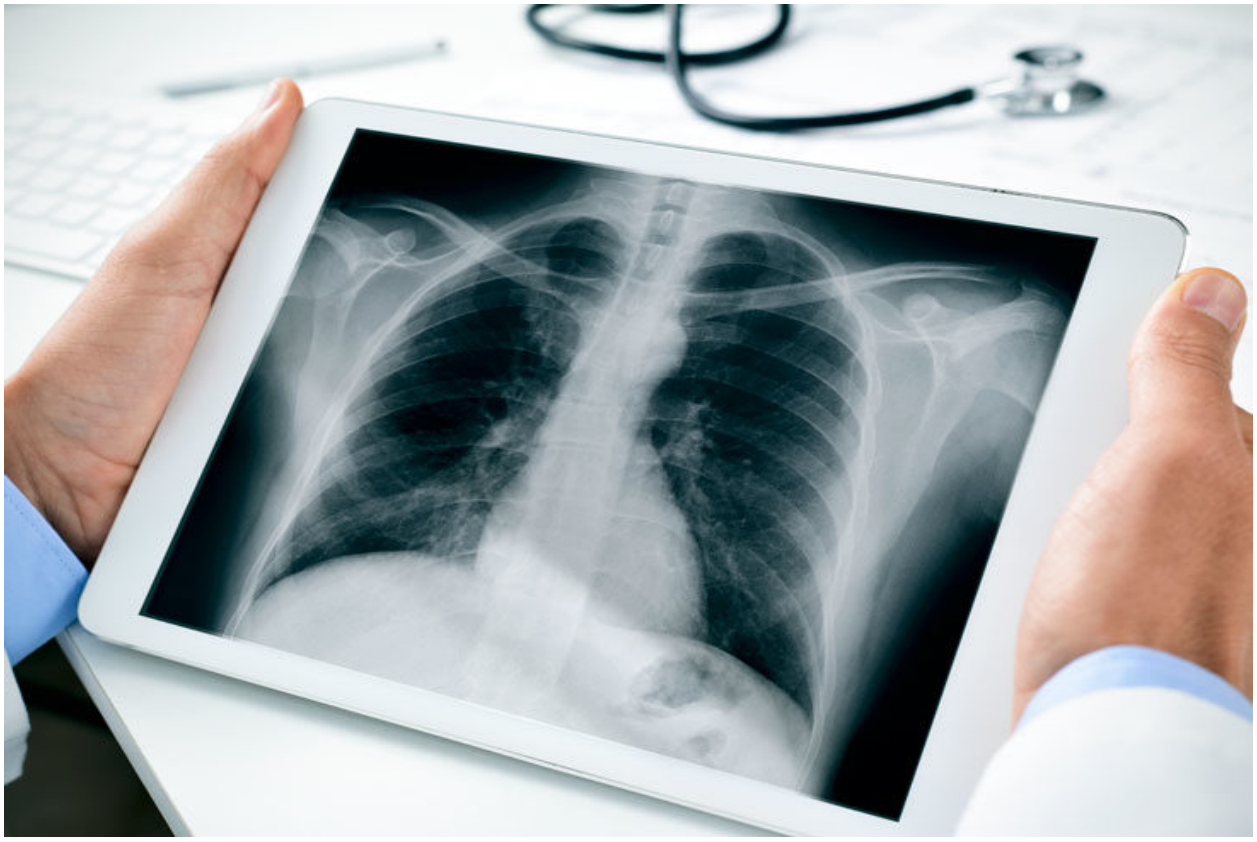
Group Final Report

Machine Learning 2

Chest X-Ray Classification



Kristin Levine
Sertan Akinci
May 2021

Introduction

The COVID-19 Radiography Database is a large collection of four different classes of chest X-rays: normal, COVID, lung opacity (non-COVID lung infections), and viral pneumonia. In the midst of the pandemic, it is more important than ever to be able to quickly diagnose and treat different types of lung infections. At a time when medical staff is greatly overworked, it would be wonderful if machine learning could help in making some of these diagnoses. The goal is to build a suitable model for this classification task and train it with the data at hand to be able to correctly diagnose new x-ray images.

Description of the Data Set

The X-rays in this dataset came from a number of different sources. See References for links.

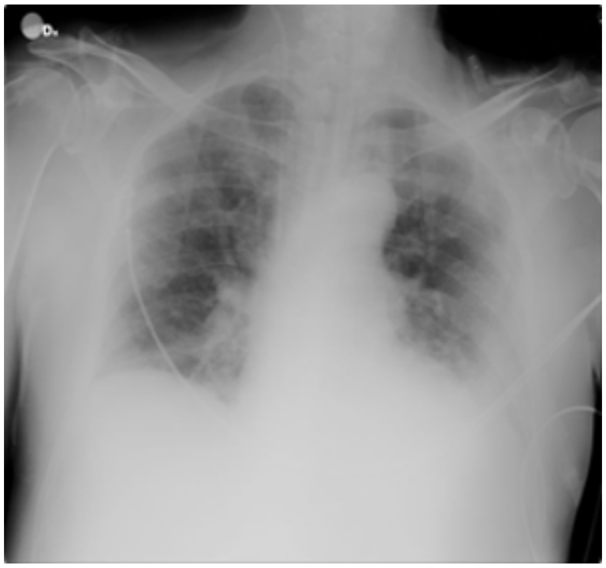
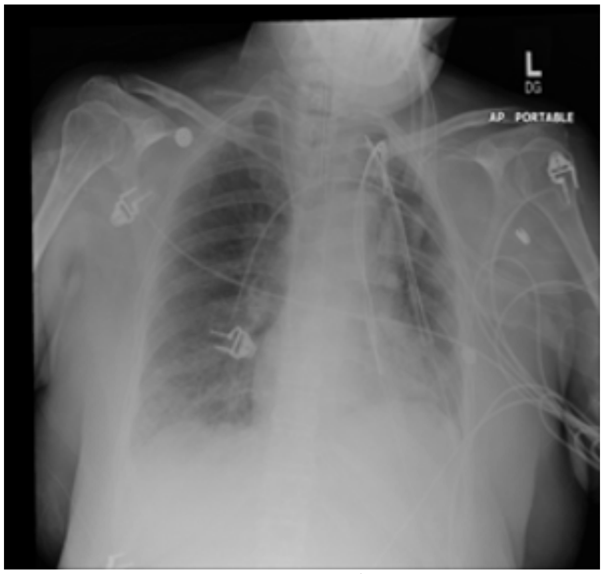
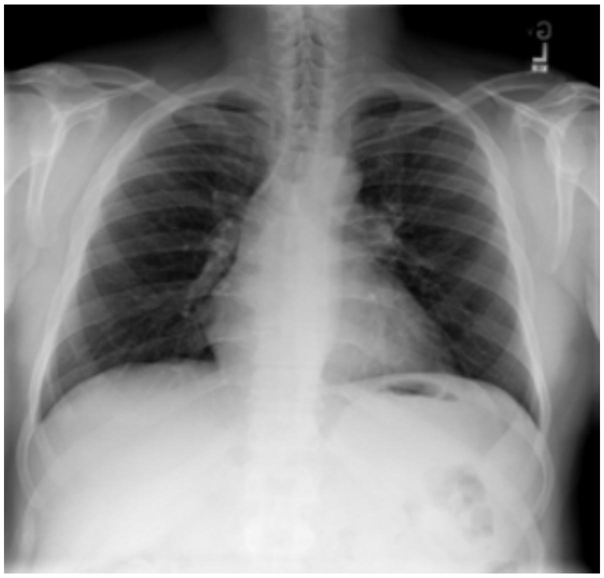
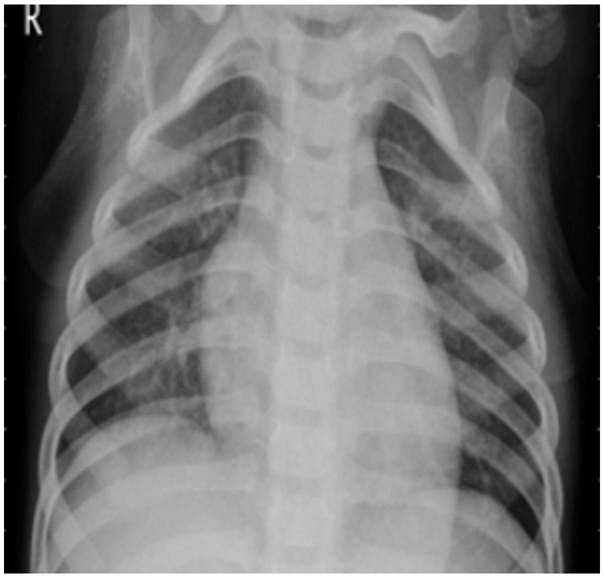
- Normal X-rays (10,192 images total):
    - 8851 RSNA [8]
    - 1341 Kaggle [9]

- COVID-19 data (3616 images total):
    - 2473 CXR images are collected from padchest dataset[1].
    - 183 CXR images from a Germany medical school[2].
    - 559 CXR image from SIRM, Github, Kaggle & Tweeter[3,4,5,6]
    - 400 CXR images from another Github source[7].

- Lung Opacity (6012 images total):
    - 6012 Lung opacity CXR images are collected from Radiological Society of North America (RSNA) CXR dataset [8]

- Viral Pneumonia (1345 images total):
    - 1345 Viral Pneumonia data are collected from the Chest X-Ray Images (pneumonia) database [9]

Examples

 Covid-19	 Lung Opacity
 Normal	 Pneumonia

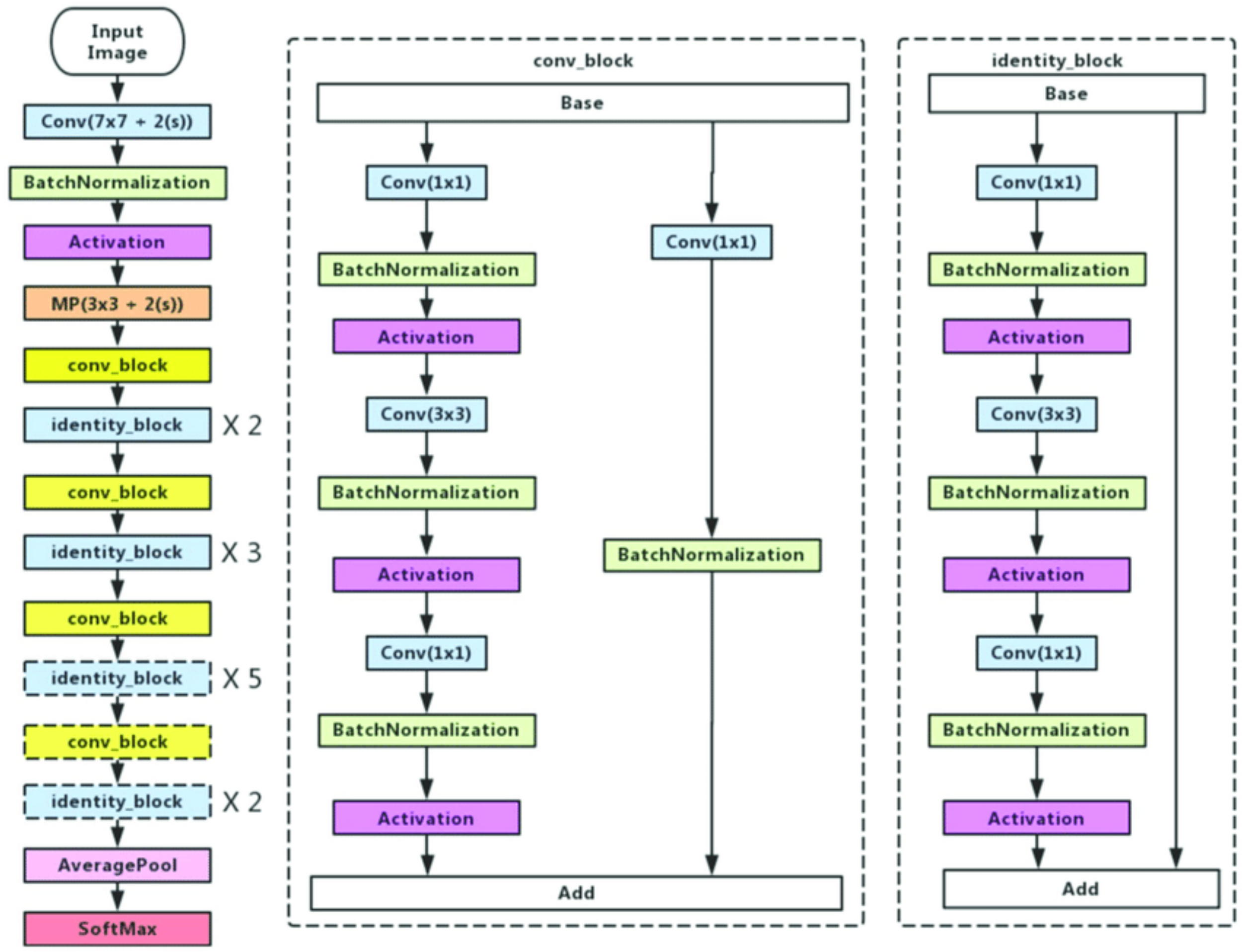
Deep Neural Network and Training Algorithm

Previous research has looked at a number of different models with two classes (normal or COVID) or three classes (normal, COVID, and viral pneumonia). We decided to look at all four classes for which we had data, as well as trying to ensemble the methods that we selected.

We used three different pre-trained models in our project: ResNet50, Xception, and VGG16. All of these models were originally trained on the ImageNet image database. Our plan was to train each of these models separately and then ensemble them together using a standard random forest algorithm. The choice of platform was Tensorflow, because we learned about it in class, but hadn't had a chance to practice. It also seemed practical to use Tensorflow with pretrained models.

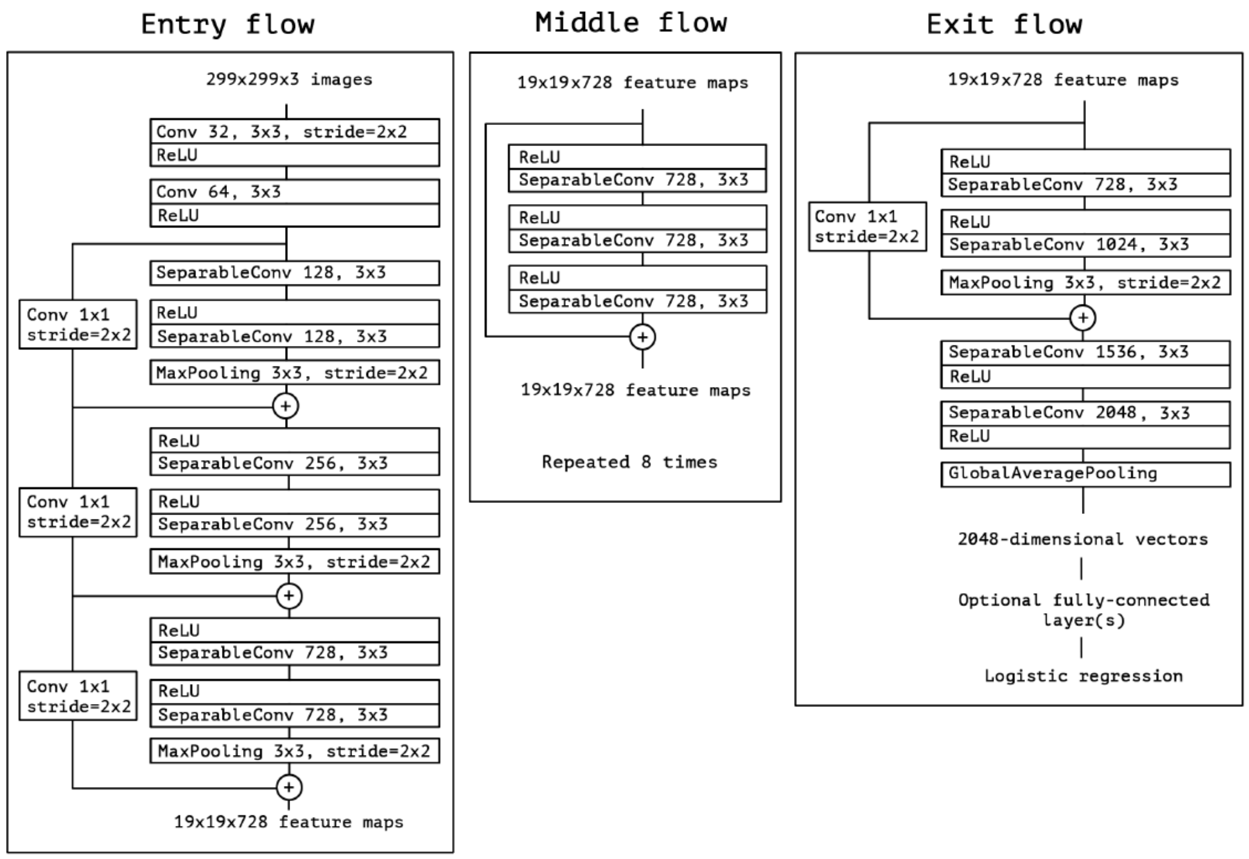
ResNet50

ResNet50 is a CNN that is 50 layers deep. It's considered a classic neural network and was the winner of the ImageNet challenge in 2015. It was the first to introduce the concept of skip connection: a signal feeding into a layer is also added to the output of another layer that is located higher up in the stack. Input shape (224, 224, 3)



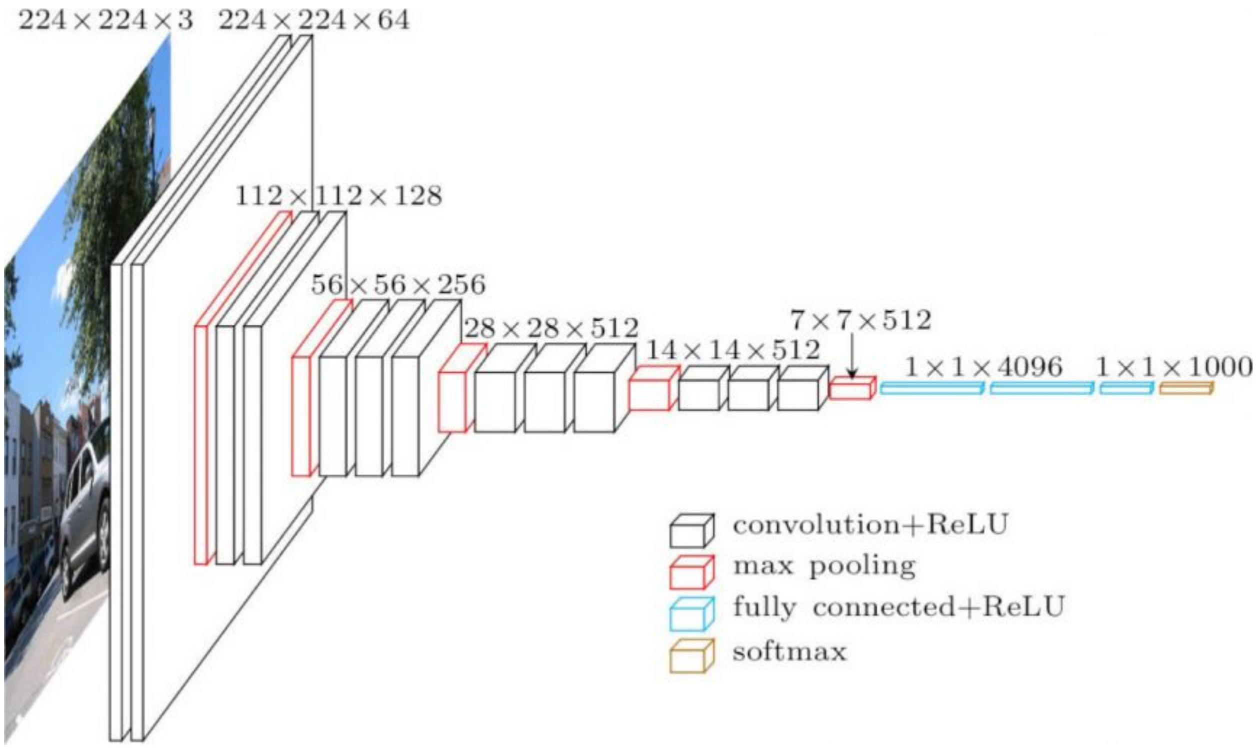
Xception

Xception is a CNN that is 71 layers deep. It dates from 2017, uses depthwise separable convolution, and is an improvement on the Inception model. Input shape (299, 299, 3)



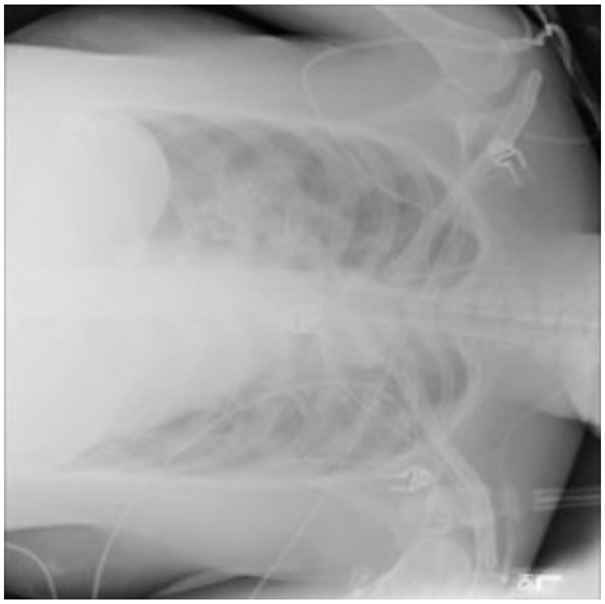
VGG16

VGG16 is another CNN named after the Visual Geometry Group from Oxford who developed it. It won the Imagenet competition in 2014 and uses smaller filters, but a deeper network. It has a very simple and classical architecture. Input shape (224, 224, 3)



Experimental Setup

The data was not split into train and test so the first step was to put aside a test dataset which consisted of 10% from each class. The remaining data was partitioned into 80% training (72% over all data) and 20% validation (18% over all data). The classes were quite unbalanced; approximately 1,200 Pneumonia cases, 3,200 Covid, 5,100 Lung Opacity, and 9,100 Normal cases. In order to gage the possible impact of this imbalance we performed an initial run and the low accuracy scores were convincing enough for us to apply over sampling while augmenting the new images for the minority classes and undersampling for the majority classes. The augmentation consisted of flipping the image horizontally, vertically, and rotating it clock and counter-clockwise.


Examples of flipped/rotated images

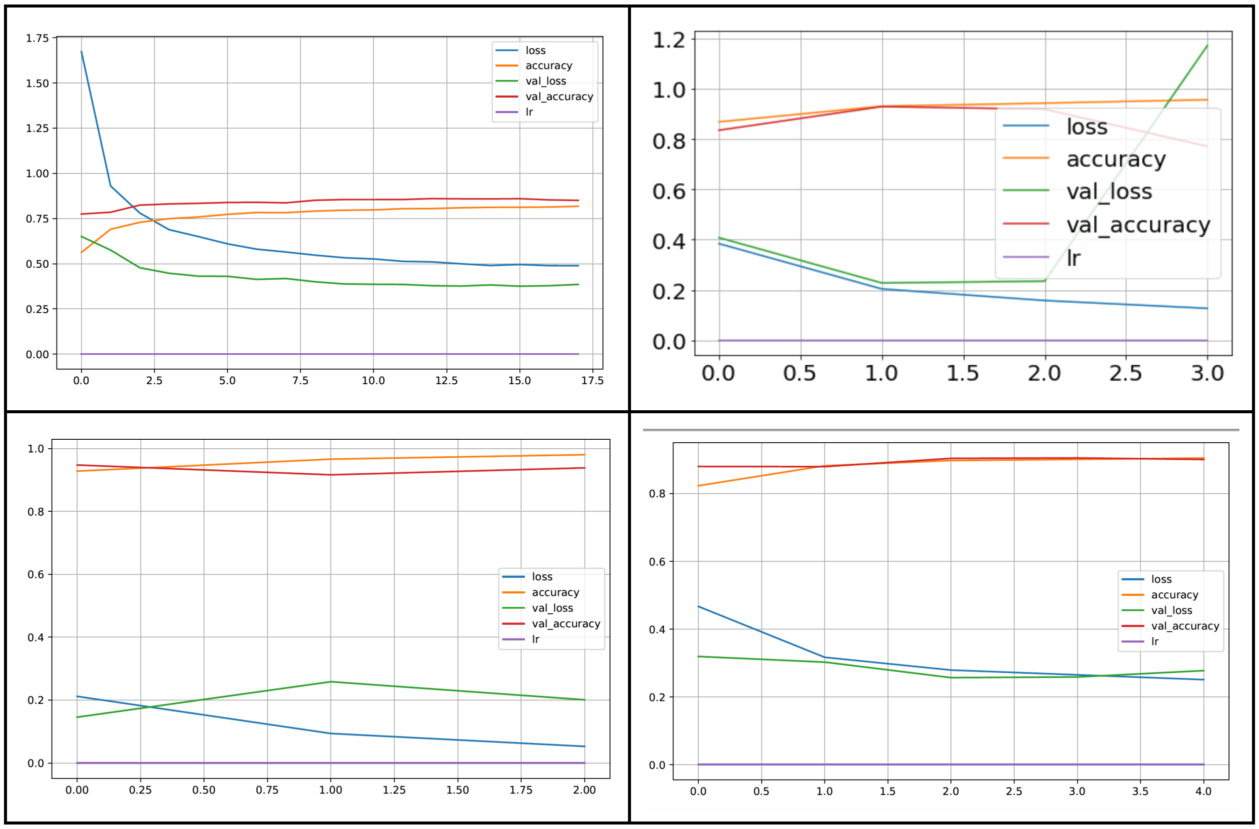
```
########## Augmenting and Oversampling Undersampling Images #####

# Oversample Viral and COVID classes #
AUG_DIR = 'COVID-19_Radiography_Dataset/Augs/'
listv = os.listdir(viral_data)
for i in range(len(listv)):
   img = cv2.imread(viral_data + listv[i])
   cv2.imwrite(viral_data + 'augfh' + str(i) + '.png', cv2.flip(img, 1))
   cv2.imwrite(viral_data + 'augfv' + str(i) + '.png', cv2.flip(img, 0))
   cv2.imwrite(viral_data + 'augrt' + str(i) + '.png', cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE))
   cv2.imwrite(viral_data + 'augrt' + str(i) + '.png', cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE))

# Undersampling NORMAL Class #####
nom_l = random.sample(os.listdir(norm_data), np.int(np.floor(len(os.listdir(norm_data)) * 0.44)))
for l in nom_l:
   os.rename(norm_data + l, AUG_DIR + l)
```

For the training and validation we loaded the data using tensorflow's image_dataset_from_directory, a convenient as well as a tricky method to read data off of a class separated folder structure. It labels the data based on the folders, splits it, batches it as specified and prepares it for training.

```
data_tr = tf.keras.preprocessing.image_dataset_from_directory(
   data_folder, labels='inferred',  class_names=None,
   color_mode='rgb', batch_size=batch, image_size=img_size, seed=random_seed,
   validation_split=split, subset='training', interpolation='bilinear'
)
```

We decided to train each of these models separately and then ensemble them together using a standard random forest algorithm. This way we were able to experiment on our models and play with the hyper parameters for the best performance. We tried training our models by first freezing the pretrained layers and letting the remaining layers learn (learning rate = 0.001) the basic structure of the images then by unfreezing the pre-trained layers and training the entire model all over again (lr = 0.0001). Another approach was to just simply train the entire model along with the pre-trained layers without freezing them first (lr = 0.0001). This method was also very effective.

We kept a close watch on the training and validation accuracy. When the training accuracy was higher than the validation accuracy, we knew we had an overfitting problem, which we addressed by using dropout layers. We also used EarlyStopping so that we could keep the epoch number high and not fall into any local minimums or fall a victim of overfitting. We also graphed our results to provide a quick visual spot check of how we were doing.



To examine our results, we created a classification report, looking at the accuracy, as well as the precision, recall and f1-score.

Results

Implementing the pre-trained models was fairly simple -- and we immediately got good individual model accuracy. However, combining the models proved more challenging. While we were getting 0.90+ accuracy on the training, validation, and test data when we were running each model alone, when we combined them in one file, loaded the weights, and ran it again on the test data, we got the following results:

<u>ResNet Testset:</u>

```
                 precision    recall  f1-score   support

          COVID       0.16      0.16      0.16       192
   Lung_Opacity       0.31      0.30      0.30       319
         Normal       0.49      0.50      0.49       541
Viral Pneumonia       0.08      0.08      0.08        71

       accuracy                           0.36      1123
      macro avg       0.26      0.26      0.26      1123
   weighted avg       0.36      0.36      0.36      1123

Accuracy :  35.61887800534283
```

<u>Xception Testset:</u>

```
                 precision    recall  f1-score   support

          COVID       0.16      0.15      0.16       192
   Lung_Opacity       0.31      0.26      0.28       319
         Normal       0.50      0.55      0.53       541
Viral Pneumonia       0.08      0.08      0.08        71

       accuracy                           0.37      1123
      macro avg       0.26      0.26      0.26      1123
   weighted avg       0.36      0.37      0.36      1123

Accuracy :  37.1326803205699
```

Ensembling our models using a random forest did improve our scores by 5%:

<u>Random Forest:</u>

```
Results Using All Features:

Classification Report:
              precision    recall  f1-score   support

         0.0       0.12      0.06      0.08        52
         1.0       0.28      0.21      0.24        98
         2.0       0.50      0.72      0.59       163
         3.0       0.00      0.00      0.00        24

    accuracy                           0.42       337
   macro avg       0.23      0.25      0.23       337
weighted avg       0.34      0.42      0.37       337

Accuracy :  42.13649851632047
```

It was great that the random forest was improving our accuracy, but our scores still seemed much too low for these powerful pre-trained models. And why were we getting such good accuracy on our training and validation set? We tried augmenting the data again, to no avail.

Finally we realized it had to do with how we were loading our test data using image_dataset_from_directory. Even though we had the same random seed, tensorflow seemed to be creating different mini-batches each time. While this randomness was good for training, it was not so great for comparing results across models. To make things worse, when we were using image_dataset_from_directory we had to load the data twice because one of the models (Xception) took a different size image as input. To get around this, we finally wrote very simple code to load the test data ourselves, which made sure it would always be in the same order. We realized we didn't really need batches for the test data, since we weren't calculating errors and updating the parameters, we were simply evaluating the model. This seemed to work and we starting getting test results that were similar to our training and validation results:

ResNet:
```
                  precision    recall  f1-score   support

           COVID       0.93      0.98      0.16       316
    Lung_Opacity       0.93      0.89      0.30       601
          Normal       0.93      0.95      0.49      1019
 Viral Pneumonia       0.95      0.94      0.08       134

        accuracy                           0.36      21125
       macro avg       0.94      0.94      0.26      21125
    weighted avg       0.94      0.94      0.94      21125

Accuracy :  93.8573423
```

Xception:
```
                  precision    recall  f1-score   support

           COVID       0.91      0.98      0.16       316
    Lung_Opacity       0.90      0.89      0.30       601
          Normal       0.95      0.95      0.49      1019
 Viral Pneumonia       0.95      0.94      0.08       134

        accuracy                           0.36      21125
       macro avg       0.92      0.93      0.91      21125
    weighted avg       0.91      0.90      0.92      21125

Accuracy :  90.2127659
```

VGG16:
```
                  precision    recall  f1-score   support

           COVID       0.94      0.99      0.96       316
    Lung_Opacity       0.92      0.92      0.92       601
          Normal       0.95      0.94      0.95      1019
 Viral Pneumonia       0.99      0.90      0.95       134

        accuracy                           0.94      21125
       macro avg       0.95      0.94      0.94      21125
    weighted avg       0.94      0.94      0.94      21125

Accuracy :  94.2316478
```

Random Forest (Combining The above 3 models):

```
Results Using All Features:

Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99       112
         1.0       0.93      0.90      0.92       167
         2.0       0.95      0.97      0.96       318
         3.0       0.97      0.97      0.97        38

    accuracy                           0.95       635
   macro avg       0.96      0.96      0.96       635
weighted avg       0.95      0.95      0.95       635

Accuracy :  95.118621102
```

Even though it was small, we were happy to increase our accuracy by a percent by applying Random Forest on the pre-trained models..

Summary and Conclusion

We were successful, with 95% accuracy, at classifying four different types of chest X-ray images using three different pre-trained models and increasing the accuracy further using a Random Forest algorithm by linking all our models together. Ensembling models is definitely a technique we'll continue to use in the future.

If we had to do it again, we might consider separating the data into only three categories: normal, COVID, and other lung infections. It wasn't quite clear from the data the difference between lung opacity and virtual pneumonia. It seems possible that "lung opacity" included examples of viral pneumonia, since it was only described as "non-COVID lung infections." If we were only interested in correctly identifying COVID infections (since that is a new type of infection that clinicians might have less experience with) it might make sense to lump the other types of infections into one category.

Finally, it would also be interesting to use more types of models -- as the ones we selected performed fairly similarly. However, previous research suggests that many of the CNN networks perform similarly on this data. Next steps would be to ensemble even more models.

References

X-Ray Images
[1]https://bimcv.cipf.es/bimcv-projects/bimcv-covid19/#1590858128006-9e640421-6711
[2]https://github.com/ml-workgroup/covid-19-image-repository/tree/master/png
[3]https://sirm.org/category/senza-categoria/covid-19/
[4]https://eurorad.org
[5]https://github.com/ieee8023/covid-chestxray-dataset
[6]https://figshare.com/articles/COVID-19_Chest_X-Ray_Image_Repository/12580328
[7]https://github.com/armiro/COVID-CXNet
[8]https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data
[9] https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

Random Forest Example:
https://github.com/amir-jafari/Data-Mining/blob/master/9-Random%20Forest/1-Example_Exercise/RF_1.py

Pre-trained Network Example:
https://github.com/yuxiaohuang/teaching/blob/master/gwu/machine_learning_I/fall_2020/code/p3_deep_learning/p3_c2_supervised_learning/p3_c2_s3_convolutional_neural_networks/case_study/case_study.ipynb

Vgg16 image,
https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/

REsNet50 image,
https://www.researchgate.net/figure/Left-ResNet50-architecture-Blocks-with-dotted-line-represents-modules-that-might-be_fig3_331364877
https://forums.fast.ai/t/whats-the-intuition-behind-skip-connections-in-resnet/63589

Xception image,
https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568

Other References

-M.E.H. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M.A. Kadir, Z.B. Mahbub, K.R. Islam, M.S. Khan, A. Iqbal, N. Al-Emadi, M.B.I. Reaz, M. T. Islam, "Can AI help in screening Viral and COVID-19 pneumonia?" IEEE Access, Vol. 8, 2020, pp. 132665 - 132676. Paper link

-Rahman, T., Khandakar, A., Qiblawey, Y., Tahir, A., Kiranyaz, S., Kashem, S.B.A., Islam, M.T., Maadeed, S.A., Zughaier, S.M., Khan, M.S. and Chowdhury, M.E., 2020. Exploring the Effect of Image Enhancement Techniques on COVID-19 Detection using Chest X-ray Images.
Paper Link

Appendix with Code:

```
######################################################################
# DESCRIPTION: This program is used to allocate a 10% Test data     #
#              as well as augment and over/under sample minority and#
#              majority classes in the training data respectively    #
######################################################################


import numpy as np
import os
import matplotlib.pyplot as plt
import random
import cv2

random.seed(13)      # so that our Test Data is the same
test_pct = 0.1       # set this equal to the percentage you want your test
data to be. Default 10%.

# Location of the data files
data_folder = "FinalProject/COVID-19_Radiography_Dataset/TrainData"
covid_data = data_folder + "/COVID/"
lung_data = data_folder + "/Lung_Opacity/"
norm_data = data_folder + "/Normal/"
viral_data = data_folder + "/Viral Pneumonia/"

#Test
test_data = "FinalProject/COVID-19_Radiography_Dataset/TestData"
covid_test = test_data + "/COVID/"
lung_test = test_data + "/Lung_Opacity/"
norm_test = test_data + "/Normal/"
viral_test = test_data + "/Viral Pneumonia/"

print(' Number of Viral data:', len(os.listdir(viral_data)))
print(' Number of Covid data:', len(os.listdir(covid_data)))
print(' Number of Normal data:', len(os.listdir(norm_data)))
print(' Number of Lung data:', len(os.listdir(lung_data)))

# Drawing a random sample of 10% of the data in each folder
cov_l = random.sample(os.listdir(covid_data),
np.int(np.floor(len(os.listdir(covid_data)) * test_pct)))
lung_l = random.sample(os.listdir(lung_data),
np.int(np.floor(len(os.listdir(lung_data)) * test_pct)))
nom_l = random.sample(os.listdir(norm_data),
np.int(np.floor(len(os.listdir(norm_data)) * test_pct)))
viral_l = random.sample(os.listdir(viral_data),
np.int(np.floor(len(os.listdir(viral_data)) * test_pct)))

#Moving the files from their respective folder to TestData folder
```

```
for l in cov_l:
   os.rename(covid_data + l, covid_test + l)
for l in lung_l:
   os.rename(lung_data + l, lung_test + l)
for l in nom_l:
   os.rename(norm_data + l, norm_test + l)
for l in viral_l:
   os.rename(viral_data + l, viral_test + l)

print(' Number of Viral data:', len(os.listdir(viral_data)))
print(' Number of Covid data:', len(os.listdir(covid_data)))
print(' Number of Normal data:', len(os.listdir(norm_data)))
print(' Number of Lung data:', len(os.listdir(lung_data)))


########## Augmenting and Oversampling Undersampling Images #####

# Oversample Viral and COVID classes
AUG_DIR = 'FinalProject/COVID-19_Radiography_Dataset/Augs/'
listc = os.listdir(covid_data)
for i in range(0,3258,8):
   img = cv2.imread(covid_data + listc[i])
   cv2.imwrite(covid_data + 'augfh' + str(i) + '.png', cv2.flip(img, 1))
   cv2.imwrite(covid_data + 'augfv' + str(i) + '.png', cv2.flip(img, 0))
   cv2.imwrite(covid_data + 'augrt' + str(i) + '.png', cv2.rotate(img,
cv2.ROTATE_90_CLOCKWISE))
   cv2.imwrite(covid_data + 'augrt' + str(i) + '.png', cv2.rotate(img,
cv2.ROTATE_90_COUNTERCLOCKWISE))

listv = os.listdir(viral_data)
for i in range(len(listv)):
   img = cv2.imread(viral_data + listv[i])
   cv2.imwrite(viral_data + 'augfh' + str(i) + '.png', cv2.flip(img, 1))
   cv2.imwrite(viral_data + 'augfv' + str(i) + '.png', cv2.flip(img, 0))
   cv2.imwrite(viral_data + 'augrt' + str(i) + '.png', cv2.rotate(img,
cv2.ROTATE_90_CLOCKWISE))
   cv2.imwrite(viral_data + 'augrt' + str(i) + '.png', cv2.rotate(img,
cv2.ROTATE_90_COUNTERCLOCKWISE))

pic = cv2.imread(AUG_DIR + 'augrn0.png')
plt.imshow(pic)
plt.show()

# Undersampling NORMAL Class #####
nom_l = random.sample(os.listdir(norm_data),
np.int(np.floor(len(os.listdir(norm_data)) * 0.44)))
for l in nom_l:
   os.rename(norm_data + l, AUG_DIR + l)
```

Here’s the code that trains our model using ResNet50. The code for Xception and VGG16 is available on our GitHub.

```
##############################################################
# DESCRIPTION: This program trains our model using RESNET50 #
#              pre-trained model.                            #
##############################################################


import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import random
import tensorflow as tf
import keras
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import cv2

random.seed(13)      # so that our Test Data is the same
test_pct = 0.1       # set this equal to the percentage you want your test
data to be. Default 10%.

# Location of the data files
abspath_curr = "COVID-19_Radiography_Dataset/"
data_folder = "COVID-19_Radiography_Dataset/TrainData/"
covid_data = data_folder + "COVID/"
lung_data = data_folder + "Lung_Opacity/"
norm_data = data_folder + "Normal/"
viral_data = data_folder + "Viral Pneumonia/"

#Test
test_data = "COVID-19_Radiography_Dataset/TestData/"
covid_test = test_data + "COVID/"
lung_test = test_data + "Lung_Opacity/"
norm_test = test_data + "Normal/"
viral_test = test_data + "Viral Pneumonia/"

print(' Number of Viral data:', len(os.listdir(viral_data)))
print(' Number of Covid data:', len(os.listdir(covid_data)))
print(' Number of Normal data:', len(os.listdir(norm_data)))
print(' Number of Lung data:', len(os.listdir(lung_data)))
#Viral 1211 (4844) Covid 3258 (4479) Normal 9176 (5140) Lung 5411
'''
########## Augmenting and Oversampling Undersampling Images #####
# Oversample Viral and COVID classes
AUG_DIR = 'FinalProject/COVID-19_Radiography_Dataset/Augs/'
list = os.listdir(covid_data)
```

```
for i in range(0,3258,8):
   img = cv2.imread(covid_data + list[i])
   cv2.imwrite(covid_data + 'augfh' + str(i) + '.png', cv2.flip(img, 1))
   cv2.imwrite(covid_data + 'augfv' + str(i) + '.png', cv2.flip(img, 0))
   cv2.imwrite(covid_data + 'augrt' + str(i) + '.png', cv2.rotate(img,
cv2.ROTATE_90_CLOCKWISE))
   cv2.imwrite(covid_data + 'augrt' + str(i) + '.png', cv2.rotate(img,
cv2.ROTATE_90_COUNTERCLOCKWISE))

pic = cv2.imread(AUG_DIR + 'augrn0.png')
plt.imshow(pic)
plt.show()

# Undersampling NORMAL Class #####
nom_l = random.sample(os.listdir(norm_data),
np.int(np.floor(len(os.listdir(norm_data)) * 0.44)))
for l in nom_l:
   os.rename(norm_data + l, AUG_DIR + l)
#################################################################
'''

batch = 32
split = 0.2
img_size = [244, 244]
random_seed = 13
n_epoch = 20

data_tr = tf.keras.preprocessing.image_dataset_from_directory(
   data_folder, labels='inferred',  class_names=None,
   color_mode='rgb', batch_size=batch, image_size=img_size, seed=random_seed,
   validation_split=split, subset='training', interpolation='bilinear')

data_val = tf.keras.preprocessing.image_dataset_from_directory(
   data_folder, labels='inferred',  class_names=None,
   color_mode='rgb', batch_size=batch, image_size=img_size, seed=random_seed,
   validation_split=split, subset='validation', interpolation='bilinear')


class_names_tr = data_tr.class_names
print(class_names_tr)
for images, labels in data_tr.take(1):
 print(images.shape, labels.shape)

'''
#Validation Data
class_names_val = data_val.class_names
print(class_names_val)
for images, labels in data_val.take(1):
 print(images.shape, labels.shape)
```

```
#Test Data
class_names_te = data_te.class_names
print(class_names_te)
for images, labels in data_te.take(1):
 print(images.shape, labels.shape)
'''

#global preprocess_input
preprocess_input = tf.keras.applications.resnet50.preprocess_input

def preprocess_pretrain(data, label):
   data_preprocessed = preprocess_input(data)
   return data_preprocessed, label

# Preprocess the training data using pretrained model
data_train = data_tr.map(preprocess_pretrain)
data_valid = data_val.map(preprocess_pretrain)


# Add the pretrained layers
pretrained_model = keras.applications.ResNet50(include_top=False,
weights='imagenet')
average_pooling =
keras.layers.GlobalAveragePooling2D()(pretrained_model.output)
drop = keras.layers.Dropout(0.2)
dropout = drop(average_pooling)
output = keras.layers.Dense(len(class_names_tr),
activation='softmax')(dropout)
model = keras.Model(inputs=pretrained_model.input, outputs=output)
model.summary()


# Freeze all the layers
for layer in pretrained_model.layers:
   layer.trainable = False

model.summary()

# ModelCheckpoint callback
model_checkpoint_cb = keras.callbacks.ModelCheckpoint(filepath=abspath_curr +
'Output/modelRESNET.h5',
                                                     save_best_only=True,
                                                     save_weights_only=True)

# EarlyStopping callback
early_stopping_cb = keras.callbacks.EarlyStopping(patience=2,
                                                 restore_best_weights=True)
```

```python
# ReduceLROnPlateau callback
reduce_lr_on_plateau_cb = keras.callbacks.ReduceLROnPlateau(factor=0.1,
                                                            patience=2)

# Compile the model
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])

# Train, evaluate and save the best model
history = model.fit(data_train,
                   epochs=n_epoch,
                   validation_data=data_valid,
                   callbacks=[model_checkpoint_cb,
                              early_stopping_cb,
                              reduce_lr_on_plateau_cb])


# Create a figure
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.tight_layout()
plt.savefig(abspath_curr + 'Output/learning_curve_freezingRes.pdf')
plt.show()



# For each layer in the pretrained model
for layer in pretrained_model.layers:
   layer.trainable = True

# Compile the model
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001),
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])

# Train, evaluate and save the best model
history = model.fit(data_train,
                   epochs=n_epoch,
                   validation_data=data_valid,
                   callbacks=[model_checkpoint_cb,
                              early_stopping_cb,
                              reduce_lr_on_plateau_cb])


# Create a figure
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.tight_layout()
plt.savefig(abspath_curr + 'Output/learning_curve_after_unfreezingREs.pdf')
plt.show()

# Reading Test Data
RESIZE_TO = 244

x, y = [], []
for path in [f for f in os.listdir(covid_test) if f[-4:] == ".png"]:
   x.append(cv2.resize(cv2.imread(covid_test + path), (RESIZE_TO,
RESIZE_TO)))
   y.append("COVID")
for path in [f for f in os.listdir(norm_test) if f[-4:] == ".png"]:
   x.append(cv2.resize(cv2.imread(norm_test + path), (RESIZE_TO, RESIZE_TO)))
   y.append("Normal")
for path in [f for f in os.listdir(viral_test) if f[-4:] == ".png"]:
   x.append(cv2.resize(cv2.imread(viral_test + path), (RESIZE_TO,
RESIZE_TO)))
   y.append("Viral")
for path in [f for f in os.listdir(lung_test) if f[-4:] == ".png"]:
   x.append(cv2.resize(cv2.imread(lung_test + path), (RESIZE_TO, RESIZE_TO)))
   y.append("Lung")



# y_test
class_names_tr = ["COVID", "Normal", "Viral", "Lung"]
y = np.array(y)
le = LabelEncoder()
le.fit(class_names_tr)
yt = le.transform(y)

# x_test
x = np.array(x)
x_test = preprocess_input(x, data_format = None)

y_pred = model.predict(x_test)
pred = np.argmax(y_pred, axis=1)

print(classification_report(yt, pred, target_names=class_names_tr))
print("Accuracy : ", accuracy_score(yt, pred) * 100)
print("\n")
```

The three models are available on our Github.

```
##################################################################
# DESCRIPTION: This program uploads all our trained models,      #
#              make predictions, and combines them to run through#
#              RANDOM FOREST                                      #
##################################################################

import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import cv2
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# FOLDER ASSIGNMENT
class_names_tr = ['COVID', 'Lung_Opacity', 'Normal', 'Viral Pneumonia']
abspath_curr = 'COVID-19_Radiography_Dataset/'
test_data = "COVID-19_Radiography_Dataset/TestData/"
covid_test = test_data + "COVID/"
lung_test = test_data + "Lung_Opacity/"
norm_test = test_data + "Normal/"
viral_test = test_data + "Viral Pneumonia/"

print(' Number of Viral data:', len(os.listdir(viral_test)))
print(' Number of Covid data:', len(os.listdir(covid_test)))
print(' Number of Normal data:', len(os.listdir(norm_test)))
print(' Number of Lung data:', len(os.listdir(lung_test)))


################# FUNCTIONS ######################

# FUNCTION TO LOAD DATA
def load_testdata(model_name, RESIZE_TO):
   x, y = [], []
   for path in os.listdir(covid_test):
       x.append(cv2.resize(cv2.imread(covid_test + path), (RESIZE_TO,
RESIZE_TO)))
       y.append("COVID")
   for path in os.listdir(norm_test):
       x.append(cv2.resize(cv2.imread(norm_test + path), (RESIZE_TO,
RESIZE_TO)))
```

```
        y.append("Normal")
    for path in os.listdir(viral_test):
        x.append(cv2.resize(cv2.imread(viral_test + path), (RESIZE_TO,
RESIZE_TO)))
        y.append("Viral")
    for path in os.listdir(lung_test):
        x.append(cv2.resize(cv2.imread(lung_test + path), (RESIZE_TO,
RESIZE_TO)))
        y.append("Lung")

    y = np.array(y)
    le = LabelEncoder()
    le.fit(["COVID", "Normal", "Viral", "Lung"])
    y = le.transform(y)

    x = np.array(x)
    if model_name == 'xception':
        x = tf.keras.applications.xception.preprocess_input(x,
data_format=None)
        print('Xception is chosen')
    elif model_name == 'resnet':
        x = tf.keras.applications.resnet50.preprocess_input(x,
data_format=None)
        print('RESNET is chosen')
    elif model_name == 'vgg16':
        x = tf.keras.applications.vgg16.preprocess_input(x, data_format=None)
        print('VGG16 is chosen')
    return x, y


# FUNCTION TO LOAD MODEL
def load_model(model_name):
    if model_name == 'xception':
        pretrained_model = keras.applications.Xception(include_top=False,
weights='imagenet')
        model_location = abspath_curr + 'Output/modelXception.h5'
        print('Xception is chosen')
    elif model_name == 'resnet':
        pretrained_model = keras.applications.ResNet50(include_top=False,
weights='imagenet')
        model_location = abspath_curr + 'Output/modelRESNET.h5'
        print('RESNET is chosen')
    elif model_name == 'vgg16':
        pretrained_model = keras.applications.VGG16(include_top=False,
weights='imagenet')
        model_location = abspath_curr + 'Output/modelVGG16.h5'
        print('VGG16 is chosen')

    # The model
    average_pooling =
keras.layers.GlobalAveragePooling2D()(pretrained_model.output)
    drop = keras.layers.Dropout(0.2)
    dropout = drop(average_pooling)
    output = keras.layers.Dense(4, activation='softmax')(dropout)
    model = keras.Model(inputs=pretrained_model.input, outputs=output)

    model.load_weights(filepath=model_location)
    model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    model.summary()

    return model


# FUNCTION FOR CLASSIFICATION REPORT
def class_report(x, y, model):
    y_pred = model.predict(x)
    pred = np.argmax(y_pred, axis=1)
    print(classification_report(y, pred, target_names=class_names_tr))
    print("Accuracy : ", accuracy_score(y, pred) * 100)
    print("\n")
    return y_pred


################# PROGRAM BEGINS ######################

# LOADING EACH MODEL AND SHOWING INDIVIDUAL PERFORMANCES
x_test, y_test = load_testdata('resnet', 244)
mymodel = load_model('resnet')
p1 = class_report(x_test, y_test, mymodel)

x_test, y_test = load_testdata('xception', 299)
mymodel = load_model('xception')
p2 = class_report(x_test, y_test, mymodel)

x_test, y_test = load_testdata('vgg16', 244)
mymodel = load_model('vgg16')
p3 = class_report(x_test, y_test, mymodel)


# COMBINING THE MODEL PREDICTIONS AND RUNNING RANDOM FOREST (ENSEMBLE MODEL)
X = np.concatenate((p1, p2, p3), axis=1)
Y = y_test
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,
random_state=100)
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)
y_pred_score = clf.predict_proba(X_test)

print("\n")
print("Results Using All Features: \n")
print("Classification Report: ")
print(classification_report(y_test, y_pred))
print("\n")
print("Accuracy : ", accuracy_score(y_test, y_pred) * 100)
print("\n")

conf_matrix = confusion_matrix(y_test, y_pred)

df_cm = pd.DataFrame(conf_matrix, index=class_names_tr,
columns=class_names_tr)
plt.figure(figsize=(15, 15))
hm = sns.heatmap(df_cm, cbar=False, annot=True, square=True, fmt='d',
annot_kws={'size': 20}, yticklabels=df_cm.columns,
                xticklabels=df_cm.columns)
hm.yaxis.set_ticklabels(hm.yaxis.get_ticklabels(), rotation=0, ha='right',
fontsize=20)
hm.xaxis.set_ticklabels(hm.xaxis.get_ticklabels(), rotation=0, ha='right',
fontsize=20)
plt.ylabel('True label', fontsize=20)
plt.xlabel('Predicted label', fontsize=20)
plt.tight_layout()
plt.show()
```