

# Notebook

January 24, 2023

## 1 Image Classification with Deep Learning Project #4

- Student name: *Aysu Erdemir*
- Student pace: *Flex*
- Scheduled project review date/time: *January, 2023*
- Instructor name: *Abhineet Kulkarni*

### 1.0.1 Overview:

- A radiology department at 'Be Well Healthcare Center' is interested in decreasing the work load of its radiologists by hiring a data scientist to develop a machine learning model to screen chest x-ray images and automatically mark those with pneumonia.
- My goal is to build a Neural Network model to classify chest x-ray images as belonging to one of the two categories: pneumonia and normal. My main purpose was to make predictions as accurately as possible while maximizing the number of True Positives (recall) and minimizing False Negatives, so that we catch as many people as possible with pneumonia (at the expense of marking some healthy people with pneumonia).

### 1.0.2 Data Understanding:

The data was obtained from [Kaggle](#). The dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care.

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.

### 1.0.3 Modeling:

1. The data was re-split into training, test and val sets using different ratios: 80% for train, 5% for validation and 15% for the test set.
2. The data was pre-processed, specifically the images were re-scaled and standardized.
3. Several versions of neural networks were built, tuned and validated:

- Artificial Neural Network ANN
- Convolutional Neural Network CNN
- Transfer Learning with VGG16
- Transfer Learning with ResNest50V2

#### 1.0.4 Evaluation:

4. Accuracy was used as the scoring metric for tuning hyperparameters and recall was used for evaluating model performance. Whilst we would prefer an overall higher accuracy, our focus is on recall as this metric is particularly important for patient safety and to minimize the legal risk.

```
[1]: # Import required packages
import pandas as pd
import numpy as np
import seaborn as sns
import math

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline

from tensorflow import keras
from keras import layers
from keras import models
from keras import optimizers
from keras import regularizers
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout ,
↳BatchNormalization
from keras.regularizers import l2
from keras.optimizers import SGD
from keras.wrappers import scikit_learn
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator

from sklearn.metrics import classification_report, accuracy_score,
↳confusion_matrix, ConfusionMatrixDisplay

import warnings
warnings.filterwarnings('ignore')

import os
```

```
[2]: # Making sure we work on CPU
# You can set environment variables in the notebook using os.environ. Do the
# following before initializing TensorFlow to limit TensorFlow to first GPU.
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="0"

# You can double check that you have the correct devices visible to TF
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 5315553729275501727
, name: "/device:XLA_CPU:0"
device_type: "XLA_CPU"
memory_limit: 17179869184
locality {
}
incarnation: 9774807852142177369
physical_device_desc: "device: XLA_CPU device"
]
```

## 2 Data Visualization

```
[3]: # specify the local directory of train, validation and test data:
train_dir = "Data/chest_xray/train/"
val_dir = "Data/chest_xray/val/"
test_dir = "Data/chest_xray/test/"
```

### 2.0.1 Display the first image with pneumonia from the train dataset:

```
[4]: os.listdir("Data/chest_xray/train/PNEUMONIA")[0:15]
```

```
[4]: ['person63_bacteria_306.jpeg',
'person1438_bacteria_3721.jpeg',
'person755_bacteria_2659.jpeg',
'person478_virus_975.jpeg',
'person661_bacteria_2553.jpeg',
'person276_bacteria_1296.jpeg',
'person1214_bacteria_3166.jpeg',
'person1353_virus_2333.jpeg',
'person26_bacteria_122.jpeg',
'person124_virus_238.jpeg',
```

```
'person1619_bacteria_4261.jpeg',
'person890_bacteria_2814.jpeg',
'person1514_bacteria_3964.jpeg',
'person1491_bacteria_3893.jpeg',
'person69_bacteria_338.jpeg']
```

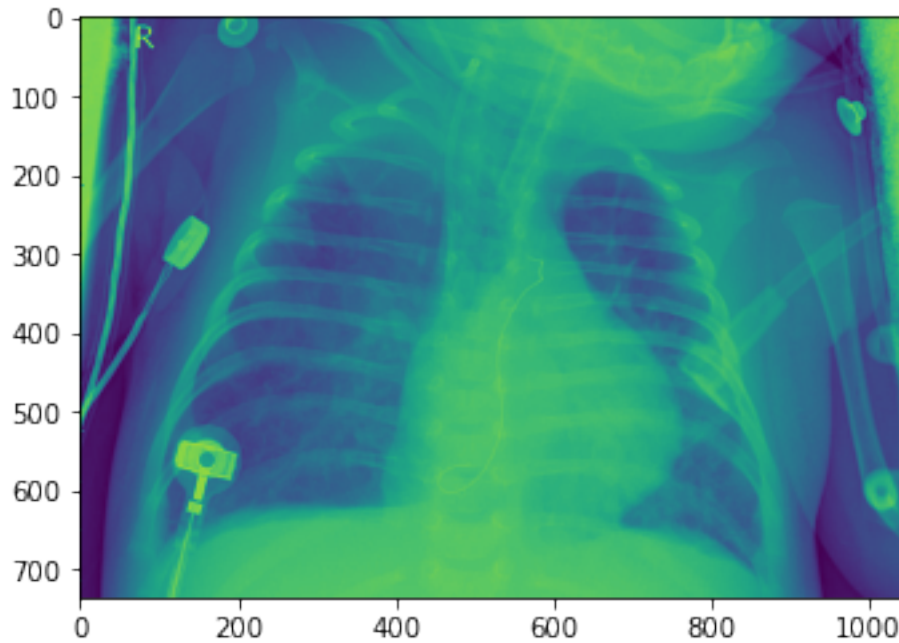
```
[5]: # List the images from the Data/chest_xray/train/PNEUMONIA directory:
pneumonia = os.listdir(train_dir+'/PNEUMONIA')

# Combine the image directory with the specific jpeg to be able to locate it.
# Read the image into an array using imread():
sample_img = plt.imread(os.path.join(train_dir+'/PNEUMONIA', pneumonia[0]))

print(f"Image name: {pneumonia[0]}")
print(image.load_img('Data/chest_xray/train/PNEUMONIA/person63_bacteria_306.
↪jpeg'))
print("-----")
print(f"Image Shape: {sample_img.shape}") # width and height
print(f"Max pixel: {sample_img.max()}")
print(f"Min pixel: {sample_img.min()}")

# Display the image
plt.imshow(sample_img);
```

```
Image name: person63_bacteria_306.jpeg
<PIL.Image.Image image mode=RGB size=1048x736 at 0x7FCD9AA1BFD0>
-----
Image Shape: (736, 1048)
Max pixel: 255
Min pixel: 0
```



```
[6]: # Print image
      print(sample_img)
```

```
[[210 208 206 ... 213 215 216]
 [210 208 206 ... 213 216 217]
 [210 208 205 ... 213 216 218]
 ...
 [ 16  15  14 ...  47  47  47]
 [ 16  15  14 ...  46  46  46]
 [ 16  15  14 ...  46  46  46]]
```

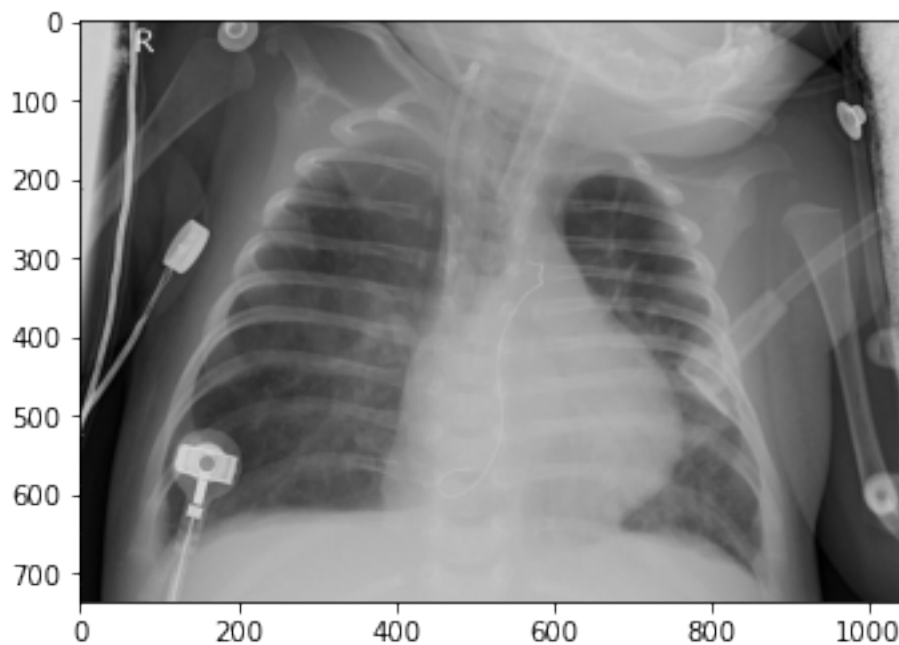
```
[7]: # Another way to visualize the image using image library:
filename = 'Data/chest_xray/train/PNEUMONIA/person63_bacteria_306.jpeg'
img = image.load_img(filename)
img_array = image.img_to_array(img) # converts image into a numpy array
img_tensor = np.expand_dims(img_array, axis=0) # expand the shape of an array,
        ↳ by inserting a new axis

# Standardize the pixel values:
img_tensor /= 255.

# Check tensor shape
print(img_tensor.shape)
# Print tensor
print(img_tensor[0][0])
```

```
# Preview the image
plt.imshow(img_tensor[0])
plt.show()
```

```
(1, 736, 1048, 3)
[[0.8235294  0.8235294  0.8235294 ]
 [0.8156863  0.8156863  0.8156863 ]
 [0.80784315 0.80784315 0.80784315]
 ...
 [0.8352941  0.8352941  0.8352941 ]
 [0.84313726 0.84313726 0.84313726]
 [0.84705883 0.84705883 0.84705883]]
```



- The dimensions of the image are 1048 pixels width and 736 pixels height.
- RGB color mode.
- The maximum pixel value is 255 and the minimum is 0.

### 2.0.2 Visualize the first 4 normal and 4 pneumonia x-rays:

```
[9]: # Specify the set of images inside pneumonia and normal
pneumonia = os.listdir("Data/chest_xray/train/PNEUMONIA")
normal = os.listdir("Data/chest_xray/train/NORMAL")

# Specify the location of the files = the directory
pneumonia_dir = "Data/chest_xray/train/PNEUMONIA"
```

```

normal_dir = "Data/chest_xray/train/NORMAL"

# Show the first four pictures from the train set of X-rays with and without
  ↳ Pneumonia

with plt.style.context('seaborn-talk'):
    fig, ax = plt.subplots(2, 4, figsize=(14,7))
    for i in range(4):
        # Combine the image directory with the specific jpeg to be able to
        ↳ locate it
        # Read the image into an array.
        img_pneumonia = plt.imread(os.path.join(pneumonia_dir, pneumonia[i]))
        img_normal = plt.imread(os.path.join(normal_dir, normal[i]))

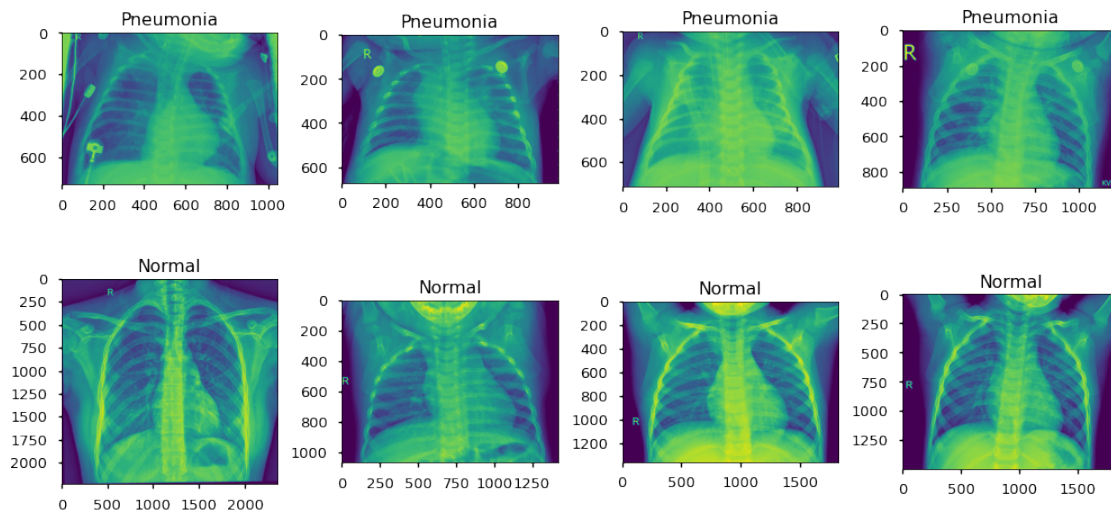
        # Display the image
        ax[0,i].imshow(img_pneumonia)
        ax[1,i].imshow(img_normal)

        # ax[0,i].set_axis_off()
        # ax[1,i].set_axis_off()

        ax[0,i].set_title("Pneumonia")
        ax[1,i].set_title("Normal")

plt.tight_layout()
plt.savefig('./images/RawImages', dpi=300, bbox_inches='tight')

```



- The normal chest X-ray (top panel) seem to depict more clear lungs without any areas of abnormal opacification.
- The chest X-ray (bottom panel) with pneumonia seem to depict less clear lungs with some

areas of opacification/consolidation.

### 2.0.3 What is total number of normal and pneumonia images in the train-test-val sets?

```
[10]: # Specify the set of images inside pneumonia and normal for train, test, val
      ↪sets:
num_pneumonia_train = (len(os.listdir("Data/chest_xray/train/PNEUMONIA")))
num_normal_train = (len(os.listdir("Data/chest_xray/train/NORMAL")))

num_pneumonia_test = (len(os.listdir("Data/chest_xray/test/PNEUMONIA")))
num_normal_test = (len(os.listdir("Data/chest_xray/test/NORMAL")))

num_pneumonia_val = (len(os.listdir("Data/chest_xray/val/PNEUMONIA")))
num_normal_val = (len(os.listdir("Data/chest_xray/val/NORMAL")))

with plt.style.context('seaborn-talk'):
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 5))
    ax1.bar(x = ["Pneumonia", "Normal"], height=[num_pneumonia_train,
    ↪num_normal_train], color=["blue", "red"])
    ax1.set_title('Train Set')
    ax2.bar(x = ["Pneumonia", "Normal"], height=[num_pneumonia_test,
    ↪num_normal_test], color=["blue", "red"])
    ax2.set_title('Test Set')
    ax3.bar(x = ["Pneumonia", "Normal"], height=[num_pneumonia_val,
    ↪num_normal_val], color=["blue", "red"])
    ax3.set_title('Val Set')

    ax1.set_ylim([0, 4000])
    ax2.set_ylim([0, 4000])
    ax3.set_ylim([0, 4000])
    plt.tight_layout()

print(f"Train Pneumonia: {num_pneumonia_train}")
print(f"Train Normal: {num_normal_train}")
print("-----")
print(f"Test Pneumonia: {num_pneumonia_test}")
print(f"Test Normal: {num_normal_test}")
print("-----")
print(f"Val Pneumonia: {num_pneumonia_val}")
print(f"Val Normal: {num_normal_val}")
print("-----")

TrainTotal = num_pneumonia_train + num_normal_train
TestTotal = num_pneumonia_test + num_normal_test
ValTotal = num_pneumonia_val + num_normal_val
Total = TrainTotal + TestTotal + ValTotal
```



```
print(f"Train Images Percentage: {np.round((TrainTotal / Total),3) }")
print(f"Test Images Percentage: {np.round((TestTotal / Total),3) }")
print(f"Val Images Percentage: {np.round((ValTotal / Total),3) }")
```

Train Pneumonia: 3876

Train Normal: 1342

Test Pneumonia: 390

Test Normal: 234

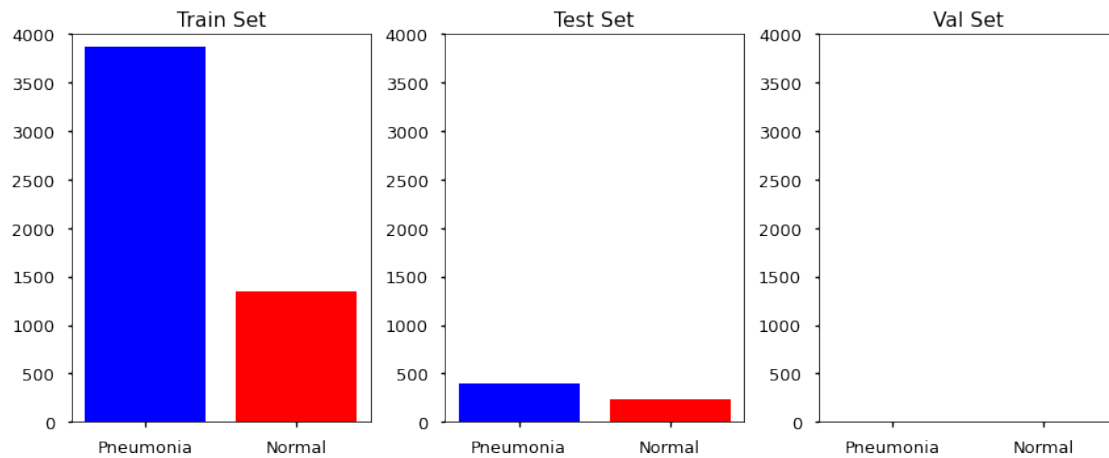
Val Pneumonia: 9

Val Normal: 9

Train Images Percentage: 0.89

Test Images Percentage: 0.106

Val Images Percentage: 0.003



- Number of images in the **validation** set appears too low with a total of 18 images and total percentage of **.3%**. We need to boost it to at about 10%.
- Number of images in the **test** set appears ideal with a total percentage of **10%**.
- Number of images in the **train** set is good with a total percentage of **89%**, but we will move some of the images from train to the validation set to boost the numbers in the val set. 80% is sufficient for the train set.

#### 2.0.4 Change the size of the Train-Test-Val sets:

- Using splitfolders package (<https://pypi.org/project/split-folders/>)
- Split the data in the folder “Data/INPUT” with “Pneumonia” and “Normal” subfolders into the output folder “Data/OUTPUT” with “train”, “test”, “val” subfolders using the pre-specified percentages.

```
[11]: # !pip install split-folders
import splitfolders

# Change the size of Train, Test and Val sets with ratios: .81, .09, .1.
# To only split into training and validation set, set a tuple to `ratio`, i.e.,
↳ `(.8, .2)`.
splitfolders.ratio("Data/INPUT", output="Data/OUTPUT",
    seed=1337, ratio=(.81, .09, .1), group_prefix=None, move=False) # default
↳ values
```

Copying files: 5856 files [00:08, 694.03 files/s]

### 2.0.5 What is total number of normal and pneumonia images in the NEW train-test-val sets?

```
[12]: num_pneumonia_train = (len(os.listdir("Data/OUTPUT/train/PNEUMONIA")))
num_normal_train = (len(os.listdir("Data/OUTPUT/train/NORMAL")))

num_pneumonia_test = (len(os.listdir("Data/OUTPUT/test/PNEUMONIA")))
num_normal_test = (len(os.listdir("Data/OUTPUT/test/NORMAL")))

num_pneumonia_val = (len(os.listdir("Data/OUTPUT/val/PNEUMONIA")))
num_normal_val = (len(os.listdir("Data/OUTPUT/val/NORMAL")))

with plt.style.context('seaborn-talk'):
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 5))
    ax1.bar(x = ["Pneumonia", "Normal"], height=[num_pneumonia_train,
↳ num_normal_train], color=["blue", "red"])
    ax1.set_title('Train Set')
    ax2.bar(x = ["Pneumonia", "Normal"], height=[num_pneumonia_test,
↳ num_normal_test], color=["blue", "red"])
    ax2.set_title('Test Set')
    ax3.bar(x = ["Pneumonia", "Normal"], height=[num_pneumonia_val,
↳ num_normal_val], color=["blue", "red"])
    ax3.set_title('Val Set')

    ax1.set_ylim([0, 4000])
    ax2.set_ylim([0, 4000])
    ax3.set_ylim([0, 4000])
    plt.tight_layout()

print(f"Train Pneumonia: {num_pneumonia_train}")
print(f"Train Normal: {num_normal_train}")
print("-----")
print(f"Test Pneumonia: {num_pneumonia_test}")
print(f"Test Normal: {num_normal_test}")
print("-----")
```

```

print(f"Val Pneumonia: {num_pneumonia_val}")
print(f"Val Normal: {num_normal_val}")
print("-----")

TrainTotal = num_pneumonia_train + num_normal_train
TestTotal = num_pneumonia_test + num_normal_test
ValTotal = num_pneumonia_val + num_normal_val
Total = TrainTotal + TestTotal + ValTotal
print(f"Train Images Percentage: {np.round((TrainTotal / Total),3) }")
print(f"Test Images Percentage: {np.round((TestTotal / Total),3) }")
print(f"Val Images Percentage: {np.round((ValTotal / Total),3) }")

```

Train Pneumonia: 3461

Train Normal: 1282

-----

Test Pneumonia: 428

Test Normal: 159

-----

Val Pneumonia: 384

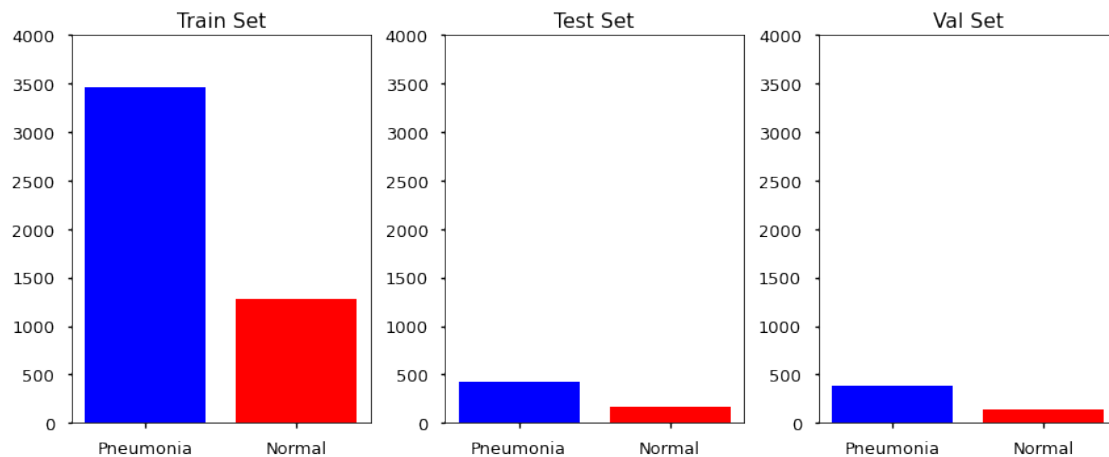
Val Normal: 142

-----

Train Images Percentage: 0.81

Test Images Percentage: 0.1

Val Images Percentage: 0.09



- New split looks much better with about 80% train, 10% test and 10% validation.

```

[13]: # Specify the new location of the train, test and val images
train_dir = "Data/OUTPUT/train/"
test_dir = "Data/OUTPUT/test/"
val_dir = "Data/OUTPUT/val/"

```

## 3 Image Preprocessing:

### 3.1 Data Normalization:

- **Pixel Normalization:** For images, a common thing that is done is to make sure each pixel value is between 0 and 1. This can be done by dividing the entire matrix by 255.
- **Size Rescaling:** Since all of the images are different sizes, we need to rescale (standardize) them using a target width and height. We will use a size of 128x128. In general, the resolutions for training CNNs usually range between  $64 \times 64$  and  $256 \times 256$ .
- **Reshaping into tensors:** Convolutional Neural Networks takes its input as **tensors**. We will use the output from `flow_from_directory()` directly to feed them into the CNNs.
- **Reshaping into vectors:** A multilayer perceptron (MLP) - fully connected artificial Neural Network (ANN) takes its input as **vectors** (single-dimensional array), not matrices (2-D grid) or tensors (generalized matrix). So we need to reshape the images into a single 16384-dimensional vector ( $128 \times 128$ ) to feed them into the ANNs.

```
[14]: print(f"Train Images Total#: {TrainTotal}")
      print(f"Test Images Total#: {TestTotal}")
      print(f"Val Images Total#: {ValTotal}")
```

Train Images Total#: 4743

Test Images Total#: 587

Val Images Total#: 526

#### 3.1.1 Format the data using Keras ImageDataGenerator():

- Rescale pixel values to be between 0-1.
- Rescale the size to be 128 x 128
- Reformat the target data into 1s (pneumonia) and 0s (normal) # `class_mode='binary'`

```
[15]: # For example, if you have 1000 images in your dataset and the batch size is
      ↪ defined as 10.
      # Then the "ImageDataGenerator" will produce 10 images in each iteration of the
      ↪ training.
      # We will use the size of the whole dataset since there are no computational
      ↪ issues with ~5000 images.

      # Each pixel is originally between 255 and 0, Rescale the data to be between 0
      ↪ and 1.
      train_datagen = ImageDataGenerator(rescale=1/255)
      test_datagen = ImageDataGenerator(rescale=1/255)
      val_datagen = ImageDataGenerator(rescale=1/255)

      # Target Size: 128 x 128: The dimensions to which all images found will be
      ↪ resized:

      # get the data from the training directory
```

```

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(128, 128),
                                                    batch_size=4743, # default
                                                    class_mode='binary')

    ↪ is 32

# get the data from the validation directory
test_generator = test_datagen.flow_from_directory(test_dir,
                                                    target_size=(128, 128),
                                                    batch_size=587,
                                                    class_mode='binary')

# get the data from the validation directory
val_generator = val_datagen.flow_from_directory(val_dir,
                                                target_size=(128, 128),
                                                batch_size=526,
                                                class_mode='binary')

print(train_generator.class_indices)

```

Found 4743 images belonging to 2 classes.

Found 587 images belonging to 2 classes.

Found 526 images belonging to 2 classes.

