equal, x, y, data, summarize, message, name)  File "/opt/tljh/user/lib/python3.7/site-packages/tensorflow/pytho
n/ops/check_ops.py", line 373, in _binary_assert
    return control_flow_ops.Assert(condition, data, summarize=summarize)  File "/opt/tljh/user/lib/python3.7/site-packages/t
ensorflow/python/util/dispatch.py", line 201, in wrapper
    return target(*args, **kwargs)  File "/opt/tljh/user/lib/python3.7/site-packages/tensorflow/python/util/tf_should_use.p
y", line 249, in wrapped
    error_in_function=error_in_function)
==================================
This is model:  <tensorflow.python.keras.engine.functional.Functional object at 0x7f35185e7650>
This is test_metrics:  {'loss': 239.81982930501303, 'accuracy': 0.5494506, 'precision': 0.024390243, 'recall': 0.5, 'auc':
0.58940643, 'f1score': 0.046511628}
This is test_generator:  <tensorflow.python.keras.preprocessing.image.DataFrameIterator object at 0x7f35185d9150>
```



In [13]:

```
## SHAP Addition to project

from keras.applications.densenet import preprocess_input, decode_predictions
print(type(model))
print(type(test_generator))
```

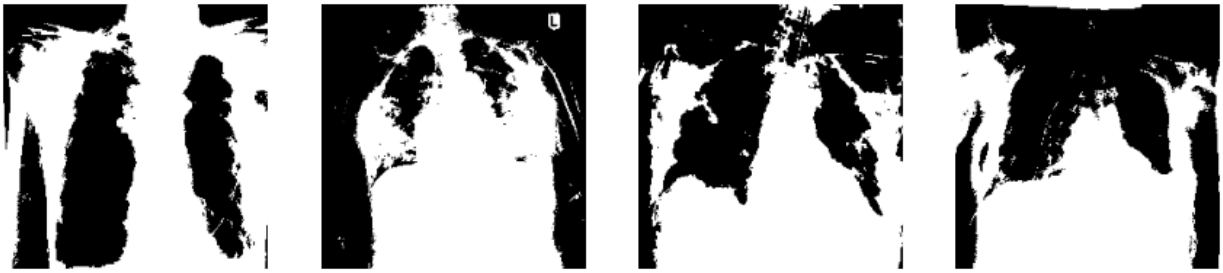
```
<class 'tensorflow.python.keras.engine.functional.Functional'>
<class 'tensorflow.python.keras.preprocessing.image.DataFrameIterator'>
```

In [14]:

```
# https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/
# Understand how to get images from ImageDataGenerator
data = {}
data['TRAIN'] = pd.read_csv(cfg['PATHS']['TRAIN_SET'])

# Create ImageDataGenerators
train_img_gen = ImageDataGenerator(preprocessing_function=remove_text,
                                   samplewise_center=True)

# Create DataFrameIterators
img_shape = (224, 224)
batch_size = 1
y_col = 'label_str'
class_mode = 'categorical'
train_generator = train_img_gen.flow_from_dataframe(dataframe=data['TRAIN'], directory=cfg['PATHS']
['RAW_DATA'],
    x_col="filename", y_col=y_col, target_size=img_shape, batch_size=batch_size,
    class_mode=class_mode, validate_filenames=False)


fig, ax = plt.subplots(nrows=1, ncols=4, figsize=(15,15))
```

```
for i in range(4):

  # convert to unsigned integers for plotting
  image = next(train_generator)[0]
    # .astype('uint8')
  X = image
  # changing size from (1, 200, 200, 3) to (200, 200, 3) for plotting the image
  image = np.squeeze(image)

  # plot raw pixel data
  ax[i].imshow(image)
  ax[i].axis('off')
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Found 1489 non-validated image filenames belonging to 2 classes.
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



In [15]:

```
print(type(X))
X
```

```
<class 'numpy.ndarray'>
```

Out[15]:

```
array([[[[  -3.1382904,   -3.1382904,   -3.1382904],
         [ -17.13829  ,  -17.13829  ,  -17.13829  ],
         [ -45.13829  ,  -45.13829  ,  -45.13829  ],
         ...,
         [ -14.13829  ,  -14.13829  ,  -14.13829  ],
         [  -8.13829  ,   -8.13829  ,   -8.13829  ],
         [  -1.1382904,   -1.1382904,   -1.1382904]],

        [[ -23.13829  ,  -23.13829  ,  -23.13829  ],
         [ -57.13829  ,  -57.13829  ,  -57.13829  ],
         [-119.13829  , -119.13829  , -119.13829  ],
         ...,
         [ -30.13829  ,  -30.13829  ,  -30.13829  ],
         [ -23.13829  ,  -23.13829  ,  -23.13829  ],
         [ -16.13829  ,  -16.13829  ,  -16.13829  ]],

        [[ -90.13829  ,  -90.13829  ,  -90.13829  ],
         [-126.13829  , -126.13829  , -126.13829  ],
         [-128.13829  , -128.13829  , -128.13829  ],
         ...,
         [ -51.13829  ,  -51.13829  ,  -51.13829  ],
         [ -45.13829  ,  -45.13829  ,  -45.13829  ],
         [ -41.13829  ,  -41.13829  ,  -41.13829  ]],

        ...,

        [[-101.13829  , -101.13829  , -101.13829  ],
         [-103.13829  , -103.13829  , -103.13829  ],
         [-105.13829  , -105.13829  , -105.13829  ],
         ...,
         [ -81.13829  ,  -81.13829  ,  -81.13829  ],
         [ -28.13829  ,  -28.13829  ,  -28.13829  ],
         [  22.86171  ,   22.86171  ,   22.86171  ]],

        [[ -98.13829  ,  -98.13829  ,  -98.13829  ],
         [-102.13829  , -102.13829  , -102.13829  ],
         [-105.13829  , -105.13829  , -105.13829  ],
         ...,
         [ -74.13829  ,  -74.13829  ,  -74.13829  ],
         [ -24.13829  ,  -24.13829  ,  -24.13829  ],
         [  25.86171  ,   25.86171  ,   25.86171  ]],

        [[ -97.13829  ,  -97.13829  ,  -97.13829  ],
```

```
        [-102.13829  , -102.13829  , -102.13829  ],
        [-104.13829  , -104.13829  , -104.13829  ],
        ...,
        [ -75.13829  ,  -75.13829  ,  -75.13829  ],
        [ -22.13829  ,  -22.13829  ,  -22.13829  ],
        [  27.86171  ,   27.86171  ,   27.86171  ]]]], dtype=float32)
```

In [16]:
```
train_img = data['TRAIN'].loc[:,'filename']
train_label = data['TRAIN'].loc[:,'label_str']
```

In [17]:
```
train_img[[4,5]]
```

Out[17]:
```
4    rsna/09629e2b-7f1e-499c-aab7-2bff196f034b.jpg
5    rsna/0c294ecf-23d6-4c56-b5e2-0482915cd102.jpg
Name: filename, dtype: object
```

In [18]:
```
data['TRAIN'].head()
```

Out[18]:

	Unnamed: 0	filename	label	label_str
0	601	covid-chestxray-dataset/images/1-s2.0-S1341321...	0	non-COVID-19
1	97	rsna/02285fa4-35b7-4af6-b88f-3cac45a7f5c8.jpg	0	non-COVID-19
2	783	covid-chestxray-dataset/images/16497_1_1.png	0	non-COVID-19
3	189	covid-chestxray-dataset/images/covid-19-infect...	0	non-COVID-19
4	665	rsna/09629e2b-7f1e-499c-aab7-2bff196f034b.jpg	0	non-COVID-19

In [ ]:
```

```

In [19]:
```
# load the covid-19 class names
import json
fname = '/home/ubuntu/covid-cxr/covid_fname_index.json'
with open(fname) as f:
    class_names = json.load(f)

print(class_names)
```

```
{'0': ['0', 'COVID-19'], '1': ['1', 'non-COVID-19']}
```

TESTing variable that matches the example from - https://shap.readthedocs.io/en/latest/example_notebooks/gradient_explainer/Explain%20an%20Intermediate%20Layer%20of%20VGG16%20on%20ImageNet.html

In [20]:
```
# Testing variables and see Layers descriptions
print("fname", fname)

for i in range(29):
    print("modelayer", i, model.layers[i].input)

# pick layer to use for Shap
layer = 6
```

```
fname /home/ubuntu/covid-cxr/covid_fname_index.json
modelayer 0 Tensor("input_1:0", shape=(None, 224, 224, 3), dtype=float32)
modelayer 1 Tensor("input_1:0", shape=(None, 224, 224, 3), dtype=float32)
modelayer 2 Tensor("conv0_0/BiasAdd:0", shape=(None, 224, 224, 16), dtype=float32)
modelayer 3 Tensor("batch_normalization/cond/Identity:0", shape=(None, 224, 224, 16), dtype=float32)
modelayer 4 Tensor("leaky_re_lu/LeakyRelu:0", shape=(None, 224, 224, 16), dtype=float32)
modelayer 5 [<tf.Tensor 'conv0_1/BiasAdd:0' shape=(None, 224, 224, 16) dtype=float32>, <tf.Tensor 'input_1:0' shape=(None, 2
24, 224, 3) dtype=float32>]
modelayer 6 Tensor("concat0/concat:0", shape=(None, 224, 224, 19), dtype=float32)
modelayer 7 Tensor("batch_normalization_1/cond/Identity:0", shape=(None, 224, 224, 19), dtype=float32)
modelayer 8 Tensor("leaky_re_lu_1/LeakyRelu:0", shape=(None, 224, 224, 19), dtype=float32)
modelayer 9 Tensor("max_pooling2d/MaxPool:0", shape=(None, 112, 112, 19), dtype=float32)
modelayer 10 Tensor("conv1_0/BiasAdd:0", shape=(None, 112, 112, 48), dtype=float32)
modelayer 11 Tensor("batch_normalization_2/cond/Identity:0", shape=(None, 112, 112, 48), dtype=float32)
modelayer 12 Tensor("leaky_re_lu_2/LeakyRelu:0", shape=(None, 112, 112, 48), dtype=float32)
modelayer 13 [<tf.Tensor 'conv1_1/BiasAdd:0' shape=(None, 112, 112, 48) dtype=float32>, <tf.Tensor 'max_pooling2d/MaxPool:0'
shape=(None, 112, 112, 19) dtype=float32>]
modelayer 14 Tensor("concat1/concat:0", shape=(None, 112, 112, 67), dtype=float32)
modelayer 15 Tensor("batch_normalization_3/cond/Identity:0", shape=(None, 112, 112, 67), dtype=float32)
modelayer 16 Tensor("leaky_re_lu_3/LeakyRelu:0", shape=(None, 112, 112, 67), dtype=float32)
```

```
modelayer 17 Tensor("max_pooling2d_1/MaxPool:0", shape=(None, 56, 56, 67), dtype=float32)
modelayer 18 Tensor("conv2_0/BiasAdd:0", shape=(None, 56, 56, 144), dtype=float32)
modelayer 19 Tensor("batch_normalization_4/cond/Identity:0", shape=(None, 56, 56, 144), dtype=float32)
modelayer 20 Tensor("leaky_re_lu_4/LeakyRelu:0", shape=(None, 56, 56, 144), dtype=float32)
modelayer 21 [<tf.Tensor 'conv2_1/BiasAdd:0' shape=(None, 56, 56, 144) dtype=float32>, <tf.Tensor 'max_pooling2d_1/MaxPool:
0' shape=(None, 56, 56, 67) dtype=float32>]
modelayer 22 Tensor("concat2/concat:0", shape=(None, 56, 56, 211), dtype=float32)
modelayer 23 Tensor("batch_normalization_5/cond/Identity:0", shape=(None, 56, 56, 211), dtype=float32)
modelayer 24 Tensor("leaky_re_lu_5/LeakyRelu:0", shape=(None, 56, 56, 211), dtype=float32)
modelayer 25 Tensor("max_pooling2d_2/MaxPool:0", shape=(None, 28, 28, 211), dtype=float32)
modelayer 26 Tensor("flatten/Reshape:0", shape=(None, 165424), dtype=float32)
modelayer 27 Tensor("dropout/cond/Identity:0", shape=(None, 165424), dtype=float32)
modelayer 28 Tensor("dense/BiasAdd:0", shape=(None, 128), dtype=float32)
```

In [21]:

```
print(type(X))
print(X.dtype)
print(X.flags)
print(X.shape)
print(X.ndim)
print(X.size)
print(X.itemsize)
print(X.flags)
print(X.strides)
```

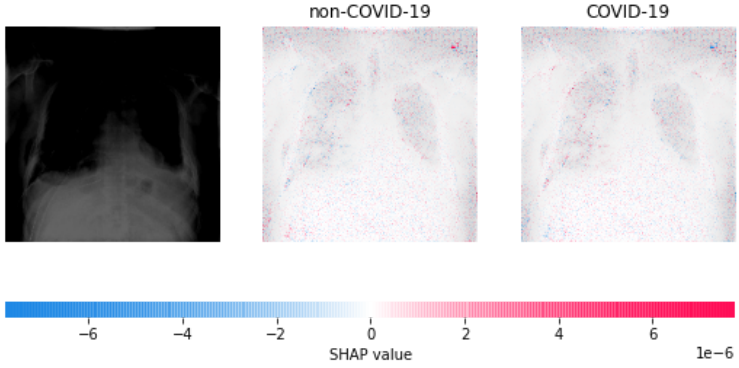
```
<class 'numpy.ndarray'>
float32
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False

(1, 224, 224, 3)
4
150528
4
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False

(602112, 2688, 12, 4)
```

In [22]:

```
# SHAP processing


to_explain = X


#print("preprocessinput", preprocess_input(X.copy()))


# explain how the input to the 7th layer of the model explains the top two classes
def map2layer(x, layer):
    feed_dict = dict(zip([model.layers[0].input], [preprocess_input(x.copy())]))
    #print("feed_dict", feed_dict)
    return K.get_session().run(model.layers[layer].input, feed_dict)


#print("modelayer", model.layers[0].input)
#print("modelayer7", model.layers[layer].input)
#print("modelayer-1", model.layers[-1].output)
#print("map_toexplain", map2layer(to_explain, layer))


e = shap.GradientExplainer((model.layers[layer].input, model.layers[-1].output),
map2layer(preprocess_input(X.copy()), layer))
shap_values, indexes = e.shap_values(map2layer(to_explain, layer), ranked_outputs=2)


#print("shapvalues", type(shap_values), shap_values)
```

```
#print("shap index", type(indexes), indexes)

# get the names for the classes
index_names = np.vectorize(lambda x: class_names[str(x)][1])(indexes)
print("index names", type(index_names), index_names)

# plot the explanations
shap.image_plot(shap_values, to_explain, index_names)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
index names <class 'numpy.ndarray'> [['non-COVID-19' 'COVID-19']]
```



In [ ]:

In [ ]: