CS6005- Deep Learning

Assignment-4

Transfer learning assignment

V.S. Suryaa

2018103610

P-Batch

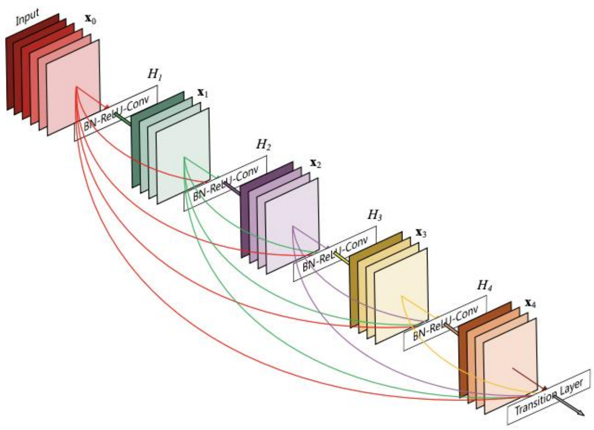
Problem statement:

To classify Chest X-ray images using transfer learning to identify Covid, Pneumonia and Healthy patients.

Transfer Learning:

Transfer learning is a strategy where a model developed for a task trained on a specific dataset is used as the starting point to solve a related problem. When transfer learning is used, the models do not require huge modifications to better adapt to the problem in hand.

In this work, we use DenseNet model for transfer learning purpose. In the DenseNet architecture, the output of each layer is connected to every subsequent layer in the model. For each layer the outputs or the feature maps of all preceding layers are used as inputs. The various inputs to each layer are concatenated to create a new feature map as the input to that layer. This concatenation of feature maps learned by different blocks increases the variation in the input of subsequent blocks and improves efficiency.



DenseNet169 architecture was used with new fully connected layers attached to it for classification purposes. The last few layers were retrained by unfreezing for better results and adaptation to the task given.

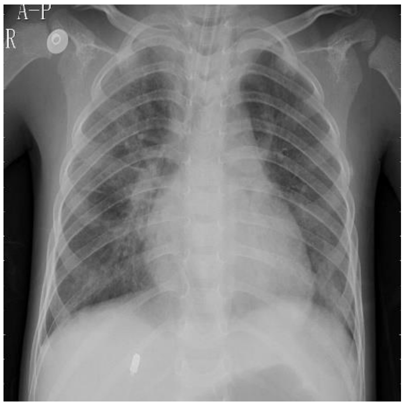
```
Total params: 13,528,387
Trainable params: 6,802,435
Non-trainable params: 6,725,952
```

After unfreezing the last few layers and attaching the new fully connected layers, the total parameters was 13,528,387 with trainable parameters as 6,802,435.

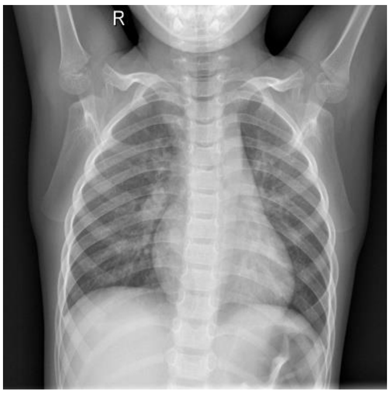
Dataset:

The dataset used is “COVID-19 Radiography Database” curated by a team of researchers from Qatar University. The database contains a total of 3,886 CXR images for Covid-19 patients along with healthy and viral pneumonia patients. There are 1,200 Covid-19 CXR images, 1,341 CXR images for Healthy patients and 1,345 CXR images for Viral Pneumonia patients. All images in the dataset are either in the Joint Photographic Expert Group (JPG/JPEG) or Portable Network Graphics (PNG) format.

Covid affected patient



Viral Pneumonia affected patient



Healthy patient

Module explanation:

1. Importing necessary libraries:

All the necessary libraries are imported for the execution of the program and visualization of the results.

2. Creating Image data generator:

Three different data generators are created for train, validation and test data. Image data generator is created with rescaling factor as 1/255 in all three. Feature-wise center is set to true transform the images to 0 mean and the images are randomly zoomed in by a factor upto 0.07 in training and validation data generators. The validation data split is 8% of the total data available for training. The target size is set as 256, hence the images will be of the size (256,256,3), where 3 denotes the RGB channels in the image. The batch size is uniformly set as 32.

3. Import DenseNet:

The DenseNet169 model trained on ImageNet data is loaded with its weights, excluding the top part (FCN). Global average pooling is performed on the output of the DenseNet and is fed into a fully connected layer with 512 units stacked with 64 units and a softmax layer of 3 units for classification. Dropout layers of 0.3 are added in the network to prevent overfitting.

The top few layers in the CNN are unfrozen to allow training the layers and updating their weights to better fit the data.

Parameter	**Value**
Learning rate	0.01
Optimizer	Stochastic Gradient Descent
Loss	Sparse Categorical Cross Entropy
Monitor	Validation loss

4. Start Training:

The model is trained for 10 epochs on the training data.

5. Prediction on test data:

The predicted values on the test data are obtained and the model is also evaluated for the accuracy and the loss on test set.

6. Visualize result:

The model's accuracy vs epochs graph and loss vs epochs graph are plotted for both training data and validation data. Classification report is generated for the test data which contains information about precision, recall and f1-score, and the confusion matrix is plotted with the heatmap for visualization.

Code:

Importing necessary libraries:

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import keras
from tensorflow.keras import layers, Input, optimizers, losses, activations, models, applications
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Convolution2D, Dense, Input, Flatten, Dropout, MaxPooling2D,
BatchNormalization, GlobalAveragePooling2D, Concatenate, BatchNormalization, GlobalAveragePooling2D, Activation
```

Creating Image data generator:

```
TARGET_SIZE = 256
train_datagen = ImageDataGenerator(rescale=1./255.,
                                     featurewise_center=True,
                                     validation_split = 0.08,
                                   zoom_range = 0.07
                                    )
test_datagen = ImageDataGenerator(rescale= 1./255)
train_data = train_datagen.flow_from_directory(
    'COVID-19 Radiography Database/Train',
    class_mode="sparse",
    batch_size=32,
    target_size=(TARGET_SIZE,TARGET_SIZE),
    shuffle=True,
    subset="training",
)

validation_data = train_datagen.flow_from_directory(
    'COVID-19 Radiography Database/Train',
    class_mode="sparse",
    batch_size=32,
    target_size=(TARGET_SIZE, TARGET_SIZE),
    shuffle=True,
    subset="validation",
)
```

```
test_data = test_datagen.flow_from_directory(
    'COVID-19 Radiography Database/Test',
    class_mode="sparse",
    batch_size=32,
    target_size=(TARGET_SIZE, TARGET_SIZE),
    shuffle=False,
)
```

```
Found 3290 images belonging to 3 classes.
Found 284 images belonging to 3 classes.
Found 312 images belonging to 3 classes.
```

Without Retraining layers:

Importing DenseNet model:

```
base_model = tf.keras.applications.DenseNet169(weights="imagenet",include_top=False,input_shape=(TARGET_SIZE,TARGET_SIZE, 3))
y1 = base_model.output
y = GlobalAveragePooling2D()(y1)
x = Dense(512, activation='relu')(y)
x=Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.3)(x)
predictions = Dense(3, activation='softmax')(x)

model=Sequential()
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(loss='sparse_categorical_crossentropy', optimizer= tf.keras.optimizers.SGD(learning_rate=0.1),metrics=['accuracy'])
model.summary()
model_save = ModelCheckpoint('weights.h5',
                             save_best_only = True,
                             save_weights_only = True,
                             monitor = 'val_loss',
                             mode = 'min', verbose = 1)
```

Start Training:

```
EPOCHS = 15
history = model.fit_generator(
      train_data,
      steps_per_epoch=train_data.samples//train_data.batch_size,
      epochs = EPOCHS,
      validation_data = validation_data,
      validation_steps=validation_data.samples//validation_data.batch_size,
      callbacks = [model_save])
```

```
Epoch 1/15
102/102 [==============================] - ETA: 0s - loss: 0.3795 - accuracy: 0.8735
Epoch 00001: val_loss improved from inf to 1.04616, saving model to weights.h5
102/102 [==============================] - 145s 1s/step - loss: 0.3795 - accuracy: 0.8735 - val_loss: 1.0462 - val_accuracy: 0.6367
Epoch 2/15
102/102 [==============================] - ETA: 0s - loss: 0.1390 - accuracy: 0.9561
Epoch 00002: val_loss improved from 1.04616 to 0.25613, saving model to weights.h5
102/102 [==============================] - 101s 992ms/step - loss: 0.1390 - accuracy: 0.9561 - val_loss: 0.2561 - val_accuracy: 0.8906
Epoch 3/15
102/102 [==============================] - ETA: 0s - loss: 0.1271 - accuracy: 0.9616
Epoch 00003: val_loss improved from 0.25613 to 0.05990, saving model to weights.h5
102/102 [==============================] - 102s 1s/step - loss: 0.1271 - accuracy: 0.9616 - val_loss: 0.0599 - val_accuracy: 0.9766
Epoch 4/15
102/102 [==============================] - ETA: 0s - loss: 0.0702 - accuracy: 0.9767
Epoch 00004: val_loss did not improve from 0.05990
102/102 [==============================] - 101s 989ms/step - loss: 0.0702 - accuracy: 0.9767 - val_loss: 0.1623 - val_accuracy: 0.9414
Epoch 5/15
102/102 [==============================] - ETA: 0s - loss: 0.0596 - accuracy: 0.9804
Epoch 00005: val_loss did not improve from 0.05990
102/102 [==============================] - 101s 985ms/step - loss: 0.0596 - accuracy: 0.9804 - val_loss: 0.0620 - val_accuracy: 0.9648
Epoch 6/15
102/102 [==============================] - ETA: 0s - loss: 0.0472 - accuracy: 0.9840
Epoch 00006: val_loss did not improve from 0.05990
102/102 [==============================] - 100s 981ms/step - loss: 0.0472 - accuracy: 0.9840 - val_loss: 0.8919 - val_accuracy: 0.7344
Epoch 7/15
102/102 [==============================] - ETA: 0s - loss: 0.0437 - accuracy: 0.9840
Epoch 00007: val_loss did not improve from 0.05990
102/102 [==============================] - 101s 988ms/step - loss: 0.0437 - accuracy: 0.9840 - val_loss: 0.2637 - val_accuracy: 0.8789
Epoch 8/15
102/102 [==============================] - ETA: 0s - loss: 0.0492 - accuracy: 0.9871
Epoch 00008: val_loss did not improve from 0.05990
102/102 [==============================] - 101s 987ms/step - loss: 0.0492 - accuracy: 0.9871 - val_loss: 0.4198 - val_accuracy: 0.8789
Epoch 9/15
102/102 [==============================] - ETA: 0s - loss: 0.0360 - accuracy: 0.9868
Epoch 00009: val_loss did not improve from 0.05990
102/102 [==============================] - 101s 990ms/step - loss: 0.0360 - accuracy: 0.9868 - val_loss: 0.2273 - val_accuracy: 0.9141
Epoch 10/15
102/102 [==============================] - ETA: 0s - loss: 0.0440 - accuracy: 0.9843
Epoch 00010: val_loss did not improve from 0.05990

Epoch 11/15
102/102 [==============================] - ETA: 0s - loss: 0.0276 - accuracy: 0.9914
Epoch 00011: val_loss did not improve from 0.05990
102/102 [==============================] - 101s 993ms/step - loss: 0.0276 - accuracy: 0.9914 - val_loss: 0.1007 - val_accuracy: 0.9648
Epoch 12/15
102/102 [==============================] - ETA: 0s - loss: 0.0258 - accuracy: 0.9908
Epoch 00012: val_loss did not improve from 0.05990
102/102 [==============================] - 101s 994ms/step - loss: 0.0258 - accuracy: 0.9908 - val_loss: 0.2959 - val_accuracy: 0.9180
Epoch 13/15
102/102 [==============================] - ETA: 0s - loss: 0.0236 - accuracy: 0.9932
Epoch 00013: val_loss did not improve from 0.05990
102/102 [==============================] - 101s 991ms/step - loss: 0.0236 - accuracy: 0.9932 - val_loss: 0.5126 - val_accuracy: 0.8594
Epoch 14/15
102/102 [==============================] - ETA: 0s - loss: 0.0130 - accuracy: 0.9969
Epoch 00014: val_loss did not improve from 0.05990
102/102 [==============================] - 102s 1000ms/step - loss: 0.0130 - accuracy: 0.9969 - val_loss: 0.0885 - val_accuracy: 0.9648
Epoch 15/15
102/102 [==============================] - ETA: 0s - loss: 0.0145 - accuracy: 0.9957
Epoch 00015: val_loss did not improve from 0.05990
102/102 [==============================] - 101s 991ms/step - loss: 0.0145 - accuracy: 0.9957 - val_loss: 0.1454 - val_accuracy: 0.9609
```

Prediction on test data:

```
pred = model.predict_generator(test_data)
y_pred = pred.argmax(axis=-1)
print(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2
 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2]
```

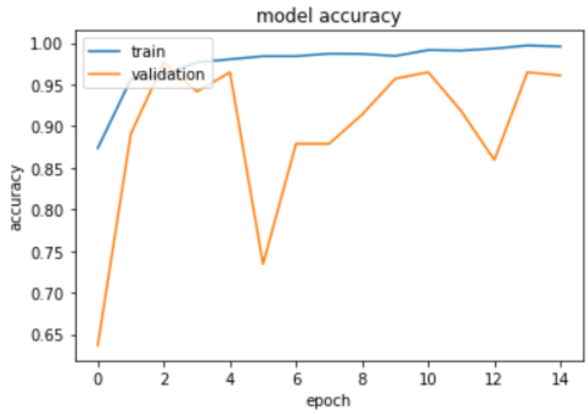
```
model.evaluate(test_data)
```

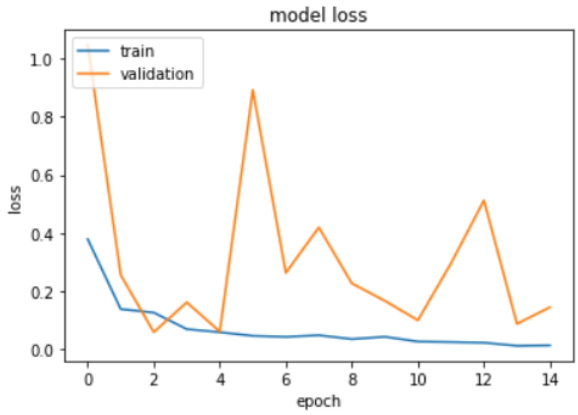
```
10/10 [==============================] - 4s 402ms/step - loss: 0.0460 - accuracy: 0.9872
[0.04603487253189087, 0.9871794581413269]
```

Visualize result:

```
print(history.history.keys())
#  "Accuracy"
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# "Loss"
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```



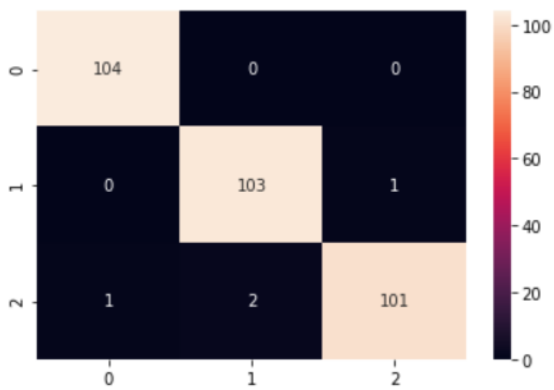



```python
from sklearn.metrics import classification_report, confusion_matrix
print('Confusion Matrix')
cm = confusion_matrix(test_data.classes, y_pred)
print(cm)
print(sns.heatmap(confusion_matrix(test_data.classes, y_pred),annot=True, fmt= "d"))
print(classification_report(test_data.classes, y_pred, digits=5))
```

```
Confusion Matrix
[[104   0   0]
 [  0 103   1]
 [  1   2 101]]
AxesSubplot(0.125,0.125;0.62x0.755)
              precision    recall  f1-score   support

           0    0.99048   1.00000   0.99522       104
           1    0.98095   0.99038   0.98565       104
           2    0.99020   0.97115   0.98058       104

    accuracy                        0.98718       312
   macro avg    0.98721   0.98718   0.98715       312
weighted avg    0.98721   0.98718   0.98715       312
```



```
recall = np.diag(cm) / np.sum(cm, axis = 1)
precision = np.diag(cm) / np.sum(cm, axis = 0)
print(np.mean(recall))
print(np.mean(precision))
```

```
0.9871794871794872
0.9872082166199814
```

Retraining few layers:

Importing DenseNet model:

```
base_model = tf.keras.applications.DenseNet169(weights="imagenet",include_top=False,input_shape=(TARGET_SIZE,TARGET_SIZE, 3))
y1 = base_model.output
y = GlobalAveragePooling2D()(y1)
x = Dense(512, activation='relu')(y)
x=Dropout(0.3)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.3)(x)
predictions = Dense(3, activation='softmax')(x)

for layer in base_model.layers[:369]:
    layer.trainable = False
for layer in base_model.layers[369:]:
    layer.trainable = True

model=Sequential()
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(loss='sparse_categorical_crossentropy', optimizer= tf.keras.optimizers.SGD(learning_rate=0.1),metrics=['accuracy']
model.summary()
model_save = ModelCheckpoint('weights.h5',
                             save_best_only = True,
                             save_weights_only = True,
                             monitor = 'val_loss',
                             mode = 'min', verbose = 1)
```

Start Training:

```
EPOCHS = 10
history = model.fit_generator(
      train_data,
      steps_per_epoch=train_data.samples//train_data.batch_size,
      epochs = EPOCHS,
      validation_data = validation_data,
      validation_steps=validation_data.samples//validation_data.batch_size,
      callbacks = [model_save])
```

```
Epoch 1/10
102/102 [==============================] - ETA: 0s - loss: 0.2834 - accuracy: 0.9067
Epoch 00001: val_loss improved from inf to 0.18882, saving model to weights.h5
102/102 [==============================] - 174s 2s/step - loss: 0.2834 - accuracy: 0.9067 - val_loss: 0.1888 - val_accuracy: 0.9258
Epoch 2/10
102/102 [==============================] - ETA: 0s - loss: 0.1284 - accuracy: 0.9543
Epoch 00002: val_loss improved from 0.18882 to 0.07443, saving model to weights.h5
102/102 [==============================] - 119s 1s/step - loss: 0.1284 - accuracy: 0.9543 - val_loss: 0.0744 - val_accuracy: 0.9805
Epoch 3/10
102/102 [==============================] - ETA: 0s - loss: 0.0718 - accuracy: 0.9785
Epoch 00003: val_loss improved from 0.07443 to 0.03955, saving model to weights.h5
102/102 [==============================] - 119s 1s/step - loss: 0.0718 - accuracy: 0.9785 - val_loss: 0.0396 - val_accuracy: 0.9844
Epoch 4/10
102/102 [==============================] - ETA: 0s - loss: 0.0737 - accuracy: 0.9770
Epoch 00004: val_loss did not improve from 0.03955
102/102 [==============================] - 118s 1s/step - loss: 0.0737 - accuracy: 0.9770 - val_loss: 0.0994 - val_accuracy: 0.9648
Epoch 5/10
102/102 [==============================] - ETA: 0s - loss: 0.0445 - accuracy: 0.9850
Epoch 00005: val_loss did not improve from 0.03955
102/102 [==============================] - 116s 1s/step - loss: 0.0445 - accuracy: 0.9850 - val_loss: 0.0973 - val_accuracy: 0.9805
Epoch 6/10
102/102 [==============================] - ETA: 0s - loss: 0.0449 - accuracy: 0.9853
Epoch 00006: val_loss did not improve from 0.03955
102/102 [==============================] - 118s 1s/step - loss: 0.0449 - accuracy: 0.9853 - val_loss: 0.1341 - val_accuracy: 0.9727
Epoch 7/10
102/102 [==============================] - ETA: 0s - loss: 0.0353 - accuracy: 0.9883
Epoch 00007: val_loss did not improve from 0.03955
102/102 [==============================] - 118s 1s/step - loss: 0.0353 - accuracy: 0.9883 - val_loss: 0.1640 - val_accuracy: 0.9336
Epoch 8/10
102/102 [==============================] - ETA: 0s - loss: 0.0362 - accuracy: 0.9917
Epoch 00008: val_loss did not improve from 0.03955
102/102 [==============================] - 119s 1s/step - loss: 0.0362 - accuracy: 0.9917 - val_loss: 0.0673 - val_accuracy: 0.9766
Epoch 9/10
102/102 [==============================] - ETA: 0s - loss: 0.0165 - accuracy: 0.9951
Epoch 00009: val_loss did not improve from 0.03955
102/102 [==============================] - 113s 1s/step - loss: 0.0165 - accuracy: 0.9951 - val_loss: 0.1738 - val_accuracy: 0.9727
Epoch 10/10
102/102 [==============================] - ETA: 0s - loss: 0.0271 - accuracy: 0.9890
Epoch 00010: val_loss did not improve from 0.03955
102/102 [==============================] - 109s 1s/step - loss: 0.0271 - accuracy: 0.9890 - val_loss: 0.0616 - val_accuracy: 0.9727
```

Prediction on test data:

```python
pred = model.predict_generator(test_data)
y_pred = pred.argmax(axis=-1)
print(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

```python
model.evaluate(test_data)
```

```
10/10 [==============================] - 5s 464ms/step - loss: 0.0249 - accuracy: 0.9968
[0.02493244595825672, 0.9967948794364929]
```

Test data loss: 0.0249

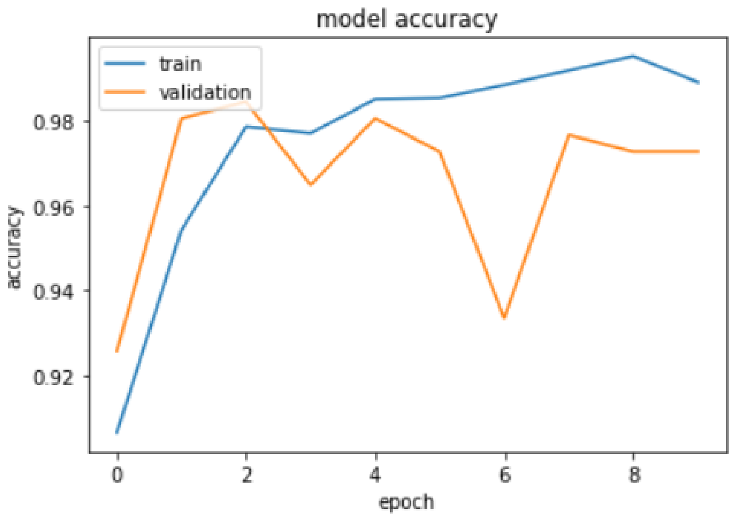
Test data accuracy: 99.679%

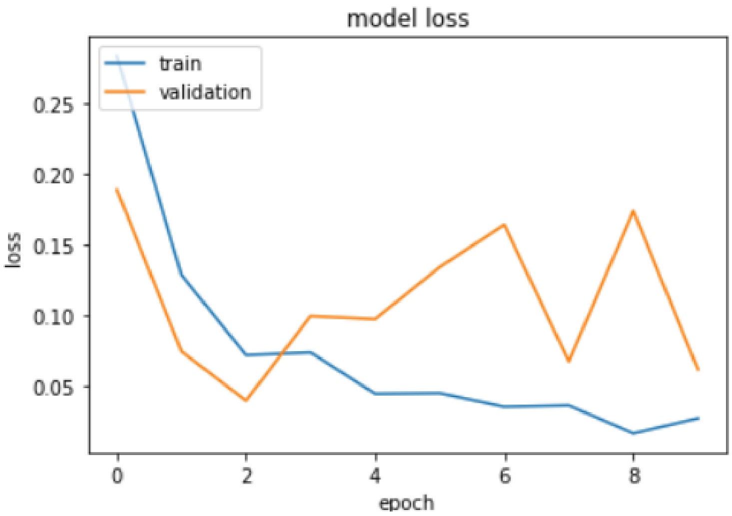
Visualize results:

```
print(history.history.keys())
#  "Accuracy"
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
# "Loss"
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Accuracy and Loss Vs Epochs for Training and Validation:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





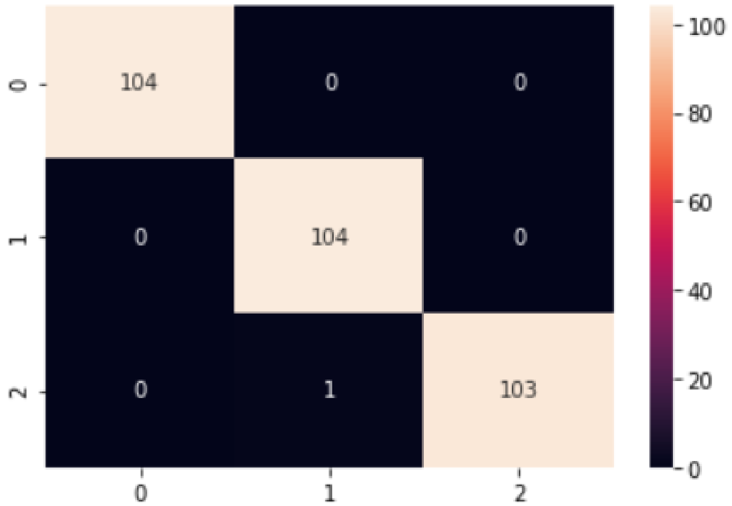
Classification report on test data:

```
from sklearn.metrics import classification_report, confusion_matrix
print('Confusion Matrix')
cm = confusion_matrix(test_data.classes, y_pred)
print(cm)
print(sns.heatmap(confusion_matrix(test_data.classes, y_pred),annot=True, fmt= "d"))
print(classification_report(test_data.classes, y_pred, digits=5))
```

```
Confusion Matrix
[[104   0   0]
 [  0 104   0]
 [  0   1 103]]
AxesSubplot(0.125,0.125;0.62x0.755)
              precision    recall  f1-score   support

           0    1.00000   1.00000   1.00000       104
           1    0.99048   1.00000   0.99522       104
           2    1.00000   0.99038   0.99517       104

    accuracy                        0.99679       312
   macro avg    0.99683   0.99679   0.99679       312
weighted avg    0.99683   0.99679   0.99679       312
```




```
recall = np.diag(cm) / np.sum(cm, axis = 1)
precision = np.diag(cm) / np.sum(cm, axis = 0)
print(np.mean(recall))
print(np.mean(precision))
```

```
0.9967948717948718
0.9968253968253968
```

Results:

The last few layers retrained model performed really well on the test data with just one misclassification on the whole testing.

The final results on the test data are:

Parameter	Without Retraining	With Retrained layers
Accuracy	98.718	99.679
Precision	98.721	99.683
Recall	98.718	99.679
F1-score	98.715	99.679
Loss	0.04603	0.0249

Conclusion:

The DenseNet 169 model with last few layers retrained and used for the classification of Chest X-ray images into Covid, Pneumonia and Healthy performed well on the testing data using transfer learning approach.

References:

1. Dataset: https://www.kaggle.com/tawsifurrahman/covid19-radiography-database
2. DenseNet: G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261-2269, doi: 10.1109/CVPR.2017.243.