{'NORMAL': 0, 'PNEUMONIA': 1}

4743 divisors: 17, 31, 51, 93, 153,

```

[16]: train_generator.image_shape, test_generator.image_shape, val_generator.
    ↪ image_shape

```

```

[16]: ((128, 128, 3), (128, 128, 3), (128, 128, 3))

```

### 3.1.2 Create the transformed data sets:

```

[17]: # next() returns the next item in the iterator = The first batch of the images,
    ↪ in our case all images.
train_image, train_label = next(train_generator)
test_image, test_label = next(test_generator)
val_image, val_label = next(val_generator)

```

```

[18]: print(train_image.shape)
print(train_label.shape)

```

```

(4743, 128, 128, 3)

```

```

(4743,)

```

```

[19]: train_image[0]

```

```

[19]: array([[0.          , 0.          , 0.          ],
             [0.          , 0.          , 0.          ],
             [0.          , 0.          , 0.          ],
             ...,
             [0.6117647 , 0.6117647 , 0.6117647 ],
             [0.8117648 , 0.8117648 , 0.8117648 ],
             [0.9960785 , 0.9960785 , 0.9960785 ]],

          [[0.          , 0.          , 0.          ],
           [0.          , 0.          , 0.          ],
           [0.          , 0.          , 0.          ],
           ...,
           [0.6039216 , 0.6039216 , 0.6039216 ],
           [0.83921576, 0.83921576, 0.83921576],
           [1.          , 1.          , 1.          ]],

          [[0.          , 0.          , 0.          ],
           [0.          , 0.          , 0.          ],
           [0.          , 0.          , 0.          ],
           ...,
           [0.5882353 , 0.5882353 , 0.5882353 ],
           [0.8078432 , 0.8078432 , 0.8078432 ],
           [0.9960785 , 0.9960785 , 0.9960785 ]],

          ...,

          [[0.          , 0.          , 0.          ],
           [0.03529412, 0.03529412, 0.03529412],
           [0.07058824, 0.07058824, 0.07058824],
           ...,
           [0.454902 , 0.454902 , 0.454902 ],
           [0.5137255 , 0.5137255 , 0.5137255 ],
           [0.77647066, 0.77647066, 0.77647066]],

          [[0.          , 0.          , 0.          ],
           [0.03529412, 0.03529412, 0.03529412],
           [0.07843138, 0.07843138, 0.07843138],
           ...,
           [0.41960788, 0.41960788, 0.41960788],
           [0.5137255 , 0.5137255 , 0.5137255 ],
           [0.75294125, 0.75294125, 0.75294125]],

          [[0.          , 0.          , 0.          ],
           [0.02352941, 0.02352941, 0.02352941],
           [0.08235294, 0.08235294, 0.08235294],
           ...,
           [0.427451 , 0.427451 , 0.427451 ]],

```

```
[0.53333336, 0.53333336, 0.53333336],
 [0.7686275 , 0.7686275 , 0.7686275 ]]], dtype=float32)
```

### 3.1.3 Reshape the transformed data into vectors:

[20]: *# Reshape images:*

```
X_train = train_image.reshape(train_image.shape[0], -1)
X_test = test_image.reshape(test_image.shape[0], -1)
X_val = val_image.reshape(val_image.shape[0], -1)

print(X_train.shape)
print(X_test.shape)
print(X_val.shape)
```

```
(4743, 49152)
(587, 49152)
(526, 49152)
```

[21]: *# Reshape labels:*

```
y_train = np.reshape(train_label, (4743,1))    # y_train2 = train_label.
    ↪ reshape(train_label.shape[0], 1)
y_test = np.reshape(test_label, (587,1))
y_val = np.reshape(val_label, (526,1))

print(y_train.shape)
print(y_test.shape)
print(y_val.shape)
```

```
(4743, 1)
(587, 1)
(526, 1)
```

### 3.1.4 Visualize some of the transformed images from the training dataset:

- The images appear in a standardized way now:

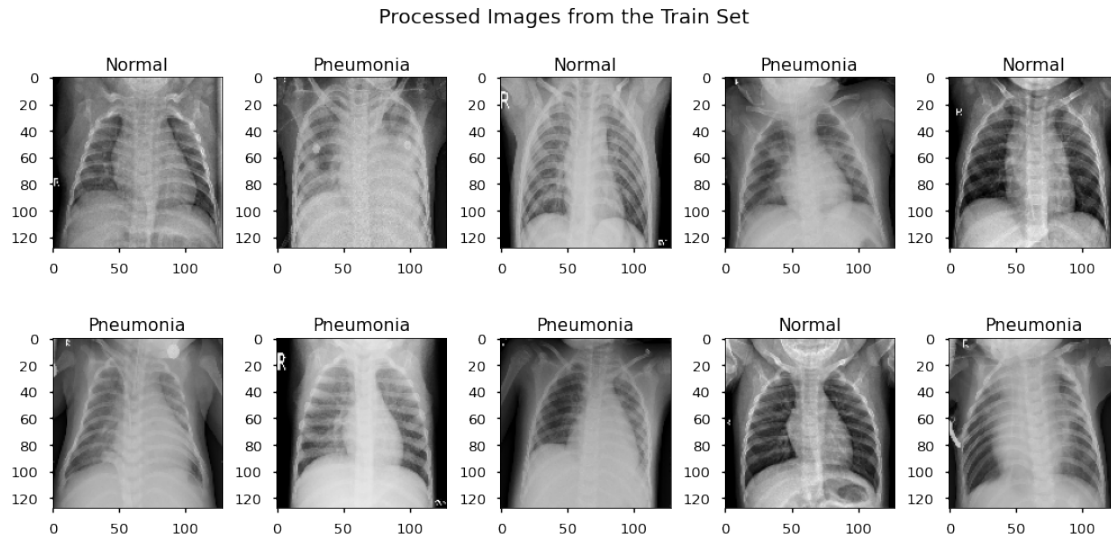
[132]: `plt.figure(figsize=(14,7))`

```
with plt.style.context('seaborn-talk'):
    for i in range(10):
        ax = plt.subplot(2,5,i+1) # This is object oriented and different from
        ↪ plt.subplots()
        plt.imshow(train_image[i])
        if train_label[i]==0:
            plt.title("Normal")
```

```

else:
    plt.title("Pneumonia")
    # plt.axis('off')
plt.suptitle('Processed Images from the Train Set',fontsize=18)
plt.tight_layout()
plt.savefig('./images/ProcessedImages', dpi=300, bbox_inches='tight')

```



## 4 MODELING:

### 5 Baseline Artificial Neural Network:

- One input layer with 5 neurons with relu activation which works well with images.
- One output layer with 1 neuron for the binary classification (normal versus pneumonia).
- Input shape is a vector with 16384 rows (128 x 128)
- Using Adam optimizer (“Adaptive Moment Estimation”) an adaptive learning rate optimizer which is widely used for training deep neural networks and in image processing. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm. Learning rate defaults to 0.001 but the contribution of gradients to updated weight varies over epochs, hence ADAPTIVE.

```

[23]: # Size of the image vector:
128*128*3

```

```

[23]: 49152

```

```

[24]: # specify the model:
model = models.Sequential()

```



```

# Add dense layers with relu activation
model.add(layers.Dense(5, activation='relu', input_shape = (49152,)))
# Add final layer with sigmoid activation
model.add(layers.Dense(1, activation='sigmoid')) # because this is a binary
↳decision task

model.compile(loss = 'binary_crossentropy', # because this is a binary
↳decision task
              optimizer = 'adam',
              metrics = ['accuracy']) # [keras.metrics.Recall()]

```

[25]: `model.summary()`

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 5)	245765
dense_1 (Dense)	(None, 1)	6

Total params: 245,771  
 Trainable params: 245,771  
 Non-trainable params: 0

[26]: `# Fit model, set epochs to 31 because 4743 is divisible to 31.`  
`Baseline_ANN = model.fit(X_train, y_train, epochs = 100, batch_size = 31,`  
`↳verbose =0,`  
`validation_data = (X_val, y_val))`

```

Epoch 1/100
153/153 [=====] - 1s 8ms/step - loss: 0.8407 -
accuracy: 0.7116 - val_loss: 0.6671 - val_accuracy: 0.7300
Epoch 2/100
153/153 [=====] - 0s 3ms/step - loss: 0.6551 -
accuracy: 0.7297 - val_loss: 0.6437 - val_accuracy: 0.7300
Epoch 3/100
153/153 [=====] - 0s 3ms/step - loss: 0.6347 -
accuracy: 0.7297 - val_loss: 0.6262 - val_accuracy: 0.7300
Epoch 4/100
153/153 [=====] - 0s 3ms/step - loss: 0.6194 -
accuracy: 0.7297 - val_loss: 0.6130 - val_accuracy: 0.7300
Epoch 5/100
153/153 [=====] - 0s 3ms/step - loss: 0.6081 -
accuracy: 0.7297 - val_loss: 0.6036 - val_accuracy: 0.7300
Epoch 6/100

```

153/153 [=====] - 0s 3ms/step - loss: 0.6000 -  
accuracy: 0.7297 - val\_loss: 0.5968 - val\_accuracy: 0.7300  
Epoch 7/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5944 -  
accuracy: 0.7297 - val\_loss: 0.5920 - val\_accuracy: 0.7300  
Epoch 8/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5906 -  
accuracy: 0.7297 - val\_loss: 0.5888 - val\_accuracy: 0.7300  
Epoch 9/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5879 -  
accuracy: 0.7297 - val\_loss: 0.5868 - val\_accuracy: 0.7300  
Epoch 10/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5863 -  
accuracy: 0.7297 - val\_loss: 0.5853 - val\_accuracy: 0.7300  
Epoch 11/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5852 -  
accuracy: 0.7297 - val\_loss: 0.5845 - val\_accuracy: 0.7300  
Epoch 12/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5845 -  
accuracy: 0.7297 - val\_loss: 0.5840 - val\_accuracy: 0.7300  
Epoch 13/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5841 -  
accuracy: 0.7297 - val\_loss: 0.5836 - val\_accuracy: 0.7300  
Epoch 14/100  
153/153 [=====] - 1s 5ms/step - loss: 0.5839 -  
accuracy: 0.7297 - val\_loss: 0.5835 - val\_accuracy: 0.7300  
Epoch 15/100  
153/153 [=====] - 1s 5ms/step - loss: 0.5838 -  
accuracy: 0.7297 - val\_loss: 0.5834 - val\_accuracy: 0.7300  
Epoch 16/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5837 -  
accuracy: 0.7297 - val\_loss: 0.5833 - val\_accuracy: 0.7300  
Epoch 17/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5833 - val\_accuracy: 0.7300  
Epoch 18/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 19/100  
153/153 [=====] - 1s 5ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 20/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 21/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 22/100

153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 23/100  
153/153 [=====] - 1s 5ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 24/100  
153/153 [=====] - 1s 5ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 25/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 26/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 27/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 28/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 29/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 30/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 31/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 32/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 33/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 34/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 35/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 36/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 37/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 38/100

153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 39/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 40/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 41/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 42/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 43/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 44/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 45/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 46/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 47/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 48/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 49/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 50/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 51/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 52/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 53/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 54/100

153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 55/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 56/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 57/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 58/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 59/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 60/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 61/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 62/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 63/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 64/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 65/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 66/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 67/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 68/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 69/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 70/100

153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 71/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 72/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 73/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 74/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 75/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 76/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 77/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 78/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 79/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 80/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 81/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 82/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 83/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 84/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 85/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 86/100

153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 87/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 88/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 89/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 90/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 91/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 92/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 93/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 94/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 95/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 96/100  
153/153 [=====] - 0s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 97/100  
153/153 [=====] - 1s 3ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 98/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 99/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300  
Epoch 100/100  
153/153 [=====] - 1s 4ms/step - loss: 0.5836 -  
accuracy: 0.7297 - val\_loss: 0.5832 - val\_accuracy: 0.7300

### 5.0.1 Evaluate the model performance:

- A traditional Neural Network and CNN uses different data shapes as the input images. Vector for ANN and tensor for CNN.
- A traditional Neural Network and CNN also differ in how accuracy metrics are reported: “accuracy” and “val\_accuracy” for ANN versus “acc” and “val\_acc” for CNN.
- Therefore, we need to specify the result from the model, Xtrain, Xtest, accuracy and valaccuracy for the function below to be generalizable:

```
[27]: def visualize_model_performance(result, Xtrainname, Xtestname, accuracy, valaccuracy):  
  
    with plt.style.context('seaborn-talk'):  
  
        # Display train and validation loss and accuracy:  
        fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16,5))  
        ax1.plot(result.history['loss'])  
        ax1.plot(result.history['val_loss'])  
        ax1.set_title("Loss")  
        ax1.legend(labels = ['train loss', 'val loss'])  
        ax2.plot(result.history[accuracy])  
        ax2.plot(result.history[valaccuracy])  
        ax2.legend(labels = ['train acc', 'val acc'])  
        ax2.set_title('Accuracy')  
  
        # Output (probability) predictions for the test set  
        y_hat_test = result.model.predict(Xtestname)  
        y_pred = np rint(y_hat_test).astype(np.int) # Round elements of the  
        array to the nearest integer.  
        y_true = y_test.astype(np.int)  
  
        # Generate a confusion matrix displaying the predictive accuracy of the  
        model on the test set:  
        cm = confusion_matrix(y_true, y_pred) # normalize = 'true'  
        disp = ConfusionMatrixDisplay(confusion_matrix=cm)  
        disp.plot(cmap = "Blues", ax=ax3)  
        ax3.set_title('Confusion Matrix - TestSet')  
  
        # Print Classification Report displaying the performance of the model  
        on the test set using various metrics:  
        print('Classification Report:')  
        print(classification_report(y_true, y_pred))  
        print('\n')  
  
        # Print final train and test loss and accuracy:  
        train_loss, train_acc = result.model.evaluate(Xtrainname, y_train);  
        test_loss, test_acc = result.model.evaluate(Xtestname, y_test);  
        print('-----')
```



```

print(f'Final Train Loss: {np.round(train_loss,4)}')
print(f'Final Test Loss: {np.round(test_loss,4)}')
print('-----')
print(f'Final Train Acc: {np.round(train_acc,4)}')
print(f'Final Test Acc: {np.round(test_acc,4)}')
print('\n')

```

```

[28]: visualize_model_performance(Baseline_ANN, X_train, X_test, "accuracy",
    ↪ "val_accuracy")

```

Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	159
1	0.73	1.00	0.84	428
accuracy			0.73	587
macro avg	0.36	0.50	0.42	587
weighted avg	0.53	0.73	0.61	587

```

149/149 [=====] - 0s 2ms/step - loss: 0.5836 -
accuracy: 0.7297
19/19 [=====] - 0s 2ms/step - loss: 0.5841 - accuracy:
0.7291

```

-----

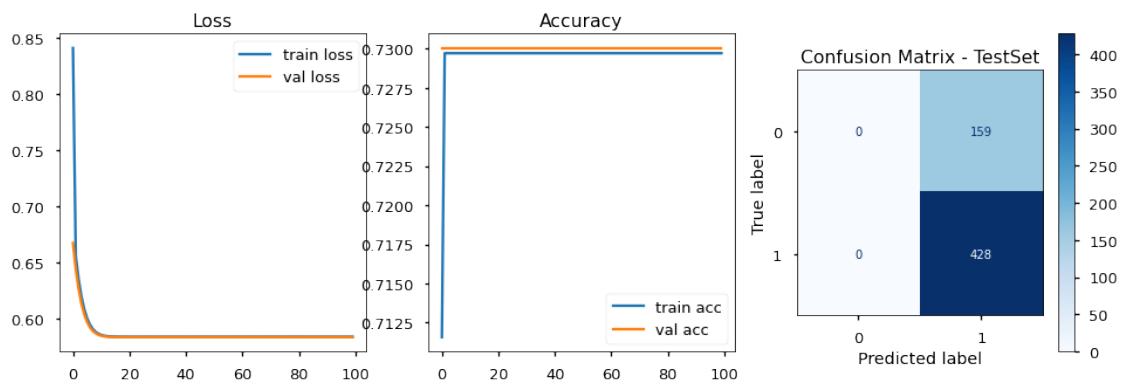
Final Train Loss: 0.5836

Final Test Loss: 0.5841

-----

Final Train Acc: 0.7297

Final Test Acc: 0.7291



- The extremely simple baseline ANN is basically predicting all x-rays as Pneumonia, which still leads to a 72% accuracy.

## 6 Bigger/Deeper Artificial Neural Network:

- Add an input layer with 128 neurons
- Add three hidden layers with 64, 32 and 10 neurons.
- Add the output layer with 1 neuron.

**Early Stopping:** Specify early stopping training for all the subsequent models so that once the model performance stops improving on a hold out validation dataset the training will stop. It helps with overfitting and it won't run for more epochs unnecessarily:

```
[29]: # Patience number of 10: the number of epochs to wait before early stop if no
      ↪ progress on the validation set.
early_stop = EarlyStopping(monitor='val_loss', patience=10,
      ↪ restore_best_weights=True)
```

```
[30]: model = models.Sequential()

      # Add dense layers with relu activation
model.add(layers.Dense(128, activation='relu', input_shape = (49152,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(10, activation='relu'))

      # Add final layer with sigmoid activation
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss = 'binary_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
```

```
[31]: model.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 128)	6291584
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 32)	2080

```

dense_5 (Dense)                (None, 10)                330
-----
dense_6 (Dense)                (None, 1)                 11
=====
Total params: 6,302,261
Trainable params: 6,302,261
Non-trainable params: 0
-----

```

```
[32]: Deeper_ANN = model.fit(X_train, y_train, epochs = 100, batch_size = 31, verbose=0,
                             validation_data = (X_val, y_val), callbacks = [early_stop])
```

```
[33]: visualize_model_performance(Deeper_ANN, X_train, X_test, "accuracy",
                                   "val_accuracy")
```

Classification Report:

	precision	recall	f1-score	support
0	0.89	0.89	0.89	159
1	0.96	0.96	0.96	428
accuracy			0.94	587
macro avg	0.93	0.92	0.92	587
weighted avg	0.94	0.94	0.94	587

```

149/149 [=====] - 1s 4ms/step - loss: 0.0927 -
accuracy: 0.9646
19/19 [=====] - 0s 4ms/step - loss: 0.1660 - accuracy:
0.9404
-----

```

```

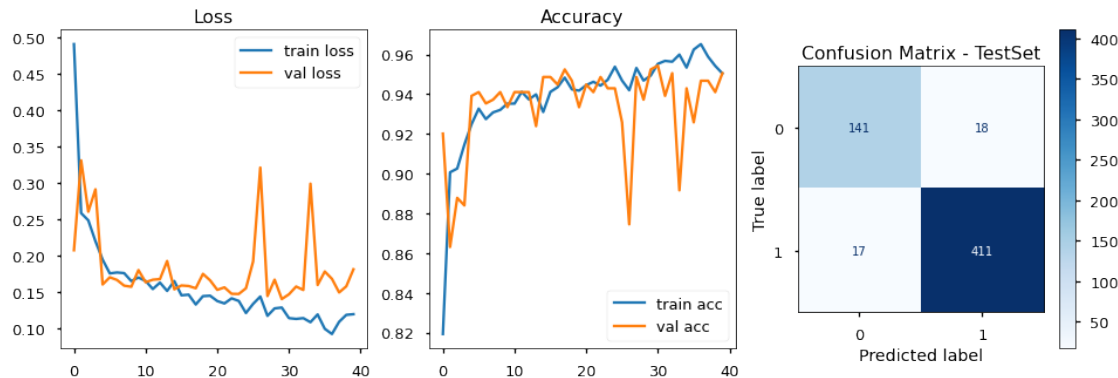
Final Train Loss: 0.0927
Final Test Loss: 0.166
-----

```

```

Final Train Acc: 0.9646
Final Test Acc: 0.9404

```



- A Deeper ANN with 5 layers improves the performance extensively achieving an overall accuracy of 94% on the test set.
- Recall for pneumonia is 96%, that is predicting true pneumonia cases as pneumonia. 17 out of 428 pneumonia cases are mislabeled as normal.

## 7 Deeper ANN with Class Weights (due to class imbalance):

- Using the same number of layers / neurons as above but adding class weights to see if it would improve performance.

```
[34]: pneumonia = os.listdir("Data/OUTPUT/train/PNEUMONIA")
normal = os.listdir("Data/OUTPUT/train/NORMAL")

weight_pneumonia = len(normal)/(len(normal) + len(pneumonia))
weight_normal = len(pneumonia)/(len(normal) + len(pneumonia))

print(len(pneumonia), len(normal))
print(f'Weight for class pneumonia: {np.round(weight_pneumonia, 2)}')
print(f'Weight for class normal: {np.round(weight_normal, 2)}')
```

```
3461 1282
Weight for class pneumonia: 0.27
Weight for class normal: 0.73
```

```
[35]: weights_dict = {0:np.round(weight_normal, 2) , 1: np.round(weight_pneumonia, 2)}
weights_dict
```

```
[35]: {0: 0.73, 1: 0.27}
```

```
[36]: Weighted_ANN = model.fit(X_train, y_train, epochs = 100, batch_size = 31,
    verbose = 0,
```

```
validation_data = (X_val, y_val), class_weight =  
weights_dict, callbacks = [early_stop])
```

```
[37]: visualize_model_performance(Weighted_ANN, X_train, X_test, "accuracy",  
val_accuracy)
```

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.88	0.90	159
1	0.96	0.97	0.96	428
accuracy			0.95	587
macro avg	0.94	0.93	0.93	587
weighted avg	0.95	0.95	0.95	587

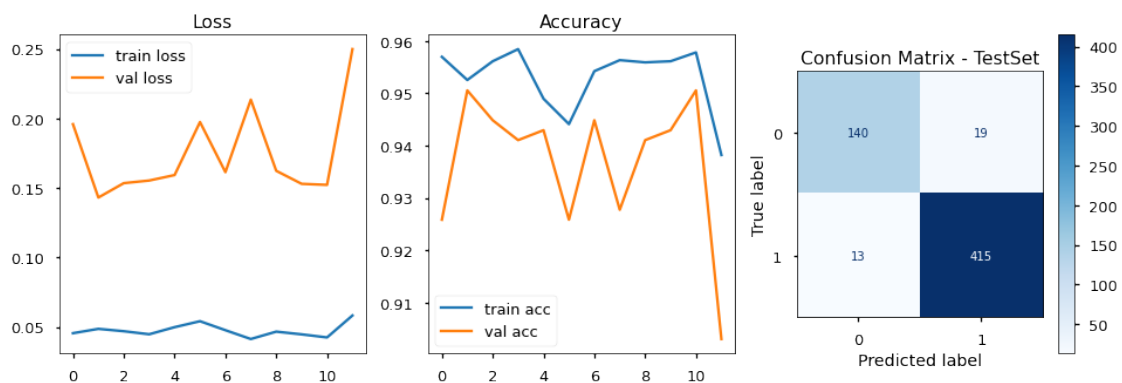
```
149/149 [=====] - 1s 4ms/step - loss: 0.0881 -  
accuracy: 0.9656  
19/19 [=====] - 0s 4ms/step - loss: 0.1633 - accuracy:  
0.9455
```

```
-----  
Final Train Loss: 0.0881
```

```
Final Test Loss: 0.1633  
-----
```

```
Final Train Acc: 0.9656
```

```
Final Test Acc: 0.9455
```



- Using class weights for the Deeper ANN with 4 layers does not seem to improve overall performance,
- But recall for pneumonia is now up to 97%. Only 13 out of 428 pneumonia cases are mislabeled

as normal.

## 8 Baseline Convolutional Neural Network:

- Baseline model with 1 convolutional layer, 1 max pooling layer, and 1 fully connected layer
- Number of output filters in the convolutional layer is 8.
- Kernel Size is 3 x 3. If your images are smaller than 128×128 you may want to consider sticking with strictly 1×1 and 3×3 filters.
- A fully connected layer with 16 neurons. Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. The FC layer helps to map the representation between the input and the output.
- An output layer with 1 neuron making the predictions.

```
[137]: model = Sequential()

# 1st Convolution and Pooling
model.add(Conv2D(8, (3, 3), activation='relu', input_shape=(128, 128, 3)))
model.add(MaxPool2D(pool_size = (2, 2)))

# Flatten
model.add(Flatten())

# Include a fully-connected layer and an output layer
model.add(Dense(activation = 'relu', units = 8)) # inner layer
model.add(Dense(activation = 'sigmoid', units = 1)) # output layer

# Compile model
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['acc'])

model.summary()
```

Model: "sequential\_14"

Layer (type)	Output Shape	Param #
conv2d_21 (Conv2D)	(None, 126, 126, 8)	224
max_pooling2d_30 (MaxPooling)	(None, 63, 63, 8)	0
flatten_12 (Flatten)	(None, 31752)	0
dense_41 (Dense)	(None, 8)	254024
dense_42 (Dense)	(None, 1)	9

Total params: 254,257

Trainable params: 254,257  
Non-trainable params: 0

-----

```
[138]: # Train the model
Baseline_CNN = model.fit(train_image, y_train, epochs = 100, batch_size=31,
    ↪ verbose = 0,
                                validation_data = (val_image, y_val), callbacks =
    ↪ [early_stop])
```

```
[139]: visualize_model_performance(Baseline_CNN, train_image, test_image, "acc",
    ↪ "val_acc")
```

Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	159
1	0.73	1.00	0.84	428
accuracy			0.73	587
macro avg	0.36	0.50	0.42	587
weighted avg	0.53	0.73	0.61	587

149/149 [=====] - 2s 14ms/step - loss: 0.5836 - acc:  
0.7297

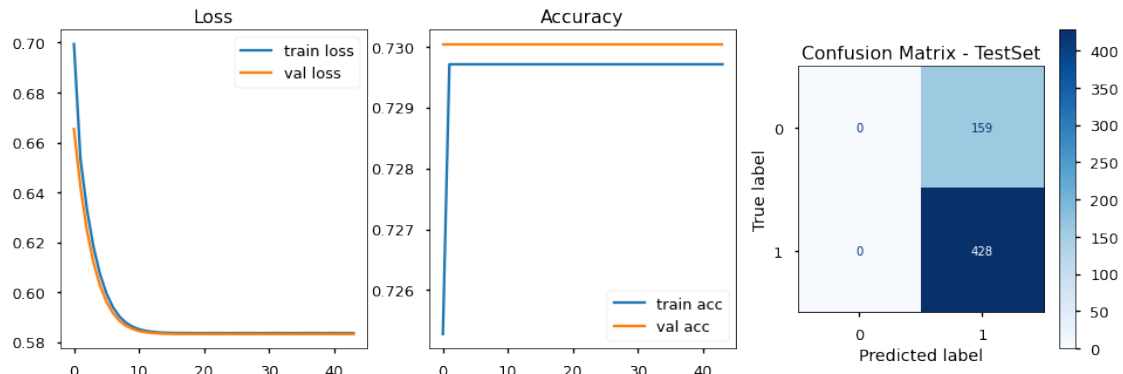
19/19 [=====] - 0s 13ms/step - loss: 0.5841 - acc:  
0.7291

-----  
Final Train Loss: 0.5836

Final Test Loss: 0.5841

-----  
Final Train Acc: 0.7297

Final Test Acc: 0.7291



- The simple baseline CNN is predicting all x-rays as Pneumonia, which still leads to a 72% accuracy.

## 9 Deeper CNN:

- I will deepen the neural network to include more layers, filters and neurons to pull more features out of the images to improve the model.
- Layers early in the network architecture (i.e., closer to the actual input image) learn fewer convolutional filters while layers deeper in the network (i.e., closer to the output predictions) will learn more filters.

```
[41]: model = Sequential()

# 1st Convolution and Pooling
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
#input_shape=(128, 128,1)
model.add(MaxPool2D(pool_size = (2, 2))) # 32 is number of filters

# 2nd Convolution and Pooling
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPool2D(pool_size = (2, 2)))

# 3rd Convolution and Pooling
model.add(Conv2D(128, (3, 3), activation="relu"))
model.add(MaxPool2D(pool_size = (2, 2)))

# Flatten
model.add(Flatten())

# activation
model.add(Dense(activation = 'relu', units = 128)) # inner layer
model.add(Dense(activation = 'relu', units = 64)) # inner layer
model.add(Dense(activation = 'sigmoid', units = 1)) # output layer
```



```
# Compile model
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['acc'])

model.summary()
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_3 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_4 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_9 (Dense)	(None, 128)	3211392
dense_10 (Dense)	(None, 64)	8256
dense_11 (Dense)	(None, 1)	65

Total params: 3,312,961  
 Trainable params: 3,312,961  
 Non-trainable params: 0

```
[42]: Deeper_CNN = model.fit(train_image, y_train, epochs = 100, batch_size=31,
                             validation_data = (val_image, y_val), callbacks = [early_stop])
```

Epoch 1/100  
 153/153 [=====] - 47s 310ms/step - loss: 0.2503 - acc: 0.9026 - val\_loss: 0.2615 - val\_acc: 0.8859  
 Epoch 2/100  
 153/153 [=====] - 49s 322ms/step - loss: 0.1446 - acc: 0.9467 - val\_loss: 0.1519 - val\_acc: 0.9506  
 Epoch 3/100

```

153/153 [=====] - 47s 306ms/step - loss: 0.1320 - acc:
0.9505 - val_loss: 0.1677 - val_acc: 0.9335
Epoch 4/100
153/153 [=====] - 47s 306ms/step - loss: 0.0990 - acc:
0.9625 - val_loss: 0.1374 - val_acc: 0.9468
Epoch 5/100
153/153 [=====] - 47s 310ms/step - loss: 0.0867 - acc:
0.9650 - val_loss: 0.1963 - val_acc: 0.9354
Epoch 6/100
153/153 [=====] - 47s 306ms/step - loss: 0.0832 - acc:
0.9652 - val_loss: 0.1190 - val_acc: 0.9544
Epoch 7/100
153/153 [=====] - 45s 297ms/step - loss: 0.0652 - acc:
0.9777 - val_loss: 0.1447 - val_acc: 0.9525
Epoch 8/100
153/153 [=====] - 45s 294ms/step - loss: 0.0504 - acc:
0.9804 - val_loss: 0.2285 - val_acc: 0.9392
Epoch 9/100
153/153 [=====] - 45s 297ms/step - loss: 0.0368 - acc:
0.9867 - val_loss: 0.1937 - val_acc: 0.9487
Epoch 10/100
153/153 [=====] - 45s 293ms/step - loss: 0.0529 - acc:
0.9787 - val_loss: 0.1532 - val_acc: 0.9392
Epoch 11/100
153/153 [=====] - 45s 295ms/step - loss: 0.0319 - acc:
0.9895 - val_loss: 0.2114 - val_acc: 0.9620
Epoch 12/100
153/153 [=====] - 47s 310ms/step - loss: 0.0302 - acc:
0.9880 - val_loss: 0.2064 - val_acc: 0.9506
Epoch 13/100
153/153 [=====] - 45s 297ms/step - loss: 0.0204 - acc:
0.9916 - val_loss: 0.2115 - val_acc: 0.9601
Epoch 14/100
153/153 [=====] - 46s 303ms/step - loss: 0.0042 - acc:
0.9992 - val_loss: 0.3029 - val_acc: 0.9639
Epoch 15/100
153/153 [=====] - 47s 309ms/step - loss: 0.0226 - acc:
0.9920 - val_loss: 0.3378 - val_acc: 0.9468
Epoch 16/100
153/153 [=====] - 48s 312ms/step - loss: 0.0145 - acc:
0.9947 - val_loss: 0.2489 - val_acc: 0.9506

```

```

[43]: visualize_model_performance(Deeper_CNN, train_image, test_image, "acc",
    ↪ "val_acc")

```

Classification Report:

precision	recall	f1-score	support
-----------	--------	----------	---------

	0	0.91	0.95	0.93	159
	1	0.98	0.96	0.97	428
accuracy				0.96	587
macro avg		0.95	0.96	0.95	587
weighted avg		0.96	0.96	0.96	587

149/149 [=====] - 11s 74ms/step - loss: 0.0516 - acc: 0.9821

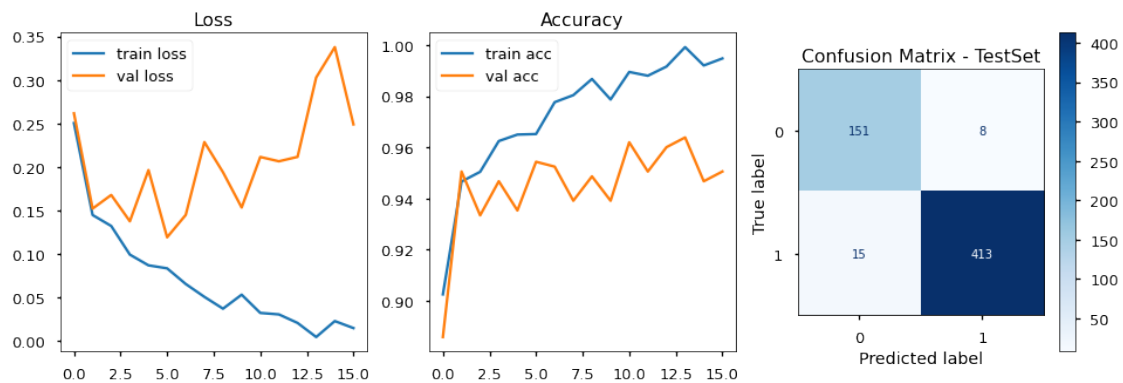
19/19 [=====] - 1s 71ms/step - loss: 0.1050 - acc: 0.9608

-----  
Final Train Loss: 0.0516

Final Test Loss: 0.105  
-----

Final Train Acc: 0.9821

Final Test Acc: 0.9608



- Using a deeper CNN overall accuracy is at 96% on the test set.
- Recall for pneumonia is high at 96%. Only 15 out of 428 pneumonia cases are mislabeled as normal.
- Recall for pneumonia is high at 95%. only 8 out of 159 was mislabeled.
- The model does not seem to overfit a much, but let's still experiment with some regularization techniques:

## 10 CNN with dropout regularization:

- Dropout regularization helps prevent neural networks from overfitting. Dropout works by randomly disabling neurons and their corresponding connections. This prevents the network from relying too much on single neurons and forces all neurons to learn to generalize better.

```
[56]: import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
[57]: model = Sequential()

# 1st Convolution and Pooling and dropout
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
↳#input_shape=(128, 128,1)
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.4)) # regularization, turn off 40% of the neurons

# 2nd Convolution and Pooling
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.4)) # regularization

# 3rd Convolution and Pooling
model.add(Conv2D(128, (3, 3), activation="relu"))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.4)) # regularization

# Flatten
model.add(Flatten())

model.add(Dense(activation = 'relu', units = 128)) # inner layer
model.add(Dropout(0.2)) # regularization
model.add(Dense(activation = 'relu', units = 64)) # inner layer
model.add(Dropout(0.2)) # regularization
model.add(Dense(activation = 'sigmoid', units = 1)) # output layer

model.save("Dropout_CNN")

# Compile model
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
↳['acc'])

model.summary()
```

```
INFO:tensorflow:Assets written to: Dropout_CNN/assets
Model: "sequential_7"
```

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_14 (MaxPooling)	(None, 63, 63, 32)	0

dropout_10 (Dropout)	(None, 63, 63, 32)	0
conv2d_15 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_15 (MaxPooling)	(None, 30, 30, 64)	0
dropout_11 (Dropout)	(None, 30, 30, 64)	0
conv2d_16 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_16 (MaxPooling)	(None, 14, 14, 128)	0
dropout_12 (Dropout)	(None, 14, 14, 128)	0
flatten_5 (Flatten)	(None, 25088)	0
dense_21 (Dense)	(None, 128)	3211392
dropout_13 (Dropout)	(None, 128)	0
dense_22 (Dense)	(None, 64)	8256
dropout_14 (Dropout)	(None, 64)	0
dense_23 (Dense)	(None, 1)	65

=====  
 Total params: 3,312,961  
 Trainable params: 3,312,961  
 Non-trainable params: 0  
 =====

```
[61]: Dropout_CNN = model.fit(train_image, y_train, epochs = 100, batch_size=31,
                             validation_data=(val_image, y_val), callbacks =[
    ↪ [early_stop] ]
```

```
Epoch 1/100
153/153 [=====] - 47s 308ms/step - loss: 0.1011 - acc:
0.9646 - val_loss: 0.1500 - val_acc: 0.9487
Epoch 2/100
153/153 [=====] - 49s 320ms/step - loss: 0.0979 - acc:
0.9620 - val_loss: 0.1282 - val_acc: 0.9449
Epoch 3/100
153/153 [=====] - 50s 329ms/step - loss: 0.0884 - acc:
0.9671 - val_loss: 0.1213 - val_acc: 0.9525
Epoch 4/100
153/153 [=====] - 49s 317ms/step - loss: 0.0915 - acc:
0.9667 - val_loss: 0.1180 - val_acc: 0.9582
Epoch 5/100
```

```

153/153 [=====] - 47s 310ms/step - loss: 0.0849 - acc:
0.9707 - val_loss: 0.1544 - val_acc: 0.9449
Epoch 6/100
153/153 [=====] - 48s 315ms/step - loss: 0.0884 - acc:
0.9652 - val_loss: 0.1251 - val_acc: 0.9487
Epoch 7/100
153/153 [=====] - 48s 311ms/step - loss: 0.0653 - acc:
0.9774 - val_loss: 0.1341 - val_acc: 0.9563
Epoch 8/100
153/153 [=====] - 50s 324ms/step - loss: 0.0768 - acc:
0.9726 - val_loss: 0.1625 - val_acc: 0.9487
Epoch 9/100
153/153 [=====] - 48s 315ms/step - loss: 0.0676 - acc:
0.9724 - val_loss: 0.1389 - val_acc: 0.9563
Epoch 10/100
153/153 [=====] - 48s 313ms/step - loss: 0.0630 - acc:
0.9739 - val_loss: 0.1315 - val_acc: 0.9525
Epoch 11/100
153/153 [=====] - 49s 317ms/step - loss: 0.0680 - acc:
0.9726 - val_loss: 0.1306 - val_acc: 0.9601
Epoch 12/100
153/153 [=====] - 54s 350ms/step - loss: 0.0621 - acc:
0.9777 - val_loss: 0.1380 - val_acc: 0.9563
Epoch 13/100
153/153 [=====] - 49s 320ms/step - loss: 0.0606 - acc:
0.9768 - val_loss: 0.1381 - val_acc: 0.9563
Epoch 14/100
153/153 [=====] - 48s 315ms/step - loss: 0.0543 - acc:
0.9791 - val_loss: 0.1490 - val_acc: 0.9620

```

```
[62]: visualize_model_performance(Dropout_CNN, train_image, test_image, "acc",
    ↪ "val_acc")
```

Classification Report:

	precision	recall	f1-score	support
0	0.93	0.94	0.93	159
1	0.98	0.97	0.97	428
accuracy			0.96	587
macro avg	0.95	0.95	0.95	587
weighted avg	0.96	0.96	0.96	587

```

149/149 [=====] - 9s 63ms/step - loss: 0.0585 - acc:
0.9812
19/19 [=====] - 1s 62ms/step - loss: 0.1007 - acc:

```

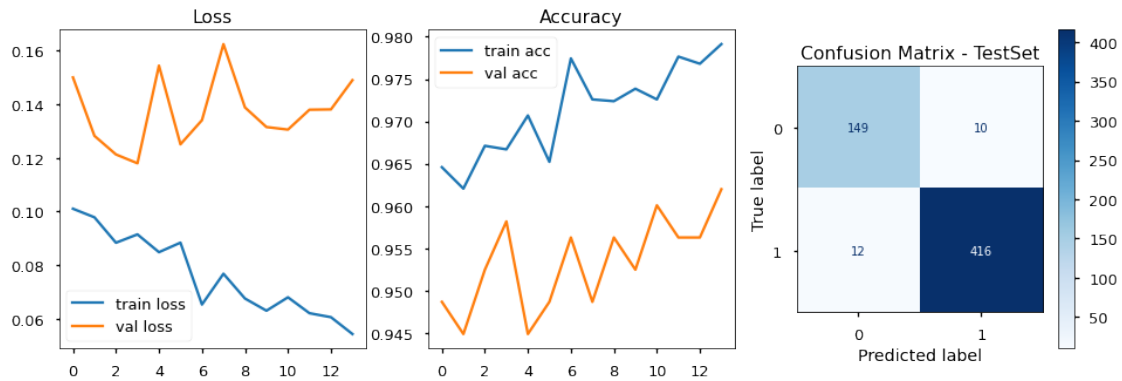
0.9625

-----  
Final Train Loss: 0.0585

Final Test Loss: 0.1007  
-----

Final Train Acc: 0.9812

Final Test Acc: 0.9625



- Using a deeper CNN with dropout regularization the overall accuracy is slightly higher: 96.3% on the test set.
- Recall for pneumonia is also slightly highed with 97%. Only 12 out of 428 pneumonia cases are mislabeled as normal.
- For the predicting the normal x-rays, only 10 out of 159 was mislabeled.

## 11 CNN with dropout and class weights:

```
[66]: Weighted_CNN = model.fit(train_image, y_train, epochs = 100, batch_size=31,  
                               validation_data=(val_image, y_val),  
                               callbacks = [early_stop], class_weight = weights_dict)
```

Epoch 1/100

153/153 [=====] - 47s 309ms/step - loss: 0.0340 - acc:  
0.9705 - val\_loss: 0.1351 - val\_acc: 0.9506

Epoch 2/100

153/153 [=====] - 47s 310ms/step - loss: 0.0313 - acc:  
0.9720 - val\_loss: 0.1334 - val\_acc: 0.9449

Epoch 3/100

153/153 [=====] - 48s 311ms/step - loss: 0.0290 - acc:  
0.9728 - val\_loss: 0.1358 - val\_acc: 0.9544

Epoch 4/100

153/153 [=====] - 51s 335ms/step - loss: 0.0318 - acc:

```

0.9707 - val_loss: 0.1237 - val_acc: 0.9639
Epoch 5/100
153/153 [=====] - 50s 326ms/step - loss: 0.0311 - acc:
0.9707 - val_loss: 0.1326 - val_acc: 0.9525
Epoch 6/100
153/153 [=====] - 50s 324ms/step - loss: 0.0259 - acc:
0.9766 - val_loss: 0.1275 - val_acc: 0.9563
Epoch 7/100
153/153 [=====] - 49s 321ms/step - loss: 0.0257 - acc:
0.9760 - val_loss: 0.1412 - val_acc: 0.9563
Epoch 8/100
153/153 [=====] - 48s 312ms/step - loss: 0.0290 - acc:
0.9722 - val_loss: 0.1386 - val_acc: 0.9582
Epoch 9/100
153/153 [=====] - 50s 324ms/step - loss: 0.0245 - acc:
0.9758 - val_loss: 0.1542 - val_acc: 0.9525
Epoch 10/100
153/153 [=====] - 48s 314ms/step - loss: 0.0275 - acc:
0.9758 - val_loss: 0.1617 - val_acc: 0.9487
Epoch 11/100
153/153 [=====] - 50s 325ms/step - loss: 0.0235 - acc:
0.9770 - val_loss: 0.1709 - val_acc: 0.9563
Epoch 12/100
153/153 [=====] - 50s 325ms/step - loss: 0.0256 - acc:
0.9766 - val_loss: 0.1670 - val_acc: 0.9506
Epoch 13/100
153/153 [=====] - 50s 325ms/step - loss: 0.0245 - acc:
0.9743 - val_loss: 0.1569 - val_acc: 0.9544
Epoch 14/100
153/153 [=====] - 48s 317ms/step - loss: 0.0277 - acc:
0.9736 - val_loss: 0.1639 - val_acc: 0.9544

```

```
[67]: visualize_model_performance(Weighted_CNN, train_image, test_image, "acc", ↵
      ↪ "val_acc")
```

Classification Report:

	precision	recall	f1-score	support
0	0.91	0.94	0.93	159
1	0.98	0.96	0.97	428
accuracy			0.96	587
macro avg	0.94	0.95	0.95	587
weighted avg	0.96	0.96	0.96	587

```
149/149 [=====] - 10s 70ms/step - loss: 0.0496 - acc:
```



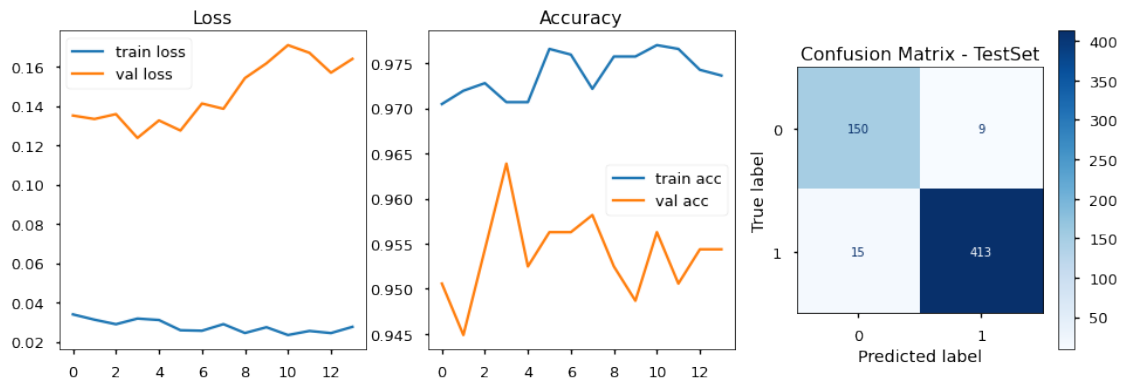
```
0.9827
19/19 [=====] - 1s 71ms/step - loss: 0.1088 - acc:
0.9591
```

```
-----
Final Train Loss: 0.0496
```

```
Final Test Loss: 0.1088
```

```
-----
Final Train Acc: 0.9827
```

```
Final Test Acc: 0.9591
```



- Adding class weights does not change the results much, it even made the recall value for pneumonue a bit worse by increasing the weight for normal cases. Since we care about detecting true positives - pneumonia we will not use class weights.

## 12 CNN with L2 regularization:

- L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero).
- L2 regularization combats overfitting by forcing weights to be small.

```
[51]: model = Sequential()

# 1st Convolution and Pooling and dropout
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3),
    ↪kernel_regularizer=regularizers.l2(0.005)))
model.add(MaxPool2D(pool_size = (2, 2)))

# 2nd Convolution and Pooling
model.add(Conv2D(64, (3, 3), activation="relu", kernel_regularizer=regularizers.
    ↪l2(0.005))) # default is 0.01
model.add(MaxPool2D(pool_size = (2, 2)))
```

```

# 3rd Convolution and Pooling
model.add(Conv2D(128, (3, 3), activation="relu",
    ↪kernel_regularizer=regularizers.l2(0.005)))
model.add(MaxPool2D(pool_size = (2, 2)))

# Flatten
model.add(Flatten())

# Add dense layers
model.add(Dense(activation = 'relu', units = 128,
    ↪kernel_regularizer=regularizers.l2(0.005))) # inner layer
model.add(Dense(activation = 'relu', units = 64,
    ↪kernel_regularizer=regularizers.l2(0.005))) # inner layer
model.add(Dense(activation = 'sigmoid', units = 1)) # output layer

# Compile model
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
    ↪['acc'])

model.summary()

```

Model: "sequential\_5"

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_9 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_9 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_10 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_10 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten_3 (Flatten)	(None, 25088)	0
dense_15 (Dense)	(None, 128)	3211392
dense_16 (Dense)	(None, 64)	8256
dense_17 (Dense)	(None, 1)	65

Total params: 3,312,961

Trainable params: 3,312,961  
Non-trainable params: 0

-----

```
[52]: L2_CNN = model.fit(train_image, y_train, epochs = 100, batch_size=31,  
                        validation_data=(val_image, y_val), callbacks =[  
                            ↪[early_stop])
```

Epoch 1/100

153/153 [=====] - 44s 289ms/step - loss: 0.8820 - acc:  
0.8433 - val\_loss: 0.4094 - val\_acc: 0.9259

Epoch 2/100

153/153 [=====] - 44s 289ms/step - loss: 0.3960 - acc:  
0.9220 - val\_loss: 0.3745 - val\_acc: 0.9202

Epoch 3/100

153/153 [=====] - 44s 288ms/step - loss: 0.3257 - acc:  
0.9336 - val\_loss: 0.3021 - val\_acc: 0.9316

Epoch 4/100

153/153 [=====] - 44s 289ms/step - loss: 0.2809 - acc:  
0.9471 - val\_loss: 0.2779 - val\_acc: 0.9354

Epoch 5/100

153/153 [=====] - 44s 288ms/step - loss: 0.2723 - acc:  
0.9416 - val\_loss: 0.2409 - val\_acc: 0.9506

Epoch 6/100

153/153 [=====] - 44s 285ms/step - loss: 0.2642 - acc:  
0.9422 - val\_loss: 0.2445 - val\_acc: 0.9487

Epoch 7/100

153/153 [=====] - 44s 287ms/step - loss: 0.2408 - acc:  
0.9479 - val\_loss: 0.2464 - val\_acc: 0.9468

Epoch 8/100

153/153 [=====] - 44s 288ms/step - loss: 0.2373 - acc:  
0.9492 - val\_loss: 0.2286 - val\_acc: 0.9506

Epoch 9/100

153/153 [=====] - 44s 285ms/step - loss: 0.2215 - acc:  
0.9517 - val\_loss: 0.2312 - val\_acc: 0.9392

Epoch 10/100

153/153 [=====] - 44s 287ms/step - loss: 0.2108 - acc:  
0.9528 - val\_loss: 0.2058 - val\_acc: 0.9525

Epoch 11/100

153/153 [=====] - 44s 286ms/step - loss: 0.2199 - acc:  
0.9486 - val\_loss: 0.2167 - val\_acc: 0.9468

Epoch 12/100

153/153 [=====] - 44s 288ms/step - loss: 0.2127 - acc:  
0.9549 - val\_loss: 0.1986 - val\_acc: 0.9525

Epoch 13/100

153/153 [=====] - 45s 293ms/step - loss: 0.2044 - acc:  
0.9532 - val\_loss: 0.1941 - val\_acc: 0.9544

Epoch 14/100

153/153 [=====] - 44s 285ms/step - loss: 0.1889 - acc:  
0.9591 - val\_loss: 0.1868 - val\_acc: 0.9582  
Epoch 15/100  
153/153 [=====] - 44s 286ms/step - loss: 0.1846 - acc:  
0.9608 - val\_loss: 0.2108 - val\_acc: 0.9449  
Epoch 16/100  
153/153 [=====] - 44s 286ms/step - loss: 0.1805 - acc:  
0.9591 - val\_loss: 0.2039 - val\_acc: 0.9506  
Epoch 17/100  
153/153 [=====] - 44s 285ms/step - loss: 0.1875 - acc:  
0.9549 - val\_loss: 0.1962 - val\_acc: 0.9468  
Epoch 18/100  
153/153 [=====] - 44s 286ms/step - loss: 0.1763 - acc:  
0.9606 - val\_loss: 0.1788 - val\_acc: 0.9601  
Epoch 19/100  
153/153 [=====] - 44s 287ms/step - loss: 0.1756 - acc:  
0.9599 - val\_loss: 0.1826 - val\_acc: 0.9601  
Epoch 20/100  
153/153 [=====] - 44s 286ms/step - loss: 0.1705 - acc:  
0.9629 - val\_loss: 0.1854 - val\_acc: 0.9563  
Epoch 21/100  
153/153 [=====] - 44s 287ms/step - loss: 0.1676 - acc:  
0.9614 - val\_loss: 0.1947 - val\_acc: 0.9411  
Epoch 22/100  
153/153 [=====] - 43s 283ms/step - loss: 0.1732 - acc:  
0.9604 - val\_loss: 0.2181 - val\_acc: 0.9354  
Epoch 23/100  
153/153 [=====] - 43s 284ms/step - loss: 0.1604 - acc:  
0.9646 - val\_loss: 0.1890 - val\_acc: 0.9525  
Epoch 24/100  
153/153 [=====] - 44s 284ms/step - loss: 0.1647 - acc:  
0.9610 - val\_loss: 0.1747 - val\_acc: 0.9601  
Epoch 25/100  
153/153 [=====] - 44s 284ms/step - loss: 0.1648 - acc:  
0.9631 - val\_loss: 0.1820 - val\_acc: 0.9506  
Epoch 26/100  
153/153 [=====] - 43s 284ms/step - loss: 0.1556 - acc:  
0.9673 - val\_loss: 0.1907 - val\_acc: 0.9525  
Epoch 27/100  
153/153 [=====] - 43s 284ms/step - loss: 0.1656 - acc:  
0.9629 - val\_loss: 0.2437 - val\_acc: 0.9297  
Epoch 28/100  
153/153 [=====] - 44s 285ms/step - loss: 0.1602 - acc:  
0.9614 - val\_loss: 0.1787 - val\_acc: 0.9506  
Epoch 29/100  
153/153 [=====] - 43s 283ms/step - loss: 0.1584 - acc:  
0.9633 - val\_loss: 0.1770 - val\_acc: 0.9525  
Epoch 30/100

153/153 [=====] - 43s 283ms/step - loss: 0.1575 - acc:  
 0.9646 - val\_loss: 0.1711 - val\_acc: 0.9544  
 Epoch 31/100  
 153/153 [=====] - 43s 283ms/step - loss: 0.1507 - acc:  
 0.9667 - val\_loss: 0.2032 - val\_acc: 0.9411  
 Epoch 32/100  
 153/153 [=====] - 43s 284ms/step - loss: 0.1410 - acc:  
 0.9690 - val\_loss: 0.1886 - val\_acc: 0.9506  
 Epoch 33/100  
 153/153 [=====] - 43s 281ms/step - loss: 0.1510 - acc:  
 0.9669 - val\_loss: 0.1738 - val\_acc: 0.9525  
 Epoch 34/100  
 153/153 [=====] - 43s 282ms/step - loss: 0.1485 - acc:  
 0.9673 - val\_loss: 0.1833 - val\_acc: 0.9506  
 Epoch 35/100  
 153/153 [=====] - 43s 284ms/step - loss: 0.1466 - acc:  
 0.9684 - val\_loss: 0.1659 - val\_acc: 0.9563  
 Epoch 36/100  
 153/153 [=====] - 43s 279ms/step - loss: 0.1406 - acc:  
 0.9690 - val\_loss: 0.1724 - val\_acc: 0.9506  
 Epoch 37/100  
 153/153 [=====] - 43s 281ms/step - loss: 0.1357 - acc:  
 0.9743 - val\_loss: 0.1664 - val\_acc: 0.9601  
 Epoch 38/100  
 153/153 [=====] - 43s 280ms/step - loss: 0.1402 - acc:  
 0.9692 - val\_loss: 0.2229 - val\_acc: 0.9392  
 Epoch 39/100  
 153/153 [=====] - 43s 281ms/step - loss: 0.1376 - acc:  
 0.9730 - val\_loss: 0.1647 - val\_acc: 0.9563  
 Epoch 40/100  
 153/153 [=====] - 43s 283ms/step - loss: 0.1376 - acc:  
 0.9713 - val\_loss: 0.1850 - val\_acc: 0.9487  
 Epoch 41/100  
 153/153 [=====] - 43s 281ms/step - loss: 0.1310 - acc:  
 0.9751 - val\_loss: 0.1648 - val\_acc: 0.9563  
 Epoch 42/100  
 153/153 [=====] - 43s 280ms/step - loss: 0.1348 - acc:  
 0.9707 - val\_loss: 0.1534 - val\_acc: 0.9639  
 Epoch 43/100  
 153/153 [=====] - 43s 280ms/step - loss: 0.1399 - acc:  
 0.9703 - val\_loss: 0.1661 - val\_acc: 0.9563  
 Epoch 44/100  
 153/153 [=====] - 43s 279ms/step - loss: 0.1418 - acc:  
 0.9682 - val\_loss: 0.1654 - val\_acc: 0.9582  
 Epoch 45/100  
 153/153 [=====] - 43s 280ms/step - loss: 0.1355 - acc:  
 0.9749 - val\_loss: 0.1600 - val\_acc: 0.9582  
 Epoch 46/100

```

153/153 [=====] - 43s 281ms/step - loss: 0.1263 - acc:
0.9751 - val_loss: 0.1744 - val_acc: 0.9544
Epoch 47/100
153/153 [=====] - 43s 281ms/step - loss: 0.1285 - acc:
0.9739 - val_loss: 0.1558 - val_acc: 0.9601
Epoch 48/100
153/153 [=====] - 43s 279ms/step - loss: 0.1255 - acc:
0.9745 - val_loss: 0.1881 - val_acc: 0.9487
Epoch 49/100
153/153 [=====] - 43s 279ms/step - loss: 0.1325 - acc:
0.9739 - val_loss: 0.1712 - val_acc: 0.9544
Epoch 50/100
153/153 [=====] - 43s 278ms/step - loss: 0.1361 - acc:
0.9713 - val_loss: 0.1684 - val_acc: 0.9582
Epoch 51/100
153/153 [=====] - 43s 279ms/step - loss: 0.1239 - acc:
0.9783 - val_loss: 0.1659 - val_acc: 0.9563
Epoch 52/100
153/153 [=====] - 43s 279ms/step - loss: 0.1224 - acc:
0.9770 - val_loss: 0.2201 - val_acc: 0.9430

```

```
[53]: visualize_model_performance(L2_CNN, train_image, test_image, "acc", "val_acc")
```

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.92	0.91	159
1	0.97	0.96	0.97	428
accuracy			0.95	587
macro avg	0.94	0.94	0.94	587
weighted avg	0.95	0.95	0.95	587

```

149/149 [=====] - 10s 65ms/step - loss: 0.1196 - acc:
0.9791
19/19 [=====] - 1s 62ms/step - loss: 0.1803 - acc:
0.9506

```

-----

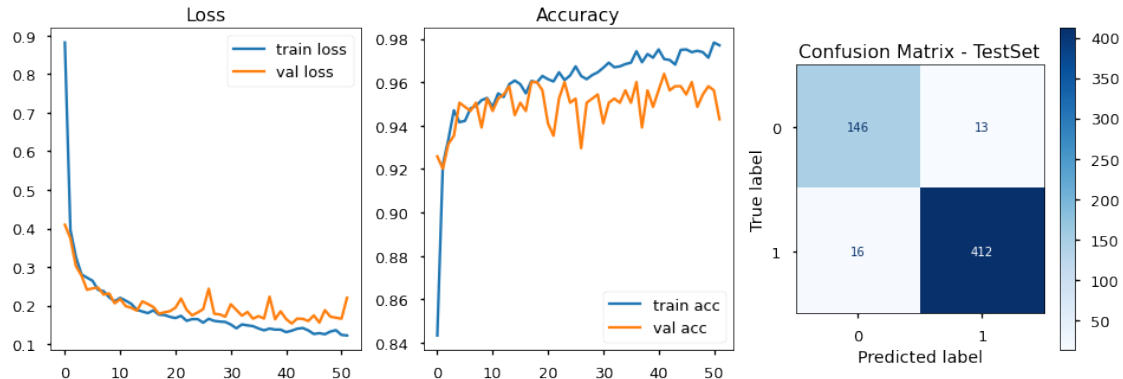
Final Train Loss: 0.1196

Final Test Loss: 0.1803

-----

Final Train Acc: 0.9791

Final Test Acc: 0.9506



- Using a deeper CNN with L2 regularization (instead of dropout) achieves similar but slightly worse results:
- The overall accuracy is: 95% on the test set.
- Recall for pneumonia is 96%. 16 out of 428 pneumonia cases are mislabeled as normal.
- 13 out of 159 normal cases were mislabeled.

### 13 CNN with dropout and L2 regularization:

```
[68]: model = Sequential()

# 1st Convolution and Pooling and dropout
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)))
    ↪ #input_shape=(128, 128, 1)
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.4)) # regularization

# 2nd Convolution and Pooling
model.add(Conv2D(64, (3, 3), activation="relu", kernel_regularizer=regularizers.
    ↪ l2(0.005))) # default is 0.01
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.4)) # regularization

# 3rd Convolution and Pooling
model.add(Conv2D(128, (3, 3), activation="relu",
    ↪ kernel_regularizer=regularizers.l2(0.005)))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.4)) # regularization

# Flatten
model.add(Flatten())
```

```

model.add(Dense(activation = 'relu', units = 128,
    ↪kernel_regularizer=regularizers.l2(0.005))) # inner layer
model.add(Dropout(0.2)) # regularization

model.add(Dense(activation = 'relu', units = 64,
    ↪kernel_regularizer=regularizers.l2(0.005))) # inner layer
model.add(Dropout(0.2)) # regularization

model.add(Dense(activation = 'sigmoid', units = 1)) # output layer

model.save("Dropout_L2_CNN")

# Compile model
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
    ↪['acc'])

model.summary()

```

INFO:tensorflow:Assets written to: Dropout\_L2\_CNN/assets  
Model: "sequential\_8"

Layer (type)	Output Shape	Param #
conv2d_17 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_17 (MaxPooling)	(None, 63, 63, 32)	0
dropout_15 (Dropout)	(None, 63, 63, 32)	0
conv2d_18 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_18 (MaxPooling)	(None, 30, 30, 64)	0
dropout_16 (Dropout)	(None, 30, 30, 64)	0
conv2d_19 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_19 (MaxPooling)	(None, 14, 14, 128)	0
dropout_17 (Dropout)	(None, 14, 14, 128)	0
flatten_6 (Flatten)	(None, 25088)	0
dense_24 (Dense)	(None, 128)	3211392
dropout_18 (Dropout)	(None, 128)	0



dense_25 (Dense)	(None, 64)	8256
-----		
dropout_19 (Dropout)	(None, 64)	0
-----		
dense_26 (Dense)	(None, 1)	65
=====		
Total params: 3,312,961		
Trainable params: 3,312,961		
Non-trainable params: 0		
-----		

```
[69]: Dropout_L2_CNN = model.fit(train_image, y_train, epochs = 100, batch_size=31,
                                validation_data=(val_image, y_val), callbacks =[
    ↪ [early_stop])
```

```
Epoch 1/100
153/153 [=====] - 48s 311ms/step - loss: 1.3092 - acc:
0.7563 - val_loss: 0.6555 - val_acc: 0.8441
Epoch 2/100
153/153 [=====] - 51s 336ms/step - loss: 0.4962 - acc:
0.8805 - val_loss: 0.4205 - val_acc: 0.9163
Epoch 3/100
153/153 [=====] - 52s 341ms/step - loss: 0.3728 - acc:
0.9125 - val_loss: 0.3564 - val_acc: 0.9259
Epoch 4/100
153/153 [=====] - 52s 341ms/step - loss: 0.3416 - acc:
0.9203 - val_loss: 0.3686 - val_acc: 0.8954
Epoch 5/100
153/153 [=====] - 49s 322ms/step - loss: 0.3091 - acc:
0.9226 - val_loss: 0.3544 - val_acc: 0.9049
Epoch 6/100
153/153 [=====] - 49s 320ms/step - loss: 0.3170 - acc:
0.9222 - val_loss: 0.3184 - val_acc: 0.9240
Epoch 7/100
153/153 [=====] - 49s 321ms/step - loss: 0.2886 - acc:
0.9315 - val_loss: 0.2854 - val_acc: 0.9354
Epoch 8/100
153/153 [=====] - 49s 320ms/step - loss: 0.2853 - acc:
0.9275 - val_loss: 0.2733 - val_acc: 0.9411
Epoch 9/100
153/153 [=====] - 49s 323ms/step - loss: 0.2565 - acc:
0.9393 - val_loss: 0.2499 - val_acc: 0.9468
Epoch 10/100
153/153 [=====] - 49s 319ms/step - loss: 0.2703 - acc:
0.9338 - val_loss: 0.2507 - val_acc: 0.9373
Epoch 11/100
153/153 [=====] - 49s 323ms/step - loss: 0.2534 - acc:
0.9399 - val_loss: 0.2740 - val_acc: 0.9278
```

Epoch 12/100  
153/153 [=====] - 49s 320ms/step - loss: 0.2500 - acc: 0.9418 - val\_loss: 0.2349 - val\_acc: 0.9430  
Epoch 13/100  
153/153 [=====] - 49s 318ms/step - loss: 0.2437 - acc: 0.9420 - val\_loss: 0.2188 - val\_acc: 0.9487  
Epoch 14/100  
153/153 [=====] - 49s 322ms/step - loss: 0.2435 - acc: 0.9408 - val\_loss: 0.2322 - val\_acc: 0.9430  
Epoch 15/100  
153/153 [=====] - 49s 321ms/step - loss: 0.2321 - acc: 0.9448 - val\_loss: 0.2231 - val\_acc: 0.9525  
Epoch 16/100  
153/153 [=====] - 52s 337ms/step - loss: 0.2391 - acc: 0.9458 - val\_loss: 0.2164 - val\_acc: 0.9487  
Epoch 17/100  
153/153 [=====] - 54s 352ms/step - loss: 0.2202 - acc: 0.9496 - val\_loss: 0.2497 - val\_acc: 0.9449  
Epoch 18/100  
153/153 [=====] - 49s 324ms/step - loss: 0.2510 - acc: 0.9420 - val\_loss: 0.2442 - val\_acc: 0.9354  
Epoch 19/100  
153/153 [=====] - 49s 322ms/step - loss: 0.2369 - acc: 0.9405 - val\_loss: 0.2755 - val\_acc: 0.9316  
Epoch 20/100  
153/153 [=====] - 48s 314ms/step - loss: 0.2375 - acc: 0.9439 - val\_loss: 0.2384 - val\_acc: 0.9373  
Epoch 21/100  
153/153 [=====] - 48s 317ms/step - loss: 0.2221 - acc: 0.9502 - val\_loss: 0.2548 - val\_acc: 0.9278  
Epoch 22/100  
153/153 [=====] - 51s 332ms/step - loss: 0.2353 - acc: 0.9386 - val\_loss: 0.2186 - val\_acc: 0.9468  
Epoch 23/100  
153/153 [=====] - 53s 345ms/step - loss: 0.2247 - acc: 0.9490 - val\_loss: 0.2284 - val\_acc: 0.9411  
Epoch 24/100  
153/153 [=====] - 51s 336ms/step - loss: 0.2134 - acc: 0.9528 - val\_loss: 0.2637 - val\_acc: 0.9278  
Epoch 25/100  
153/153 [=====] - 52s 341ms/step - loss: 0.2249 - acc: 0.9483 - val\_loss: 0.2147 - val\_acc: 0.9449  
Epoch 26/100  
153/153 [=====] - 50s 324ms/step - loss: 0.2245 - acc: 0.9467 - val\_loss: 0.2064 - val\_acc: 0.9506  
Epoch 27/100  
153/153 [=====] - 49s 321ms/step - loss: 0.2350 - acc: 0.9454 - val\_loss: 0.2577 - val\_acc: 0.9335

Epoch 28/100  
153/153 [=====] - 52s 342ms/step - loss: 0.2270 - acc:  
0.9502 - val\_loss: 0.2323 - val\_acc: 0.9449  
Epoch 29/100  
153/153 [=====] - 51s 333ms/step - loss: 0.2157 - acc:  
0.9511 - val\_loss: 0.2037 - val\_acc: 0.9487  
Epoch 30/100  
153/153 [=====] - 50s 329ms/step - loss: 0.2265 - acc:  
0.9437 - val\_loss: 0.2326 - val\_acc: 0.9373  
Epoch 31/100  
153/153 [=====] - 50s 330ms/step - loss: 0.2161 - acc:  
0.9481 - val\_loss: 0.2324 - val\_acc: 0.9449  
Epoch 32/100  
153/153 [=====] - 50s 324ms/step - loss: 0.2135 - acc:  
0.9507 - val\_loss: 0.2066 - val\_acc: 0.9506  
Epoch 33/100  
153/153 [=====] - 48s 315ms/step - loss: 0.2191 - acc:  
0.9517 - val\_loss: 0.2018 - val\_acc: 0.9468  
Epoch 34/100  
153/153 [=====] - 50s 328ms/step - loss: 0.2006 - acc:  
0.9524 - val\_loss: 0.2178 - val\_acc: 0.9525  
Epoch 35/100  
153/153 [=====] - 50s 324ms/step - loss: 0.2298 - acc:  
0.9445 - val\_loss: 0.2303 - val\_acc: 0.9449  
Epoch 36/100  
153/153 [=====] - 54s 355ms/step - loss: 0.2195 - acc:  
0.9483 - val\_loss: 0.2206 - val\_acc: 0.9525  
Epoch 37/100  
153/153 [=====] - 52s 338ms/step - loss: 0.2139 - acc:  
0.9502 - val\_loss: 0.2221 - val\_acc: 0.9582  
Epoch 38/100  
153/153 [=====] - 51s 337ms/step - loss: 0.2045 - acc:  
0.9519 - val\_loss: 0.2260 - val\_acc: 0.9392  
Epoch 39/100  
153/153 [=====] - 52s 337ms/step - loss: 0.2090 - acc:  
0.9502 - val\_loss: 0.2115 - val\_acc: 0.9468  
Epoch 40/100  
153/153 [=====] - 53s 348ms/step - loss: 0.2045 - acc:  
0.9519 - val\_loss: 0.2028 - val\_acc: 0.9468  
Epoch 41/100  
153/153 [=====] - 49s 319ms/step - loss: 0.2071 - acc:  
0.9532 - val\_loss: 0.2047 - val\_acc: 0.9411  
Epoch 42/100  
153/153 [=====] - 50s 328ms/step - loss: 0.2027 - acc:  
0.9549 - val\_loss: 0.2193 - val\_acc: 0.9525  
Epoch 43/100  
153/153 [=====] - 50s 329ms/step - loss: 0.2111 - acc:  
0.9505 - val\_loss: 0.2486 - val\_acc: 0.9373

```
[70]: visualize_model_performance(Dropout_L2_CNN, train_image, test_image, "acc",
    ↪ "val_acc")
```

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.91	0.91	159
1	0.97	0.97	0.97	428
accuracy			0.95	587
macro avg	0.94	0.94	0.94	587
weighted avg	0.95	0.95	0.95	587

149/149 [=====] - 11s 71ms/step - loss: 0.1749 - acc: 0.9658

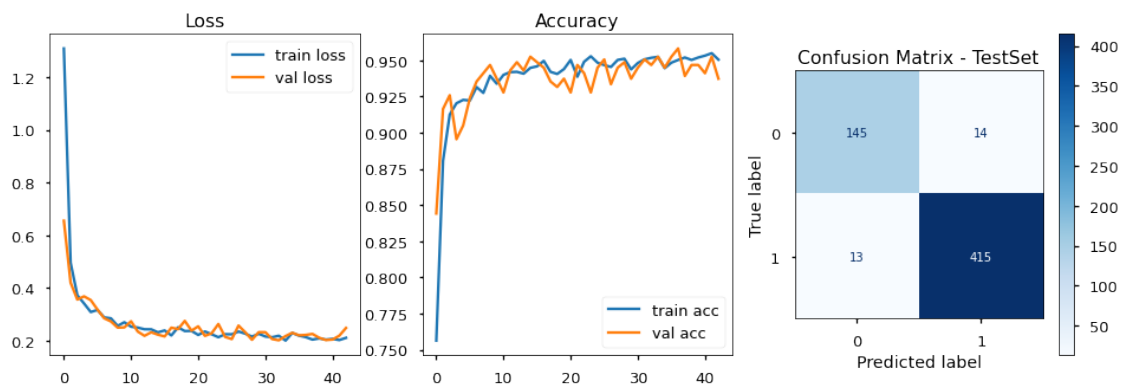
19/19 [=====] - 1s 66ms/step - loss: 0.1975 - acc: 0.9540

-----  
Final Train Loss: 0.1749

Final Test Loss: 0.1975  
-----

Final Train Acc: 0.9658

Final Test Acc: 0.954



- Using a deeper CNN with dropout as well as L2 does not change results much: accuracy is a still high at 95% on the test set.
- Recall for pneumonia is also still high with 97%. 13 out of 428 pneumonia cases are mislabeled as normal.

## 14 CNN with dropout and lower learning rate:

- One of the key hyperparameters to set in order to train a neural network is the learning rate for gradient descent. This parameter scales the magnitude of our weight updates in order to minimize the network's loss function.
- If your learning rate is set too low, training will progress very slowly as you are making very tiny updates to the weights in your network. However, if your learning rate is set too high, it can cause undesirable divergent behavior in your loss function.

```
[71]: model = keras.models.load_model("Dropout_CNN")
```

WARNING:tensorflow:No training configuration found in save file, so the model was \*not\* compiled. Compile it manually.

```
[72]: optm = optimizers.Adam(learning_rate=0.0005) # default is 0.001

model.compile(optimizer = optm, loss = 'binary_crossentropy', metrics = ['acc'])

LowLearnRate_CNN = model.fit(train_image, y_train, epochs = 100, batch_size=31,
                             validation_data=(val_image, y_val),
                             callbacks = [early_stop])
```

Epoch 1/100

153/153 [=====] - 50s 326ms/step - loss: 0.5059 - acc: 0.7774 - val\_loss: 0.3982 - val\_acc: 0.8840

Epoch 2/100

153/153 [=====] - 50s 324ms/step - loss: 0.2356 - acc: 0.9119 - val\_loss: 0.2438 - val\_acc: 0.9106

Epoch 3/100

153/153 [=====] - 51s 331ms/step - loss: 0.1836 - acc: 0.9289 - val\_loss: 0.2678 - val\_acc: 0.9125

Epoch 4/100

153/153 [=====] - 49s 323ms/step - loss: 0.1658 - acc: 0.9397 - val\_loss: 0.1711 - val\_acc: 0.9354

Epoch 5/100

153/153 [=====] - 50s 330ms/step - loss: 0.1554 - acc: 0.9427 - val\_loss: 0.1691 - val\_acc: 0.9240

Epoch 6/100

153/153 [=====] - 51s 334ms/step - loss: 0.1475 - acc: 0.9473 - val\_loss: 0.1489 - val\_acc: 0.9316

Epoch 7/100

153/153 [=====] - 48s 316ms/step - loss: 0.1434 - acc: 0.9488 - val\_loss: 0.1380 - val\_acc: 0.9506

Epoch 8/100

153/153 [=====] - 49s 322ms/step - loss: 0.1324 - acc: 0.9524 - val\_loss: 0.1518 - val\_acc: 0.9335

Epoch 9/100

153/153 [=====] - 53s 344ms/step - loss: 0.1242 - acc:

```

0.9534 - val_loss: 0.1542 - val_acc: 0.9468
Epoch 10/100
153/153 [=====] - 48s 314ms/step - loss: 0.1084 - acc:
0.9597 - val_loss: 0.1113 - val_acc: 0.9525
Epoch 11/100
153/153 [=====] - 48s 315ms/step - loss: 0.1023 - acc:
0.9631 - val_loss: 0.1112 - val_acc: 0.9563
Epoch 12/100
153/153 [=====] - 51s 336ms/step - loss: 0.0950 - acc:
0.9650 - val_loss: 0.1042 - val_acc: 0.9620
Epoch 13/100
153/153 [=====] - 49s 321ms/step - loss: 0.1039 - acc:
0.9629 - val_loss: 0.1261 - val_acc: 0.9487
Epoch 14/100
153/153 [=====] - 48s 312ms/step - loss: 0.0887 - acc:
0.9677 - val_loss: 0.1057 - val_acc: 0.9601
Epoch 15/100
153/153 [=====] - 49s 318ms/step - loss: 0.0773 - acc:
0.9713 - val_loss: 0.1093 - val_acc: 0.9563
Epoch 16/100
153/153 [=====] - 48s 312ms/step - loss: 0.0828 - acc:
0.9688 - val_loss: 0.1161 - val_acc: 0.9601
Epoch 17/100
153/153 [=====] - 48s 311ms/step - loss: 0.0717 - acc:
0.9734 - val_loss: 0.1234 - val_acc: 0.9506
Epoch 18/100
153/153 [=====] - 48s 311ms/step - loss: 0.0682 - acc:
0.9739 - val_loss: 0.1337 - val_acc: 0.9563
Epoch 19/100
153/153 [=====] - 50s 325ms/step - loss: 0.0662 - acc:
0.9758 - val_loss: 0.1050 - val_acc: 0.9639
Epoch 20/100
153/153 [=====] - 47s 310ms/step - loss: 0.0626 - acc:
0.9747 - val_loss: 0.1258 - val_acc: 0.9563
Epoch 21/100
153/153 [=====] - 47s 310ms/step - loss: 0.0554 - acc:
0.9806 - val_loss: 0.1107 - val_acc: 0.9677
Epoch 22/100
153/153 [=====] - 49s 317ms/step - loss: 0.0554 - acc:
0.9781 - val_loss: 0.1313 - val_acc: 0.9563

```

```
[73]: visualize_model_performance(LowLearnRate_CNN, train_image, test_image, "acc",
    ↪ "val_acc")
```

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.94	0.94	159

1	0.98	0.98	0.98	428
accuracy			0.97	587
macro avg	0.96	0.96	0.96	587
weighted avg	0.97	0.97	0.97	587

149/149 [=====] - 10s 69ms/step - loss: 0.0757 - acc: 0.9743

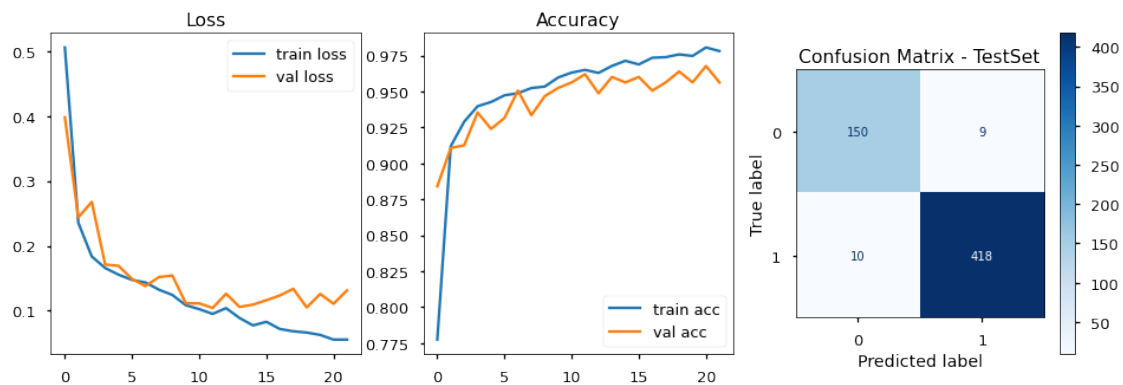
19/19 [=====] - 1s 65ms/step - loss: 0.1058 - acc: 0.9676

-----  
Final Train Loss: 0.0757

Final Test Loss: 0.1058  
-----

Final Train Acc: 0.9743

Final Test Acc: 0.9676



- Adding a lower learning rate to dropout regularization slightly improved the model:
- Accuracy is still high at 96.8% on the test set.
- Recall for pneumonia is higher with 98%.
- Recall for normal is higher with 94%.

```
[140]: # Display separately for the powerpoint:
with plt.style.context('seaborn-talk'):

    fig, ax1 = plt.subplots(figsize=(5,5))

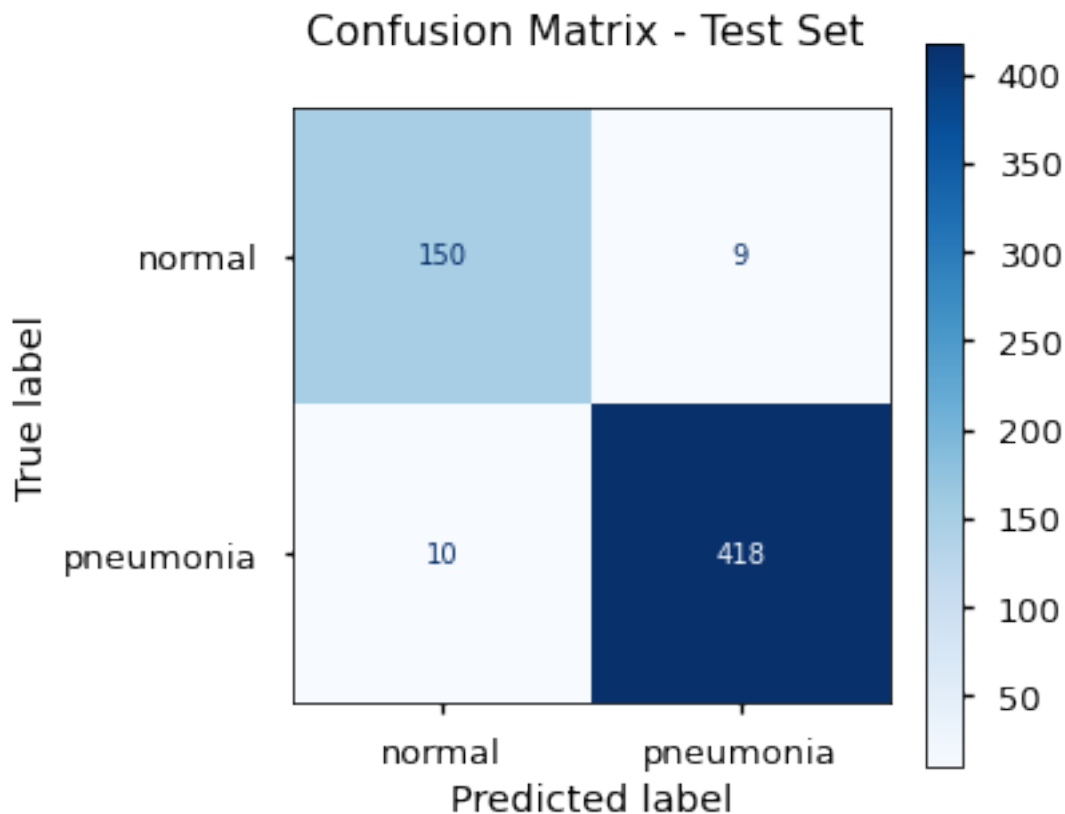
    # Output (probability) predictions for the test set
    y_hat_test = LowLearnRate_CNN.model.predict(test_image)
```

```

y_pred = np rint(y_hat_test).astype(np.int) # Round elements of the array
↳ to the nearest integer.
y_true = y_test.astype(np.int)

# Generate a confusion matrix displaying the predictive accuracy of the
↳ model on the test set:
cm = confusion_matrix(y_true, y_pred) # normalize = 'true'
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['normal',
↳ 'pneumonia'])
disp.plot(cmap = "Blues", ax=ax1)
ax1.set_title('Confusion Matrix - Test Set \n')
plt.savefig('./images/CNN_confusionmatrix_cnn', dpi=300, bbox_inches='tight')

```



## 15 Transfer Learning with VGG16:

- Transfer learning (TL) focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. (Source: [https://en.wikipedia.org/wiki/Transfer\\_learning](https://en.wikipedia.org/wiki/Transfer_learning))



- It is quite popular in deep learning where pre-trained models are used as the starting point on Computer Vision (CV) tasks. This way one can get the benefit of using complex models developed by others as start point and on top of it build more.

### 15.0.1 Adjustments to default VGG16:

- Use the weights as it was in original model, so we set weights = 'imagenet'
- Change the image size from the input layer so we can use the model on our images: (128, 128, 3)
- Remove the top layers by setting: include\_top=False to include your own fully connected layers with adjusted neuron parameters.
- Add a final layer with a binary classification output.
- We will not train the default model layers. We will only train the final added layers. So, we will set properties for trainable = False.

### 15.0.2 The default VGG16 model:

```
[74]: from tensorflow.keras.applications import VGG16
```

```
[75]: # the default VGG16 model
model_vgg16=VGG16()
model_vgg16.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 224, 224, 3]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0

block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
=====		
Total params: 138,357,544		
Trainable params: 138,357,544		
Non-trainable params: 0		
-----		

### 15.0.3 Create the base VGG16 model:

```
[76]: base_model = VGG16(include_top = False, weights = "imagenet", input_shape = (128, 128, 3))
      # all the layers intrainable - freeze the layers (except last ones we will be adding)
      base_model.trainable = False
      base_model.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
=====		
input_2 (InputLayer)	[(None, 128, 128, 3)]	0
block1_conv1 (Conv2D)	(None, 128, 128, 64)	1792
block1_conv2 (Conv2D)	(None, 128, 128, 64)	36928

```

-----
block1_pool (MaxPooling2D) (None, 64, 64, 64) 0
-----
block2_conv1 (Conv2D) (None, 64, 64, 128) 73856
-----
block2_conv2 (Conv2D) (None, 64, 64, 128) 147584
-----
block2_pool (MaxPooling2D) (None, 32, 32, 128) 0
-----
block3_conv1 (Conv2D) (None, 32, 32, 256) 295168
-----
block3_conv2 (Conv2D) (None, 32, 32, 256) 590080
-----
block3_conv3 (Conv2D) (None, 32, 32, 256) 590080
-----
block3_pool (MaxPooling2D) (None, 16, 16, 256) 0
-----
block4_conv1 (Conv2D) (None, 16, 16, 512) 1180160
-----
block4_conv2 (Conv2D) (None, 16, 16, 512) 2359808
-----
block4_conv3 (Conv2D) (None, 16, 16, 512) 2359808
-----
block4_pool (MaxPooling2D) (None, 8, 8, 512) 0
-----
block5_conv1 (Conv2D) (None, 8, 8, 512) 2359808
-----
block5_conv2 (Conv2D) (None, 8, 8, 512) 2359808
-----
block5_conv3 (Conv2D) (None, 8, 8, 512) 2359808
-----
block5_pool (MaxPooling2D) (None, 4, 4, 512) 0
=====
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
-----

```

#### 15.0.4 Create the full VGG16 model:

```

[77]: model = Sequential()
      model.add(base_model)
      model.add(Flatten())

      # Add the fully connected layers
      model.add(Dense(128, activation = "relu"))
      model.add(Dropout(0.4)) # regularization

```

```

model.add(Dense(64, activation = "relu"))
model.add(Dropout(0.4)) # regularization
model.add(Dense(1, activation = "sigmoid"))

# You could freeze layers here too:
# for layer in base_model.layers[:-1]: # or for layer in base_model.layers:
#     layer.trainable = False

model.summary()

```

Model: "sequential\_9"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 4, 4, 512)	14714688
flatten_7 (Flatten)	(None, 8192)	0
dense_27 (Dense)	(None, 128)	1048704
dropout_20 (Dropout)	(None, 128)	0
dense_28 (Dense)	(None, 64)	8256
dropout_21 (Dropout)	(None, 64)	0
dense_29 (Dense)	(None, 1)	65

=====  
 Total params: 15,771,713  
 Trainable params: 1,057,025  
 Non-trainable params: 14,714,688  
 =====

```

[78]: # Compile model, select loss function and optimizer to use
model.compile(loss = 'binary_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])

```

```

[79]: # Fit model
VGG16 = model.fit(train_image, y_train, epochs = 100, batch_size=31,
                  validation_data=(val_image, y_val),
                  callbacks = [early_stop])

```

Epoch 1/100

153/153 [=====] - 241s 2s/step - loss: 0.3194 - accuracy: 0.8678 - val\_loss: 0.1442 - val\_accuracy: 0.9544

Epoch 2/100

153/153 [=====] - 239s 2s/step - loss: 0.1745 -  
accuracy: 0.9391 - val\_loss: 0.1317 - val\_accuracy: 0.9525  
Epoch 3/100  
153/153 [=====] - 236s 2s/step - loss: 0.1573 -  
accuracy: 0.9448 - val\_loss: 0.1314 - val\_accuracy: 0.9601  
Epoch 4/100  
153/153 [=====] - 235s 2s/step - loss: 0.1391 -  
accuracy: 0.9526 - val\_loss: 0.1138 - val\_accuracy: 0.9525  
Epoch 5/100  
153/153 [=====] - 235s 2s/step - loss: 0.1275 -  
accuracy: 0.9559 - val\_loss: 0.1130 - val\_accuracy: 0.9544  
Epoch 6/100  
153/153 [=====] - 234s 2s/step - loss: 0.1241 -  
accuracy: 0.9566 - val\_loss: 0.1268 - val\_accuracy: 0.9544  
Epoch 7/100  
153/153 [=====] - 233s 2s/step - loss: 0.1137 -  
accuracy: 0.9585 - val\_loss: 0.1063 - val\_accuracy: 0.9563  
Epoch 8/100  
153/153 [=====] - 232s 2s/step - loss: 0.1162 -  
accuracy: 0.9606 - val\_loss: 0.1149 - val\_accuracy: 0.9506  
Epoch 9/100  
153/153 [=====] - 231s 2s/step - loss: 0.1047 -  
accuracy: 0.9627 - val\_loss: 0.1209 - val\_accuracy: 0.9506  
Epoch 10/100  
153/153 [=====] - 231s 2s/step - loss: 0.1051 -  
accuracy: 0.9574 - val\_loss: 0.1074 - val\_accuracy: 0.9582  
Epoch 11/100  
153/153 [=====] - 231s 2s/step - loss: 0.0897 -  
accuracy: 0.9639 - val\_loss: 0.1108 - val\_accuracy: 0.9601  
Epoch 12/100  
153/153 [=====] - 232s 2s/step - loss: 0.0880 -  
accuracy: 0.9667 - val\_loss: 0.1192 - val\_accuracy: 0.9563  
Epoch 13/100  
153/153 [=====] - 233s 2s/step - loss: 0.0926 -  
accuracy: 0.9650 - val\_loss: 0.1216 - val\_accuracy: 0.9525  
Epoch 14/100  
153/153 [=====] - 234s 2s/step - loss: 0.0893 -  
accuracy: 0.9661 - val\_loss: 0.1084 - val\_accuracy: 0.9563  
Epoch 15/100  
153/153 [=====] - 235s 2s/step - loss: 0.0846 -  
accuracy: 0.9713 - val\_loss: 0.1013 - val\_accuracy: 0.9639  
Epoch 16/100  
153/153 [=====] - 235s 2s/step - loss: 0.0880 -  
accuracy: 0.9673 - val\_loss: 0.1099 - val\_accuracy: 0.9582  
Epoch 17/100  
153/153 [=====] - 235s 2s/step - loss: 0.0779 -  
accuracy: 0.9675 - val\_loss: 0.1231 - val\_accuracy: 0.9601  
Epoch 18/100

```

153/153 [=====] - 236s 2s/step - loss: 0.0668 -
accuracy: 0.9751 - val_loss: 0.1100 - val_accuracy: 0.9639
Epoch 19/100
153/153 [=====] - 236s 2s/step - loss: 0.0723 -
accuracy: 0.9709 - val_loss: 0.0933 - val_accuracy: 0.9620
Epoch 20/100
153/153 [=====] - 235s 2s/step - loss: 0.0792 -
accuracy: 0.9644 - val_loss: 0.1713 - val_accuracy: 0.9506
Epoch 21/100
153/153 [=====] - 240s 2s/step - loss: 0.0760 -
accuracy: 0.9703 - val_loss: 0.1167 - val_accuracy: 0.9601
Epoch 22/100
153/153 [=====] - 245s 2s/step - loss: 0.0521 -
accuracy: 0.9802 - val_loss: 0.1588 - val_accuracy: 0.9601
Epoch 23/100
153/153 [=====] - 242s 2s/step - loss: 0.0633 -
accuracy: 0.9734 - val_loss: 0.1345 - val_accuracy: 0.9639
Epoch 24/100
153/153 [=====] - 244s 2s/step - loss: 0.0641 -
accuracy: 0.9743 - val_loss: 0.1570 - val_accuracy: 0.9601
Epoch 25/100
153/153 [=====] - 242s 2s/step - loss: 0.0652 -
accuracy: 0.9736 - val_loss: 0.1138 - val_accuracy: 0.9601
Epoch 26/100
153/153 [=====] - 236s 2s/step - loss: 0.0727 -
accuracy: 0.9709 - val_loss: 0.1688 - val_accuracy: 0.9620
Epoch 27/100
153/153 [=====] - 237s 2s/step - loss: 0.0616 -
accuracy: 0.9755 - val_loss: 0.1143 - val_accuracy: 0.9639
Epoch 28/100
153/153 [=====] - 235s 2s/step - loss: 0.0635 -
accuracy: 0.9745 - val_loss: 0.1395 - val_accuracy: 0.9639
Epoch 29/100
153/153 [=====] - 235s 2s/step - loss: 0.0539 -
accuracy: 0.9768 - val_loss: 0.1153 - val_accuracy: 0.9639

```

```
[80]: visualize_model_performance(VGG16, train_image, test_image, "accuracy",
    ↪ "val_accuracy")
```

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.91	0.92	159
1	0.97	0.98	0.97	428
accuracy			0.96	587
macro avg	0.95	0.94	0.95	587
weighted avg	0.96	0.96	0.96	587

```
149/149 [=====] - 210s 1s/step - loss: 0.0379 -
accuracy: 0.9871
19/19 [=====] - 25s 1s/step - loss: 0.0960 - accuracy:
0.9591
```

-----

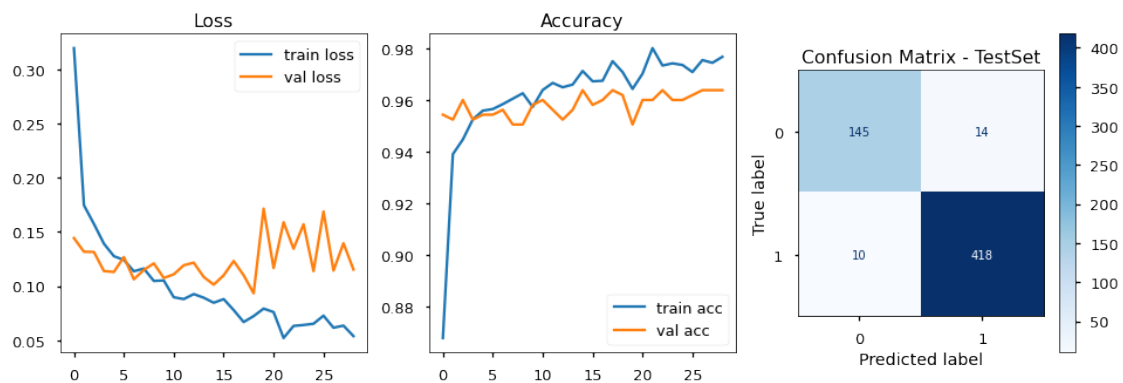
Final Train Loss: 0.0379

Final Test Loss: 0.096

-----

Final Train Acc: 0.9871

Final Test Acc: 0.9591



- Using transfer learning with VGG16 we get very similar results.
- Overall accuracy is 95.9% on the test set.
- Recall for pneumonia is high with 98%. Only 10 out of 428 pneumonia cases are mislabeled as normal.
- Recall for normal is slightly lower with 91%. 14 out of 159 was mislabeled.

## 16 Transfer Learning with RESNET50V2:

### 16.0.1 Default RESNET50V2:

```
[81]: from keras.applications.resnet_v2 import ResNet50V2
```

```
[82]: # the default VGG16 model
model_ResNet50V2=keras.applications.resnet_v2.ResNet50V2()
model_ResNet50V2.summary()
```

Model: "resnet50v2"

-----

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 224, 224, 3)]	0	
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_3[0][0]
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	conv1_pad[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_conv[0][0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
conv2_block1_preact_bn (BatchNormali	(None, 56, 56, 64)	256	pool1_pool[0][0]
conv2_block1_preact_relu (Activ	(None, 56, 56, 64)	0	conv2_block1_preact_bn[0][0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4096	conv2_block1_preact_relu[0][0]
conv2_block1_1_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_block1_1_conv[0][0]
conv2_block1_1_relu (Activation	(None, 56, 56, 64)	0	conv2_block1_1_bn[0][0]
conv2_block1_2_pad (ZeroPadding	(None, 58, 58, 64)	0	conv2_block1_1_relu[0][0]
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36864	conv2_block1_2_pad[0][0]



```

conv2_block1_2_bn (BatchNormali (None, 56, 56, 64) 256
conv2_block1_2_conv[0][0]
-----
conv2_block1_2_relu (Activation (None, 56, 56, 64) 0
conv2_block1_2_bn[0][0]
-----
conv2_block1_0_conv (Conv2D) (None, 56, 56, 256) 16640
conv2_block1_preact_relu[0][0]
-----
conv2_block1_3_conv (Conv2D) (None, 56, 56, 256) 16640
conv2_block1_2_relu[0][0]
-----
conv2_block1_out (Add) (None, 56, 56, 256) 0
conv2_block1_0_conv[0][0]
conv2_block1_3_conv[0][0]
-----
conv2_block2_preact_bn (BatchNo (None, 56, 56, 256) 1024
conv2_block1_out[0][0]
-----
conv2_block2_preact_relu (Activ (None, 56, 56, 256) 0
conv2_block2_preact_bn[0][0]
-----
conv2_block2_1_conv (Conv2D) (None, 56, 56, 64) 16384
conv2_block2_preact_relu[0][0]
-----
conv2_block2_1_bn (BatchNormali (None, 56, 56, 64) 256
conv2_block2_1_conv[0][0]
-----
conv2_block2_1_relu (Activation (None, 56, 56, 64) 0
conv2_block2_1_bn[0][0]
-----
conv2_block2_2_pad (ZeroPadding (None, 58, 58, 64) 0
conv2_block2_1_relu[0][0]
-----
conv2_block2_2_conv (Conv2D) (None, 56, 56, 64) 36864
conv2_block2_2_pad[0][0]
-----

```

```

-----
conv2_block2_2_bn (BatchNormali (None, 56, 56, 64) 256
conv2_block2_2_conv[0][0]
-----
conv2_block2_2_relu (Activation (None, 56, 56, 64) 0
conv2_block2_2_bn[0][0]
-----
conv2_block2_3_conv (Conv2D) (None, 56, 56, 256) 16640
conv2_block2_2_relu[0][0]
-----
conv2_block2_out (Add) (None, 56, 56, 256) 0
conv2_block1_out[0][0]
conv2_block2_3_conv[0][0]
-----
conv2_block3_preact_bn (BatchNo (None, 56, 56, 256) 1024
conv2_block2_out[0][0]
-----
conv2_block3_preact_relu (Activ (None, 56, 56, 256) 0
conv2_block3_preact_bn[0][0]
-----
conv2_block3_1_conv (Conv2D) (None, 56, 56, 64) 16384
conv2_block3_preact_relu[0][0]
-----
conv2_block3_1_bn (BatchNormali (None, 56, 56, 64) 256
conv2_block3_1_conv[0][0]
-----
conv2_block3_1_relu (Activation (None, 56, 56, 64) 0
conv2_block3_1_bn[0][0]
-----
conv2_block3_2_pad (ZeroPadding (None, 58, 58, 64) 0
conv2_block3_1_relu[0][0]
-----
conv2_block3_2_conv (Conv2D) (None, 28, 28, 64) 36864
conv2_block3_2_pad[0][0]
-----
conv2_block3_2_bn (BatchNormali (None, 28, 28, 64) 256
conv2_block3_2_conv[0][0]

```

```

-----
-----
conv2_block3_2_relu (Activation (None, 28, 28, 64) 0
conv2_block3_2_bn[0][0]
-----
-----
max_pooling2d_20 (MaxPooling2D) (None, 28, 28, 256) 0
conv2_block2_out[0][0]
-----
-----
conv2_block3_3_conv (Conv2D) (None, 28, 28, 256) 16640
conv2_block3_2_relu[0][0]
-----
-----
conv2_block3_out (Add) (None, 28, 28, 256) 0
max_pooling2d_20[0][0]
conv2_block3_3_conv[0][0]
-----
-----
conv3_block1_preact_bn (BatchNo (None, 28, 28, 256) 1024
conv2_block3_out[0][0]
-----
-----
conv3_block1_preact_relu (Activ (None, 28, 28, 256) 0
conv3_block1_preact_bn[0][0]
-----
-----
conv3_block1_1_conv (Conv2D) (None, 28, 28, 128) 32768
conv3_block1_preact_relu[0][0]
-----
-----
conv3_block1_1_bn (BatchNormali (None, 28, 28, 128) 512
conv3_block1_1_conv[0][0]
-----
-----
conv3_block1_1_relu (Activation (None, 28, 28, 128) 0
conv3_block1_1_bn[0][0]
-----
-----
conv3_block1_2_pad (ZeroPadding (None, 30, 30, 128) 0
conv3_block1_1_relu[0][0]
-----
-----
conv3_block1_2_conv (Conv2D) (None, 28, 28, 128) 147456
conv3_block1_2_pad[0][0]
-----
-----
conv3_block1_2_bn (BatchNormali (None, 28, 28, 128) 512

```

```

conv3_block1_2_conv[0][0]
-----
-----
conv3_block1_2_relu (Activation (None, 28, 28, 128) 0
conv3_block1_2_bn[0][0]
-----
-----
conv3_block1_0_conv (Conv2D)      (None, 28, 28, 512) 131584
conv3_block1_preact_relu[0][0]
-----
-----
conv3_block1_3_conv (Conv2D)      (None, 28, 28, 512) 66048
conv3_block1_2_relu[0][0]
-----
-----
conv3_block1_out (Add)             (None, 28, 28, 512) 0
conv3_block1_0_conv[0][0]
conv3_block1_3_conv[0][0]
-----
-----
conv3_block2_preact_bn (BatchNo (None, 28, 28, 512) 2048
conv3_block1_out[0][0]
-----
-----
conv3_block2_preact_relu (Activ (None, 28, 28, 512) 0
conv3_block2_preact_bn[0][0]
-----
-----
conv3_block2_1_conv (Conv2D)      (None, 28, 28, 128) 65536
conv3_block2_preact_relu[0][0]
-----
-----
conv3_block2_1_bn (BatchNormali (None, 28, 28, 128) 512
conv3_block2_1_conv[0][0]
-----
-----
conv3_block2_1_relu (Activation (None, 28, 28, 128) 0
conv3_block2_1_bn[0][0]
-----
-----
conv3_block2_2_pad (ZeroPadding (None, 30, 30, 128) 0
conv3_block2_1_relu[0][0]
-----
-----
conv3_block2_2_conv (Conv2D)      (None, 28, 28, 128) 147456
conv3_block2_2_pad[0][0]
-----
-----

```

```

conv3_block2_2_bn (BatchNormali (None, 28, 28, 128) 512
conv3_block2_2_conv[0][0]
-----
conv3_block2_2_relu (Activation (None, 28, 28, 128) 0
conv3_block2_2_bn[0][0]
-----
conv3_block2_3_conv (Conv2D) (None, 28, 28, 512) 66048
conv3_block2_2_relu[0][0]
-----
conv3_block2_out (Add) (None, 28, 28, 512) 0
conv3_block1_out[0][0]
conv3_block2_3_conv[0][0]
-----
conv3_block3_preact_bn (BatchNo (None, 28, 28, 512) 2048
conv3_block2_out[0][0]
-----
conv3_block3_preact_relu (Activ (None, 28, 28, 512) 0
conv3_block3_preact_bn[0][0]
-----
conv3_block3_1_conv (Conv2D) (None, 28, 28, 128) 65536
conv3_block3_preact_relu[0][0]
-----
conv3_block3_1_bn (BatchNormali (None, 28, 28, 128) 512
conv3_block3_1_conv[0][0]
-----
conv3_block3_1_relu (Activation (None, 28, 28, 128) 0
conv3_block3_1_bn[0][0]
-----
conv3_block3_2_pad (ZeroPadding (None, 30, 30, 128) 0
conv3_block3_1_relu[0][0]
-----
conv3_block3_2_conv (Conv2D) (None, 28, 28, 128) 147456
conv3_block3_2_pad[0][0]
-----
conv3_block3_2_bn (BatchNormali (None, 28, 28, 128) 512
conv3_block3_2_conv[0][0]
-----

```

-----  
conv3\_block3\_2\_relu (Activation (None, 28, 28, 128) 0  
conv3\_block3\_2\_bn[0][0]

-----  
conv3\_block3\_3\_conv (Conv2D) (None, 28, 28, 512) 66048  
conv3\_block3\_2\_relu[0][0]

-----  
conv3\_block3\_out (Add) (None, 28, 28, 512) 0  
conv3\_block2\_out[0][0]  
conv3\_block3\_3\_conv[0][0]

