PneumoStack: A Novel Approach to Automated Pneumonia and COVID-19 Diagnosis with Chest X-Ray Analysis via Convolutional Neural Networks and Stacked Generalization

Sonya Jin

Table of Contents

1. Abstract

Pneumonia is the single largest infectious cause of death in children worldwide, accounting for 15% of all deaths of children under 5 years old. Regarding the current pandemic, chest X-ray (CXR) analysis is needed to rectify false negatives from RT-PCR in COVID diagnosis, emphasizing the need to improve diagnostic accuracy. As CXRs are the principal diagnostic tool for pneumonia, automating medical image analysis with medical image classification can aid radiologists in expediting and improving the diagnostic process. Research in deep learning for medical image analysis has utilized individual transfer learning neural networks as well as neural network ensembles constructed by means such as bootstrap aggregation and soft-voting. This study presents a novel stacked model for CXR analysis composed of three CNN architectures: InceptionResNetV2, Xception, and ResNet50. All three pre-trained models were trained on a chest X-ray dataset for binary classification and multi-class classification and ensembled via stacked generalization into a neural network meta-learner. The proposed stacked model (Pneumostack) achieved an accuracy of 95.4% in three-category classification (COVID-19, non-COVID pneumonia, and normal) and 99.8% in binary classification (normal and pneumonia), outperforming any one of its single constituent classifiers and other models presented in current literature. Surpassing existing transfer learning models and ensembles, Pneumostack opens doors to higher performance in automated CXR analysis and other CNN applications in medicine.

2. Introduction

Pneumonia is a lung infection that causes the alveoli of the lungs to fill with pus, causing symptoms such as coughing, difficulty breathing, and fever [2]. It can be caused by bacteria, *Streptococcus pneumoniae* being the most common, and viruses, including SARS-CoV-2 [2]. Complications of pneumonia, if the disease is left untreated, include acute respiratory distress (ARDS), respiratory failure, necrotizing pneumonia, pleural disorders, organ damage, and sepsis [2]. Chest X-rays (CXRs) are the principal diagnostic tool, or the "gold standard", for pneumonia diagnosis [1]. Lobar and lobular consolidation are characteristic of bacterial pneumonia, while interstitial opacities are characteristic of viral pneumonia [3]. Automated diagnosis methods can extract these characteristic features, minimizing false predictions from human intervention. In the event that trained radiologists are limited, automated diagnosis can reduce child mortality rates in regions where pneumonia is most prevalent - South Asia and sub-Saharan Africa [1]. COVID-19 As Chest X-rays (CXR) are the principal diagnostic tool for pneumonia [2], automating medical image analysis with medical image classification can aid radiologists in expediting and improving the diagnostic process in time and accuracy.

Coronavirus disease (COVID-19) is an infectious disease that causes mild to moderate respiratory illness [28]. In rare cases, COVID-19 can lead to severe respiratory problems, kidney failure, or death [28]. Currently, the principal diagnostic method is the reverse-transcription polymerase chain reaction (RT-PCR) laboratory test that detects RNA specific to the SARS-CoV-2 virus with the nasopharyngeal or oropharyngeal swab [29]. Additionally, patterns of COVID-19 can be identified on CXRs. Reported typical radiological findings include multifocal and bilateral ground glass opacities and consolidations with peripheral and basal predominance [30]. Unique features of COVID-19 pneumonia are peripheral air space opacities and bilateral lower lobe consolidations with lower-lung distribution [31]. Recent reports have revealed that RT-PCR has a sensitivity as low as 60%-71% for detecting COVID-19, while CXRs have a sensitivity of 69% [4], presenting the possibility for CXR analysis rectifying false negative findings in RT-PCR in COVID-19 diagnosis and the need to improve CXR analysis accuracy.

2.1. Relevant Work

CNNs have been at the forefront of automated CXR analysis research in effort to detect pneumonia and COVID-19. Wang et al. [16] constructed COVID-Net, a tailored CNN for the detection of COVID-19 with a projection-expansion-projection-extension (PEPX) design pattern. With three classes (non-COVID pneumonia, COVID-19, normal), the model achieved an accuracy of 93.3%. Apostolopoulos et al. [17] used VGG-19 as a base model for three classes and achieved an accuracy of 87%. Umer et al. [18] proposed COVINet, a CNN approach with three convolutional layers, a max pooling layer, an average pooling layer, and four FC layers. COVINet achieved an accuracy of 89.9% with three classes. Nishio et. al. [19] used VGG-16 for the detection of three classes and achieved an accuracy of 83.68% with a combination of data augmentation methods - conventional and mixup. For binary classification, many approaches were proposed, such as the MADE-based CNN [20] with 92.55% accuracy, Deep CNN [23] with 93% accuracy, and a weighted voting ensemble [33] with a 72.26% accuracy.

2.2. Aim

In contrast to other ensembling methods and the use of individual transfer learning models, the aim of this study is to present a stacked convolutional neural network meta-learner of transfer learning CNNs with stacked generalization in effort to achieve higher performance than any one of its constituent classifiers and existing individual models in binary and multiclass pneumonia CXR classification.

2.3. Convolutional Neural Networks

Convolutional neural networks (CNNs), a deep learning algorithm, has shown unsurpassed success in varying image classification tasks due to their capabilities of automated unsupervised feature extraction and dimensionality reduction, making it suitable for CXR analysis [3]. A CNN consists of an input layer, hidden convolutional layers, ReLU layers, pooling layers, fully-connected (FC) layers, and an output layer. The convolutional layer applies a convolution operation to the input from a subarea of the previous layer, passing the generated feature map on to the next layer. The ReLU layer then applies an activation function on the passed feature map to increase non-linearity in the network, removing negative values from the map by setting them to zero. Pooling downsamples the detection of feature maps and decreases training time. Finally, the FC layers drive the final classification predictions by taking the output of the hidden layers and giving the final probabilities for each label [3]. A CNN requires less data preprocessing and reaches better results than other classification algorithms due to its capability of capturing spatial and temporal dependencies in an image [3]. Moreover, CNNs convolve learned features with input data through 2D convolutional layers to extract high-level features, making this network ideal for processing 2D images, such as CXRs.

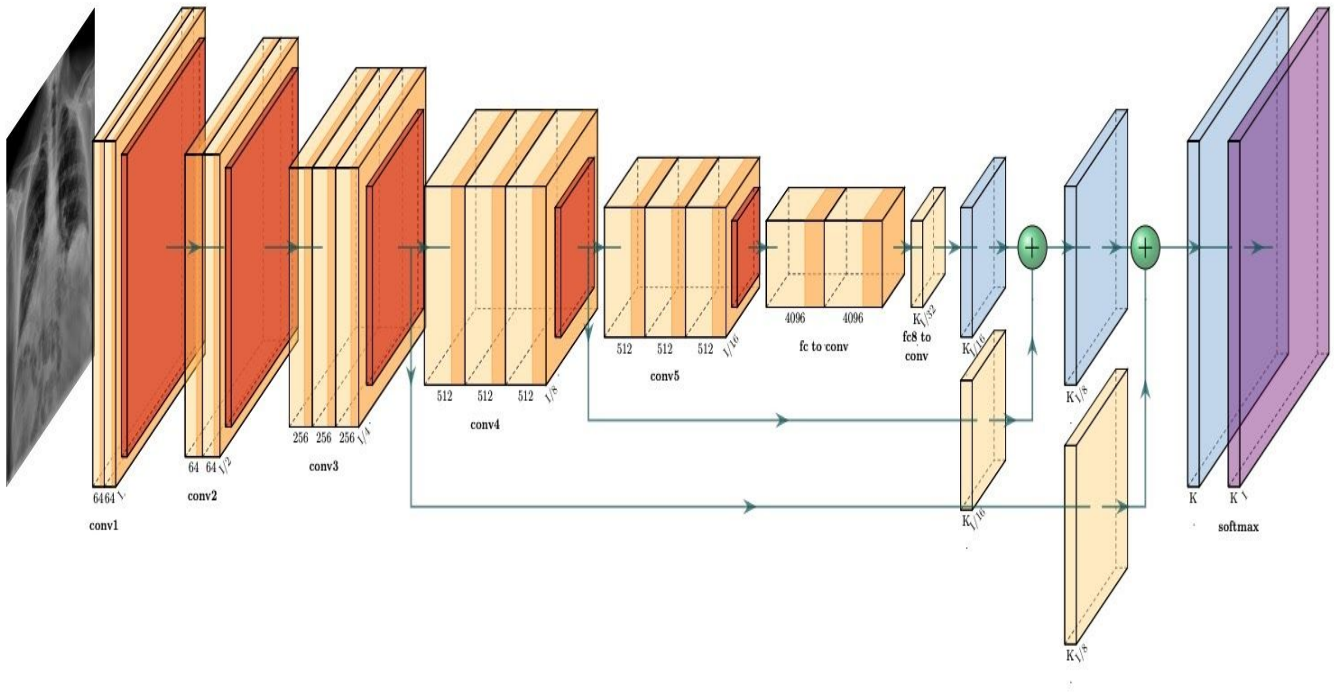


Figure 1. *Sample CNN Architecture*

2.4. Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as a starting point for another task [6]. In the application for image classification, transfer learning models are sourced from base models pre-trained on the ImageNet 1000-class classification competition with over 1,000,000 images [6]. With transfer learning, one can achieve higher accuracy with a small dataset as the pre-trained weights can already recognize generic image features in earlier layers, eliminating the need to train a network from scratch with suboptimal weights. The source model is fine-tuned with the train-freezing of later layers that are more dataset-specific and then trained on the dataset. The main benefits of transfer learning include a higher y-intercept, slope, and asymptote in performance [8].

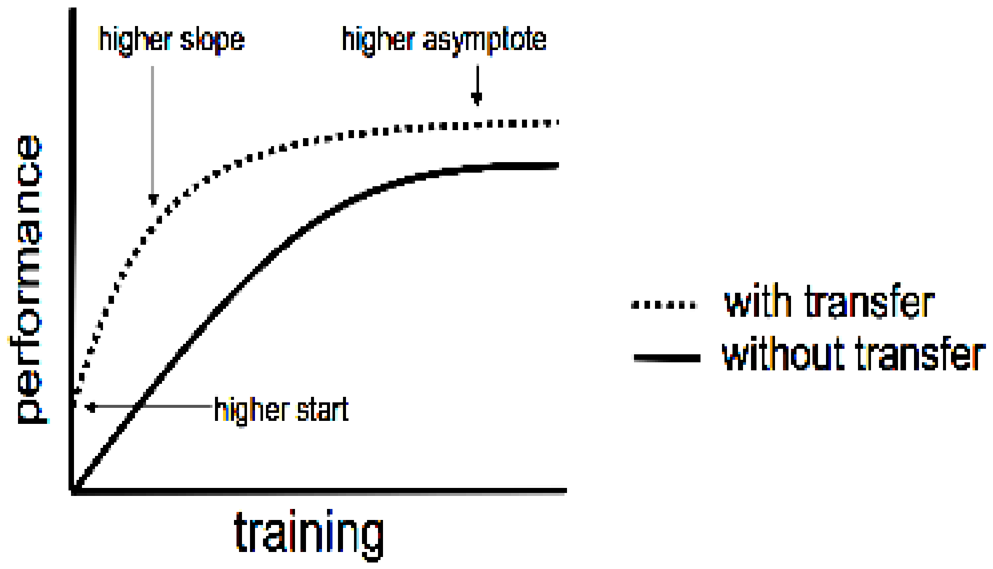


Figure 2. *The benefits of transfer learning* [8]

2.4.1. Xception

Xception is a convolutional neural network architecture composed of a linear stack of depthwise separable convolution layers with residual connections [10]. The architecture has 36 convolutional layers forming the feature extraction base which are structured into 14 models with outlined linear residual connections [10]. This model outperformed VGGNet, ResNet, and InceptionV3 in ImageNet with 94.5% [15].

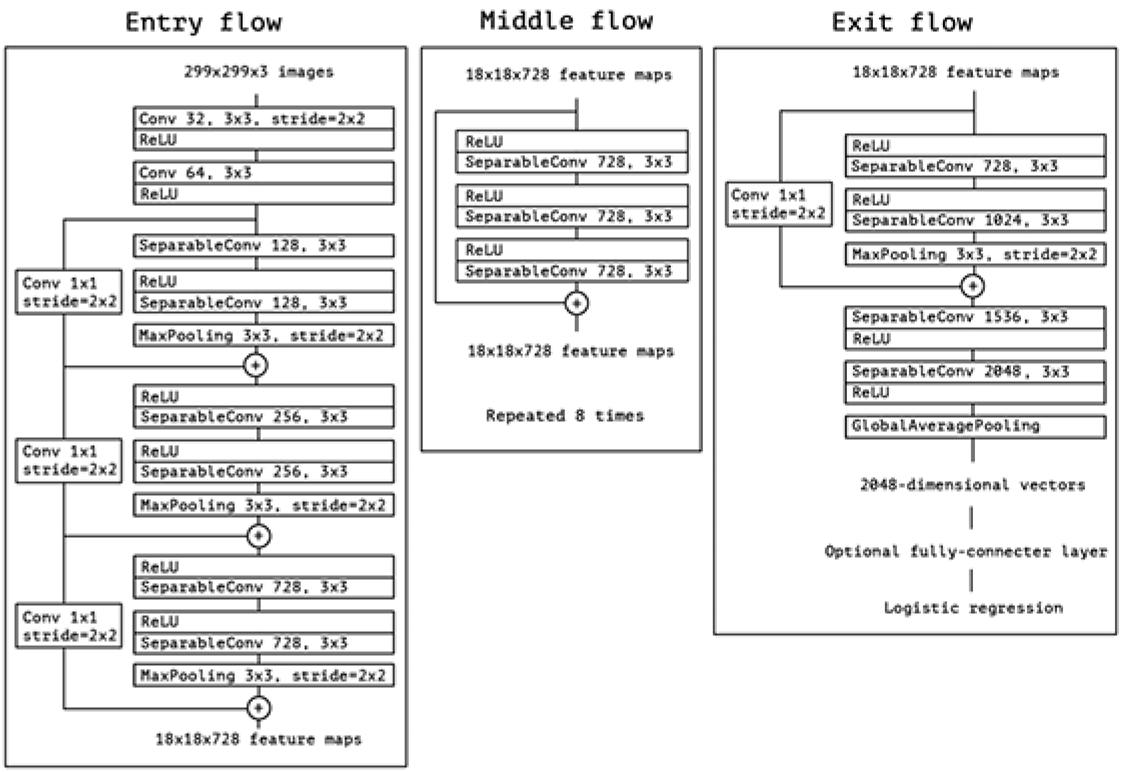


Figure 3. *Xception Architecture* [10]

2.4.2. InceptionResNetV2

Similarly, InceptionResNetV2 combines the Inception architecture with residual connections that replaces the filter concatenation stage of Inception. Each Inception block is followed by a filter expansion layer that scales up the dimensionality of the filters. Inception-ResNetV2 also has batch-normalization only on top of the traditional Inception layers, but not on top of the summations to increase the overall number of Inception blocks [14]. This model outperformed InceptionV3 and ResNet152 on ImageNet with a 94.6% performance.

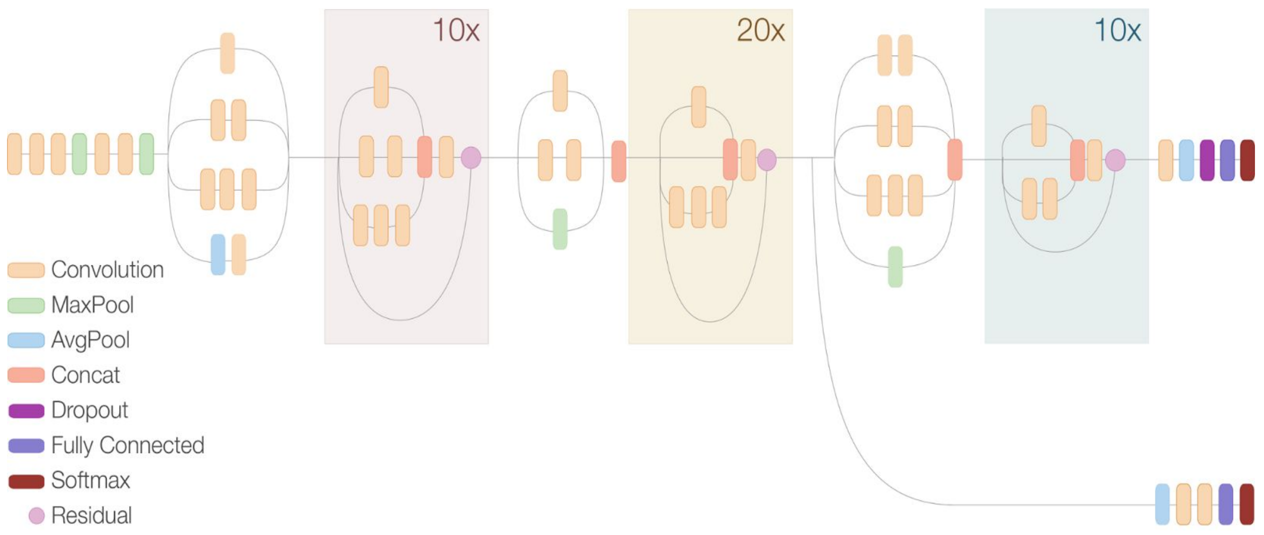


Figure 4. *Compressed View of InceptionResNetV2 Architecture* [14]

2.4.3. ResNet50

The ResNet50 architecture introduces the concept of skip-wise connections, which allows the training of extremely deep neural networks with 50+ layers successfully without degradation [13]. Previously, this was impossible due to the vanishing gradient problem that persists as more layers are added to a neural network, abruptly degrading performance [13]. The model has 48 convolution layers along with a max pooling layer and an average pooling layer [13]. ResNet50 was the winner of ImageNet 2015 with a 93% accuracy [15].

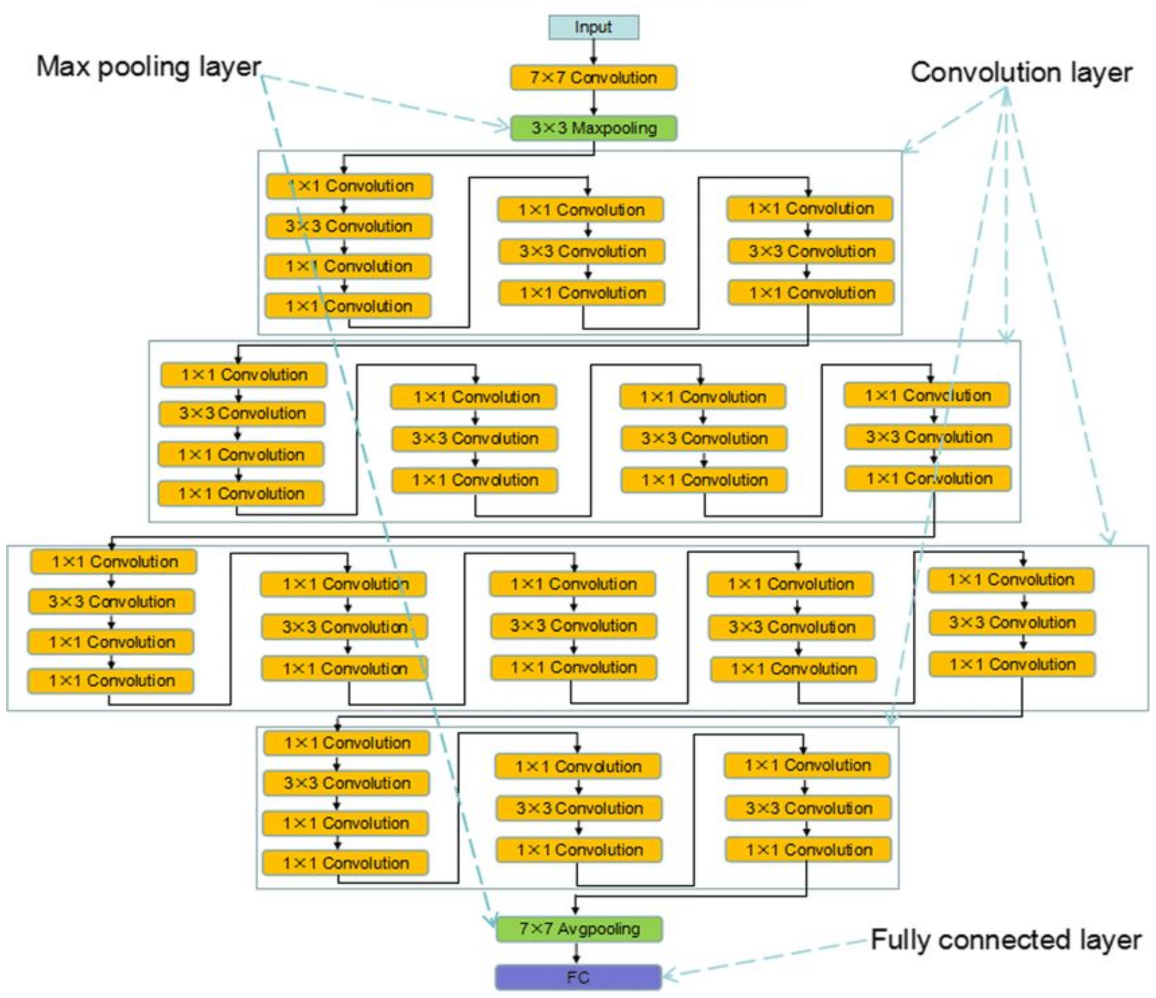


Figure 5. *ResNet50 Architecture* [13]

2.5. Ensemble learning

Ensemble learning is a method used to maximize detection performance by combining the results of single constituent algorithms [4]. The purpose of ensemble learning is to harness the capabilities of a range of well-performing models on a classification task and manipulate the predictions of the models to construct an ensemble that outperforms any single model in the ensemble [9]. One frequently used ensemble method is bootstrap aggregating, which involves the creation of random samples of training data with replacement [32]. A model is then built for each sample, and the results of the multiple models are combined with average or majority voting [32]. Another method is boosting - an iterative technique that adjusts the weight of an observation considering the preceding classification - which decreases bias error [32]. However, a drawback of this method is that it tends to overfit the training data [32]. Furthermore, stacked generalization is an ensembling method that involves constructing a meta-model that trains on the predictions made by its base models on out-of-sample data [9]. The base models (Level-0 models) fit on the training data, and the predictions are compiled. The meta-model (Level-1 model) then learns how to best combine the predictions of the base models [9]. To reap the benefits of different CNN architectures, stacked generalization was the ensembling method of choice for this study.

3. Methods

3.1. Dataset

In this work, the Cohen et. al COVID-19 Image Data Collection [11] dataset was modified and used. The 5829-image dataset comprises of a collection of COVID-19 (461), non-COVID-19 viral pneumonia (1414), bacterial pneumonia (2521), and normal (1433) X-rays collected at the Guangzhou Women and Children's Medical Center. The dataset also contains X-ray images of the fungal *Pneumocystis* pneumonia and lipoid pneumonia, but these images were removed for this study as they are not caused by viral or bacterial strains.

.

Label	Number of Images
Normal	1433
Viral Pneumonia Viruses included: *Influenza, SARS*	1414
Bacterial Pneumonia Bacteria included: *Chlamydia pneumoniae, Streptococcus pneumoniae, E. coli, Klebsiella pneumoniae, Legionella, Mycoplasma pneumoniae*	2521
COVID-19	461

Table 1. *Number of images by label in dataset*

Figure 6 shows samples of a normal, viral pneumonia, bacterial pneumonia, and COVID-19 scan. The scans are as follows: (A) Normal scan, (B) Viral pneumonia, (C) Bacterial pneumonia, (D) COVID-19 pneumonia.

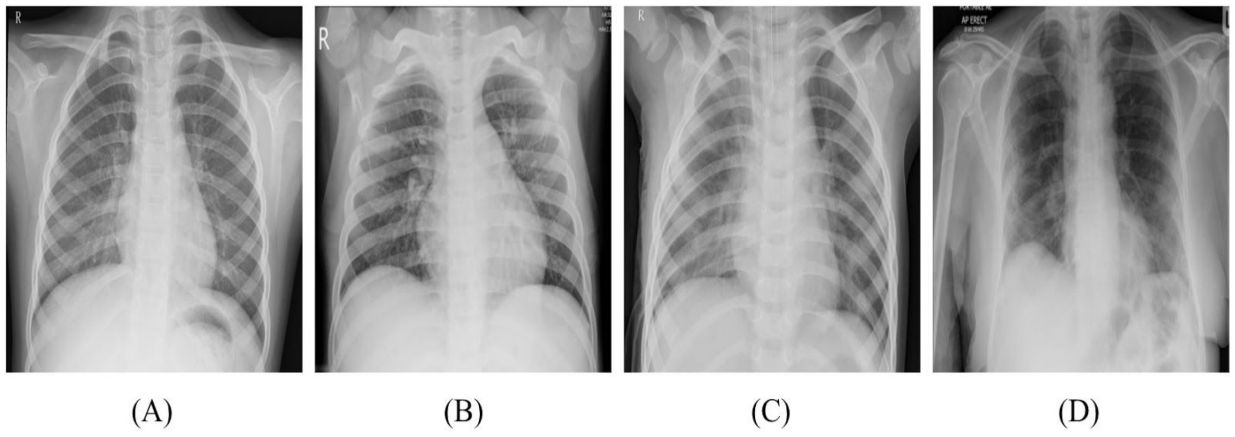


Figure 6. *Data samples*

3.2. Data Augmentation

Augmentation can aid in the transform invariant approach of feature-learning in CNNs with the inclusion of invariant transformations [34]. The dataset was augmented with four transformations: horizontal flip, rotation, vertical shift, and horizontal shift. The images were horizontally shifted by 10%, vertically shifted by 10%, rotated by 15 degrees clockwise, and flipped along the horizontal axis. Figure 3 shows nine images with the applied transformations specified above at random.

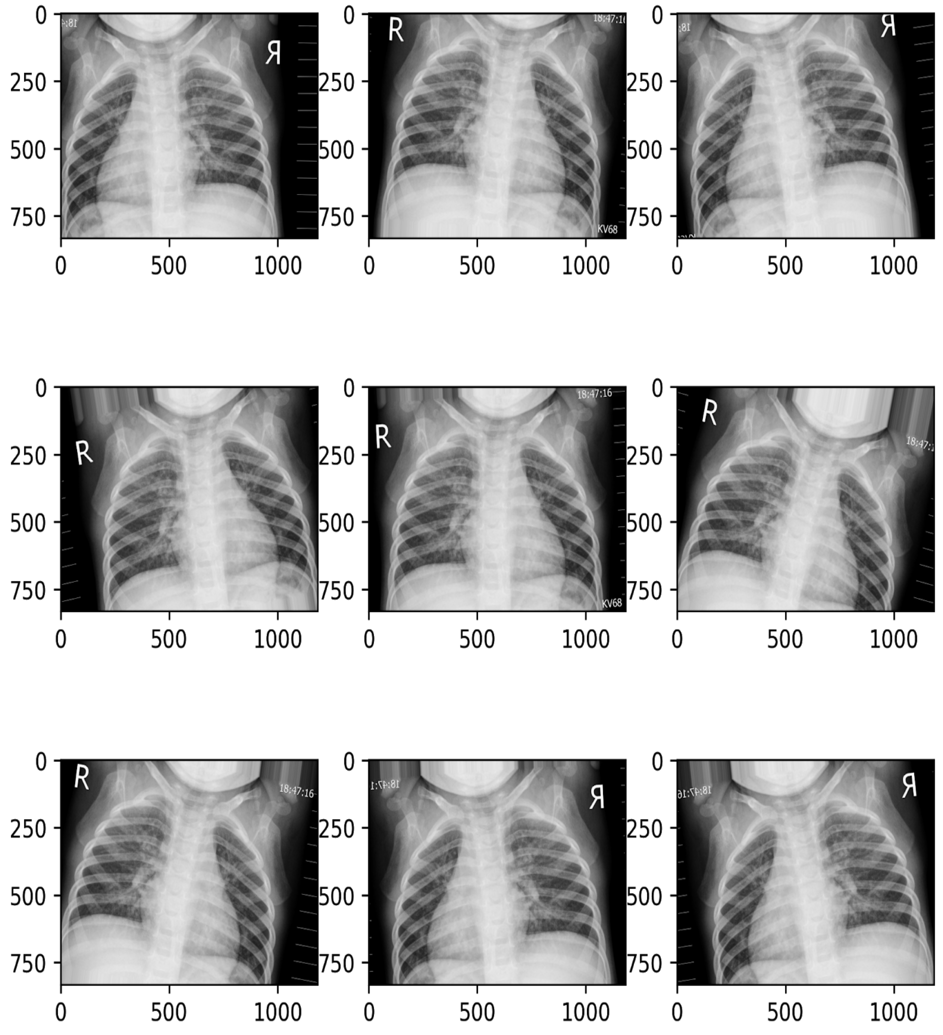


Figure 7. *Nine images with randomly applied data augmentation techniques*

3.3. Implementation of Transfer Learning

Xception, InceptionResNetV2, and ResNet50 were constructed and pre-trained weights were loaded from ImageNet. The first 10 layers were frozen. After the convolution layers of the source model, a global max pooling layer, three dropout layers, and three FC Dense layers were added. Rectified linear unit was used as the activation function for the first FC layer. In binary classification, the activation function of the last FC layer was sigmoid. In three-category classification, softmax was used in place of sigmoid. All models were compiled with Adam optimization. Binary and categorical cross-entropy were used to calculate loss for binary and three-class classification, respectively, as defined below:

$$CE = -\sum_{i=1}^{C'=2} t_i log(s_i) = -t_1 log(s_1) - (1 - t_1) log(1 - s_1)$$

Figure 8. *Binary cross-entropy*

Where:

- C_1 and C_2 are the two classes (pneumonia vs. normal)
- t_1 [0,1] and s_1 are the ground truth and the score for C_1
- t_2 = 1 - t1 and s_2 = 1 - s_2; they are the groundtruth and the score for C_2

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

Figure 9. *Categorical cross-entropy*

Where:

- C is the number of classes
- s_p is the predicted score for the positive class

In training, callbacks such as ModelCheckpoint, EarlyStopping, and ReduceLROnPlateau were used to prevent the model from overfitting on training data.

3.4. Stacked Model

All layers in each of the ensemble's individual models were frozen. A dataset with the predictions of Xception, InceptionResNetV2, and ResNet50 was constructed after the training of individual models. A sequential CNN architecture was used to construct the stacked model with a Flatten layer that takes 3 inputs for binary classification and 9 inputs for three-category classification, a FC Dense layer with ReLU activation, and a FC Dense layer with sigmoid (binary)/softmax (three-category) activation. The model was compiled with Adam optimization and either binary or categorical cross entropy.

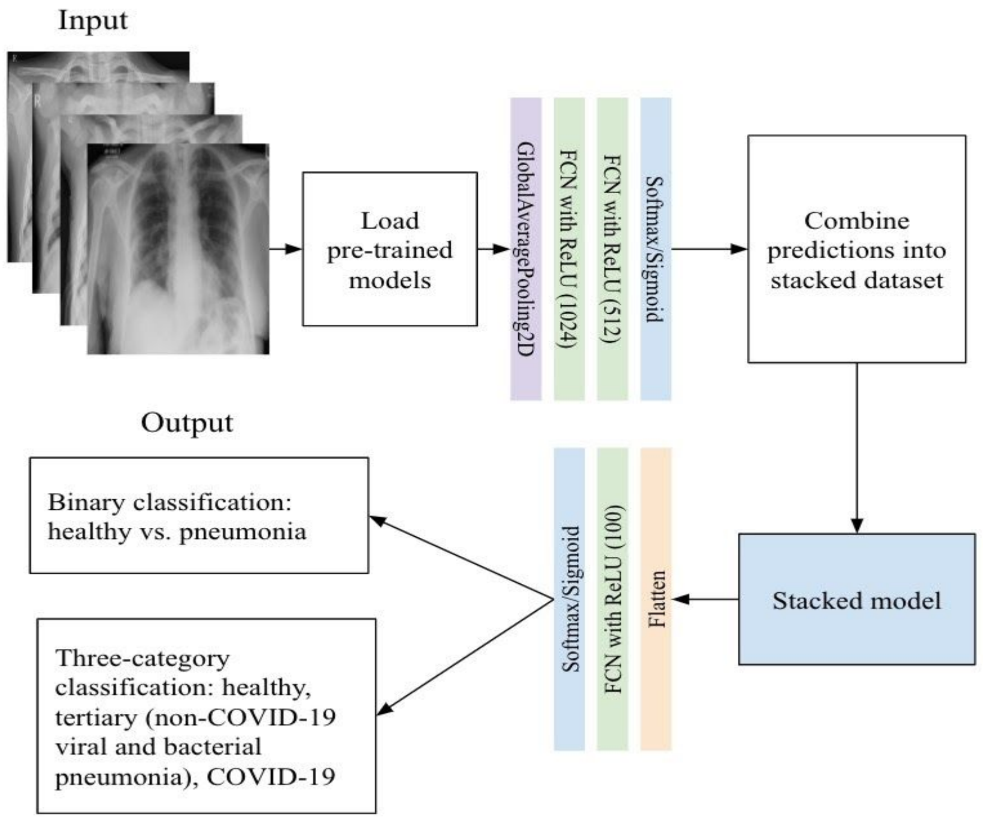

Input
Load pre-trained models
GlobalAveragePooling2D
FCN with ReLU (1024)
FCN with ReLU (512)
Softmax/Sigmoid
Combine predictions into stacked dataset
Stacked model
Flatten
FCN with ReLU (100)
Softmax/Sigmoid
Output
Binary classification: healthy vs. pneumonia
Three-category classification: healthy, tertiary (non-COVID-19 viral and bacterial pneumonia), COVID-19

Figure 10. *Schematic diagram of proposed model*

3.5. Performance Metrics

To avoid the accuracy paradox in binary classification, multiple performance metrics besides accuracy were used to evaluate the individual models and the stacked model. The performance metrics used for binary classification are the Sørensen–Dice coefficient, AUC and accuracy for binary classification. Accuracy, precision, and recall were used for three-category classification.

The Sørensen–Dice coefficient (F_1 score) is the weighted average of precision and recall and is a popular metric for binary classification. The highest value is 1.0, indicating perfect precision and recall, and the lowest possible value is 0. AUC [0,1] represents the degree of separability and measures the model's capability of distinguishing between classes by computing the area under the receiver-operating characteristic (ROC) curve. Precision quantifies the number of true positive class predictions, while recall is the percentage of true predictions classified. Accuracy is the fraction of correct predictions over the total number of predictions.

$$Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

$$Precision = \frac{T_p}{T_p + F_p}$$

$$Recall = \frac{T_p}{T_p + T_n}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Figure 11. *Definition of Performance Metrics*

Where:

- T_p = number of true positives

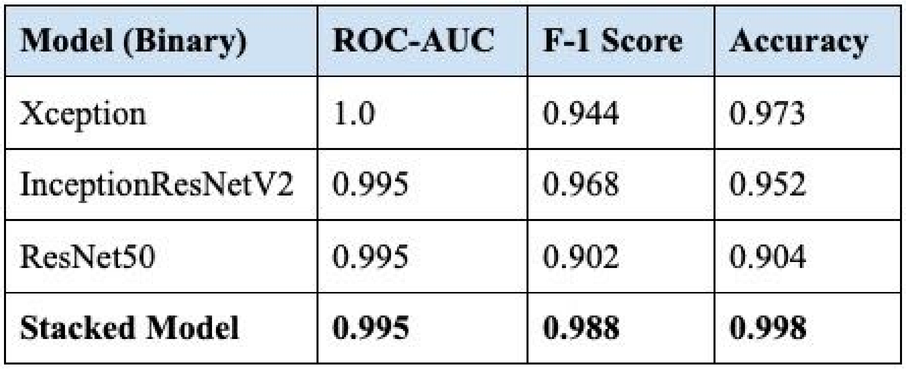
- T_n = number of true negatives
- F_p = number of false positives
- F_n = number of false negatives

3.6. Operating System

Deep learning models were constructed, trained, and evaluated on Google Colaboratory with 1xTesla K80 GPU, 2496 CUDA cores, 12GB GDDR5 VRAM, 2vCPU @2.3Ghz, 12.6 GB RAM, and 64 GB disk space.

4. Results

4.1. Binary Classification Results



Model (Binary)	ROC-AUC	F-1 Score	Accuracy
Xception	1.0	0.944	0.973
InceptionResNetV2	0.995	0.968	0.952
ResNet50	0.995	0.902	0.904
Stacked Model	**0.995**	**0.988**	**0.998**

Table 2. *Binary classification results*

The stacked model outperformed all constituent classifiers - Xception, InceptionResNetV2, and ResNet50 in binary classification with an accuracy 0.998 and a Sørensen–Dice coefficient of 0.988.

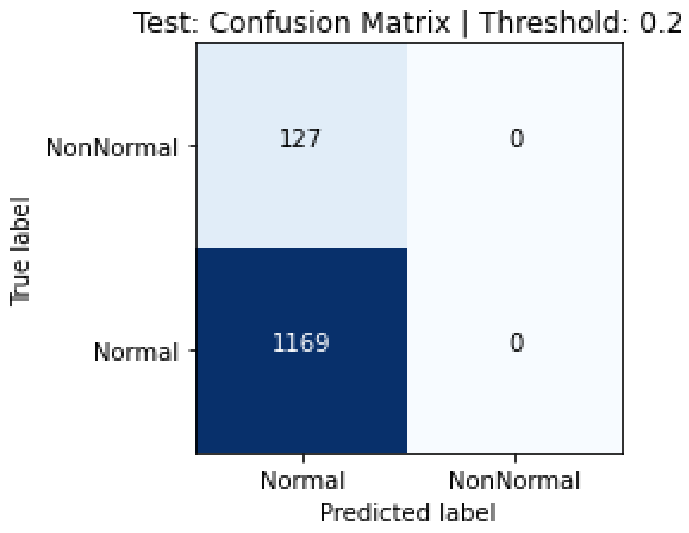


Figure 12. *Stacked model binary classification unnormalized confusion matrix*

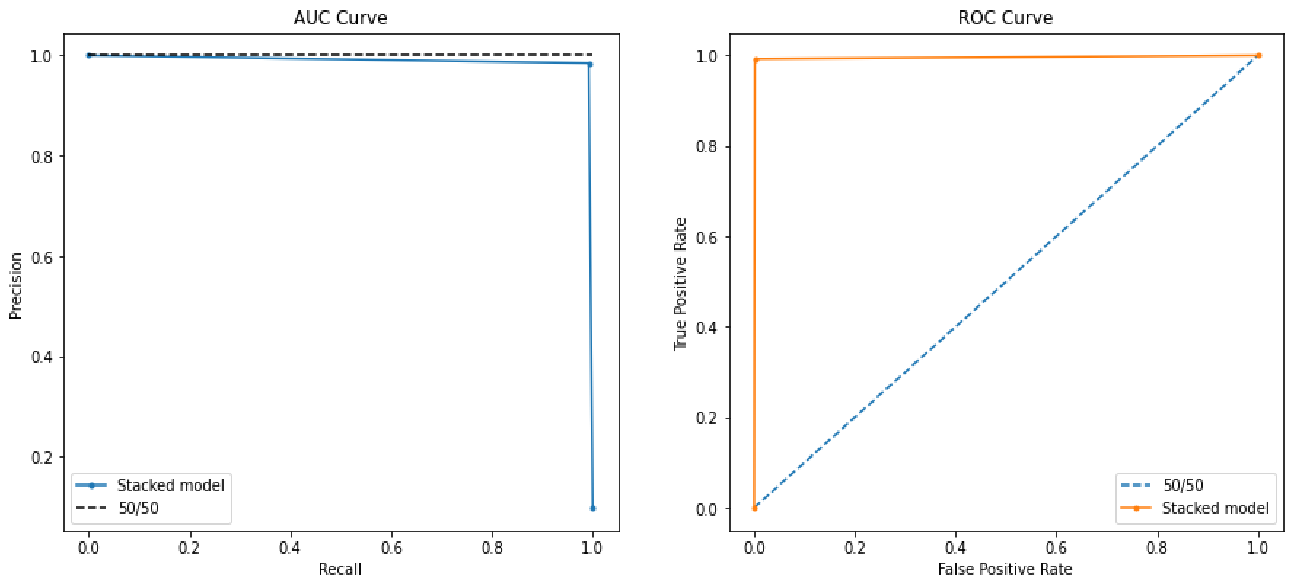


Figure 13. *Stacked model binary classification Precision-Recall and ROC Curves*

4.2. Three-Category Classification Results

Model (Three-class)	**Precision**	**Recall**	**Accuracy**
Xception	0.922	0.939	0.942
InceptionResNetV2	0.901	0.940	0.944
ResNet50	0.884	0.878	0.890
Stacked Model	**0.911**	**0.953**	**0.954**

Table 3. *Stacked model three-category classification results*

The stacked model outperformed all constituent classifiers - Xception, InceptionResNetV2, and ResNet50 in three-category classification with an accuracy of 0.954.

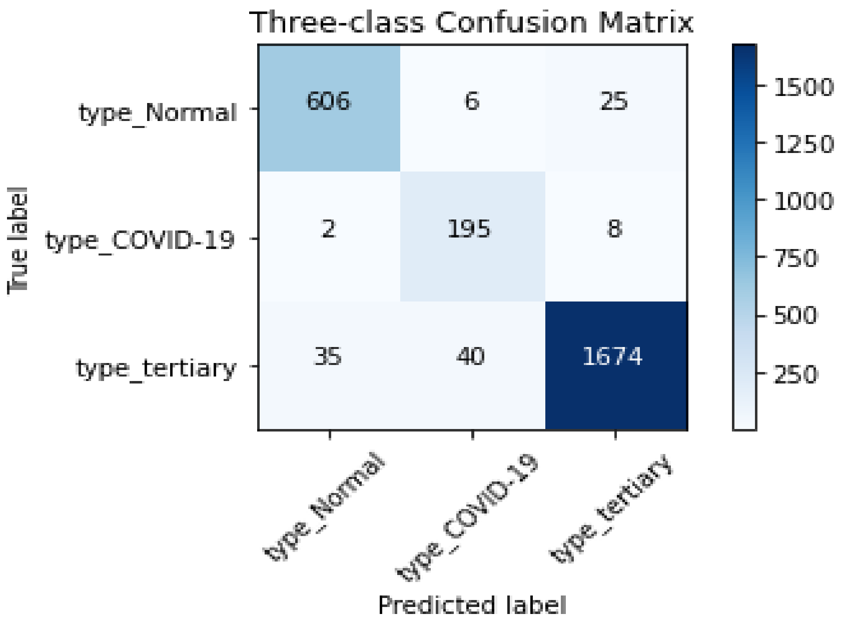
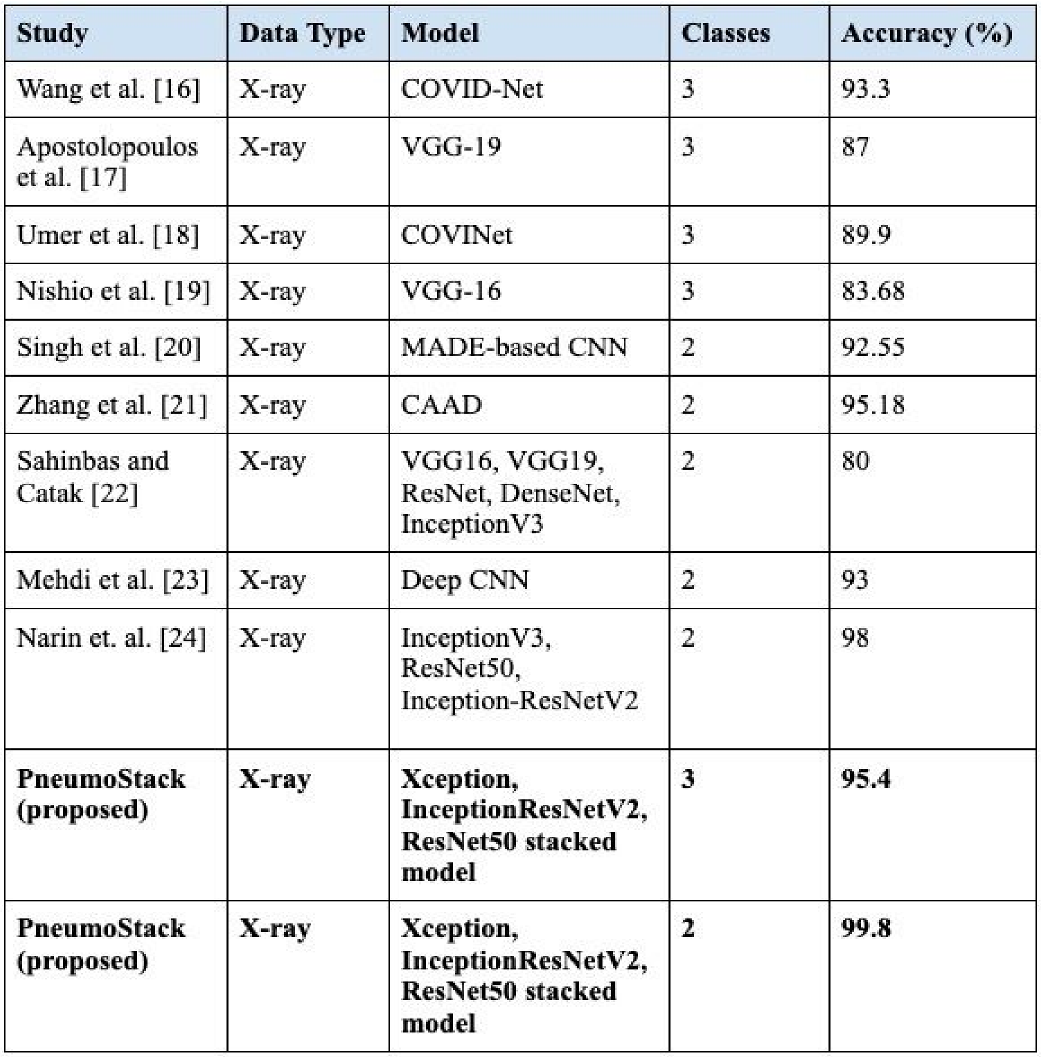


Figure 14. *Three-category unnormalized confusion matrix*

5. Conclusion

The stacked model performed significantly better than its constituent models (Xception, InceptionResNetV2, and ResNet50), as well as existing models used for pneumonia binary/multiclass classification.



Study	Data Type	Model	Classes	Accuracy (%)
Wang et al. [16]	X-ray	COVID-Net	3	93.3
Apostolopoulos et al. [17]	X-ray	VGG-19	3	87
Umer et al. [18]	X-ray	COVINet	3	89.9
Nishio et al. [19]	X-ray	VGG-16	3	83.68
Singh et al. [20]	X-ray	MADE-based CNN	2	92.55
Zhang et al. [21]	X-ray	CAAD	2	95.18
Sahinbas and Catak [22]	X-ray	VGG16, VGG19, ResNet, DenseNet, InceptionV3	2	80
Mehdi et al. [23]	X-ray	Deep CNN	2	93
Narin et. al. [24]	X-ray	InceptionV3, ResNet50, Inception-ResNetV2	2	98
PneumoStack (proposed)	**X-ray**	**Xception, InceptionResNetV2, ResNet50 stacked model**	**3**	**95.4**
PneumoStack (proposed)	**X-ray**	**Xception, InceptionResNetV2, ResNet50 stacked model**	**2**	**99.8**

Table 4. *Comparison of PneumoStack to models in other studies*

The results of this study indicate that an ensemble Xception, InceptionResNetV2, and ResNet50 constructed via stacked generalization has the potential to outperform existing methods used for automated pneumonia/COVID-19 diagnosis. Limitations of this study include developing and validating the proposed model on a public dataset. CXR characteristics from these public datasets may differ from those found in clinical data. Future steps include investigating PneumoStack performance on clinical CXRs to validate usability in a clinical setting. To counter class imbalance in the dataset, the data augmentation may be redone with synthetic minority oversampling technique (SMOTE). To investigate if this superior performance projects onto other applied CNN tasks in medicine, Pneumostack may be used in other medical imaging tasks such as MRI analysis for the early detection of neurodegenerative disease, differential gene analysis, and biomarker identification. Ultimately, Pneumostack opens the doors to higher CXR analysis performance in automated pneumonia and COVID-19 diagnosis, potentially paving the way for higher performance in various applied computer vision tasks in medicine.

6. Code

6.1. Data Preprocessing for Individual Transfer Learning Models

```
#Importation of libraries
import cv2
import glob
import h5py
import shutil
import keras
import imgaug as aug
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mimg
import imgaug.augmenters as augment
import tensorflow as tf
from os import listdir, makedirs, getcwd, remove
from os.path import isfile, join, abspath, exists, isdir, expanduser
from PIL import Image
from pathlib import Path
from skimage.io import imread
from skimage.transform import resize

from keras.models import Sequential, Model
from keras.applications.xception import Xception
from keras.applications.resnet50 import ResNet50
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.preprocessing.image import ImageDataGenerator,load_img,
img_to_array
from keras.preprocessing import image
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Input,
Flatten, SeparableConv2D,GlobalAveragePooling2D
from keras.layers import GlobalMaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.merge import Concatenate
from keras.models import Model
from keras import backend as K
from keras.optimizers import Adam, SGD, RMSprop
from keras.utils.vis_utils import plot_model
from keras.callbacks import ModelCheckpoint, Callback,
EarlyStopping,EarlyStopping,TensorBoard,ReduceLROnPlateau,CSVLogger,Learni
ngRateScheduler
from keras.utils import to_categorical

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
color = sns.color_palette()
%matplotlib inline


from google.colab import drive
drive.mount('/content/drive')


def normal_nonnormal(x):
   if x == 'Normal':
       return x
   else:
       return 'Non-Normal'
df = pd.read_csv('/content/drive/My Drive/CombinedImages.zip (Unzipped
Files)/CombinedImages/CombinedUpdated.csv')

na_fill = {'VirusCategory1': 'Normal'}
df = df.fillna(value = na_fill) #switch na to normal (dataset error)

df.VirusCategory1 = df.VirusCategory1.map(normal_nonnormal)
df = df.join(pd.get_dummies(df.VirusCategory1.values, prefix = 'type'))
#one hot
df = df[['ImagePath', 'VirusCategory1', 'type_Non-Normal']] #only columns
needed
X = df[['ImagePath', 'VirusCategory1']]
y = df[['type_Non-Normal']]
train, test = train_test_split(df)
x_train, x_test, y_train, y_test = train_test_split(X,y, random_state =
10, stratify = X.VirusCategory1.values,
                                                  train_size = .90)


print(x_train.VirusCategory1.value_counts())
x_train = x_train.drop('VirusCategory1', axis = 1)
x_test = x_test.drop('VirusCategory1', axis = 1)

def get_image_value(path):
   #This function will retrieve the RGB array for an image given its path
   img = image.load_img(path, target_size = (71, 71,3))
   img = image.img_to_array(img)

   return img/255


def get_data(df):
   #This function will retrieve the paths for each item within a sample,
and call get_image_value to retrieve the RGB array for each image
   from tqdm import tqdm
   img_list = []
   for path in tqdm(df.ImagePath.values, desc = 'Gathering Image Arrays'):
       path = f'/content/drive/My Drive/CombinedImages.zip (Unzipped
Files)/CombinedImages/all/{path}'
       img_list.append(get_image_value(path))
   return np.array(img_list).squeeze()
x_test = get_data(x_test)
x_train = get_data(x_train)

augmentation =ImageDataGenerator(rotation_range = 15, width_shift_range =
.1, height_shift_range = .1,
                                                            horizontal_flip
= True, fill_mode = 'nearest') #augmentation
augmentation.fit(x_train)
```

6.2. Xception Construction and Training

```
def get_Xception():
  base_model = tf.keras.applications.Xception(
   include_top=False,
   weights="imagenet",
   input_tensor=None,
   input_shape= (225,225, 3),
   pooling=None,
   classes=1000,
   classifier_activation="softmax",
)


 for layer in base_model.layers[:-12]:
   layer.trainable = False

 for layer in base_model.layers:
   print(layer,layer.trainable)

 model = Sequential()
 model.add(base_model)
 model.add(GlobalMaxPooling2D())
 model.add(Dense(1024,activation='relu'))
 model.add(Dropout(0.5))
 model.add(Dense(512,activation='relu'))
 model.add(Dropout(0.5))
 model.add(Dense(1,activation='sigmoid'))#softmax for three-category, 3
inputs
 model.summary()

 plot_model(model, to_file='model_architecture.png', show_shapes=True,
show_layer_names=True)
  opt = SGD(lr=1e-4,momentum=0.95)
 opt1 = Adam(lr=1e-4)


 model.compile(
   loss='binary_crossentropy',#categorical_crossentropy for three-class
   optimizer=opt1,
   metrics=['accuracy']
 )

 return model


from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau

early_stopping = EarlyStopping(monitor='val_loss', verbose = 1,
patience=5, min_delta = .002) #prevents overfitting
model_checkpoint = ModelCheckpoint('xception.h5', verbose = 1,
save_best_only=True,
                                 monitor = 'val_loss', min_delta = .002)
#saves weight as val loss decreases

lr_plat = ReduceLROnPlateau(patience = 3, mode = 'min') #adjusts learning
rate if loss plateaus
epochs = 50
batch_size = 32
normal_model = get_Xception_normal()
normal_history = normal_model.fit(augmentation.flow(x_train, y_train,
batch_size = batch_size),
               epochs = epochs,
        callbacks = [early_stopping, model_checkpoint, lr_plat],
validation_data = (x_test, y_test), verbose= 1) #training
```

6.3. InceptionResNetV2 Construction and Training

```
def get_InceptionResNetV2_normal():

 base_model = tf.keras.applications.InceptionResNetV2(include_top=False,
                 input_shape = (225,225,3),
                 weights = 'imagenet')

 for layer in base_model.layers[:-12]:
   layer.trainable = False

 for layer in base_model.layers:
   print(layer,layer.trainable)

 model = Sequential()
 model.add(base_model)
 model.add(GlobalAveragePooling2D())
 model.add(Dense(1024,activation='relu'))
 model.add(Dropout(0.5))
 model.add(Dense(512,activation='relu'))
 model.add(Dropout(0.5))
 model.add(Dense(1,activation='sigmoid')) #softmax for three-category, 3
inputs
 model.summary()

 plot_model(model, to_file='model_architecture.png', show_shapes=True,
show_layer_names=True)
  opt = SGD(lr=1e-4,momentum=0.95)
 opt1 = Adam(lr=1e-4)


 model.compile(
   loss='binary_crossentropy',#categorical_crossentropy for three-class

   optimizer=opt1,
   metrics=['accuracy']
 )

 return model
```

```
model_checkpoint = ModelCheckpoint('InceptionResNetV2.h5', verbose = 1,
save_best_only=True,
                                  monitor = 'val_loss', min_delta = .002)
#saves weight as val loss decreases

lr_plat = ReduceLROnPlateau(patience = 3, mode = 'min') #adjusts learning
rate if loss plateaus
epochs = 50
batch_size = 32
normal_model = get_InceptionResNetV2_normal()
normal_history = normal_model.fit(augmentation.flow(x_train, y_train,
batch_size = batch_size),
               epochs = epochs,
        callbacks = [early_stopping, model_checkpoint, lr_plat],
validation_data = (x_test, y_test), verbose= 1)
```

6.4. ResNet50 Construction and Training

```
def get_ResNet50_normal():

 base_model = applications.resnet50.ResNet50(weights= 'imagenet',
include_top=False, input_shape= (75,75,3))

 for layer in base_model.layers[:-12]:
   layer.trainable = False

 for layer in base_model.layers:
   print(layer,layer.trainable)

 model = Sequential()
 model.add(base_model)
 model.add(GlobalAveragePooling2D())
 model.add(Dense(1024,activation='relu'))
 model.add(Dropout(0.5))
 model.add(Dense(512,activation='relu'))
 model.add(Dropout(0.5))
 model.add(Dense(1,activation='sigmoid'))#softmax for three-category, 3
inputs
```

```
 model.summary()

 plot_model(model, to_file='model_architecture.png', show_shapes=True,
show_layer_names=True)
  opt = SGD(lr=1e-4,momentum=0.95)
 opt1 = Adam(lr=1e-4)


 model.compile(
   loss='binary_crossentropy',#categorical_crossentropy for three-class

   optimizer=opt1,
   metrics=['accuracy']
 )

 return model


model_checkpoint = ModelCheckpoint('resnet50.h5', verbose = 1,
save_best_only=True,
                                 monitor = 'val_loss', min_delta = .002)
#saves weight as val loss decreases

lr_plat = ReduceLROnPlateau(patience = 3, mode = 'min') #adjusts learning
rate if loss plateaus
epochs = 50
batch_size = 32
normal_model = get_ResNet50_normal()
normal_history = normal_model.fit(augmentation.flow(x_train, y_train,
batch_size = batch_size),
               epochs = epochs,
        callbacks = [early_stopping, model_checkpoint, lr_plat],
validation_data = (x_test, y_test), verbose= 1)
```

6.5. Data Preprocessing for Stacked Model (Binary)

```
def normal_nonnormal(x):
   if x == 'Normal':
       return x
   else:
       return 'Non-Normal'
df = pd.read_csv('/content/drive/My Drive/CombinedImages.zip (Unzipped
Files)/CombinedImages/CombinedUpdated.csv')
#df = pd.read_csv('../CombinedImages/CombinedUpdated.csv')
na_fill = {'VirusCategory1': 'Normal'}
df = df.fillna(value = na_fill) #switch na to normal (dataset error)

df.VirusCategory1 = df.VirusCategory1.map(normal_nonnormal)
df = df.join(pd.get_dummies(df.VirusCategory1.values, prefix = 'type'))
#one hot
df = df[['ImagePath', 'VirusCategory1', 'type_Non-Normal']] #only columns
needed
X = df[['ImagePath', 'VirusCategory1']]
y = df[['type_Non-Normal']]

x_train, x_test, y_train, y_test = train_test_split(X,y, random_state =
10, stratify = X.VirusCategory1.values,
                                                  train_size = .80)


print(x_train.VirusCategory1.value_counts())
x_train = x_train.drop('VirusCategory1', axis = 1) #only using this
category to stratify
x_test = x_test.drop('VirusCategory1', axis = 1)

def get_image_value(path):
   '''This function will retrive the RGB array for an image given its
path'''
   img = image.load_img(path, target_size = (225,225,3))
   img = image.img_to_array(img)

   return img/255


def get_data(df):
   '''This function will retrive the paths for each item within a sample,
and call get_image_value to retrieve
   the RGB array for each image'''
   from tqdm import tqdm
   img_list = []
   for path in tqdm(df.ImagePath.values, desc = 'Gathering Image Arrays'):
       path = f'/content/drive/My Drive/CombinedImages.zip (Unzipped
Files)/CombinedImages/all/{path}'
       img_list.append(get_image_value(path))
   return np.array(img_list).squeeze()
x_test = get_data(x_test)
x_train = get_data(x_train)



from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator
augmentation =ImageDataGenerator(rotation_range = 15, width_shift_range =
.1, height_shift_range = .1,
                                                             horizontal_flip
= True, fill_mode = 'nearest') #augmentation
augmentation.fit(x_train)


import keras
from keras.models import load_model

def load_all_models(n_models):
 all_models = list()
 for i in range(n_models):
   # define filename for this ensemble
   filename = 'models/model_' + str(i + 1) + '.h5'
   # load model from file
   model = load_model(filename)
   # add to list of members
   all_models.append(model)
   print('>loaded %s' % filename)
 return all_models

n_members = 3
members = load_all_models(n_members)
print('Loaded %d models' % len(members))
```

6.6. Data Preprocessing for Stacked Model (Three-Category)

```
def normal_nonnormal(x):
   if x == 'Normal':
       return x
   else:
       return 'Non-Normal'
df = pd.read_csv('/content/drive/My Drive/CombinedImages.zip (Unzipped
Files)/CombinedImages/data.csv')


print(df)


na_fill = {'VirusCategory1': 'Normal'}
# na_fill2 = {'VirusCategory2': 'Normal2'}
df = df.fillna(value = na_fill) #switch na to normal (dataset error)
# df = df.fillna(value = na_fill2)
print(df.VirusCategory1.unique())#print class labels in dataset

def class_label1(x):
 if x == 'Normal' or x == 'No Finding':
     y = 'Normal'
 elif x == 'COVID-19' or x == 'COVID-19, ARDS' :
     y = 'COVID-19'
 #elif x != 'Pneumocystis' or x != 'Lipoid':
   #y = 'tertiary'
 elif x in ['Bacterial', 'bacteria']:#, 'E.Coli','Chlamydophila',
'Klebsiella','Legionella','Mycoplasma Bacterial Pneumonia', 'bacteria',
'Virus']:
     y = 'tertiary'
 else:
     y = x
 return y

print(df.VirusCategory1.map(class_label1))
df.VirusCategory1 = df.VirusCategory1.map(class_label1)

print(df.VirusCategory1)
```

```
df = df.join(pd.get_dummies(df.VirusCategory1.values, prefix = 'type'))
#one hot

# df = df[['ImagePath', 'VirusCategory1', 'type_COVID-19',
'type_Bacterial', 'type_virus']] #only columns needed
X = df[['ImagePath', 'VirusCategory1']]
y = df[['type_Normal', 'type_COVID-19', 'type_tertiary']]
print(df)

x_train, x_test, y_train, y_test = train_test_split(X,y, random_state =
45,
                                                    train_size = 0.9)
#stratify = X.VirusCategory1.values)

print(x_train.VirusCategory1.value_counts())
x_train = x_train.drop('VirusCategory1', axis = 1) #only using this
category to stratify
x_test = x_test.drop('VirusCategory1', axis = 1)

def get_image_value(path):
   '''This function will retrive the RGB array for an image given its
path'''
   img = image.load_img(path, target_size = (425,425,3))
   img = image.img_to_array(img)

   return img/255


def get_data(df):
   '''This function will retrive the paths for each item within a sample,
and call get_image_value to retrieve
   the RGB array for each image'''
   from tqdm import tqdm
   img_list = []
   for path in tqdm(df.ImagePath.values, desc = 'Gathering Image Arrays'):
       path = f'/content/drive/My Drive/CombinedImages.zip (Unzipped
Files)/CombinedImages/all/{path}'
       img_list.append(get_image_value(path))
   return np.array(img_list).squeeze()
x_test = get_data(x_test)
```

```
x_train = get_data(x_train)

def stacked_dataset(members, inputX):
 stackX = None
 for i in range(len(members)):
   model = members[i]
   # make prediction
   yhat = model.predict(inputX, verbose=0)
   # stack predictions into [rows, members, probabilities]
   if stackX is None:
     stackX = yhat
   else:
     stackX = numpy.dstack((stackX, yhat))
   # flatten predictions to [rows, members x probabilities]
 stackX = stackX.reshape((stackX.shape[0],
stackX.shape[1]*stackX.shape[2]))
 return stackX
```

6.7. Stacked Model Construction and Training

```
def define_stacked_model(members):
 for i in range(len(members)):
   model = members[i]
   for layer in model.layers:
     layer.trainable = False
     layer._name = 'ensemble_' + str(i+1) + '_' + layer.name
 meta_model = keras.models.Sequential()
 meta_model.add(keras.layers.Flatten(input_shape=[3,1]))#9 inputs for
three-class classification
 meta_model.add(keras.layers.Dense(100, activation='relu'))
 meta_model.add(keras.layers.Dense(1, activation='sigmoid'))#softmax for
three-class classification
 plot_model(meta_model, show_shapes=True, to_file='meta_model.png')
 meta_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])#categorical_crossentropy for three-class
classification
 return meta_model

stackX = stacked_dataset(members, x_train)
stacked_model = define_stacked_model(members)
```

```
early_stopping = EarlyStopping(monitor='val_loss', verbose = 1,
patience=5, min_delta = .002) #prevents overfitting
model_checkpoint = ModelCheckpoint('ensemble.h5', verbose = 1,
save_best_only=True,
                                 monitor = 'val_loss', min_delta = .002)
#saves weight as val loss decreases
history = stacked_model.fit(stackX, y_train, epochs=50, verbose=0,
validation_data = (stacked_dataset(members, x_test), y_test), callbacks =
[early_stopping, model_checkpoint])
```

6.8. PneumoStack Evaluation and Performance Visualization for Binary Classification

```
def stacked_prediction(members, model, inputX):
 # create dataset using ensemble
 stackedX = stacked_dataset(members, inputX)
 # make a prediction
 yhat = model.predict(stackedX)
 return yhat

#print accuracy for individual models
for model in members:
 _, acc = model.evaluate(x_test, y_test, verbose=0)
 print('Model Accuracy: %.3f' % acc)

from sklearn.metrics import accuracy_score
yhat = stacked_prediction(members, stacked_model, x_test)
print('test', y_test)
print('hat', yhat)
yhat = np.rint(yhat)
acc = accuracy_score(y_test, yhat)
print('Stacked Test Accuracy: %.3f' % acc)

train_loss = history.history['loss']
train_acc = history.history['accuracy']
test_loss = history.history['val_loss']
test_acc = history.history['val_accuracy']
epochs = [i for i in range(1, len(test_acc)+1)]

fig, ax = plt.subplots(1,2, figsize = (15,5))
```

```
ax[0].plot(epochs, train_loss, label = 'Train Loss')
ax[0].plot(epochs, test_loss, label = 'Test Loss')
ax[0].set_title('Train/Test Loss')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Loss')
ax[0].legend()

ax[1].plot(epochs, train_acc, label = 'Train Accuracy')
ax[1].plot(epochs, test_acc, label = 'Test Accuracy')
ax[1].set_title('Train/Test Accuracy')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend()

from sklearn.metrics import roc_curve, roc_auc_score,
precision_recall_curve, f1_score, auc

stacked_model.load_weights('ensemble.h5') #load the best weights before
overfitting

y_test_precision, y_test_recall, spec = precision_recall_curve(y_test,
yhat)
y_test_predict = np.where(yhat >= .5, 1, 0).ravel()
y_test_f1= f1_score(y_test, y_test_predict)
y_test_auc = auc(y_test_recall, y_test_precision)
no_skill = len(y_test[y_test==1]) / len(y_test)

import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,2, figsize = (15,6))

ax[0].plot(y_test_recall, y_test_precision, marker='.', label='Stacked
model')
ax[0].plot([0, 1], [no_skill, no_skill], linestyle='--', label='50/50',
color = 'Black')
ax[0].set_xlabel('Recall')
ax[0].set_ylabel('Precision')
ax[0].set_title(f'Precision Recall')
ax[0].legend()
```

```
#ROC CURVE
ns_probs = [0 for i in range(len(y_test))]
ns_auc = roc_auc_score(y_test, ns_probs)
y_test_roc = roc_auc_score(y_test, yhat)

ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
y_test_fpr, y_test_tpr, threshold = roc_curve(y_test, yhat)
ax[1].plot(ns_fpr, ns_tpr, linestyle='--', label='50/50')
ax[1].plot(y_test_fpr, y_test_tpr, marker='.', label='Stacked model')
ax[1].set_xlabel('False Positive Rate')
ax[1].set_ylabel('True Positive Rate')
ax[1].set_title(f'ROC Curve')
ax[1].legend()
plt.show()


pd.DataFrame({'F1 Score': round(y_test_f1, 3), 'AUC': round(y_test_auc,
3), 'ROC':round(y_test_roc, 3)}, index = [0])

import itertools
import seaborn as sns

#confusion matrix
def plot_confusion_matrix(y_test,y_train, y_train_prob,
y_test_prob,thresholds, classes,
                         cmap=plt.cm.Blues):
   fig, ax = plt.subplots(len(thresholds),2, figsize = (10,10))

   for idx, thresh in enumerate(thresholds):
       y_test_predict = np.where(y_test_prob >= thresh, 1, 0)
       y_train_predict = np.where(y_train_prob >= thresh, 1, 0)
       train_cm = confusion_matrix(y_train, y_train_predict)
       test_cm = confusion_matrix(y_test, y_test_predict)

       #test confusion
       ax[idx, 0].imshow(test_cm,  cmap=plt.cm.Blues)

       ax[idx, 0].set_title(f'Test: Confusion Matrix | Threshold:
{thresh}')
```

```
        ax[idx, 0].set_ylabel('True label')
        ax[idx, 0].set_xlabel('Predicted label')

        class_names = classes
        tick_marks = np.arange(len(class_names))
        ax[idx, 0].set_xticks(tick_marks)
        ax[idx,0].set_xticklabels(class_names)
        ax[idx, 0].set_yticks(tick_marks)
        ax[idx, 0].set_yticklabels(class_names)

        th = test_cm.max() / 2.

        for i, j in itertools.product(range(test_cm.shape[0]),
range(test_cm.shape[1])):
                ax[idx, 0].text(j, i, f'{test_cm[i, j]}',# |
{int(round(test_cm[i,j]/test_cm.ravel().sum(),5)*100)}%',
                         horizontalalignment='center',
                         color='white' if test_cm[i, j] > th else 'black')
        ax[idx, 0].set_ylim([-.5,1.5])

        #TRAIN CONFUSION
        ax[idx, 1].imshow(train_cm,  cmap=plt.cm.Blues)

        ax[idx, 1].set_title(f'Train: Confusion Matrix | Threshold:
{thresh}')
        ax[idx, 1].set_ylabel('True label')
        ax[idx, 1].set_xlabel('Predicted label')

        class_names = classes
        tick_marks = np.arange(len(class_names))
        ax[idx, 1].set_xticks(tick_marks)
        ax[idx,1].set_xticklabels(class_names)
        ax[idx, 1].set_yticks(tick_marks)
        ax[idx, 1].set_yticklabels(class_names)


        th = train_cm.max() / 2.

        for i, j in itertools.product(range(train_cm.shape[0]),
range(train_cm.shape[1])):
```

```
                ax[idx, 1].text(j, i, f'{train_cm[i, j]}',# |
{int(round(train_cm[i,j]/train_cm.ravel().sum(),5)*100)}%',
                        horizontalalignment='center',
                        color='white' if train_cm[i, j] > th else 'black')
        ax[idx, 1].set_ylim([-.5,1.5])
    plt.tight_layout()
    plt.show()


plot_confusion_matrix(y_train = y_train, y_test = y_test, y_train_prob =
stacked_prediction(members, stacked_model, x_train),
                     y_test_prob = yhat, classes = ['Normal',
'NonNormal'], thresholds = [.2, .5,.6])
```

6.9. PneumoStack Evaluation and Performance Visualization for Three-Category Classification

```
from numpy import argmax
y_pred = normal_model1.predict(x_train)
y_test.columns = [0, 1, 2]
y_test.idxmax(axis=1)
y_pred.shape
a = y_test.idxmax(axis=1)
b = argmax(y_pred, axis=1)

import keras
from keras.models import load_model

def load_all_models(n_models):
 all_models = list()
 for i in range(n_models):
   # define filename for this ensemble
   filename = 'models/model_' + str(i + 1) + '.h5'
   # load model from file
   model = load_model(filename)
   # add to list of members
   all_models.append(model)
   print('>loaded %s' % filename)
 return all_models
```

```python
n_members = 3
members = load_all_models(n_members)
print('Loaded %d models' % len(members))

from sklearn.metrics import accuracy_score

yhat = stacked_prediction(members, stacked_model, x_test)
print('test', y_test)
print('hat', yhat)
yhat = np.rint(yhat)
#yhat = argmax(yhat, axis=1)
acc = accuracy_score(y_test, yhat)
print('Stacked Test Accuracy: %.3f' % acc)

train_loss = history.history['loss']
train_acc = history.history['accuracy']
test_loss = history.history['val_loss']
test_acc = history.history['val_accuracy']
epochs = [i for i in range(1, len(test_acc)+1)]

fig, ax = plt.subplots(1,2, figsize = (15,5))
ax[0].plot(epochs, train_loss, label = 'Train Loss')
ax[0].plot(epochs, test_loss, label = 'Test Loss')
ax[0].set_title('Train/Test Loss')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Loss')
ax[0].legend()

ax[1].plot(epochs, train_acc, label = 'Train Accuracy')
ax[1].plot(epochs, test_acc, label = 'Test Accuracy')
ax[1].set_title('Train/Test Accuracy')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend()

from sklearn.metrics import multilabel_confusion_matrix
from numpy import argmax
# confusion_matrix(y_test.idxmax(axis=1).tolist(), argmax(yhat, axis = 1),
labels=['type_Normal', 'type_COVID-19', 'type_virus', 'type_Bacterial'])
```

```python
cm = confusion_matrix(y_test.idxmax(axis=1).tolist(), argmax(yhat, axis =
1))

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
   """
   This function prints and plots the confusion matrix.
   Normalization can be applied by setting `normalize=True`.
   """
   import itertools
   if normalize:
       cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
       print("Normalized confusion matrix")
   else:
       print('Confusion matrix, without normalization')

   print(cm)
   plt.figure(dpi: 400, figsize: [100,100])
   plt.imshow(cm, interpolation='nearest', cmap=cmap)
   plt.title(title)
   plt.colorbar()
   tick_marks = np.arange(len(classes))
   plt.xticks(tick_marks, classes, rotation=45)
   plt.yticks(tick_marks, classes)

   fmt = '.2f' if normalize else 'd'
   thresh = cm.max() / 2.
   for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
       plt.text(j, i, format(cm[i, j], fmt),
                horizontalalignment="center",
                color="white" if cm[i, j] > thresh else "black")

   plt.ylabel('True label')
   plt.xlabel('Predicted label')
   plt.tight_layout()
```

```
plot_confusion_matrix(cm, classes = ['type_Normal', 'type_COVID-19',
'type_tertiary'], normalize = False, title = 'Three-class Confusion
Matrix', cmap=plt.cm.Blues)

recall = np.diag(cm) / np.sum(cm, axis = 1)
precision = np.diag(cm) / np.sum(cm, axis = 0)
recall = np.mean(recall)
precision = np.mean(precision)
pd.DataFrame({'Precision': round(precision, 3), 'Recall': round(recall,
3), 'Accuracy':round(acc, 3)}, index = [0])
```

7. References

1. Pneumonia. (2019). Retrieved 12 February 2021, from https://www.who.int/news-room/fact-sheets/detail/pneumonia
2. Pneumonia | NHLBI, NIH. (2021). Retrieved 12 February 2021, from https://www.nhlbi.nih.gov/health-topics/pneumonia
3. A. Ghaderi and V. Athitsos, "Selective unsupervised feature learning with Convolutional Neural Network (S-CNN)," *2016 23rd International Conference on Pattern Recognition (ICPR)*, Cancun, 2016, pp. 2486-2490, doi: 10.1109/ICPR.2016.7900009.
4. Bai, H., Hsieh, B., Xiong, Z., Halsey, K., Choi, J., & Tran, T. et al. (2020). Performance of Radiologists in Differentiating COVID-19 from Non-COVID-19 Viral Pneumonia at Chest CT. *Radiology*, *296*(2), E46-E54. doi: 10.1148/radiol.2020200823
5. Saha, S. (2021). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Towards Data Science. Retrieved 12 February 2021, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164
6. Zhang, J. (1999). Developing robust non-linear models through bootstrap aggregated neural networks. Neurocomputing, 25(1-3), 93-113. doi: 10.1016/s0925-2312(99)00054-5
7. Ensemble Transfer Learning Framework for Vessel Size Estimation from 2D Images - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Ensemble-transfer-learning-using-pretrained-CNN-model-initialized-with-weights-trained-on_fig4_333619654 [accessed 28 Feb, 2021]
8. Brownlee, J. (2017). A Gentle Introduction to Transfer Learning for Deep Learning.

Retrieved 28 February 2021, from https://machinelearningmastery.com/transfer-learning-for-deep-learning/

9. Wolpert, D. (1992). Stacked generalization. *Neural Networks*, *5*(2), 241-259. doi: 10.1016/s0893-6080(05)80023-1

10. Detecting Behavioral Microsleeps from EEG Power Spectra - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Schematic-diagram-of-stacked-generalization_fig2_5898833 [accessed 28 Feb, 2021]

11. Cohen, Joseph, Paul Morrison, and Lan Dao. "COVID-19 Image Data Collection: Prospective Predictions Are The Future " *arXiv 2006 11988* (2021): n. pag. Web 25 Feb 2021

12. Chollet, F. (2016). Xception: Deep Learning with Depthwise Separable Convolutions. Retrieved 31 January 2021, from https://arxiv.org/abs/1610.02357

13. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. Retrieved 31 January 2021, from https://arxiv.org/abs/1512.03385

14. Alemi. Alex. (2021). Improving Inception and Image Classification in TensorFlow. Retrieved 28 February 2021, from https://ai.googleblog.com/2016/08/improving-inception-and-image.html

15. ImageNet Winning CNN Architectures (ILSVRC) | Data Science and Machine Learning. (2021). Retrieved 28 February 2021, from https://www.kaggle.com/getting-started/149448

16. Wang, L., Lin, Z.Q. & Wong, A. COVID-Net: a tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images. *Sci Rep* 10,

19549 (2020). https://doi.org/10.1038/s41598-020-76550-z

17. Apostolopoulos, I. D., & Mpesiana, T. A. (2020). Covid-19: automatic detection from X-ray images utilizing transfer learning with convolutional neural networks. *Physical and engineering sciences in medicine,* 43(2), 635–640. https://doi.org/10.1007/s13246-020-00865-4

18. Umer, M., Ashraf, I., Ullah, S. et al. COVINet: a convolutional neural network approach for predicting COVID-19 from chest X-ray images. *J Ambient Intell Human Comput* (2021). https://doi.org/10.1007/s12652-021-02917-3

19. Nishio, M., Noguchi, S., Matsuo, H., & Murakami, T. (2020). Automatic classification between COVID-19 pneumonia, non-COVID-19 pneumonia, and the healthy on chest X-ray image: combination of data augmentation methods. *Scientific Reports*, *10*(1). doi: 10.1038/s41598-020-74539-2

20. Singh D, Kumar V, Yadav V, and Kaur M. Deep Convolutional Neural Networks based Classification model for COVID-19 Infected Patients using Chest Xray Images. *International Journal of Pattern Recognition and Artificial Intelligence*, https://doi.org/10.1142/S0218001421510046, 2020.

21. J. Zhang et al., "Viral Pneumonia Screening on Chest X-rays Using Confidence-Aware Anomaly Detection," in *IEEE Transactions on Medical Imaging*, doi: 10.1109/TMI.2020.3040950.

22. Sahinbas K, and Catak FO. Transfer Learning Based Convolutional Neural Network for COVID-19 Detection with X-Ray Images. https://www.ozgurcatak.org/files/papers/covid19-deep-learning.pdf, 2020

23. Jamil M, and Hussain I. Automatic Detection of COVID-19 Infection from Chest X-ray

using Deep Learning. medRxiv, https://doi.org/10.1101/2020.05.10.20097063, 2020.

24. Narin A, Kaya C, and Pamuk Z. Automatic Detection of Coronavirus Disease (COVID19) Using X-ray Images and Deep Convolutional Neural Networks. arXiv:2003.10849v1, 2020

25. Osterburg, Stephen. "Implementation of Xception Model." *Coding*, stephan-osterburg.gitbook.io/coding/coding/ml-dl/tensorfow/ch3-xception/implementation-of-xception-model.

26. Aguas, Kenneth. (2020). A guide to transfer learning with Keras using ResNet50. Retrieved 5 February 2021, from https://medium.com/@kenneth.ca95/a-guide-to-transfer-learning-with-keras-using-resnet50-a81a4a28084b

27. Brownlee, J. (2018). Stacking Ensemble for Deep Learning Neural Networks in Python. Retrieved 5 February 2021, from https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks/

28. What Is Coronavirus?. (2021). Retrieved 6 March 2021, from https://www.hopkinsmedicine.org/health/conditions-and-diseases/coronavirus

29. RT-PCR Testing. (2021). Retrieved 6 March 2021, from https://www.idsociety.org/covid-19-real-time-learning-network/diagnostics/RT-pcr-testing/

30. Rousan, L.A., Elobeid, E., Karrar, M. *et al.* Chest x-ray findings and temporal lung changes in patients with COVID-19 pneumonia. *BMC Pulm Med* 20, 245 (2020). https://doi.org/10.1186/s12890-020-01286-5

31. Jacobi, A., Chung, M., Bernheim, A., & Eber, C. (2020). Portable chest X-ray in coronavirus disease-19 (COVID-19): A pictorial review. *Clinical Imaging*, *64*, 35-42. doi: 10.1016/j.clinimag.2020.04.001

32. Gianchandani, N., Jaiswal, A., Singh, D. *et al.* Rapid COVID-19 diagnosis using ensemble deep transfer learning models from chest radiographic images. *J Ambient Intell Human Comput* (2020). https://doi.org/10.1007/s12652-020-02669-6

33. H. Ko, H. Ha, H. Cho, K. Seo and J. Lee, "Pneumonia Detection with Weighted Voting Ensemble of CNN Models," *2019 2nd International Conference on Artificial Intelligence and Big Data (ICAIBD)*, Chengdu, China, 2019, pp. 306-310, doi: 10.1109/ICAIBD.2019.8837042.

34. Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. *J Big Data* 6, 60 (2019). https://doi.org/10.1186/s40537-019-0197-0