-----  
conv3\_block4\_preact\_bn (BatchNo (None, 28, 28, 512) 2048  
conv3\_block3\_out[0][0]

-----  
conv3\_block4\_preact\_relu (Activ (None, 28, 28, 512) 0  
conv3\_block4\_preact\_bn[0][0]

-----  
conv3\_block4\_1\_conv (Conv2D) (None, 28, 28, 128) 65536  
conv3\_block4\_preact\_relu[0][0]

-----  
conv3\_block4\_1\_bn (BatchNormali (None, 28, 28, 128) 512  
conv3\_block4\_1\_conv[0][0]

-----  
conv3\_block4\_1\_relu (Activation (None, 28, 28, 128) 0  
conv3\_block4\_1\_bn[0][0]

-----  
conv3\_block4\_2\_pad (ZeroPadding (None, 30, 30, 128) 0  
conv3\_block4\_1\_relu[0][0]

-----  
conv3\_block4\_2\_conv (Conv2D) (None, 14, 14, 128) 147456  
conv3\_block4\_2\_pad[0][0]

-----  
conv3\_block4\_2\_bn (BatchNormali (None, 14, 14, 128) 512  
conv3\_block4\_2\_conv[0][0]

-----  
conv3\_block4\_2\_relu (Activation (None, 14, 14, 128) 0  
conv3\_block4\_2\_bn[0][0]

```

-----
-----
max_pooling2d_21 (MaxPooling2D) (None, 14, 14, 512) 0
conv3_block3_out[0][0]
-----
-----
conv3_block4_3_conv (Conv2D) (None, 14, 14, 512) 66048
conv3_block4_2_relu[0][0]
-----
-----
conv3_block4_out (Add) (None, 14, 14, 512) 0
max_pooling2d_21[0][0]
conv3_block4_3_conv[0][0]
-----
-----
conv4_block1_preact_bn (BatchNormali (None, 14, 14, 512) 2048
conv3_block4_out[0][0]
-----
-----
conv4_block1_preact_relu (Activ (None, 14, 14, 512) 0
conv4_block1_preact_bn[0][0]
-----
-----
conv4_block1_1_conv (Conv2D) (None, 14, 14, 256) 131072
conv4_block1_preact_relu[0][0]
-----
-----
conv4_block1_1_bn (BatchNormali (None, 14, 14, 256) 1024
conv4_block1_1_conv[0][0]
-----
-----
conv4_block1_1_relu (Activation (None, 14, 14, 256) 0
conv4_block1_1_bn[0][0]
-----
-----
conv4_block1_2_pad (ZeroPadding (None, 16, 16, 256) 0
conv4_block1_1_relu[0][0]
-----
-----
conv4_block1_2_conv (Conv2D) (None, 14, 14, 256) 589824
conv4_block1_2_pad[0][0]
-----
-----
conv4_block1_2_bn (BatchNormali (None, 14, 14, 256) 1024
conv4_block1_2_conv[0][0]
-----
-----
conv4_block1_2_relu (Activation (None, 14, 14, 256) 0

```

conv4\_block1\_2\_bn[0][0]

-----  
-----  
conv4\_block1\_0\_conv (Conv2D) (None, 14, 14, 1024) 525312  
conv4\_block1\_preact\_relu[0][0]  
-----

-----  
conv4\_block1\_3\_conv (Conv2D) (None, 14, 14, 1024) 263168  
conv4\_block1\_2\_relu[0][0]  
-----

-----  
conv4\_block1\_out (Add) (None, 14, 14, 1024) 0  
conv4\_block1\_0\_conv[0][0]  
conv4\_block1\_3\_conv[0][0]  
-----

-----  
conv4\_block2\_preact\_bn (BatchNormali (None, 14, 14, 1024) 4096  
conv4\_block1\_out[0][0]  
-----

-----  
conv4\_block2\_preact\_relu (Activ (None, 14, 14, 1024) 0  
conv4\_block2\_preact\_bn[0][0]  
-----

-----  
conv4\_block2\_1\_conv (Conv2D) (None, 14, 14, 256) 262144  
conv4\_block2\_preact\_relu[0][0]  
-----

-----  
conv4\_block2\_1\_bn (BatchNormali (None, 14, 14, 256) 1024  
conv4\_block2\_1\_conv[0][0]  
-----

-----  
conv4\_block2\_1\_relu (Activation (None, 14, 14, 256) 0  
conv4\_block2\_1\_bn[0][0]  
-----

-----  
conv4\_block2\_2\_pad (ZeroPadding (None, 16, 16, 256) 0  
conv4\_block2\_1\_relu[0][0]  
-----

-----  
conv4\_block2\_2\_conv (Conv2D) (None, 14, 14, 256) 589824  
conv4\_block2\_2\_pad[0][0]  
-----

-----  
conv4\_block2\_2\_bn (BatchNormali (None, 14, 14, 256) 1024  
conv4\_block2\_2\_conv[0][0]  
-----  
-----



```

conv4_block2_2_relu (Activation (None, 14, 14, 256) 0
conv4_block2_2_bn[0][0]
-----

conv4_block2_3_conv (Conv2D)      (None, 14, 14, 1024) 263168
conv4_block2_2_relu[0][0]
-----

conv4_block2_out (Add)              (None, 14, 14, 1024) 0
conv4_block1_out[0][0]
conv4_block2_3_conv[0][0]
-----

conv4_block3_preact_bn (BatchNo (None, 14, 14, 1024) 4096
conv4_block2_out[0][0]
-----

conv4_block3_preact_relu (Activ (None, 14, 14, 1024) 0
conv4_block3_preact_bn[0][0]
-----

conv4_block3_1_conv (Conv2D)      (None, 14, 14, 256) 262144
conv4_block3_preact_relu[0][0]
-----

conv4_block3_1_bn (BatchNormali (None, 14, 14, 256) 1024
conv4_block3_1_conv[0][0]
-----

conv4_block3_1_relu (Activation (None, 14, 14, 256) 0
conv4_block3_1_bn[0][0]
-----

conv4_block3_2_pad (ZeroPadding (None, 16, 16, 256) 0
conv4_block3_1_relu[0][0]
-----

conv4_block3_2_conv (Conv2D)      (None, 14, 14, 256) 589824
conv4_block3_2_pad[0][0]
-----

conv4_block3_2_bn (BatchNormali (None, 14, 14, 256) 1024
conv4_block3_2_conv[0][0]
-----

conv4_block3_2_relu (Activation (None, 14, 14, 256) 0
conv4_block3_2_bn[0][0]
-----

```

-----  
conv4\_block3\_3\_conv (Conv2D) (None, 14, 14, 1024) 263168  
conv4\_block3\_2\_relu[0][0]  
-----

-----  
conv4\_block3\_out (Add) (None, 14, 14, 1024) 0  
conv4\_block2\_out[0][0]  
conv4\_block3\_3\_conv[0][0]  
-----

-----  
conv4\_block4\_preact\_bn (BatchNormali (None, 14, 14, 1024) 4096  
conv4\_block3\_out[0][0]  
-----

-----  
conv4\_block4\_preact\_relu (Activ (None, 14, 14, 1024) 0  
conv4\_block4\_preact\_bn[0][0]  
-----

-----  
conv4\_block4\_1\_conv (Conv2D) (None, 14, 14, 256) 262144  
conv4\_block4\_preact\_relu[0][0]  
-----

-----  
conv4\_block4\_1\_bn (BatchNormali (None, 14, 14, 256) 1024  
conv4\_block4\_1\_conv[0][0]  
-----

-----  
conv4\_block4\_1\_relu (Activation (None, 14, 14, 256) 0  
conv4\_block4\_1\_bn[0][0]  
-----

-----  
conv4\_block4\_2\_pad (ZeroPadding (None, 16, 16, 256) 0  
conv4\_block4\_1\_relu[0][0]  
-----

-----  
conv4\_block4\_2\_conv (Conv2D) (None, 14, 14, 256) 589824  
conv4\_block4\_2\_pad[0][0]  
-----

-----  
conv4\_block4\_2\_bn (BatchNormali (None, 14, 14, 256) 1024  
conv4\_block4\_2\_conv[0][0]  
-----

-----  
conv4\_block4\_2\_relu (Activation (None, 14, 14, 256) 0  
conv4\_block4\_2\_bn[0][0]  
-----

-----  
conv4\_block4\_3\_conv (Conv2D) (None, 14, 14, 1024) 263168  
conv4\_block4\_2\_relu[0][0]  
-----

```

-----
conv4_block4_out (Add)          (None, 14, 14, 1024) 0
conv4_block3_out[0][0]
conv4_block4_3_conv[0][0]
-----

conv4_block5_preact_bn (BatchNo (None, 14, 14, 1024) 4096
conv4_block4_out[0][0]
-----

conv4_block5_preact_relu (Activ (None, 14, 14, 1024) 0
conv4_block5_preact_bn[0][0]
-----

conv4_block5_1_conv (Conv2D)    (None, 14, 14, 256) 262144
conv4_block5_preact_relu[0][0]
-----

conv4_block5_1_bn (BatchNormali (None, 14, 14, 256) 1024
conv4_block5_1_conv[0][0]
-----

conv4_block5_1_relu (Activation (None, 14, 14, 256) 0
conv4_block5_1_bn[0][0]
-----

conv4_block5_2_pad (ZeroPadding (None, 16, 16, 256) 0
conv4_block5_1_relu[0][0]
-----

conv4_block5_2_conv (Conv2D)    (None, 14, 14, 256) 589824
conv4_block5_2_pad[0][0]
-----

conv4_block5_2_bn (BatchNormali (None, 14, 14, 256) 1024
conv4_block5_2_conv[0][0]
-----

conv4_block5_2_relu (Activation (None, 14, 14, 256) 0
conv4_block5_2_bn[0][0]
-----

conv4_block5_3_conv (Conv2D)    (None, 14, 14, 1024) 263168
conv4_block5_2_relu[0][0]
-----

conv4_block5_out (Add)          (None, 14, 14, 1024) 0

```

```

conv4_block4_out[0][0]
conv4_block5_3_conv[0][0]
-----
conv4_block6_preact_bn (BatchNo (None, 14, 14, 1024) 4096
conv4_block5_out[0][0]
-----
conv4_block6_preact_relu (Activ (None, 14, 14, 1024) 0
conv4_block6_preact_bn[0][0]
-----
conv4_block6_1_conv (Conv2D) (None, 14, 14, 256) 262144
conv4_block6_preact_relu[0][0]
-----
conv4_block6_1_bn (BatchNormali (None, 14, 14, 256) 1024
conv4_block6_1_conv[0][0]
-----
conv4_block6_1_relu (Activation (None, 14, 14, 256) 0
conv4_block6_1_bn[0][0]
-----
conv4_block6_2_pad (ZeroPadding (None, 16, 16, 256) 0
conv4_block6_1_relu[0][0]
-----
conv4_block6_2_conv (Conv2D) (None, 7, 7, 256) 589824
conv4_block6_2_pad[0][0]
-----
conv4_block6_2_bn (BatchNormali (None, 7, 7, 256) 1024
conv4_block6_2_conv[0][0]
-----
conv4_block6_2_relu (Activation (None, 7, 7, 256) 0
conv4_block6_2_bn[0][0]
-----
max_pooling2d_22 (MaxPooling2D) (None, 7, 7, 1024) 0
conv4_block5_out[0][0]
-----
conv4_block6_3_conv (Conv2D) (None, 7, 7, 1024) 263168
conv4_block6_2_relu[0][0]
-----

```

conv4_block6_out (Add)	(None, 7, 7, 1024)	0
max_pooling2d_22[0][0]		
conv4_block6_3_conv[0][0]		
-----		
conv5_block1_preact_bn (BatchNo	(None, 7, 7, 1024)	4096
conv4_block6_out[0][0]		
-----		
conv5_block1_preact_relu (Activ	(None, 7, 7, 1024)	0
conv5_block1_preact_bn[0][0]		
-----		
conv5_block1_1_conv (Conv2D)	(None, 7, 7, 512)	524288
conv5_block1_preact_relu[0][0]		
-----		
conv5_block1_1_bn (BatchNormali	(None, 7, 7, 512)	2048
conv5_block1_1_conv[0][0]		
-----		
conv5_block1_1_relu (Activation	(None, 7, 7, 512)	0
conv5_block1_1_bn[0][0]		
-----		
conv5_block1_2_pad (ZeroPadding	(None, 9, 9, 512)	0
conv5_block1_1_relu[0][0]		
-----		
conv5_block1_2_conv (Conv2D)	(None, 7, 7, 512)	2359296
conv5_block1_2_pad[0][0]		
-----		
conv5_block1_2_bn (BatchNormali	(None, 7, 7, 512)	2048
conv5_block1_2_conv[0][0]		
-----		
conv5_block1_2_relu (Activation	(None, 7, 7, 512)	0
conv5_block1_2_bn[0][0]		
-----		
conv5_block1_0_conv (Conv2D)	(None, 7, 7, 2048)	2099200
conv5_block1_preact_relu[0][0]		
-----		
conv5_block1_3_conv (Conv2D)	(None, 7, 7, 2048)	1050624
conv5_block1_2_relu[0][0]		
-----		

```

-----
conv5_block1_out (Add)          (None, 7, 7, 2048)    0
conv5_block1_0_conv[0][0]
conv5_block1_3_conv[0][0]
-----

conv5_block2_preact_bn (BatchNo (None, 7, 7, 2048)    8192
conv5_block1_out[0][0]
-----

conv5_block2_preact_relu (Activ (None, 7, 7, 2048)    0
conv5_block2_preact_bn[0][0]
-----

conv5_block2_1_conv (Conv2D)    (None, 7, 7, 512)    1048576
conv5_block2_preact_relu[0][0]
-----

conv5_block2_1_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block2_1_conv[0][0]
-----

conv5_block2_1_relu (Activation (None, 7, 7, 512)    0
conv5_block2_1_bn[0][0]
-----

conv5_block2_2_pad (ZeroPadding (None, 9, 9, 512)    0
conv5_block2_1_relu[0][0]
-----

conv5_block2_2_conv (Conv2D)    (None, 7, 7, 512)    2359296
conv5_block2_2_pad[0][0]
-----

conv5_block2_2_bn (BatchNormali (None, 7, 7, 512)    2048
conv5_block2_2_conv[0][0]
-----

conv5_block2_2_relu (Activation (None, 7, 7, 512)    0
conv5_block2_2_bn[0][0]
-----

conv5_block2_3_conv (Conv2D)    (None, 7, 7, 2048)    1050624
conv5_block2_2_relu[0][0]
-----

conv5_block2_out (Add)          (None, 7, 7, 2048)    0
conv5_block1_out[0][0]

```

conv5\_block2\_3\_conv[0][0]

-----  
-----  
conv5\_block3\_preact\_bn (BatchNo (None, 7, 7, 2048) 8192  
conv5\_block2\_out[0][0]

-----  
-----  
conv5\_block3\_preact\_relu (Activ (None, 7, 7, 2048) 0  
conv5\_block3\_preact\_bn[0][0]

-----  
-----  
conv5\_block3\_1\_conv (Conv2D) (None, 7, 7, 512) 1048576  
conv5\_block3\_preact\_relu[0][0]

-----  
-----  
conv5\_block3\_1\_bn (BatchNormali (None, 7, 7, 512) 2048  
conv5\_block3\_1\_conv[0][0]

-----  
-----  
conv5\_block3\_1\_relu (Activation (None, 7, 7, 512) 0  
conv5\_block3\_1\_bn[0][0]

-----  
-----  
conv5\_block3\_2\_pad (ZeroPadding (None, 9, 9, 512) 0  
conv5\_block3\_1\_relu[0][0]

-----  
-----  
conv5\_block3\_2\_conv (Conv2D) (None, 7, 7, 512) 2359296  
conv5\_block3\_2\_pad[0][0]

-----  
-----  
conv5\_block3\_2\_bn (BatchNormali (None, 7, 7, 512) 2048  
conv5\_block3\_2\_conv[0][0]

-----  
-----  
conv5\_block3\_2\_relu (Activation (None, 7, 7, 512) 0  
conv5\_block3\_2\_bn[0][0]

-----  
-----  
conv5\_block3\_3\_conv (Conv2D) (None, 7, 7, 2048) 1050624  
conv5\_block3\_2\_relu[0][0]

-----  
-----  
conv5\_block3\_out (Add) (None, 7, 7, 2048) 0  
conv5\_block2\_out[0][0]  
conv5\_block3\_3\_conv[0][0]

```

post_bn (BatchNormalization)      (None, 7, 7, 2048)    8192
conv5_block3_out[0][0]

-----

post_relu (Activation)            (None, 7, 7, 2048)    0          post_bn[0][0]

-----

avg_pool (GlobalAveragePooling2) (None, 2048)          0          post_relu[0][0]

-----

predictions (Dense)              (None, 1000)          2049000    avg_pool[0][0]
=====
Total params: 25,613,800
Trainable params: 25,568,360
Non-trainable params: 45,440
-----
-----

```

## 16.0.2 Adjustments to default RESNET50:

- Use the weights as it was in original model, so we set weights = 'imagenet'
- Change the image size from the input layer so we can use the model on our images: (128, 128, 3)
- Remove the top layers by setting: include\_top=False
- Add a final layer with a binary classification output.
- We will not train the default model layers. We will only train the final added layers. So, we will set properties for trainable = False.

```

[109]: base_model = keras.applications.resnet_v2.ResNet50V2(weights = "imagenet",
                                                    input_shape = (128, 128, 3),
                                                    pooling = "max", include_top =
↪False,
                                                    classes = 2)

base_model.trainable = False
base_model.summary()

```

Model: "resnet50v2"

```

-----

Layer (type)                 Output Shape          Param #   Connected to
=====
input_5 (InputLayer)         [(None, 128, 128, 3) 0
-----

conv1_pad (ZeroPadding2D)    (None, 134, 134, 3)  0          input_5[0][0]

```



```

-----
conv1_conv (Conv2D)          (None, 64, 64, 64)  9472      conv1_pad[0][0]
-----
pool1_pad (ZeroPadding2D)    (None, 66, 66, 64)  0
conv1_conv[0][0]
-----
pool1_pool (MaxPooling2D)    (None, 32, 32, 64)  0      pool1_pad[0][0]
-----
conv2_block1_preact_bn (BatchNo (None, 32, 32, 64)  256
pool1_pool[0][0]
-----
conv2_block1_preact_relu (Activ (None, 32, 32, 64)  0
conv2_block1_preact_bn[0][0]
-----
conv2_block1_1_conv (Conv2D)  (None, 32, 32, 64)  4096
conv2_block1_preact_relu[0][0]
-----
conv2_block1_1_bn (BatchNormali (None, 32, 32, 64)  256
conv2_block1_1_conv[0][0]
-----
conv2_block1_1_relu (Activation (None, 32, 32, 64)  0
conv2_block1_1_bn[0][0]
-----
conv2_block1_2_pad (ZeroPadding (None, 34, 34, 64)  0
conv2_block1_1_relu[0][0]
-----
conv2_block1_2_conv (Conv2D)    (None, 32, 32, 64)  36864
conv2_block1_2_pad[0][0]
-----
conv2_block1_2_bn (BatchNormali (None, 32, 32, 64)  256
conv2_block1_2_conv[0][0]
-----
conv2_block1_2_relu (Activation (None, 32, 32, 64)  0
conv2_block1_2_bn[0][0]
-----
-----

```

conv2\_block1\_0\_conv (Conv2D) (None, 32, 32, 256) 16640  
conv2\_block1\_preact\_relu[0][0]

conv2\_block1\_3\_conv (Conv2D) (None, 32, 32, 256) 16640  
conv2\_block1\_2\_relu[0][0]

conv2\_block1\_out (Add) (None, 32, 32, 256) 0  
conv2\_block1\_0\_conv[0][0]  
conv2\_block1\_3\_conv[0][0]

conv2\_block2\_preact\_bn (BatchNormaliz (None, 32, 32, 256) 1024  
conv2\_block1\_out[0][0]

conv2\_block2\_preact\_relu (Activation (None, 32, 32, 256) 0  
conv2\_block2\_preact\_bn[0][0]

conv2\_block2\_1\_conv (Conv2D) (None, 32, 32, 64) 16384  
conv2\_block2\_preact\_relu[0][0]

conv2\_block2\_1\_bn (BatchNormalizati (None, 32, 32, 64) 256  
conv2\_block2\_1\_conv[0][0]

conv2\_block2\_1\_relu (Activation) (None, 32, 32, 64) 0  
conv2\_block2\_1\_bn[0][0]

conv2\_block2\_2\_pad (ZeroPadding) (None, 34, 34, 64) 0  
conv2\_block2\_1\_relu[0][0]

conv2\_block2\_2\_conv (Conv2D) (None, 32, 32, 64) 36864  
conv2\_block2\_2\_pad[0][0]

conv2\_block2\_2\_bn (BatchNormalizati (None, 32, 32, 64) 256  
conv2\_block2\_2\_conv[0][0]

conv2\_block2\_2\_relu (Activation) (None, 32, 32, 64) 0  
conv2\_block2\_2\_bn[0][0]

```
-----
conv2_block2_3_conv (Conv2D)      (None, 32, 32, 256) 16640
conv2_block2_2_relu[0][0]
-----
```

```
-----
conv2_block2_out (Add)            (None, 32, 32, 256) 0
conv2_block1_out[0][0]
conv2_block2_3_conv[0][0]
-----
```

```
-----
conv2_block3_preact_bn (BatchNo (None, 32, 32, 256) 1024
conv2_block2_out[0][0]
-----
```

```
-----
conv2_block3_preact_relu (Activ (None, 32, 32, 256) 0
conv2_block3_preact_bn[0][0]
-----
```

```
-----
conv2_block3_1_conv (Conv2D)      (None, 32, 32, 64) 16384
conv2_block3_preact_relu[0][0]
-----
```

```
-----
conv2_block3_1_bn (BatchNormali (None, 32, 32, 64) 256
conv2_block3_1_conv[0][0]
-----
```

```
-----
conv2_block3_1_relu (Activation (None, 32, 32, 64) 0
conv2_block3_1_bn[0][0]
-----
```

```
-----
conv2_block3_2_pad (ZeroPadding (None, 34, 34, 64) 0
conv2_block3_1_relu[0][0]
-----
```

```
-----
conv2_block3_2_conv (Conv2D)      (None, 16, 16, 64) 36864
conv2_block3_2_pad[0][0]
-----
```

```
-----
conv2_block3_2_bn (BatchNormali (None, 16, 16, 64) 256
conv2_block3_2_conv[0][0]
-----
```

```
-----
conv2_block3_2_relu (Activation (None, 16, 16, 64) 0
conv2_block3_2_bn[0][0]
-----
```

```
-----
max_pooling2d_26 (MaxPooling2D) (None, 16, 16, 256) 0
conv2_block2_out[0][0]
-----
```

```

-----
conv2_block3_3_conv (Conv2D)      (None, 16, 16, 256) 16640
conv2_block3_2_relu[0][0]
-----

conv2_block3_out (Add)            (None, 16, 16, 256) 0
max_pooling2d_26[0][0]
conv2_block3_3_conv[0][0]
-----

conv3_block1_preact_bn (BatchNo (None, 16, 16, 256) 1024
conv2_block3_out[0][0]
-----

conv3_block1_preact_relu (Activ (None, 16, 16, 256) 0
conv3_block1_preact_bn[0][0]
-----

conv3_block1_1_conv (Conv2D)      (None, 16, 16, 128) 32768
conv3_block1_preact_relu[0][0]
-----

conv3_block1_1_bn (BatchNormali (None, 16, 16, 128) 512
conv3_block1_1_conv[0][0]
-----

conv3_block1_1_relu (Activation (None, 16, 16, 128) 0
conv3_block1_1_bn[0][0]
-----

conv3_block1_2_pad (ZeroPadding (None, 18, 18, 128) 0
conv3_block1_1_relu[0][0]
-----

conv3_block1_2_conv (Conv2D)      (None, 16, 16, 128) 147456
conv3_block1_2_pad[0][0]
-----

conv3_block1_2_bn (BatchNormali (None, 16, 16, 128) 512
conv3_block1_2_conv[0][0]
-----

conv3_block1_2_relu (Activation (None, 16, 16, 128) 0
conv3_block1_2_bn[0][0]
-----

conv3_block1_0_conv (Conv2D)      (None, 16, 16, 512) 131584

```

conv3\_block1\_preact\_relu[0][0]

-----  
-----  
conv3\_block1\_3\_conv (Conv2D) (None, 16, 16, 512) 66048  
conv3\_block1\_2\_relu[0][0]

-----  
-----  
conv3\_block1\_out (Add) (None, 16, 16, 512) 0  
conv3\_block1\_0\_conv[0][0]  
conv3\_block1\_3\_conv[0][0]

-----  
-----  
conv3\_block2\_preact\_bn (BatchNormaliz (None, 16, 16, 512) 2048  
conv3\_block1\_out[0][0]

-----  
-----  
conv3\_block2\_preact\_relu (Activation (None, 16, 16, 512) 0  
conv3\_block2\_preact\_bn[0][0]

-----  
-----  
conv3\_block2\_1\_conv (Conv2D) (None, 16, 16, 128) 65536  
conv3\_block2\_preact\_relu[0][0]

-----  
-----  
conv3\_block2\_1\_bn (BatchNormalizati (None, 16, 16, 128) 512  
conv3\_block2\_1\_conv[0][0]

-----  
-----  
conv3\_block2\_1\_relu (Activation (None, 16, 16, 128) 0  
conv3\_block2\_1\_bn[0][0]

-----  
-----  
conv3\_block2\_2\_pad (ZeroPadding (None, 18, 18, 128) 0  
conv3\_block2\_1\_relu[0][0]

-----  
-----  
conv3\_block2\_2\_conv (Conv2D) (None, 16, 16, 128) 147456  
conv3\_block2\_2\_pad[0][0]

-----  
-----  
conv3\_block2\_2\_bn (BatchNormalizati (None, 16, 16, 128) 512  
conv3\_block2\_2\_conv[0][0]

-----  
-----  
conv3\_block2\_2\_relu (Activation (None, 16, 16, 128) 0  
conv3\_block2\_2\_bn[0][0]

conv3\_block2\_3\_conv (Conv2D) (None, 16, 16, 512) 66048  
conv3\_block2\_2\_relu[0][0]

conv3\_block2\_out (Add) (None, 16, 16, 512) 0  
conv3\_block1\_out[0][0]  
conv3\_block2\_3\_conv[0][0]

conv3\_block3\_preact\_bn (BatchNormali (None, 16, 16, 512) 2048  
conv3\_block2\_out[0][0]

conv3\_block3\_preact\_relu (Activ (None, 16, 16, 512) 0  
conv3\_block3\_preact\_bn[0][0]

conv3\_block3\_1\_conv (Conv2D) (None, 16, 16, 128) 65536  
conv3\_block3\_preact\_relu[0][0]

conv3\_block3\_1\_bn (BatchNormali (None, 16, 16, 128) 512  
conv3\_block3\_1\_conv[0][0]

conv3\_block3\_1\_relu (Activation (None, 16, 16, 128) 0  
conv3\_block3\_1\_bn[0][0]

conv3\_block3\_2\_pad (ZeroPadding (None, 18, 18, 128) 0  
conv3\_block3\_1\_relu[0][0]

conv3\_block3\_2\_conv (Conv2D) (None, 16, 16, 128) 147456  
conv3\_block3\_2\_pad[0][0]

conv3\_block3\_2\_bn (BatchNormali (None, 16, 16, 128) 512  
conv3\_block3\_2\_conv[0][0]

conv3\_block3\_2\_relu (Activation (None, 16, 16, 128) 0  
conv3\_block3\_2\_bn[0][0]

conv3\_block3\_3\_conv (Conv2D) (None, 16, 16, 512) 66048  
conv3\_block3\_2\_relu[0][0]

```

-----
conv3_block3_out (Add)          (None, 16, 16, 512)  0
conv3_block2_out[0][0]
conv3_block3_3_conv[0][0]
-----

conv3_block4_preact_bn (BatchNo (None, 16, 16, 512)  2048
conv3_block3_out[0][0]
-----

conv3_block4_preact_relu (Activ (None, 16, 16, 512)  0
conv3_block4_preact_bn[0][0]
-----

conv3_block4_1_conv (Conv2D)    (None, 16, 16, 128)  65536
conv3_block4_preact_relu[0][0]
-----

conv3_block4_1_bn (BatchNormali (None, 16, 16, 128)  512
conv3_block4_1_conv[0][0]
-----

conv3_block4_1_relu (Activation (None, 16, 16, 128)  0
conv3_block4_1_bn[0][0]
-----

conv3_block4_2_pad (ZeroPadding (None, 18, 18, 128)  0
conv3_block4_1_relu[0][0]
-----

conv3_block4_2_conv (Conv2D)    (None, 8, 8, 128)   147456
conv3_block4_2_pad[0][0]
-----

conv3_block4_2_bn (BatchNormali (None, 8, 8, 128)   512
conv3_block4_2_conv[0][0]
-----

conv3_block4_2_relu (Activation (None, 8, 8, 128)   0
conv3_block4_2_bn[0][0]
-----

max_pooling2d_27 (MaxPooling2D) (None, 8, 8, 512)   0
conv3_block3_out[0][0]
-----

conv3_block4_3_conv (Conv2D)    (None, 8, 8, 512)   66048
conv3_block4_2_relu[0][0]

```

```

-----
-----
conv3_block4_out (Add)          (None, 8, 8, 512)    0
max_pooling2d_27[0][0]
conv3_block4_3_conv[0][0]
-----
-----
conv4_block1_preact_bn (BatchNo (None, 8, 8, 512)    2048
conv3_block4_out[0][0]
-----
-----
conv4_block1_preact_relu (Activ (None, 8, 8, 512)    0
conv4_block1_preact_bn[0][0]
-----
-----
conv4_block1_1_conv (Conv2D)    (None, 8, 8, 256)    131072
conv4_block1_preact_relu[0][0]
-----
-----
conv4_block1_1_bn (BatchNormali (None, 8, 8, 256)    1024
conv4_block1_1_conv[0][0]
-----
-----
conv4_block1_1_relu (Activation (None, 8, 8, 256)    0
conv4_block1_1_bn[0][0]
-----
-----
conv4_block1_2_pad (ZeroPadding (None, 10, 10, 256)  0
conv4_block1_1_relu[0][0]
-----
-----
conv4_block1_2_conv (Conv2D)    (None, 8, 8, 256)    589824
conv4_block1_2_pad[0][0]
-----
-----
conv4_block1_2_bn (BatchNormali (None, 8, 8, 256)    1024
conv4_block1_2_conv[0][0]
-----
-----
conv4_block1_2_relu (Activation (None, 8, 8, 256)    0
conv4_block1_2_bn[0][0]
-----
-----
conv4_block1_0_conv (Conv2D)    (None, 8, 8, 1024)   525312
conv4_block1_preact_relu[0][0]
-----
-----
conv4_block1_3_conv (Conv2D)    (None, 8, 8, 1024)   263168

```



```

conv4_block1_2_relu[0][0]
-----
-----
conv4_block1_out (Add)          (None, 8, 8, 1024)    0
conv4_block1_0_conv[0][0]
conv4_block1_3_conv[0][0]
-----
-----
conv4_block2_preact_bn (BatchNo (None, 8, 8, 1024)    4096
conv4_block1_out[0][0]
-----
-----
conv4_block2_preact_relu (Activ (None, 8, 8, 1024)    0
conv4_block2_preact_bn[0][0]
-----
-----
conv4_block2_1_conv (Conv2D)     (None, 8, 8, 256)    262144
conv4_block2_preact_relu[0][0]
-----
-----
conv4_block2_1_bn (BatchNormali (None, 8, 8, 256)    1024
conv4_block2_1_conv[0][0]
-----
-----
conv4_block2_1_relu (Activation (None, 8, 8, 256)    0
conv4_block2_1_bn[0][0]
-----
-----
conv4_block2_2_pad (ZeroPadding (None, 10, 10, 256)    0
conv4_block2_1_relu[0][0]
-----
-----
conv4_block2_2_conv (Conv2D)     (None, 8, 8, 256)    589824
conv4_block2_2_pad[0][0]
-----
-----
conv4_block2_2_bn (BatchNormali (None, 8, 8, 256)    1024
conv4_block2_2_conv[0][0]
-----
-----
conv4_block2_2_relu (Activation (None, 8, 8, 256)    0
conv4_block2_2_bn[0][0]
-----
-----
conv4_block2_3_conv (Conv2D)     (None, 8, 8, 1024)    263168
conv4_block2_2_relu[0][0]
-----
-----

```

```

conv4_block2_out (Add)          (None, 8, 8, 1024)    0
conv4_block1_out[0][0]
conv4_block2_3_conv[0][0]
-----

conv4_block3_preact_bn (BatchNo (None, 8, 8, 1024)    4096
conv4_block2_out[0][0]
-----

conv4_block3_preact_relu (Activ (None, 8, 8, 1024)    0
conv4_block3_preact_bn[0][0]
-----

conv4_block3_1_conv (Conv2D)    (None, 8, 8, 256)    262144
conv4_block3_preact_relu[0][0]
-----

conv4_block3_1_bn (BatchNormali (None, 8, 8, 256)    1024
conv4_block3_1_conv[0][0]
-----

conv4_block3_1_relu (Activation (None, 8, 8, 256)    0
conv4_block3_1_bn[0][0]
-----

conv4_block3_2_pad (ZeroPadding (None, 10, 10, 256)  0
conv4_block3_1_relu[0][0]
-----

conv4_block3_2_conv (Conv2D)    (None, 8, 8, 256)    589824
conv4_block3_2_pad[0][0]
-----

conv4_block3_2_bn (BatchNormali (None, 8, 8, 256)    1024
conv4_block3_2_conv[0][0]
-----

conv4_block3_2_relu (Activation (None, 8, 8, 256)    0
conv4_block3_2_bn[0][0]
-----

conv4_block3_3_conv (Conv2D)    (None, 8, 8, 1024)    263168
conv4_block3_2_relu[0][0]
-----

conv4_block3_out (Add)          (None, 8, 8, 1024)    0
conv4_block2_out[0][0]
conv4_block3_3_conv[0][0]

```

```

-----
-----
conv4_block4_preact_bn (BatchNo (None, 8, 8, 1024) 4096
conv4_block3_out[0][0]
-----
-----
conv4_block4_preact_relu (Activ (None, 8, 8, 1024) 0
conv4_block4_preact_bn[0][0]
-----
-----
conv4_block4_1_conv (Conv2D) (None, 8, 8, 256) 262144
conv4_block4_preact_relu[0][0]
-----
-----
conv4_block4_1_bn (BatchNormali (None, 8, 8, 256) 1024
conv4_block4_1_conv[0][0]
-----
-----
conv4_block4_1_relu (Activation (None, 8, 8, 256) 0
conv4_block4_1_bn[0][0]
-----
-----
conv4_block4_2_pad (ZeroPadding (None, 10, 10, 256) 0
conv4_block4_1_relu[0][0]
-----
-----
conv4_block4_2_conv (Conv2D) (None, 8, 8, 256) 589824
conv4_block4_2_pad[0][0]
-----
-----
conv4_block4_2_bn (BatchNormali (None, 8, 8, 256) 1024
conv4_block4_2_conv[0][0]
-----
-----
conv4_block4_2_relu (Activation (None, 8, 8, 256) 0
conv4_block4_2_bn[0][0]
-----
-----
conv4_block4_3_conv (Conv2D) (None, 8, 8, 1024) 263168
conv4_block4_2_relu[0][0]
-----
-----
conv4_block4_out (Add) (None, 8, 8, 1024) 0
conv4_block3_out[0][0]
conv4_block4_3_conv[0][0]
-----
-----
conv4_block5_preact_bn (BatchNo (None, 8, 8, 1024) 4096

```

```

conv4_block4_out[0][0]
-----
-----
conv4_block5_preact_relu (Activ (None, 8, 8, 1024)    0
conv4_block5_preact_bn[0][0]
-----
-----
conv4_block5_1_conv (Conv2D)      (None, 8, 8, 256)    262144
conv4_block5_preact_relu[0][0]
-----
-----
conv4_block5_1_bn (BatchNormali (None, 8, 8, 256)    1024
conv4_block5_1_conv[0][0]
-----
-----
conv4_block5_1_relu (Activation (None, 8, 8, 256)    0
conv4_block5_1_bn[0][0]
-----
-----
conv4_block5_2_pad (ZeroPadding (None, 10, 10, 256)  0
conv4_block5_1_relu[0][0]
-----
-----
conv4_block5_2_conv (Conv2D)      (None, 8, 8, 256)    589824
conv4_block5_2_pad[0][0]
-----
-----
conv4_block5_2_bn (BatchNormali (None, 8, 8, 256)    1024
conv4_block5_2_conv[0][0]
-----
-----
conv4_block5_2_relu (Activation (None, 8, 8, 256)    0
conv4_block5_2_bn[0][0]
-----
-----
conv4_block5_3_conv (Conv2D)      (None, 8, 8, 1024)   263168
conv4_block5_2_relu[0][0]
-----
-----
conv4_block5_out (Add)             (None, 8, 8, 1024)   0
conv4_block4_out[0][0]
conv4_block5_3_conv[0][0]
-----
-----
conv4_block6_preact_bn (BatchNo (None, 8, 8, 1024)   4096
conv4_block5_out[0][0]
-----
-----

```

```

conv4_block6_preact_relu (Activ (None, 8, 8, 1024)    0
conv4_block6_preact_bn[0][0]

-----

conv4_block6_1_conv (Conv2D)      (None, 8, 8, 256)    262144
conv4_block6_preact_relu[0][0]

-----

conv4_block6_1_bn (BatchNormali (None, 8, 8, 256)    1024
conv4_block6_1_conv[0][0]

-----

conv4_block6_1_relu (Activation (None, 8, 8, 256)    0
conv4_block6_1_bn[0][0]

-----

conv4_block6_2_pad (ZeroPadding (None, 10, 10, 256)  0
conv4_block6_1_relu[0][0]

-----

conv4_block6_2_conv (Conv2D)      (None, 4, 4, 256)    589824
conv4_block6_2_pad[0][0]

-----

conv4_block6_2_bn (BatchNormali (None, 4, 4, 256)    1024
conv4_block6_2_conv[0][0]

-----

conv4_block6_2_relu (Activation (None, 4, 4, 256)    0
conv4_block6_2_bn[0][0]

-----

max_pooling2d_28 (MaxPooling2D) (None, 4, 4, 1024)  0
conv4_block5_out[0][0]

-----

conv4_block6_3_conv (Conv2D)      (None, 4, 4, 1024)   263168
conv4_block6_2_relu[0][0]

-----

conv4_block6_out (Add)             (None, 4, 4, 1024)   0
max_pooling2d_28[0][0]
conv4_block6_3_conv[0][0]

-----

conv5_block1_preact_bn (BatchNo (None, 4, 4, 1024)   4096
conv4_block6_out[0][0]

```

```

-----
conv5_block1_preact_relu (Activ (None, 4, 4, 1024)    0
conv5_block1_preact_bn[0][0]

-----

conv5_block1_1_conv (Conv2D)      (None, 4, 4, 512)    524288
conv5_block1_preact_relu[0][0]

-----

conv5_block1_1_bn (BatchNormali (None, 4, 4, 512)    2048
conv5_block1_1_conv[0][0]

-----

conv5_block1_1_relu (Activation (None, 4, 4, 512)    0
conv5_block1_1_bn[0][0]

-----

conv5_block1_2_pad (ZeroPadding (None, 6, 6, 512)    0
conv5_block1_1_relu[0][0]

-----

conv5_block1_2_conv (Conv2D)      (None, 4, 4, 512)    2359296
conv5_block1_2_pad[0][0]

-----

conv5_block1_2_bn (BatchNormali (None, 4, 4, 512)    2048
conv5_block1_2_conv[0][0]

-----

conv5_block1_2_relu (Activation (None, 4, 4, 512)    0
conv5_block1_2_bn[0][0]

-----

conv5_block1_0_conv (Conv2D)      (None, 4, 4, 2048)    2099200
conv5_block1_preact_relu[0][0]

-----

conv5_block1_3_conv (Conv2D)      (None, 4, 4, 2048)    1050624
conv5_block1_2_relu[0][0]

-----

conv5_block1_out (Add)            (None, 4, 4, 2048)    0
conv5_block1_0_conv[0][0]
conv5_block1_3_conv[0][0]

-----

conv5_block2_preact_bn (BatchNo (None, 4, 4, 2048)    8192
conv5_block1_out[0][0]

```

```

-----
conv5_block2_preact_relu (Activ (None, 4, 4, 2048)    0
conv5_block2_preact_bn[0][0]
-----
conv5_block2_1_conv (Conv2D)      (None, 4, 4, 512)    1048576
conv5_block2_preact_relu[0][0]
-----
conv5_block2_1_bn (BatchNormali (None, 4, 4, 512)    2048
conv5_block2_1_conv[0][0]
-----
conv5_block2_1_relu (Activation (None, 4, 4, 512)    0
conv5_block2_1_bn[0][0]
-----
conv5_block2_2_pad (ZeroPadding (None, 6, 6, 512)    0
conv5_block2_1_relu[0][0]
-----
conv5_block2_2_conv (Conv2D)      (None, 4, 4, 512)    2359296
conv5_block2_2_pad[0][0]
-----
conv5_block2_2_bn (BatchNormali (None, 4, 4, 512)    2048
conv5_block2_2_conv[0][0]
-----
conv5_block2_2_relu (Activation (None, 4, 4, 512)    0
conv5_block2_2_bn[0][0]
-----
conv5_block2_3_conv (Conv2D)      (None, 4, 4, 2048)    1050624
conv5_block2_2_relu[0][0]
-----
conv5_block2_out (Add)            (None, 4, 4, 2048)    0
conv5_block1_out[0][0]
conv5_block2_3_conv[0][0]
-----
conv5_block3_preact_bn (BatchNo (None, 4, 4, 2048)    8192
conv5_block2_out[0][0]
-----
conv5_block3_preact_relu (Activ (None, 4, 4, 2048)    0

```

```

conv5_block3_preact_bn[0][0]
-----
-----
conv5_block3_1_conv (Conv2D)      (None, 4, 4, 512)      1048576
conv5_block3_preact_relu[0][0]
-----
-----
conv5_block3_1_bn (BatchNormali (None, 4, 4, 512)      2048
conv5_block3_1_conv[0][0]
-----
-----
conv5_block3_1_relu (Activation (None, 4, 4, 512)      0
conv5_block3_1_bn[0][0]
-----
-----
conv5_block3_2_pad (ZeroPadding (None, 6, 6, 512)      0
conv5_block3_1_relu[0][0]
-----
-----
conv5_block3_2_conv (Conv2D)      (None, 4, 4, 512)      2359296
conv5_block3_2_pad[0][0]
-----
-----
conv5_block3_2_bn (BatchNormali (None, 4, 4, 512)      2048
conv5_block3_2_conv[0][0]
-----
-----
conv5_block3_2_relu (Activation (None, 4, 4, 512)      0
conv5_block3_2_bn[0][0]
-----
-----
conv5_block3_3_conv (Conv2D)      (None, 4, 4, 2048)     1050624
conv5_block3_2_relu[0][0]
-----
-----
conv5_block3_out (Add)             (None, 4, 4, 2048)     0
conv5_block2_out[0][0]
conv5_block3_3_conv[0][0]
-----
-----
post_bn (BatchNormalization)      (None, 4, 4, 2048)     8192
conv5_block3_out[0][0]
-----
-----
post_relu (Activation)            (None, 4, 4, 2048)     0          post_bn[0][0]
-----
-----
max_pool (GlobalMaxPooling2D)     (None, 2048)           0          post_relu[0][0]

```



```

=====
Total params: 23,564,800
Trainable params: 0
Non-trainable params: 23,564,800
-----

```

```

[114]: model = Sequential()
model.add(base_model)
model.add(Flatten())

# Add the fully connected layers
model.add(Dense(128, activation = "relu"))
model.add(Dropout(0.4)) # regularization
model.add(Dense(64, activation = "relu"))
model.add(Dropout(0.4)) # regularization
model.add(Dense(1, activation = "sigmoid"))

model.summary()
model.save("RESNET50V2");

```

Model: "sequential\_12"

Layer (type)	Output Shape	Param #
resnet50v2 (Functional)	(None, 2048)	23564800
flatten_10 (Flatten)	(None, 2048)	0
dense_36 (Dense)	(None, 128)	262272
dropout_26 (Dropout)	(None, 128)	0
dense_37 (Dense)	(None, 64)	8256
dropout_27 (Dropout)	(None, 64)	0
dense_38 (Dense)	(None, 1)	65

```

=====
Total params: 23,835,393
Trainable params: 270,593
Non-trainable params: 23,564,800
-----
INFO:tensorflow:Assets written to: RESNET50V2/assets

```

```
[116]: # Compile model
model.compile(optimizer = "adam", loss = "binary_crossentropy", metrics =_
↳ ["accuracy"])
```

```
[119]: # Fit model
RESNET50V2 = model.fit(train_image, y_train, epochs = 100, batch_size=31,
                        validation_data=(val_image, y_val),
                        callbacks = [early_stop])
```

Epoch 1/100

153/153 [=====] - 81s 531ms/step - loss: 0.1940 -  
accuracy: 0.9188 - val\_loss: 0.1557 - val\_accuracy: 0.9487

Epoch 2/100

153/153 [=====] - 82s 534ms/step - loss: 0.1647 -  
accuracy: 0.9346 - val\_loss: 0.1468 - val\_accuracy: 0.9468

Epoch 3/100

153/153 [=====] - 82s 539ms/step - loss: 0.1474 -  
accuracy: 0.9435 - val\_loss: 0.1432 - val\_accuracy: 0.9582

Epoch 4/100

153/153 [=====] - 82s 536ms/step - loss: 0.1232 -  
accuracy: 0.9473 - val\_loss: 0.1405 - val\_accuracy: 0.9544

Epoch 5/100

153/153 [=====] - 82s 536ms/step - loss: 0.1100 -  
accuracy: 0.9542 - val\_loss: 0.1601 - val\_accuracy: 0.9582

Epoch 6/100

153/153 [=====] - 83s 541ms/step - loss: 0.1095 -  
accuracy: 0.9585 - val\_loss: 0.1600 - val\_accuracy: 0.9449

Epoch 7/100

153/153 [=====] - 82s 539ms/step - loss: 0.1060 -  
accuracy: 0.9608 - val\_loss: 0.1645 - val\_accuracy: 0.9544

Epoch 8/100

153/153 [=====] - 83s 541ms/step - loss: 0.1091 -  
accuracy: 0.9595 - val\_loss: 0.1431 - val\_accuracy: 0.9620

Epoch 9/100

153/153 [=====] - 84s 547ms/step - loss: 0.0916 -  
accuracy: 0.9593 - val\_loss: 0.1656 - val\_accuracy: 0.9563

Epoch 10/100

153/153 [=====] - 82s 538ms/step - loss: 0.0923 -  
accuracy: 0.9597 - val\_loss: 0.1581 - val\_accuracy: 0.9601

Epoch 11/100

153/153 [=====] - 82s 539ms/step - loss: 0.0928 -  
accuracy: 0.9646 - val\_loss: 0.1425 - val\_accuracy: 0.9544

Epoch 12/100

153/153 [=====] - 83s 541ms/step - loss: 0.0858 -  
accuracy: 0.9633 - val\_loss: 0.1765 - val\_accuracy: 0.9563

Epoch 13/100

153/153 [=====] - 84s 546ms/step - loss: 0.0716 -

```

accuracy: 0.9669 - val_loss: 0.1581 - val_accuracy: 0.9525
Epoch 14/100
153/153 [=====] - 85s 552ms/step - loss: 0.0893 -
accuracy: 0.9616 - val_loss: 0.1810 - val_accuracy: 0.9525

```

```
[120]: visualize_model_performance(RESNET50V2, train_image, test_image, "accuracy",
    ↪"val_accuracy")
```

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.92	0.92	159
1	0.97	0.97	0.97	428
accuracy			0.96	587
macro avg	0.95	0.95	0.95	587
weighted avg	0.96	0.96	0.96	587

```

149/149 [=====] - 70s 472ms/step - loss: 0.0731 -
accuracy: 0.9717
19/19 [=====] - 8s 445ms/step - loss: 0.1048 -
accuracy: 0.9591

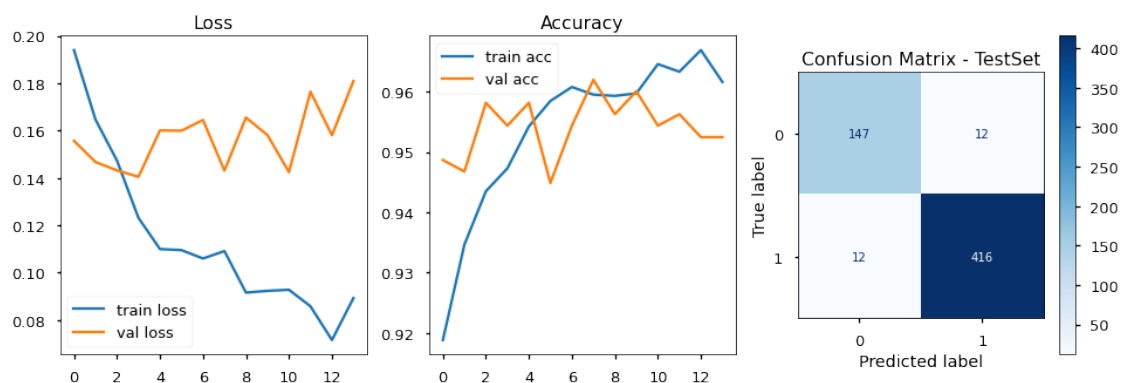
```

-----  
Final Train Loss: 0.0731

Final Test Loss: 0.1048  
-----

Final Train Acc: 0.9717

Final Test Acc: 0.9591



- Using transfet learning with RESNET5V2 we get similarly good results.

- Overall accuracy is 95.9% on the test set.
- Recall for pneumonia is high with 97%. Only 12 out of 428 pneumonia cases are mislabeled as normal.
- Recall for normal is at 92%. 12 out of 159 was mislabeled.

## 17 Best Performing Model?

```
[121]: train_acc_LowLearnRate_CNN = np.round(LowLearnRate_CNN.model.
        ↪evaluate(train_image, y_train)[1],3)
val_acc_LowLearnRate_CNN = np.round(LowLearnRate_CNN.model.evaluate(val_image,
        ↪y_val)[1],3)

train_acc_VGG16 = np.round(VGG16.model.evaluate(train_image, y_train)[1],3)
val_acc_VGG16 = np.round(VGG16.model.evaluate(val_image, y_val)[1],3)

train_acc_RESNET50V2 = np.round(RESNET50V2.model.evaluate(train_image,
        ↪y_train)[1],3)
val_acc_RESNET50V2 = np.round(RESNET50V2.model.evaluate(val_image, y_val)[1],3)
```

```
149/149 [=====] - 0s 3ms/step - loss: 0.0881 -
accuracy: 0.9656
17/17 [=====] - 0s 4ms/step - loss: 0.1431 - accuracy:
0.9506
149/149 [=====] - 9s 61ms/step - loss: 0.0757 - acc:
0.9743
17/17 [=====] - 1s 59ms/step - loss: 0.1042 - acc:
0.9620
149/149 [=====] - 202s 1s/step - loss: 0.0379 -
accuracy: 0.9871
17/17 [=====] - 21s 1s/step - loss: 0.0933 - accuracy:
0.9620
149/149 [=====] - 80s 535ms/step - loss: 0.0731 -
accuracy: 0.9717
17/17 [=====] - 8s 453ms/step - loss: 0.1405 -
accuracy: 0.9544
```

```
[124]: with plt.style.context('seaborn-talk'):
        fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(14,4))

        ax1.plot(LowLearnRate_CNN.history['acc'])
        ax1.plot(LowLearnRate_CNN.history['val_acc'])
        ax1.set_title('CNN Accuracy')
        ax1.legend(labels = [f'train_acc: {train_acc_LowLearnRate_CNN}', f'val_acc:
        ↪{val_acc_LowLearnRate_CNN}'])
        ax1.set_ylim([0.80, 1])
        ax1.set_xlim([0, 30])
```

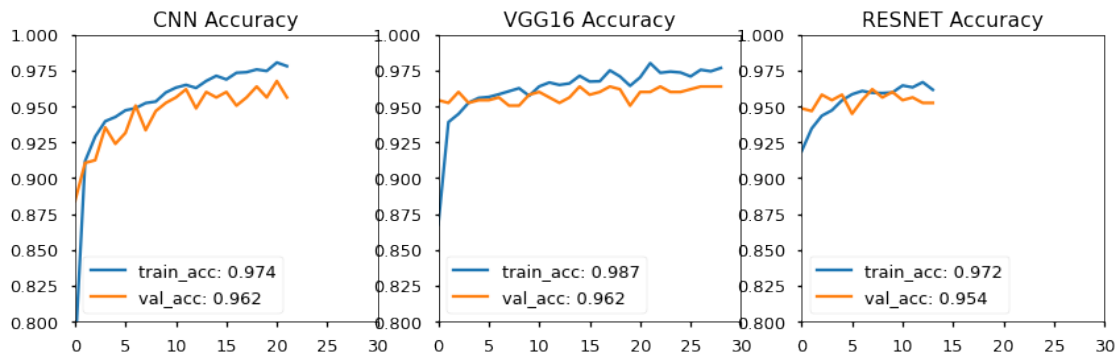
```

ax2.plot(VGG16.history['accuracy'])
ax2.plot(VGG16.history['val_accuracy'])
ax2.set_title('VGG16 Accuracy')
ax2.legend(labels = [f'train_acc: {train_acc_VGG16}', f'val_acc:␣
↪{val_acc_VGG16}'])
ax2.set_ylim([0.80, 1])
ax2.set_xlim([0, 30])

ax3.plot(RESNET50V2.history['accuracy'])
ax3.plot(RESNET50V2.history['val_accuracy'])
ax3.set_title('RESNET Accuracy')
ax3.legend(labels = [f'train_acc: {train_acc_RESNET50V2}', f'val_acc:␣
↪{val_acc_RESNET50V2}'])
ax3.set_ylim([0.80, 1])
ax3.set_xlim([0, 30])

plt.savefig('./images/CompareModels_train_val_acc', dpi=300,␣
↪bbox_inches='tight')

```



- All CNN models performed similarly well with no apparent signs of overfitting as seen by the loss and accuracy trends for the training and validation sets

```

[125]: with plt.style.context('seaborn-talk'):

fig, axs = plt.subplots(1, 3, figsize=(14,5))
fig.suptitle("Confusion matrix for best performing models \n", fontsize=18)

for ax, result, modelname in zip(axs.
↪ravel(), [LowLearnRate_CNN, VGG16, RESNET50V2], ['CNN', 'VGG16', 'RESNET']):
    y_hat_test = result.model.predict(test_image)
    y_pred = np rint(y_hat_test).astype(np.int)
    y_true = y_test.astype(np.int)
    cm = confusion_matrix(y_true, y_pred) # normalize = 'true'

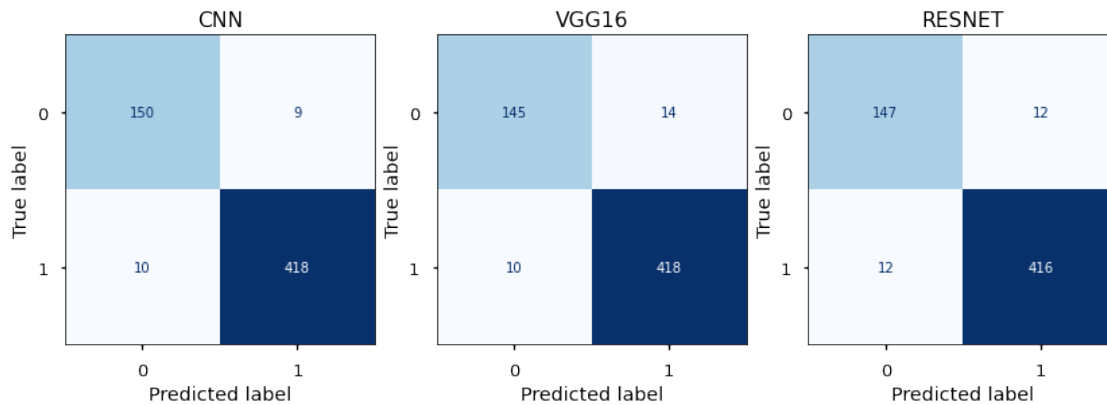
```

```

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap = "Blues", ax=ax, colorbar =False)
ax.set_title(modelname)
plt.savefig('./images/CompareModels_ConfusionMatrices', dpi=300,
           bbox_inches='tight')

```

Confusion matrix for best performing models



- All models reached overall accuracy levels of 95-96% and recall values of 97-98% for the pneumonia cases for the tets set.
- **CNN with dropout regularization and a lower learning rate** was chosen as the final model since it gave the best performance on test dataset by missing only 10 pneumonia-positive cases out of 428, and 9 out of 159 normal cases. It is also simpler than VGG or ResNet models.
- Using tuned CNN results on the test set were:
  - overall accuracy score of 97%,
  - recall score of 94% for class normal, and 98% for class pneumonia
  - f1 score of 94% for class normal, and 98% for class pneumonia

## 18 Visualize Intermediate Activations:

```

[91]: best_model = keras.models.load_model("Dropout_CNN")
      best_model.summary()

```

WARNING:tensorflow:No training configuration found in save file, so the model was \*not\* compiled. Compile it manually.  
Model: "sequential\_7"

Layer (type)	Output Shape	Param #
=====		

conv2d_14 (Conv2D)	(None, 126, 126, 32)	896
-----		
max_pooling2d_14 (MaxPooling)	(None, 63, 63, 32)	0
-----		
dropout_10 (Dropout)	(None, 63, 63, 32)	0
-----		
conv2d_15 (Conv2D)	(None, 61, 61, 64)	18496
-----		
max_pooling2d_15 (MaxPooling)	(None, 30, 30, 64)	0
-----		
dropout_11 (Dropout)	(None, 30, 30, 64)	0
-----		
conv2d_16 (Conv2D)	(None, 28, 28, 128)	73856
-----		
max_pooling2d_16 (MaxPooling)	(None, 14, 14, 128)	0
-----		
dropout_12 (Dropout)	(None, 14, 14, 128)	0
-----		
flatten_5 (Flatten)	(None, 25088)	0
-----		
dense_21 (Dense)	(None, 128)	3211392
-----		
dropout_13 (Dropout)	(None, 128)	0
-----		
dense_22 (Dense)	(None, 64)	8256
-----		
dropout_14 (Dropout)	(None, 64)	0
-----		
dense_23 (Dense)	(None, 1)	65
=====		
Total params: 3,312,961		
Trainable params: 3,312,961		
Non-trainable params: 0		
-----		

### 18.0.1 Load a sample Image, Transform the Image to a Tensor and Visualize:

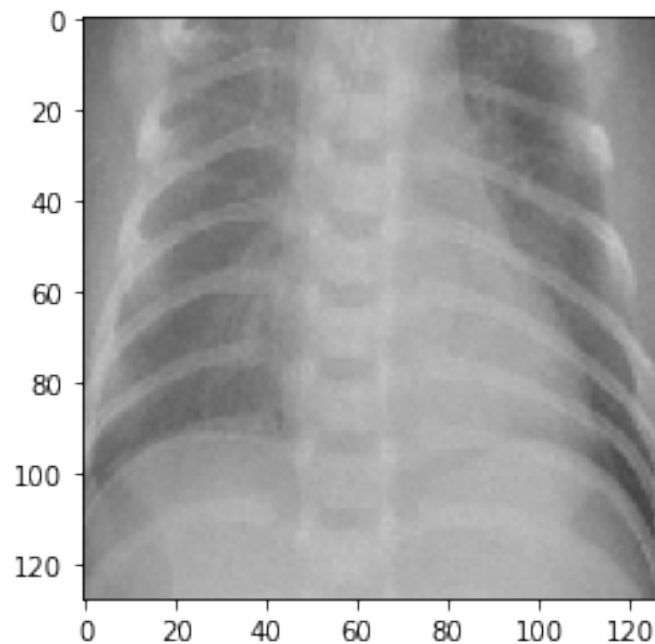
```
[92]: os.listdir("Data/OUTPUT/train/PNEUMONIA")[0:15]
```

```
[92]: ['person63_bacteria_306.jpeg',
      'person1438_bacteria_3721.jpeg',
      'person478_virus_975.jpeg',
      'person661_bacteria_2553.jpeg',
      'person1214_bacteria_3166.jpeg',
      'person1353_virus_2333.jpeg',
      'person26_bacteria_122.jpeg',
      'person1619_bacteria_4261.jpeg',
```

```
'person890_bacteria_2814.jpeg',  
'person147_bacteria_706.jpeg',  
'person1491_bacteria_3893.jpeg',  
'person69_bacteria_338.jpeg',  
'person100_bacteria_482.jpeg',  
'person321_bacteria_1489.jpeg',  
'person281_bacteria_1329.jpeg']
```

```
[93]: filename = 'Data/chest_xray/train/PNEUMONIA/person69_bacteria_338.jpeg'  
img = image.load_img(filename, target_size=(128, 128))  
img_tensor = image.img_to_array(img)  
img_tensor = np.expand_dims(img_tensor, axis=0)  
img_tensor /= 255.  
  
# Check tensor shape  
print(img_tensor.shape)  
  
# Preview the image  
plt.imshow(img_tensor[0])  
plt.show()
```

```
(1, 128, 128, 3)
```





### 18.0.2 Plot Feature Maps:

- From the model summary below we can see that the initial three layers output feature maps that have 32 channels each.
- Subsequently, the next three layers have 64 channels each and ext three layers have 128 channels each.
- As we move deeper into the layers of the CNN we can detect successively more abstract patterns.

Specifically, the models are comprised of small linear filters and the result of applying filters called activation maps, or more generally, feature maps.

Both filters and feature maps can be visualized

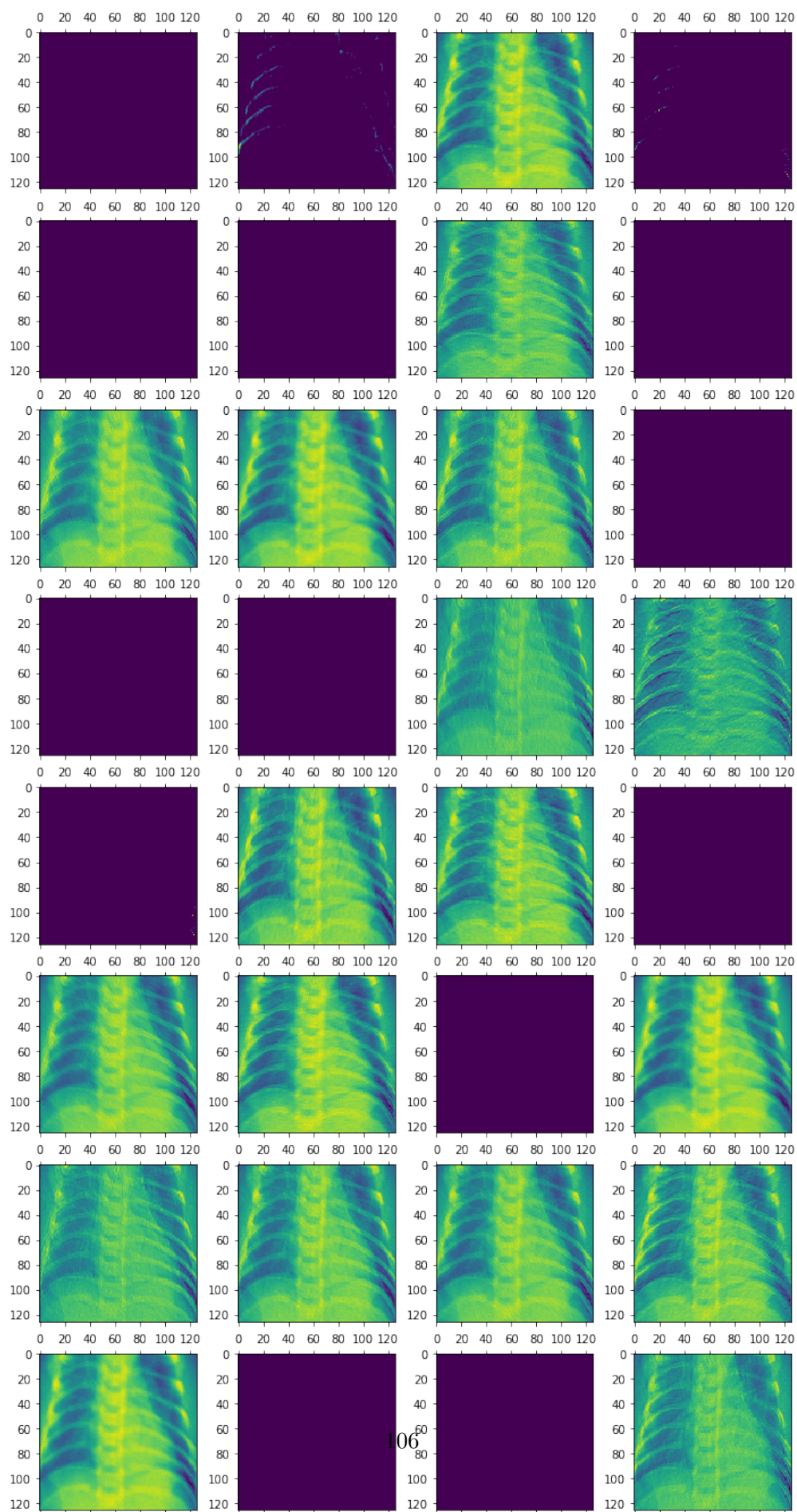
### 18.0.3 Visualize all 32 of the channels from the first activation function.

- Extract the feature maps, or layer outputs from each of the activation functions in the model.
- Generate models that transform the image from its raw state to these feature map.
- Take these transformations and visualize each channel for each feature map.

he idea of visualizing a feature map for a specific input image would be to understand what features of the input are detected or preserved in the feature maps.

```
[94]: # xtract the feature maps, or layer outputs from each of the activation ↵  
      ↵functions in the model.  
layer_outputs = [layer.output for layer in best_model.layers[:8]]  
  
# Generate models that transform the image from its raw state to these feature ↵  
      ↵map  
activation_model = models.Model(inputs=best_model.input, outputs=layer_outputs)  
  
# Returns an array for each activation layer  
activations = activation_model.predict(img_tensor)
```

```
[95]: # Visualize all 32 of the channels from the first activation function:  
  
fig, axes = plt.subplots(8, 4, figsize=(12,24))  
for i in range(32):  
    row = i//4  
    column = i%4  
    ax = axes[row, column]  
    first_layer_activation = activations[0] # first activation channel  
    ax.matshow(first_layer_activation[0, :, :, i], cmap='viridis')  
# In the case of the blank images displayed, this indicates that the patterns ↵  
    ↵were not present in the current image.
```



#### 18.0.4 Visualize a single channel for each of the activation layers:

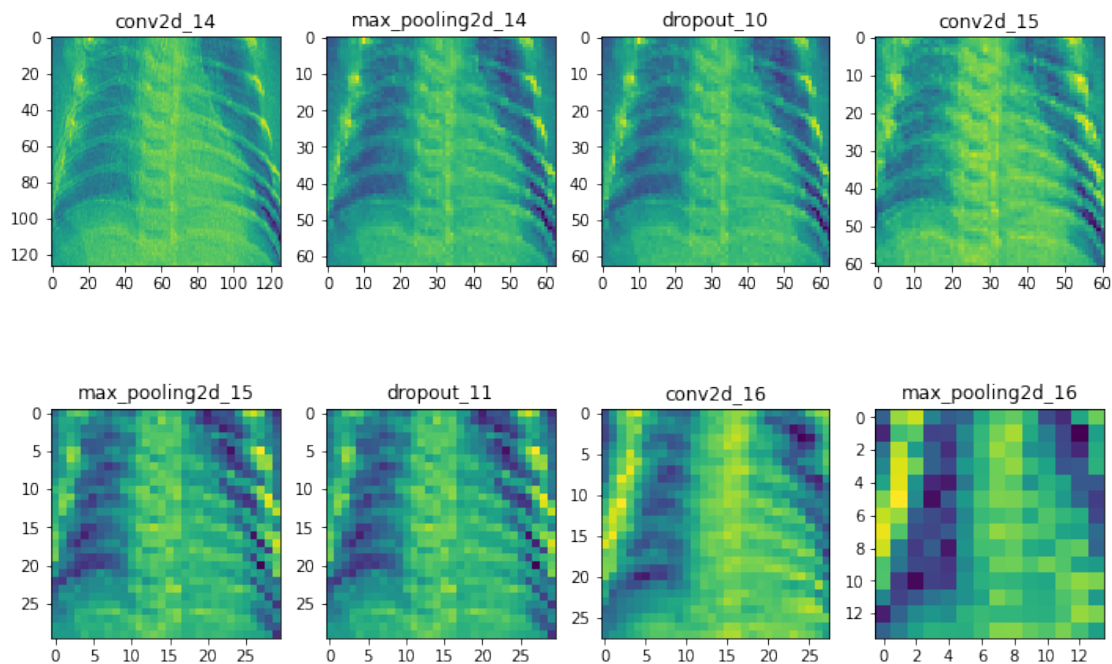
- Below is the visualization of the 25th channel for each of the activation layers. (Recall that there are more channels in later layers.)

```
[106]: fig, axes = plt.subplots(2,4, figsize=(12,8))

layer_names = []
for layer in best_model.layers[:8]: # layers until the 8th layer
    layer_names.append(layer.name)

for i in range(8):
    row = i//4
    column = i%4
    ax = axes[row, column]
    cur_layer = activations[i]
    ax.matshow(cur_layer[0, :, :, 24], cmap='viridis')
    ax.xaxis.set_ticks_position('bottom')
    ax.set_title(layer_names[i])

plt.savefig('./images/ActivationChannels', dpi=300, bbox_inches='tight')
```



- Layer layers are more abstract representations. This demonstrates how the representations

learned by CNN architectures become increasingly abstract with the depth of the layers.

The expectation would be that the feature maps close to the input detect small or fine-grained detail, whereas feature maps close to the output of the model capture more general features.

### 18.0.5 Visualize each of the channels for each of feature maps of the convolutional layers:

- Code taken from <https://github.com/learn-co-curriculum/dsc-visualizing-activation-functions-lab/tree/solution>

```
[108]: from keras import models
import math

# Extract model layer outputs
layer_outputs = [layer.output for layer in best_model.layers[:8]]

# Create a model for displaying the feature maps
activation_model = models.Model(inputs=best_model.input, outputs=layer_outputs)

activations = activation_model.predict(img_tensor)

# Extract Layer Names for Labelling
layer_names = []
for layer in best_model.layers[:8]:
    layer_names.append(layer.name)

total_features = sum([a.shape[-1] for a in activations])
total_features

n_cols = 16
n_rows = math.ceil(total_features / n_cols)

iteration = 0
fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(n_cols, n_rows*1.
↪5))

for layer_n, layer_activation in enumerate(activations):
    n_channels = layer_activation.shape[-1]
    for ch_idx in range(n_channels):
        row = iteration // n_cols
        column = iteration % n_cols

        ax = axes[row, column]

        channel_image = layer_activation[0,
                                         :, :,
```

```

                                ch_idx]
    # Post-process the feature to make it visually palatable
    channel_image -= channel_image.mean()
    channel_image /= channel_image.std()
    channel_image *= 64
    channel_image += 128
    channel_image = np.clip(channel_image, 0, 255).astype('uint8')

    ax.imshow(channel_image, aspect='auto', cmap='viridis')
    ax.get_xaxis().set_ticks([])
    ax.get_yaxis().set_ticks([])

    if ch_idx == 0:
        ax.set_title(layer_names[layer_n], fontsize=10)
        iteration += 1

fig.subplots_adjust(hspace=1.25)
plt.show()

```



## 18.1 Recommendations

- Neural network may be used to aid the healthcare professional in stream-lining the diagnosing process when classifying x-ray images. This may allow for a quicker return time and greater patient satisfaction.
- Catching as many people with pneumonia as possible is particularly important for early intervention. Use of ANN for image classification might lead to more positive outcomes because the positive patients can begin treatment right away as opposed to waiting lengthy periods of time until they hear from the readings of the radiologist and interpretation of the doctor.
- Such process would also reduce any radiologist's/doctor's stress at having to look through a great deal of images. They could instead use their time to more rigorously go over the images that fall more into the grey zone based on the model predictions.
- Less time the doctors expend looking at images to arrive at a diagnosis, the more time they can allocate to dealing with more demanding and complex procedures.

## 18.2 Limitations and Next Steps

- We can use data augmentation methods to increase the size of the training data set which could improve model performance on unseen data.
- We can crop the images to exclude the electrodes and the R script from the display which negatively affect the image processing algorithm.
- We could address the class imbalance issue using oversampling techniques which could again improve performance.

**Exporting to PDF using nbconvert:** 1. install nbconvert: ! pip install nbconvert 2. install MacTeX from [tps://tug.org/mactex/](https://tug.org/mactex/) 3. ! export PATH=/Library/TeX/texbin:\$PATH 4. ! jupyter nbconvert --to PDF NOTEBOOKNAME.ipynb

```
[141]: ! export PATH=/Library/TeX/texbin:$PATH
```

```
[142]: ! jupyter nbconvert --to PDF NOTEBOOKNAME.ipynb
```

```
[NbConvertApp] WARNING | pattern 'NOTEBOOKNAME.ipynb' matched no files
This application is used to convert notebook files (*.ipynb) to various other
formats.
```

```
WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.
```

Options

=====

The options below are convenience aliases to configurable class-options, as listed in the "Equivalent to" description-line of the aliases.

To see all configurable class-options for some <cmd>, use:

```
<cmd> --help-all
```

```

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
    include the error message in the cell output (the default behaviour is to abort
    conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
    default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False]
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False]
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True]
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True]
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the

```



system.

- Equivalent to: [--WebPDFExporter.allow\_chromium\_download=True]
- disable-chromium-sandbox
  - Disable chromium security sandbox when converting to PDF..
  - Equivalent to: [--WebPDFExporter.disable\_sandbox=True]
- show-input
  - Shows code input. This flag is only useful for dejavu users.
  - Equivalent to: [--TemplateExporter.exclude\_input=False]
- embed-images
  - Embed the images as base64 dataurls in the output. This flag is only useful for the HTML/WebPDF/Slides exports.
  - Equivalent to: [--HTMLExporter.embed\_images=True]
- log-level=<Enum>
  - Set the log level by value or name.
  - Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL']
  - Default: 30
  - Equivalent to: [--Application.log\_level]
- config=<Unicode>
  - Full path of a config file.
  - Default: ''
  - Equivalent to: [--JupyterApp.config\_file]
- to=<Unicode>
  - The export format to be used, either one of the built-in formats ['asciidoc', 'custom', 'html', 'html\_ch', 'html\_embed', 'html\_toc', 'html\_with\_lenvs', 'html\_with\_toclenvs', 'latex', 'latex\_with\_lenvs', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'selectLanguage', 'slides', 'slides\_with\_lenvs', 'webpdf'] or a dotted object name that represents the import path for an ``Exporter`` class
  - Default: ''
  - Equivalent to: [--NbConvertApp.export\_format]
- template=<Unicode>
  - Name of the template to use
  - Default: ''
  - Equivalent to: [--TemplateExporter.template\_name]
- template-file=<Unicode>
  - Name of the template file to use
  - Default: None
  - Equivalent to: [--TemplateExporter.template\_file]
- theme=<Unicode>
  - Template specific theme(e.g. the name of a JupyterLab CSS theme distributed as prebuilt extension for the lab template)
  - Default: 'light'
  - Equivalent to: [--HTMLExporter.theme]
- writer=<DottedObjectName>
  - Writer class used to write the results of the conversion
  - Default: 'FilesWriter'
  - Equivalent to: [--NbConvertApp.writer\_class]

```

--post=<DottedOrNone>
    PostProcessor class used to write the results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files. can only be used when converting
    one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults to output to the directory of each
    notebook. To recover previous default behaviour (outputting to the current
    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x). This defaults to the reveal CDN,
    but can be any url pointing to a copy of reveal.js.
    For speaker notes to work, this must be a relative path to a local copy of
    reveal.js: e.g., "reveal.js".
    If a relative path is given, it must be a subdirectory of the current
    directory (from which the server is run).
    See the usage documentation
    (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-slideshow)
    for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write. Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

```

## Examples

-----

The simplest way to use nbconvert is

```
> jupyter nbconvert mynotebook.ipynb --to html
```

Options include ['asciidoc', 'custom', 'html', 'html\_ch', 'html\_embed', 'html\_toc', 'html\_with\_lenvs', 'html\_with\_toclenvs', 'latex', 'latex\_with\_lenvs', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'selectLanguage', 'slides', 'slides\_with\_lenvs', 'webpdf'].

```
> jupyter nbconvert --to latex mynotebook.ipynb
```

Both HTML and LaTeX support multiple output templates. LaTeX includes

'base', 'article' and 'report'. HTML includes 'basic', 'lab' and 'classic'. You can specify the flavor of the format used.

```
> jupyter nbconvert --to html --template lab mynotebook.ipynb
```

You can also pipe the output to stdout, rather than a file

```
> jupyter nbconvert mynotebook.ipynb --stdout
```

PDF is generated via latex

```
> jupyter nbconvert mynotebook.ipynb --to pdf
```

You can get (and serve) a Reveal.js-powered slideshow

```
> jupyter nbconvert myslides.ipynb --to slides --post serve
```

Multiple notebooks can be given at the command line in a couple of different ways:

```
> jupyter nbconvert notebook*.ipynb
> jupyter nbconvert notebook1.ipynb notebook2.ipynb
```

or you can specify the notebooks list in a config file, containing::

```
c.NbConvertApp.notebooks = ["my_notebook.ipynb"]
```

```
> jupyter nbconvert --config mycfg.py
```

To see all available configurables, use `--help-all`.

[ ]: