Image Classification with Deep Learning-Project#4

- Student name: ***Deepali Sharma***
- Student pace: ***Flex***
- Scheduled project review date/time: ***April, 2023***
- Instructor name: ***Abhineet Kulkarni***

Business Problem

- A private hospital in remote location has only one on-site Radiologist. Instead of hiring another radiologist that will cause them significantly more money (also difficulty convincing doctors to come to this location), they want to hire a data scientist who can develop a machine learning model that can accurately detect pneumonia from chest X-ray images, which could ultimately help the Radiologist diagnose and treat patients more quickly and effectively. This way then can have a quick turn-around even when Radiologist is out sick or on vacations!

Analysis Approach

- This is an image classification problem that we will tackle using different neural networks and pre-trained modules. This analysis used followimg models:
  - Artificial Neural Networks, also known as Neural Nets)(ANN or NN).
  - Convolutional Neural Networks (CNNs)
  - Pre-trained Modules: We will use Xception and RESNET101
- We will implement various techniques to improve model performance and avoid overfitting such as Dropout, L2 Regularization and varying learning rate for ANNs and CNNs. We also use data augmentation to train our models on more data with various modifications
- We will use confusion matrix as the performance metric. In particular we want to minimize the false negatives for pneumonia cases as we dont want patients with pneumonia to be mis-diagonsed.

Executive Summary

- The **CNN model with deeper layers** and trained on augmented data performed the best. The recall score for the pneumonia cases was 99% and for normal cases was 76%
- The second best model was the pre-trained model **Xception** with additional layers added to it. The recall score for pneumonia cases was 96% and for normal cases was 81%

In [1]:
```
#import system related libs
import os, sys, shutil,time
print(sys.executable)

# import basic libs
import pandas as pd
import numpy as np
import random
import math
import datetime

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

#import sklearn libs
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from sklearn import preprocessing
#from sklearn.metrics import classification_report, accuracy_score, co
#from sklearn.metrics import roc_curve, auc
#from sklearn.metrics import plot_confusion_matrix # plot_confusion_ma
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import RocCurveDisplay, roc_curve, roc_auc_score,

#import NN/Keras related libs

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
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

#import warnings
import warnings
warnings.filterwarnings('ignore')
```

```
/usr/local/anaconda3/bin/python

Using TensorFlow backend.
```

In [2]:
```
original_start = datetime.datetime.now()
start = datetime.datetime.now()
```

Preprocessing Data

- Count the number of files available
- Look at the images before doing any data processing

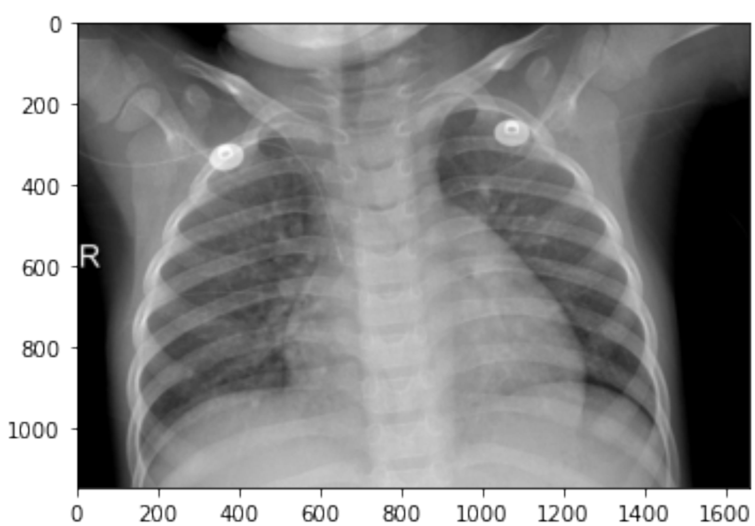
In [3]:
```
train_dir_pneum  = 'train/PNEUMONIA'
train_dir_normal = 'train/NORMAL'
test_dir_pneum   = 'test/PNEUMONIA'
test_dir_normal  = 'test/NORMAL'
total_pneum  = len(os.listdir(train_dir_pneum)) + len(os.listdir(test_
total_normal = len(os.listdir(train_dir_normal)) + len(os.listdir(test
total_images = total_pneum+total_normal

print('Train Dataset:')
print('There are', len(os.listdir(train_dir_pneum)),
      'Pneumonia images(' ,round(len(os.listdir(train_dir_pneum))/tota
print('There are', len(os.listdir(train_dir_normal)),
      'Normal images(' ,round(len(os.listdir(train_dir_normal))/total_
print('\n\nTest Dataset:')
print('There are', len(os.listdir(test_dir_pneum)),
      'Pneumonia images(' ,round(len(os.listdir(test_dir_pneum))/total
print('There are', len(os.listdir(test_dir_normal)),
      'Normal images(' ,round(len(os.listdir(test_dir_normal))/total_i
```

```
Train Dataset:
There are 3883 Pneumonia images( 0.66 )
There are 1349 Normal images( 0.23 )


Test Dataset:
There are 390 Pneumonia images( 0.07 )
There are 234 Normal images( 0.04 )
```

- There is roughly **89%** data in train set and **10%** in test dataset
- Lets just randomly pick one image and look at its dimensions, pixels and color information

In [5]:
```
#generate random number between 0 to 3800
img_num = np.random.randint(3800)
print(f"Image number displayed will be: {img_num}")
img_path = os.listdir(train_dir_pneum)[img_num]


img = image.load_img(os.path.join(train_dir_pneum, img_path))#, target
img_tensor = image.img_to_array(img)

print(f"Image Shape: {img_tensor.shape}")  # width and height
print(f"Max pixel: {img_tensor.max()}")
print(f"Min pixel: {img_tensor.min()}")
#print(f"Image: {img_tensor}")

# Display the image
plt.imshow(img, cmap='gray'); # plt.imshow(img_array.astype('uint8'))
```

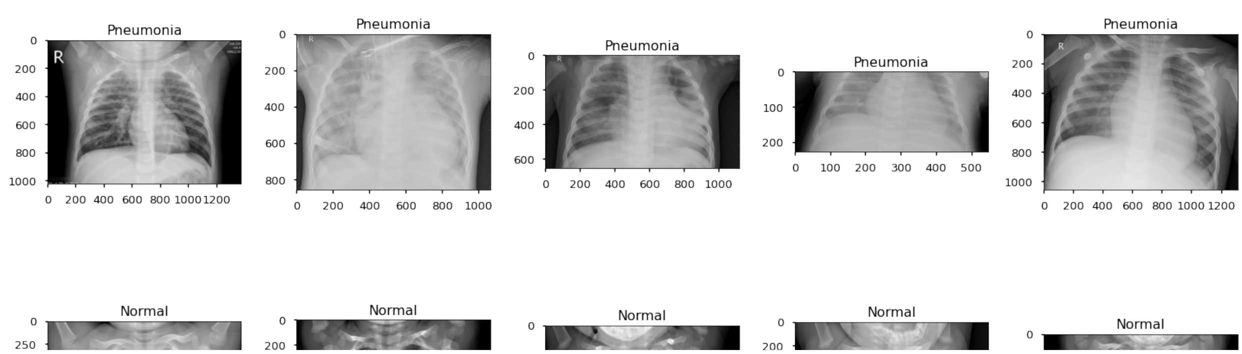
```
Image number displayed will be: 1875
Image Shape: (1145, 1658, 3)
Max pixel: 255.0
Min pixel: 0.0
```



- Lets look at a bunch of images from pneumonia and normal classes. We will pick 5 images from each class randomly

In [8]:

```
max_pixel_size =[]
min_pixel_size =[]
img_height     =[]
img_width      =[]
with plt.style.context('seaborn-talk'):
    fig, ax = plt.subplots(nrows =2, ncols = 5, figsize=(20,10))

    plt_img_pneum = np.random.randint(3883, size=5)
    plt_img_normal = np.random.randint(1349, size=5)
    for i in range(5):
        # Combine the image directory with the specific jpeg to be abl
        # Read the image into an array.
        img_path      = os.listdir(train_dir_pneum)[plt_img_pneum[i]]
        img_pneum     = image.load_img(os.path.join(train_dir_pneum, i

        img_path      = os.listdir(train_dir_normal)[plt_img_normal[i]
        img_normal    = image.load_img(os.path.join(train_dir_normal,

        img_tensor_pneum = image.img_to_array(img_pneum)
        img_tensor_normal= image.img_to_array(img_normal)
        max_pixel_size.append(img_tensor_pneum.max())
        max_pixel_size.append(img_tensor_normal.max())
        min_pixel_size.append(img_tensor_pneum.min())
        min_pixel_size.append(img_tensor_normal.min())
        img_width.append(img_tensor_pneum.shape[1])
        img_width.append(img_tensor_normal.shape[1])
        img_height.append(img_tensor_pneum.shape[0])
        img_height.append(img_tensor_normal.shape[0])
        # Display the image
        ax[0,i].imshow(img_pneum, cmap = 'gray')
        ax[1,i].imshow(img_normal, cmap = 'gray')

        # 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')
```



In [7]:

```
print(f'maximum pixel size array: {max_pixel_size}')
print(f'minimum pixel size array: {min_pixel_size}')
print(f'Image dimensions (width): {img_width}')
print(f'Image dimensions (height): {img_height}')
```

```
maximum pixel size array: [255.0, 255.0, 255.0, 255.0, 255.0, 255.0, 
255.0, 255.0, 255.0, 255.0]
minimum pixel size array: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0]
Image dimensions (width): [1152, 2458, 952, 2101, 1000, 1454, 1416, 2
088, 1568, 1836]
Image dimensions (height): [832, 1870, 528, 1606, 624, 978, 912, 2074
, 1192, 1719]
```

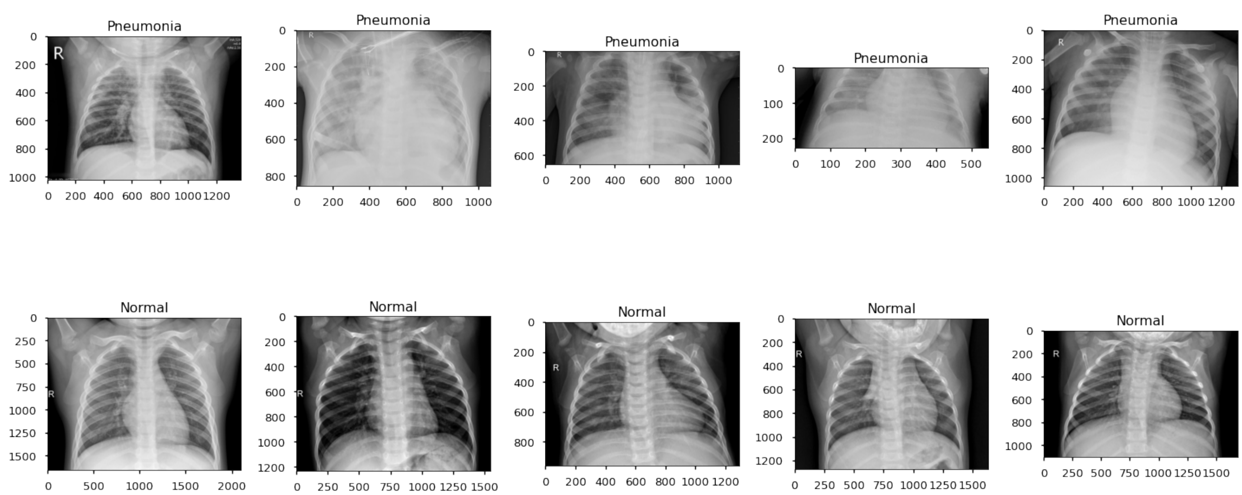
- Looking at these arraus for resolution, pixelization we conclude the following:
  - The images have *different dimesions (resolution)*
  - The pixelization for all of them ranges between 0 to 255

Transform the Image to a Tensor and Visualize Again

- We need to preprocess images into tensors wao as to use them for modeling using deep learning.
- Lets see now if rescaling affects the image quality since for modeling we will need to rescale the images. So we need to make sure that scaling doesnt result in any drastic changes.

In [9]:

```
with plt.style.context('seaborn-talk'):
    fig, ax = plt.subplots(nrows =2, ncols = 5, figsize=(20,10))
    for i in range(5):
        # Combine the image directory with the specific jpeg to be abl
        # Read the image into an array.
        img_path      = os.listdir(train_dir_pneum)[plt_img_pneum[i]]
        img_pneum     = image.load_img(os.path.join(train_dir_pneum, i

        img_path      = os.listdir(train_dir_normal)[plt_img_normal[i]
        img_normal    = image.load_img(os.path.join(train_dir_normal,

        img_tensor_pneum = image.img_to_array(img_pneum)
        img_tensor_normal= image.img_to_array(img_normal)

        img_tensor_pneum /= 255.
        img_tensor_normal /= 255.
        # Display the image
        ax[0,i].imshow(img_pneum, cmap = 'gray')
        ax[1,i].imshow(img_normal, cmap = 'gray')

        # 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()
```



- We dont see any issues with the images after scaling them down.

Splitting the train set into train and validation sets

- We use the splitfolders package (https://pypi.org/project/split-folders/ (https://pypi.org/project/split-folders/)) to achieve this
- The original train data provided by Kaggle "train" with "Pneumonia" and "Normal" subfolders was re-arranged into a new output folder "output" with "train" and "val" subfolders as well. The train foldeer contains 90% and validation folder containing 10% of the data (from original train dataset from Kaggle).
- This would leave us with 80% of total data for model training purposes, 10% for validation and 10% for test purposes (This folder is same as the original Test data provided by Kaggle).

In [113]:
```
#from PIL import Image
#import cv2
#!pip install split-folders
```

```
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
```

In [12]:
```
import splitfolders
```

In [13]:
```
# Split with a ratio.
# To only split into training and validation set, set a tuple to `rati
#splitfolders.ratio("./train", output="output",
#    seed=1337, ratio=(.9, .1), group_prefix=None, move=False) # defau
```

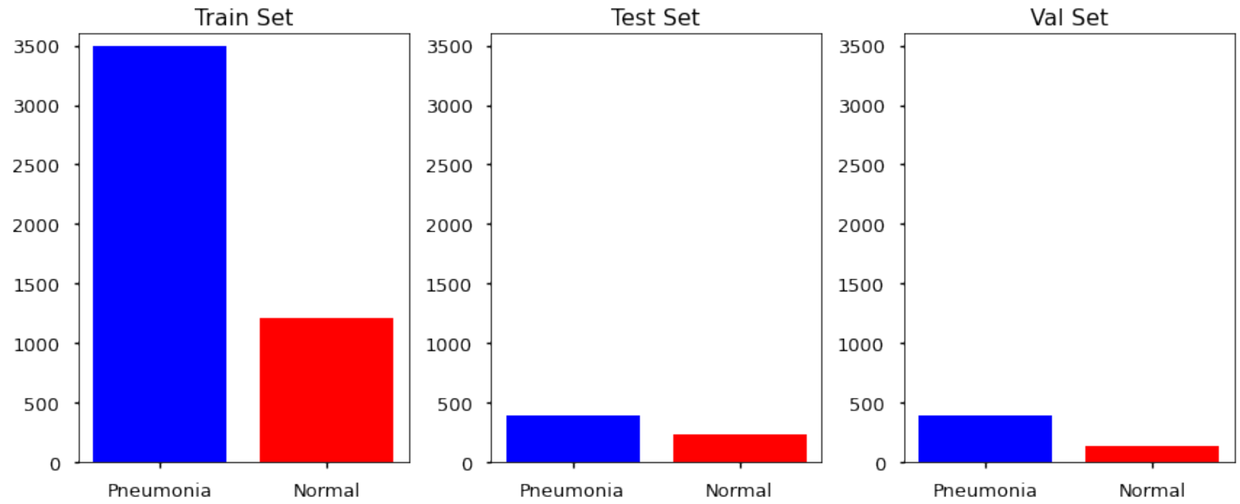
Lets check that we got the right fraction of data sets for train, test and val

In [14]:

```
num_pneum_train = (len(os.listdir("output/train/PNEUMONIA")) )
num_normal_train = (len(os.listdir("output/train/NORMAL")) )

num_pneum_test = (len(os.listdir("test/PNEUMONIA")) )
num_normal_test = (len(os.listdir("test/NORMAL")) )

num_pneum_val = (len(os.listdir("output/val/PNEUMONIA")) )
num_normal_val = (len(os.listdir("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_pneum_train, num_r
    ax1.set_title('Train Set')
    ax2.bar(x = ["Pneumonia","Normal"], height=[num_pneum_test, num_nc
    ax2.set_title('Test Set')
    ax3.bar(x = ["Pneumonia","Normal"], height=[num_pneum_val, num_nor
    ax3.set_title('Val Set')

    ax1.set_ylim([0, 3600])
    ax2.set_ylim([0, 3600])
    ax3.set_ylim([0, 3600])
    plt.tight_layout()

print(f"Train Pneumomia: {num_pneum_train}")
print(f"Train Normal: {num_normal_train}")
print("-------------------")
print(f"Test Pneumomia: {num_pneum_test}")
print(f"Test Normal: {num_normal_test}")
print("--------------------")
print(f"Val Pneumomia: {num_pneum_val}")
print(f"Val Normal: {num_normal_val}")
print("--------------------")

TrainTotal = num_pneum_train + num_normal_train
TestTotal = num_pneum_test + num_normal_test
ValTotal = num_pneum_val + num_normal_val
Total = TrainTotal + TestTotal + ValTotal
print(f"Train Images Percentage: {np.round((TrainTotal / Total),2) }")
print(f"Test Images Percentage: {np.round((TestTotal / Total),2) }")
print(f"Val Images Percentage: {np.round((ValTotal / Total),2) }")
```

```
Train Pneumomia: 3494
Train Normal: 1214
-------------------
Test Pneumomia: 390
Test Normal: 234
--------------------
Val Pneumomia: 389
Val Normal: 135
--------------------
```

```
Train Images Percentage: 0.8
Test Images Percentage: 0.11
Val Images Percentage: 0.09
```



In [15]:
```
print(f"Train Images Total#: {TrainTotal}")
print(f"Test Images Total#: {TestTotal}")
print(f"Val Images Total#: {ValTotal}")
```

```
Train Images Total#: 4708
Test Images Total#: 624
Val Images Total#: 524
```

Image preprocessing (Keras ImageDataGenerator)

This is an essential step in deep learning and computer vision tasks, such as object detection, image classification, and segmentation. We will do the following steps to prepare the images for modeling:

- Resizing and Rescaling: Images are often resized to a fixed input size, and their pixel values are rescaled to a common range. Rescaling the pixel values helps to normalize the input data and reduce the effects of lighting and contrast variations. Since all our images are of different sizes we will rescale (standardize) them using a target width and height. The resolutions for training CNNs usually range between 64 × 64 and 256 × 256. The analysis done with resolution 256x256 yielded lower perfomance for models, so I decided to use 128x128
- Normalization: Normalizing the pixel values of an image can help to reduce the effects of lighting and contrast variations. We will normalize the images by 255(the maximum pixel size in these images).
- Label the target data into 1's (pneumonia) and 0's (normal) # class_mode='binary'

In [16]:
```
train_folder = "output/train"
test_folder  = "test/"
val_folder   = "output/val"
```

In [17]:
```
IMG_SIZE=128#256
```

In [18]:
```
# get all the data in the directory test , and reshape them
test_generator = ImageDataGenerator(rescale=1./255).flow_from_director
        test_folder,
        target_size=(IMG_SIZE, IMG_SIZE),
        batch_size = TestTotal,
        class_mode='binary')

# get all the data in the directory split/validation , and reshape the
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory
        val_folder,
        target_size=(IMG_SIZE, IMG_SIZE),
        batch_size = ValTotal,
        class_mode='binary')

# get all the data in the directory split/train , and reshape them
train_generator = ImageDataGenerator(rescale=1./255).flow_from_directo
        train_folder,
        target_size=(IMG_SIZE, IMG_SIZE),
        batch_size=TrainTotal,
        class_mode='binary')
```

```
Found 624 images belonging to 2 classes.
Found 524 images belonging to 2 classes.
Found 4708 images belonging to 2 classes.
```

In [19]:
```
print(train_generator.class_indices)
print(train_generator.image_shape, test_generator.image_shape, val_ger
```

```
{'NORMAL': 0, 'PNEUMONIA': 1}
(128, 128, 3) (128, 128, 3) (128, 128, 3)
```

In [20]:
```
# create the data sets
## This will be used for CNN models as they need 3x3 input
# next() returns the next item in the iterator = The first batch of th
train_images, train_labels = next(train_generator)
test_images, test_labels = next(test_generator)
val_images, val_labels = next(val_generator)
```

In [21]:
```
# Explore your dataset again
m_train = train_images.shape[0]
num_px = train_images.shape[1]
m_test = test_images.shape[0]
m_val = val_images.shape[0]

print ("Number of training samples: " + str(m_train))
print ("Number of testing samples: " + str(m_test))
print ("Number of validation samples: " + str(m_val))
print ("train_images shape: " + str(train_images.shape))
print ("train_labels shape: " + str(train_labels.shape))
print ("test_images shape: " + str(test_images.shape))
print ("test_labels shape: " + str(test_labels.shape))
print ("val_images shape: " + str(val_images.shape))
print ("val_labels shape: " + str(val_labels.shape))
```

```
Number of training samples: 4708
Number of testing samples: 624
Number of validation samples: 524
train_images shape: (4708, 128, 128, 3)
train_labels shape: (4708,)
test_images shape: (624, 128, 128, 3)
test_labels shape: (624,)
val_images shape: (524, 128, 128, 3)
val_labels shape: (524,)
```

Modeling

1. ANN -Models:

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives signals then processes them and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold.Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times(https://en.wikipedia.org/wiki/Artificial_neural_network (https://en.wikipedia.org/wiki/Artificial_neural_network)).

1a. Baseline ANN Model

- We will use a densely connected network(ANN) as a baseline model with only one hidden layer with 10 neutrons, and an output layer with one output.
- We will use "Adam(Adaptive Adaptive Moment Estimation)" optimizer which essentially combines RMSProp and momentum by storing both the individual learning rate of RMSProp and the weighted average of momentum. The adaptive optimizers are generally faster compared to standard SGD. However, it has been argued as well that 'sgd' performs better in terms of generalization performance (https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008 (https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008)).
- The input vector has 49512 rows(128 x 128 x 3).
- For this we need to reshape our tensors into vectors.

In [22]:
```
# our train shape is 4708, 128, 128, 3. Reshaping will change it to 47
train_img = train_images.reshape(train_images.shape[0], -1)
test_img = test_images.reshape(test_images.shape[0], -1)
val_img = val_images.reshape(val_images.shape[0], -1)

print(train_img.shape)
print(test_img.shape)
print(val_img.shape)
```

```
(4708, 49152)
(624, 49152)
(524, 49152)
```

In [23]:
```
train_labels.shape
```

Out[23]:
```
(4708,)
```

In [24]:
```
# transform the labels from arrays to a 1D vector
train_y = np.reshape(train_labels, (train_images.shape[0],1))
test_y = np.reshape(test_labels, (test_images.shape[0],1))
val_y = np.reshape(val_labels, (val_images.shape[0],1))
```

In [25]:
```
print(train_y.shape)
print(test_y.shape)
print(val_y.shape)
```

```
(4708, 1)
(624, 1)
(524, 1)
```

In [26]:
```
# Size of the image vector that needs to be input to the ANN models:
n_features = train_img.shape[1]
```

In [30]:
```
#We will use "adam" optimizer. A test run with 'sgd' resulted in lower
# Initialize model
model = models.Sequential()

# First hidden layer
model.add(layers.Dense(10, activation='relu', input_shape=(n_features,
# Output layer
model.add(layers.Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
#The batch_size 44 is used as it is exact divisor of 4708(train data s
ann_baseline_model = model.fit(train_img, train_y, epochs = 100, batch
                     validation_data = (val_img, val_y))
```

In [31]:
```
results_train = model.evaluate(train_img, train_y)
```

```
148/148 [==============================] - 0s 1ms/step - loss: 0.0293
- accuracy: 0.9928
```

In [32]:
```
results_test = model.evaluate(test_img, test_y)
```

```
20/20 [==============================] - 0s 1ms/step - loss: 1.4271 -
accuracy: 0.8141
```

In [33]:
```
results_train
```

Out[33]:
```
[0.029281755909323692, 0.9927782416343689]
```

In [34]:
```
results_test
```

Out[34]:
```
[1.4270974397659302, 0.8141025900840759]
```

In [35]:
```
ann_baseline_model.history.keys()
```

Out[35]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Evaluating model performance

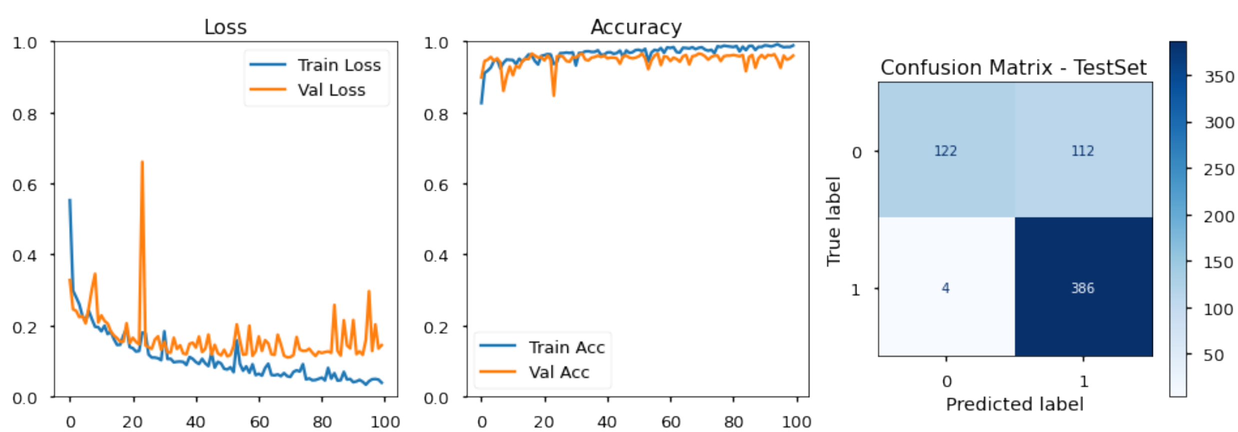
- We will define a function that takes in the model, train and test datsets, and evalutaion metrics; accuracy and validation accuracy.
- The function will return the confusion matrix and also the model performance for test and train datsets.
- We are using confusion matrix since we want to maximize the correct prediction for true pneumonia cases: i.e. recall score

In [36]:

```
# Model evaluation function
def plot_model_performance(Model, Xtrain, Xtest, Acc, Val_acc):

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

        # Diplay train and validation loss and accuracy:
        fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16,5))
        ax1.plot(Model.history['loss'])
        ax1.plot(Model.history['val_loss'])
        ax1.set_title("Loss")
        ax1.legend(labels = ['Train Loss', 'Val Loss'])
        ax1.set_ylim(0,1)
        ax2.plot(Model.history[Acc])
        ax2.plot(Model.history[Val_acc])
        ax2.legend(labels = ['Train Acc', 'Val Acc'])
        ax2.set_title('Accuracy')
        ax2.set_ylim(0,1)

        # Output (probability) predictions for the test set
        y_hat_test = Model.model.predict(Xtest)
        y_pred = np.rint(y_hat_test).astype(np.int) # Round elements o
        y_true = test_y.astype(np.int)

        # Generate a confusion matrix displaying the predictive accura
        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 th
        print('Classification Report:')
        print(classification_report(y_true, y_pred))
        print('\n')

        # Print final train and test loss and accuracy:
        train_loss, train_acc = Model.model.evaluate(Xtrain, train_y);
        test_loss, test_acc = Model.model.evaluate(Xtest, test_y);
        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')
```

In [37]:
```
plot_model_performance(ann_baseline_model,train_img,test_img,"accuracy
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.52      0.68       234
           1       0.78      0.99      0.87       390

    accuracy                           0.81       624
   macro avg       0.87      0.76      0.77       624
weighted avg       0.85      0.81      0.80       624



148/148 [==============================] - 0s 1ms/step - loss: 0.0293
- accuracy: 0.9928
20/20 [==============================] - 0s 1ms/step - loss: 1.4271 -
accuracy: 0.8141
--------------------------
Final Train Loss: 0.0293
Final Test Loss: 1.4271
--------------------------
Final Train Acc: 0.9928
Final Test Acc: 0.8141
```

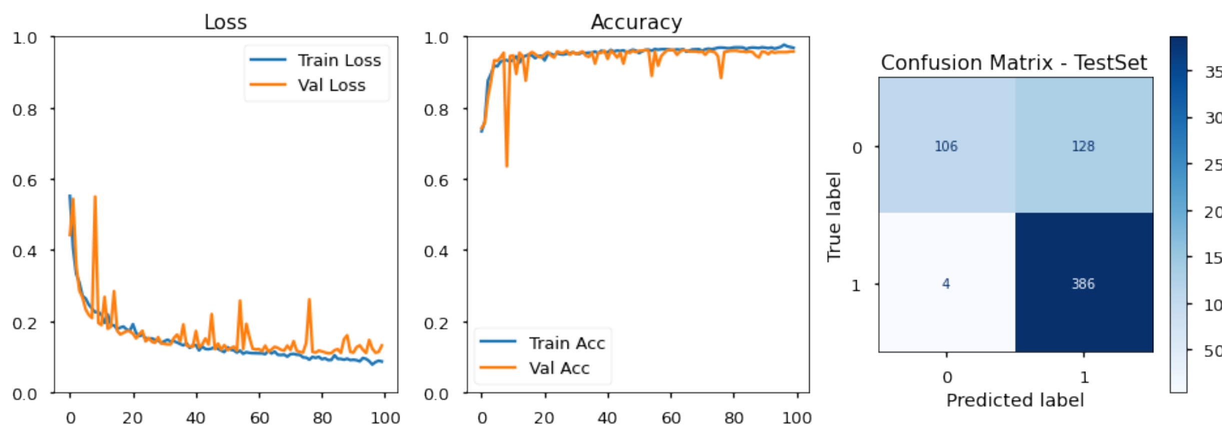


- The very basic ANN model with only one layer basically predicts correctly 99% of true pneumonia cases, whereas for normal cases the recall is 53%.

In [40]:
```
model2 = models.Sequential()

# First hidden layer
model2.add(layers.Dense(10, activation='relu', input_shape=(n_features
# Output layer
model2.add(layers.Dense(1, activation='sigmoid'))
model2.compile(optimizer='sgd',
             loss='binary_crossentropy',
             metrics=['accuracy'])
#The batch_size 44 is used as it is exact divisor of 4708(train data s
ann_baseline_sgd = model2.fit(train_img, train_y, epochs = 100, batch_
                    validation_data = (val_img, val_y))
```

In [41]:
```
plot_model_performance(ann_baseline_sgd,train_img,test_img,"accuracy",
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.45      0.62       234
           1       0.75      0.99      0.85       390

    accuracy                           0.79       624
   macro avg       0.86      0.72      0.74       624
weighted avg       0.83      0.79      0.76       624


148/148 [==============================] - 0s 1ms/step - loss: 0.0900
- accuracy: 0.9688
20/20 [==============================] - 0s 1ms/step - loss: 1.0945 -
accuracy: 0.7885
--------------------------
Final Train Loss: 0.09
Final Test Loss: 1.0945
--------------------------
Final Train Acc: 0.9688
Final Test Acc: 0.7885
```



Bigger Deeper ANN model:

- We will add an input layer with 256 neurons
- We will add four hidden layers with 128, 64, 32 and 10 neurons.
- We will add the output layer with 1 neuron.

Early Stopping:

- We will use early stopping for all the subsequent models. Early stopping checks the model performance on holdout validation dataset and once there is no improvement in performance, the training will stop. It helps with overfitting and it won't run for more epochs unnecessarily.
- The monitoring parameter that we will use for this will be Validation_loss.

In [42]:
```
# We will use patience of 10 (10 or 20 are most common). The model wil
# to make sure that there is no imporvement in model performance on Va
early_stopping = [EarlyStopping(monitor='val_loss', patience=10)]
```

In [126]:
```
model = models.Sequential()

# Add dense layers with relu activation
model.add(layers.Dense(256, activation='relu', input_shape = (n_featur
model.add(layers.Dense(128, activation='relu'))
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.save("Deeper_ANN")
model.compile(loss = 'binary_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
```

```
INFO:tensorflow:Assets written to: Deeper_ANN/assets
```

In [127]:
```
model.summary()
```

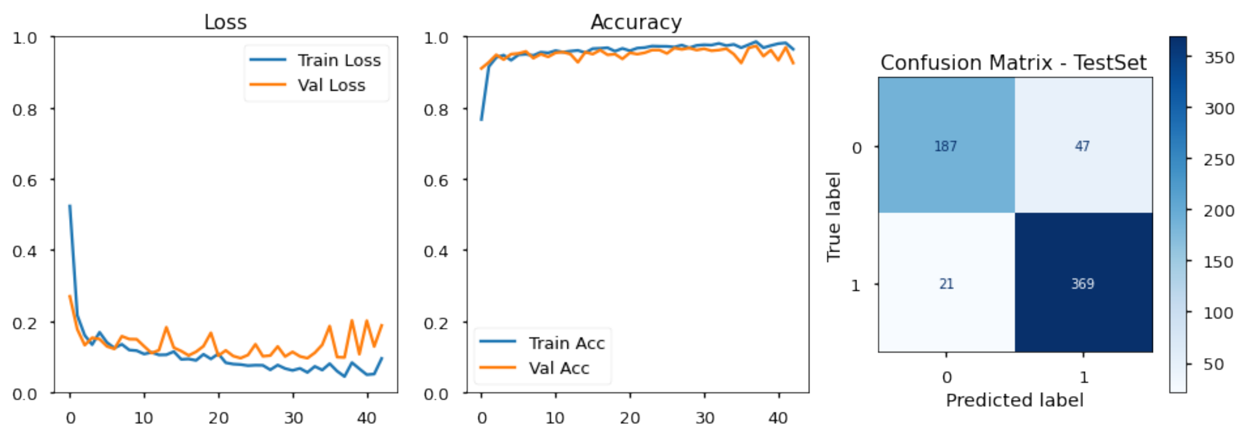
```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
dense_87 (Dense)            (None, 256)               12583168
_________________________________________________________________
dense_88 (Dense)            (None, 128)               32896
_________________________________________________________________
dense_89 (Dense)            (None, 64)                8256
_________________________________________________________________
dense_90 (Dense)            (None, 32)                2080
_________________________________________________________________
dense_91 (Dense)            (None, 10)                330
_________________________________________________________________
dense_92 (Dense)            (None, 1)                 11
=================================================================
Total params: 12,626,741
Trainable params: 12,626,741
Non-trainable params: 0
_________________________________________________________________
```

In [45]:
```
deeper_ann_model = model.fit(train_img, train_y,
                             epochs = 100, batch_size = 44,
                             verbose =1,callbacks=early_stopping,
                             validation_data = (val_img, val_y))
```

```
Epoch 37/100
107/107 [==============================] - 5s 44ms/step - loss: 0.057
5 - accuracy: 0.9816 - val_loss: 0.0995 - val_accuracy: 0.9676
Epoch 38/100
107/107 [==============================] - 5s 44ms/step - loss: 0.044
9 - accuracy: 0.9868 - val_loss: 0.0986 - val_accuracy: 0.9733
Epoch 39/100
107/107 [==============================] - 5s 43ms/step - loss: 0.080
5 - accuracy: 0.9714 - val_loss: 0.2024 - val_accuracy: 0.9447
Epoch 40/100
107/107 [==============================] - 5s 44ms/step - loss: 0.108
2 - accuracy: 0.9603 - val_loss: 0.1078 - val_accuracy: 0.9618
Epoch 41/100
107/107 [==============================] - 5s 44ms/step - loss: 0.045
9 - accuracy: 0.9818 - val_loss: 0.2019 - val_accuracy: 0.9332
Epoch 42/100
107/107 [==============================] - 5s 44ms/step - loss: 0.059
8 - accuracy: 0.9793 - val_loss: 0.1293 - val_accuracy: 0.9695
Epoch 43/100
107/107 [==============================] - 5s 43ms/step - loss: 0.072
```

In [46]:
```
plot_model_performance(deeper_ann_model,train_img,test_img,"accuracy",
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.80      0.85       234
           1       0.89      0.95      0.92       390

    accuracy                           0.89       624
   macro avg       0.89      0.87      0.88       624
weighted avg       0.89      0.89      0.89       624


148/148 [==============================] - 1s 5ms/step - loss: 0.1213
- accuracy: 0.9535
20/20 [==============================] - 0s 5ms/step - loss: 0.2972 -
accuracy: 0.8910
--------------------------
Final Train Loss: 0.1213
Final Test Loss: 0.2972
--------------------------
Final Train Acc: 0.9535
Final Test Acc: 0.891
```



- The recall score for pneumonia cases is now 95%, the model misclassified 21 cases, the recall for normal cases is still not great. Compared to baseline model it though has improved from 52% to 85%.
- The test accuracy is 89% compared to train set accuracy of 95% meaning the model is slightly overfitting.
- Lets remove the first layer with 256 neurons and see if the model does any better.
- Next steps are to include regularizations: Dropout and L2

In [47]:
```
model = models.Sequential()

# Add dense layers with relu activation
model.add(layers.Dense(128, activation='relu', input_shape = (n_featur
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'])
```

In [48]:
```
deeper_ann_model2 = model.fit(train_img, train_y,
                             epochs = 100, batch_size = 44,
                             verbose =1,callbacks=early_stopping,
                             validation_data = (val_img, val_y))
```

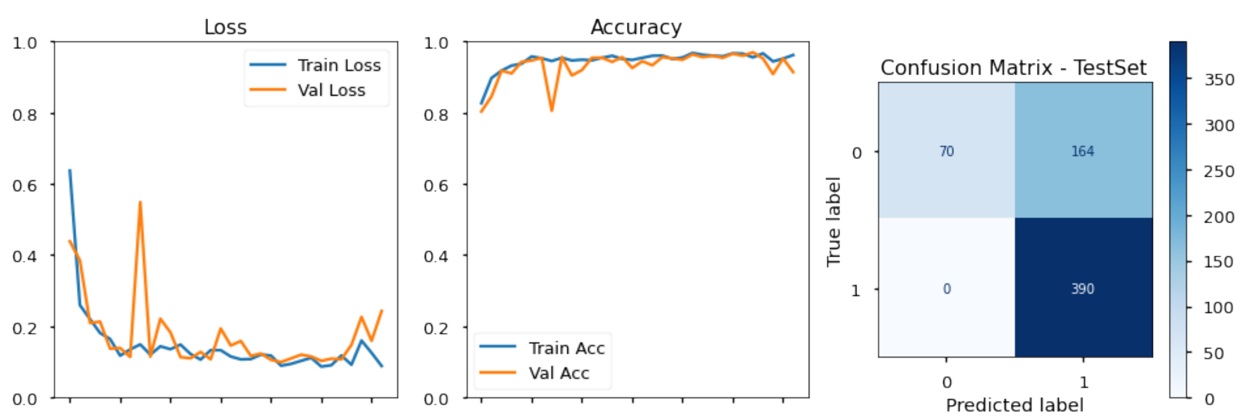
```
Epoch 1/100
107/107 [==============================] - 2s 20ms/step - loss: 1.301
1 - accuracy: 0.7288 - val_loss: 0.4386 - val_accuracy: 0.8034
Epoch 2/100
107/107 [==============================] - 2s 20ms/step - loss: 0.255
8 - accuracy: 0.8935 - val_loss: 0.3850 - val_accuracy: 0.8454
Epoch 3/100
107/107 [==============================] - 2s 20ms/step - loss: 0.239
8 - accuracy: 0.9116 - val_loss: 0.2093 - val_accuracy: 0.9179
Epoch 4/100
107/107 [==============================] - 2s 19ms/step - loss: 0.171
1 - accuracy: 0.9384 - val_loss: 0.2133 - val_accuracy: 0.9103
Epoch 5/100
107/107 [==============================] - 2s 20ms/step - loss: 0.171
1 - accuracy: 0.9329 - val_loss: 0.1374 - val_accuracy: 0.9427
Epoch 6/100
107/107 [==============================] - 2s 20ms/step - loss: 0.112
0 - accuracy: 0.9607 - val_loss: 0.1392 - val_accuracy: 0.9466
Epoch 7/100
107/107 [==============================] - 2s 19ms/step - loss: 0.144
3 - accuracy: 0.9486 - val_loss: 0.1140 - val_accuracy: 0.9542
Epoch 8/100
107/107 [==============================] - 2s 19ms/step - loss: 0.123
9 - accuracy: 0.9550 - val_loss: 0.5489 - val_accuracy: 0.8053
Epoch 9/100
107/107 [==============================] - 2s 19ms/step - loss: 0.142
7 - accuracy: 0.9439 - val_loss: 0.1151 - val_accuracy: 0.9561
Epoch 10/100
107/107 [==============================] - 2s 20ms/step - loss: 0.132
```

```
0 - accuracy: 0.9467 - val_loss: 0.2212 - val_accuracy: 0.9046
Epoch 11/100
107/107 [==============================] - 2s 20ms/step - loss: 0.140
1 - accuracy: 0.9469 - val_loss: 0.1831 - val_accuracy: 0.9198
Epoch 12/100
107/107 [==============================] - 2s 19ms/step - loss: 0.153
7 - accuracy: 0.9452 - val_loss: 0.1136 - val_accuracy: 0.9542
Epoch 13/100
107/107 [==============================] - 2s 19ms/step - loss: 0.113
3 - accuracy: 0.9583 - val_loss: 0.1103 - val_accuracy: 0.9542
Epoch 14/100
107/107 [==============================] - 2s 19ms/step - loss: 0.105
5 - accuracy: 0.9597 - val_loss: 0.1281 - val_accuracy: 0.9427
Epoch 15/100
107/107 [==============================] - 2s 19ms/step - loss: 0.118
1 - accuracy: 0.9578 - val_loss: 0.1075 - val_accuracy: 0.9561
Epoch 16/100
107/107 [==============================] - 2s 19ms/step - loss: 0.130
8 - accuracy: 0.9490 - val_loss: 0.1938 - val_accuracy: 0.9256
Epoch 17/100
107/107 [==============================] - 2s 20ms/step - loss: 0.132
2 - accuracy: 0.9429 - val_loss: 0.1461 - val_accuracy: 0.9447
Epoch 18/100
107/107 [==============================] - 2s 19ms/step - loss: 0.113
9 - accuracy: 0.9590 - val_loss: 0.1587 - val_accuracy: 0.9332
Epoch 19/100
107/107 [==============================] - 2s 20ms/step - loss: 0.106
3 - accuracy: 0.9633 - val_loss: 0.1169 - val_accuracy: 0.9561
Epoch 20/100
107/107 [==============================] - 2s 20ms/step - loss: 0.116
9 - accuracy: 0.9536 - val_loss: 0.1237 - val_accuracy: 0.9523
Epoch 21/100
107/107 [==============================] - 2s 20ms/step - loss: 0.111
8 - accuracy: 0.9521 - val_loss: 0.1059 - val_accuracy: 0.9485
Epoch 22/100
107/107 [==============================] - 2s 19ms/step - loss: 0.099
6 - accuracy: 0.9629 - val_loss: 0.0996 - val_accuracy: 0.9637
Epoch 23/100
107/107 [==============================] - 2s 19ms/step - loss: 0.091
7 - accuracy: 0.9652 - val_loss: 0.1100 - val_accuracy: 0.9561
Epoch 24/100
107/107 [==============================] - 2s 19ms/step - loss: 0.114
7 - accuracy: 0.9557 - val_loss: 0.1203 - val_accuracy: 0.9599
Epoch 25/100
107/107 [==============================] - 2s 19ms/step - loss: 0.118
0 - accuracy: 0.9559 - val_loss: 0.1153 - val_accuracy: 0.9542
Epoch 26/100
107/107 [==============================] - 2s 19ms/step - loss: 0.083
4 - accuracy: 0.9675 - val_loss: 0.1031 - val_accuracy: 0.9656
Epoch 27/100
```

```
107/107 [==============================] - 2s 19ms/step - loss: 0.091
2 - accuracy: 0.9645 - val_loss: 0.1091 - val_accuracy: 0.9599
Epoch 28/100
107/107 [==============================] - 2s 19ms/step - loss: 0.102
3 - accuracy: 0.9616 - val_loss: 0.1074 - val_accuracy: 0.9695
Epoch 29/100
107/107 [==============================] - 2s 19ms/step - loss: 0.095
0 - accuracy: 0.9662 - val_loss: 0.1481 - val_accuracy: 0.9523
Epoch 30/100
107/107 [==============================] - 2s 19ms/step - loss: 0.102
6 - accuracy: 0.9650 - val_loss: 0.2263 - val_accuracy: 0.9084
Epoch 31/100
107/107 [==============================] - 2s 19ms/step - loss: 0.137
7 - accuracy: 0.9434 - val_loss: 0.1594 - val_accuracy: 0.9523
Epoch 32/100
107/107 [==============================] - 2s 20ms/step - loss: 0.099
5 - accuracy: 0.9561 - val_loss: 0.2430 - val_accuracy: 0.9141
```

In [49]:
```
plot_model_performance(deeper_ann_model2,train_img,test_img,"accuracy"
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.30      0.46       234
           1       0.70      1.00      0.83       390

    accuracy                           0.74       624
   macro avg       0.85      0.65      0.64       624
weighted avg       0.81      0.74      0.69       624


148/148 [==============================] - 1s 4ms/step - loss: 0.1779
- accuracy: 0.9280
20/20 [==============================] - 0s 4ms/step - loss: 1.4547 -
accuracy: 0.7372
--------------------------
Final Train Loss: 0.1779
Final Test Loss: 1.4547
--------------------------
Final Train Acc: 0.928
Final Test Acc: 0.7372
```



- The model performed badly, so will keep the original configuration with 256 neurons in the in

1b. Dropout Regularization

- Apply a dropout rate of 30% to the all layers

In [50]:
```
model = models.Sequential()

# Add dense layers with relu activation
model.add(layers.Dropout(0.3, input_shape=(n_features,)))

model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dropout(0.3))

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

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

In [51]:
```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
dropout (Dropout)           (None, 49152)             0
_________________________________________________________________
dense_17 (Dense)            (None, 256)               12583168
_________________________________________________________________
dropout_1 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_18 (Dense)            (None, 128)               32896
_________________________________________________________________
dropout_2 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_19 (Dense)            (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_20 (Dense)            (None, 32)                2080
_________________________________________________________________
dropout_4 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_21 (Dense)            (None, 10)                330
_________________________________________________________________
dropout_5 (Dropout)         (None, 10)                0
_________________________________________________________________
dense_22 (Dense)            (None, 1)                 11
=================================================================
Total params: 12,626,741
Trainable params: 12,626,741
Non-trainable params: 0
_________________________________________________________________
```

In [52]:
```
dropout_ann_model = model.fit(train_img, train_y,
                              epochs = 100, batch_size = 22,
                              verbose =1,callbacks=early_stopping,
                              validation_data = (val_img, val_y))
```

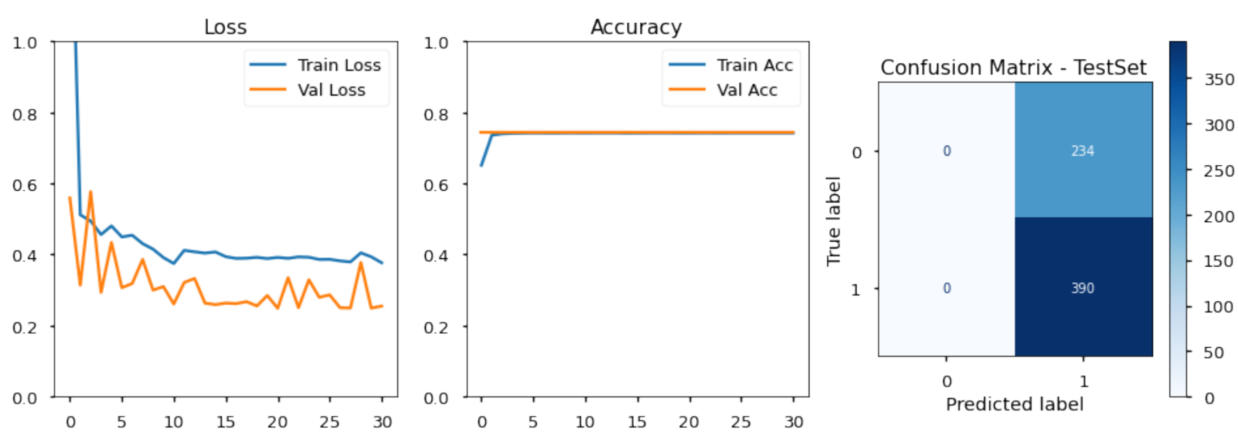
```
Epoch 1/100
214/214 [==============================] - 8s 35ms/step - loss: 3.034
1 - accuracy: 0.6450 - val_loss: 0.5591 - val_accuracy: 0.7424
Epoch 2/100
214/214 [==============================] - 7s 35ms/step - loss: 0.537
3 - accuracy: 0.7290 - val_loss: 0.3139 - val_accuracy: 0.7424
Epoch 3/100
214/214 [==============================] - 7s 35ms/step - loss: 0.477
7 - accuracy: 0.7374 - val_loss: 0.5770 - val_accuracy: 0.7424
Epoch 4/100
```

```
Epoch 4/100
214/214 [==============================] - 7s 34ms/step - loss: 0.506
1 - accuracy: 0.7260 - val_loss: 0.2934 - val_accuracy: 0.7424
Epoch 5/100
214/214 [==============================] - 8s 35ms/step - loss: 0.484
7 - accuracy: 0.7469 - val_loss: 0.4338 - val_accuracy: 0.7424
Epoch 6/100
214/214 [==============================] - 7s 35ms/step - loss: 0.455
9 - accuracy: 0.7334 - val_loss: 0.3066 - val_accuracy: 0.7424
Epoch 7/100
214/214 [==============================] - 7s 34ms/step - loss: 0.465
4 - accuracy: 0.7454 - val_loss: 0.3185 - val_accuracy: 0.7424
Epoch 8/100
214/214 [==============================] - 7s 34ms/step - loss: 0.410
8 - accuracy: 0.7423 - val_loss: 0.3862 - val_accuracy: 0.7424
Epoch 9/100
214/214 [==============================] - 7s 35ms/step - loss: 0.419
5 - accuracy: 0.7363 - val_loss: 0.2999 - val_accuracy: 0.7424
Epoch 10/100
214/214 [==============================] - 7s 34ms/step - loss: 0.393
8 - accuracy: 0.7413 - val_loss: 0.3097 - val_accuracy: 0.7424
Epoch 11/100
214/214 [==============================] - 7s 35ms/step - loss: 0.370
5 - accuracy: 0.7489 - val_loss: 0.2608 - val_accuracy: 0.7424
Epoch 12/100
214/214 [==============================] - 7s 35ms/step - loss: 0.390
9 - accuracy: 0.7418 - val_loss: 0.3212 - val_accuracy: 0.7424
Epoch 13/100
214/214 [==============================] - 7s 35ms/step - loss: 0.398
0 - accuracy: 0.7513 - val_loss: 0.3326 - val_accuracy: 0.7424
Epoch 14/100
214/214 [==============================] - 7s 34ms/step - loss: 0.394
0 - accuracy: 0.7466 - val_loss: 0.2633 - val_accuracy: 0.7424
Epoch 15/100
214/214 [==============================] - 7s 34ms/step - loss: 0.390
2 - accuracy: 0.7530 - val_loss: 0.2590 - val_accuracy: 0.7424
Epoch 16/100
214/214 [==============================] - 7s 34ms/step - loss: 0.383
6 - accuracy: 0.7498 - val_loss: 0.2633 - val_accuracy: 0.7424
Epoch 17/100
214/214 [==============================] - 7s 34ms/step - loss: 0.390
2 - accuracy: 0.7361 - val_loss: 0.2618 - val_accuracy: 0.7424
Epoch 18/100
214/214 [==============================] - 7s 34ms/step - loss: 0.394
8 - accuracy: 0.7323 - val_loss: 0.2673 - val_accuracy: 0.7424
Epoch 19/100
214/214 [==============================] - 7s 34ms/step - loss: 0.377
9 - accuracy: 0.7597 - val_loss: 0.2553 - val_accuracy: 0.7424
Epoch 20/100
214/214 [==============================] - 7s 34ms/step - loss: 0.389
```

```
0 - accuracy: 0.7281 - val_loss: 0.2846 - val_accuracy: 0.7424
Epoch 21/100
214/214 [==============================] - 7s 33ms/step - loss: 0.395
2 - accuracy: 0.7420 - val_loss: 0.2487 - val_accuracy: 0.7424
Epoch 22/100
214/214 [==============================] - 7s 35ms/step - loss: 0.386
8 - accuracy: 0.7424 - val_loss: 0.3347 - val_accuracy: 0.7424
Epoch 23/100
214/214 [==============================] - 7s 34ms/step - loss: 0.421
1 - accuracy: 0.7328 - val_loss: 0.2504 - val_accuracy: 0.7424
Epoch 24/100
214/214 [==============================] - 7s 34ms/step - loss: 0.393
3 - accuracy: 0.7383 - val_loss: 0.3289 - val_accuracy: 0.7424
Epoch 25/100
214/214 [==============================] - 7s 34ms/step - loss: 0.385
2 - accuracy: 0.7513 - val_loss: 0.2795 - val_accuracy: 0.7424
Epoch 26/100
214/214 [==============================] - 7s 34ms/step - loss: 0.387
5 - accuracy: 0.7400 - val_loss: 0.2865 - val_accuracy: 0.7424
Epoch 27/100
214/214 [==============================] - 7s 34ms/step - loss: 0.389
9 - accuracy: 0.7411 - val_loss: 0.2503 - val_accuracy: 0.7424
Epoch 28/100
214/214 [==============================] - 7s 34ms/step - loss: 0.386
4 - accuracy: 0.7367 - val_loss: 0.2495 - val_accuracy: 0.7424
Epoch 29/100
214/214 [==============================] - 7s 34ms/step - loss: 0.395
2 - accuracy: 0.7402 - val_loss: 0.3776 - val_accuracy: 0.7424
Epoch 30/100
214/214 [==============================] - 7s 35ms/step - loss: 0.420
0 - accuracy: 0.7411 - val_loss: 0.2492 - val_accuracy: 0.7424
Epoch 31/100
214/214 [==============================] - 7s 34ms/step - loss: 0.382
3 - accuracy: 0.7331 - val_loss: 0.2548 - val_accuracy: 0.7424
```

In [53]:
```
plot_model_performance(droput_ann_model,train_img,test_img,"accuracy",
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       234
           1       0.62      1.00      0.77       390

    accuracy                           0.62       624
   macro avg       0.31      0.50      0.38       624
weighted avg       0.39      0.62      0.48       624



148/148 [==============================] - 1s 5ms/step - loss: 0.2373
- accuracy: 0.7421
20/20 [==============================] - 0s 5ms/step - loss: 0.9622 -
accuracy: 0.6250
--------------------------
Final Train Loss: 0.2373
Final Test Loss: 0.9622
--------------------------
Final Train Acc: 0.7421
Final Test Acc: 0.625
```



* This model did not perform well, it mislabeled all the normal cases as pneumonia ones.

1c. L2 Regularization

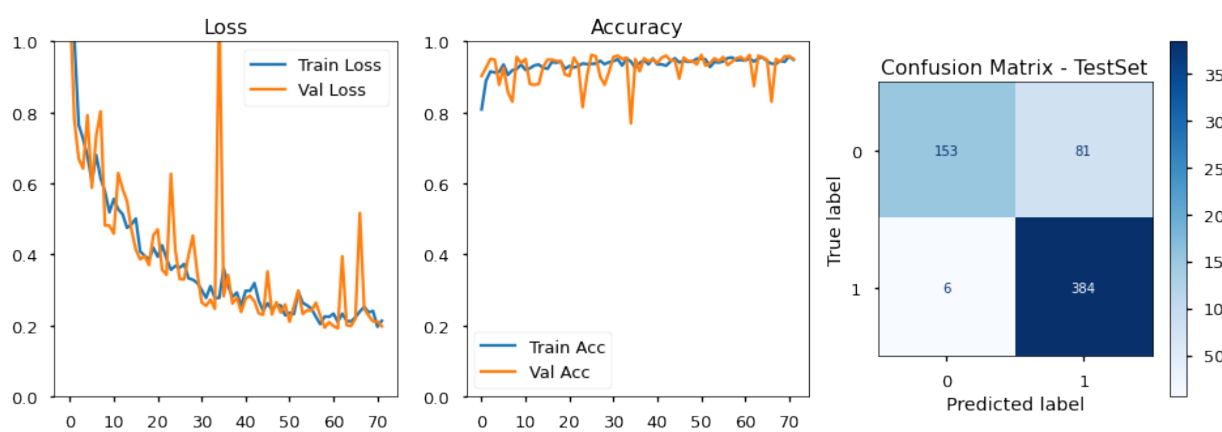
- Lets add an L2 regularizer and see what happens

In [54]:
```
random.seed(123)
L2_model = models.Sequential()

# Add the input and first hidden layer
L2_model.add(layers.Dense(256, activation='relu', kernel_regularizer=re

# Add another hidden layer
L2_model.add(layers.Dense(128, kernel_regularizer=regularizers.l2(0.005
# Add another hidden layer
L2_model.add(layers.Dense(64, kernel_regularizer=regularizers.l2(0.005

# Add an output layer
L2_model.add(layers.Dense(32, kernel_regularizer=regularizers.l2(0.005

L2_model.add(layers.Dense(10, kernel_regularizer=regularizers.l2(0.005


L2_model.add(layers.Dense(1, activation='sigmoid'))

L2_model.compile(loss = 'binary_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])

# Train the model
L2_ann_model = L2_model.fit(train_img, train_y,
                            epochs = 100, batch_size = 44,
                            verbose =1,callbacks=early_stopping,
                            validation_data = (val_img, val_y))
```

```
Epoch 51/100
107/107 [==============================] - 8s 73ms/step - loss: 0.239
1 - accuracy: 0.9510 - val_loss: 0.2108 - val_accuracy: 0.9618
Epoch 52/100
107/107 [==============================] - 8s 74ms/step - loss: 0.232
4 - accuracy: 0.9516 - val_loss: 0.2554 - val_accuracy: 0.9313
Epoch 53/100
107/107 [==============================] - 8s 74ms/step - loss: 0.254
7 - accuracy: 0.9388 - val_loss: 0.2986 - val_accuracy: 0.9389
Epoch 54/100
107/107 [==============================] - 8s 74ms/step - loss: 0.268
5 - accuracy: 0.9457 - val_loss: 0.2330 - val_accuracy: 0.9523
Epoch 55/100
107/107 [==============================] - 8s 73ms/step - loss: 0.255
3 - accuracy: 0.9392 - val_loss: 0.2428 - val_accuracy: 0.9447
Epoch 56/100
107/107 [==============================] - 8s 75ms/step - loss: 0.217
8 - accuracy: 0.9555 - val_loss: 0.2419 - val_accuracy: 0.9561
Epoch 57/100
```

In [55]:
```
plot_model_performance(L2_ann_model,train_img,test_img,"accuracy", "va
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.65      0.78       234
           1       0.83      0.98      0.90       390

    accuracy                           0.86       624
   macro avg       0.89      0.82      0.84       624
weighted avg       0.88      0.86      0.85       624


148/148 [==============================] - 2s 14ms/step - loss: 0.177
7 - accuracy: 0.9650
20/20 [==============================] - 0s 14ms/step - loss: 0.5024
- accuracy: 0.8606
-------------------------
Final Train Loss: 0.1777
Final Test Loss: 0.5024
-------------------------
Final Train Acc: 0.965
Final Test Acc: 0.8606
```



- This performed better than the dropout version. However our basic deeper ann model performs better than all of these. So we will stci with that one as our version for ANN model

Early stopping modified:

- Looking at the documentaion, I realized that reesults amy improve if we have weights distribution true in the early stopping. Lets implement that and see if the model improves

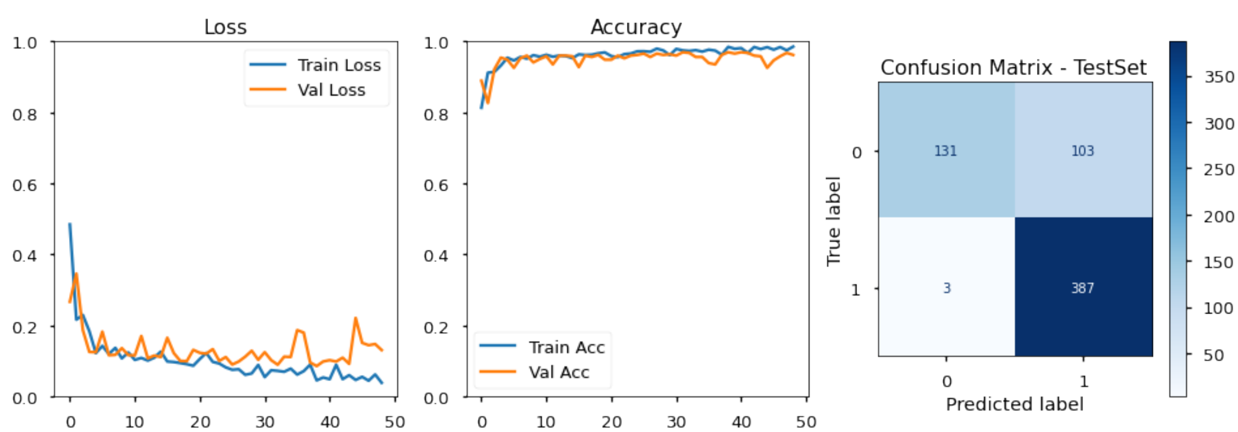
In [56]:
```
#Despite the default value of restore_weights being set to False, whic
#We have no problem in keeping another copy of the model in memory (i.
#the most sensible value is restore_best_weights=True.
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_be
```

In [57]:
```
ann_model = models.Sequential()

# Add dense layers with relu activation
ann_model.add(layers.Dense(256, activation='relu', input_shape = (n_fe
ann_model.add(layers.Dense(128, activation='relu'))
ann_model.add(layers.Dense(64, activation='relu'))
ann_model.add(layers.Dense(32, activation='relu'))
ann_model.add(layers.Dense(10, activation='relu'))

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

ann_model.compile(loss = 'binary_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
ann_final = ann_model.fit(train_img, train_y,
                             epochs = 100, batch_size = 44,
                             verbose =1,callbacks=early_stop,
                             validation_data = (val_img, val_y))
```

```
Epoch 21/100
107/107 [==============================] - 5s 45ms/step - loss: 0.090
8 - accuracy: 0.9654 - val_loss: 0.1233 - val_accuracy: 0.9485
Epoch 22/100
107/107 [==============================] - 5s 45ms/step - loss: 0.103
2 - accuracy: 0.9637 - val_loss: 0.1208 - val_accuracy: 0.9599
Epoch 23/100
107/107 [==============================] - 5s 45ms/step - loss: 0.076
9 - accuracy: 0.9715 - val_loss: 0.1337 - val_accuracy: 0.9523
Epoch 24/100
107/107 [==============================] - 5s 45ms/step - loss: 0.092
2 - accuracy: 0.9667 - val_loss: 0.1006 - val_accuracy: 0.9599
Epoch 25/100
107/107 [==============================] - 5s 46ms/step - loss: 0.072
0 - accuracy: 0.9753 - val_loss: 0.1113 - val_accuracy: 0.9618
Epoch 26/100
107/107 [==============================] - 5s 44ms/step - loss: 0.072
7 - accuracy: 0.9717 - val_loss: 0.0903 - val_accuracy: 0.9656
Epoch 27/100
107/107 [==============================] - 5s 45ms/step - loss: 0.090
```

In [59]:
```
plot_model_performance(ann_final,train_img,test_img,"accuracy", "val_a
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.56      0.71       234
           1       0.79      0.99      0.88       390

    accuracy                           0.83       624
   macro avg       0.88      0.78      0.80       624
weighted avg       0.86      0.83      0.82       624



148/148 [==============================] - 1s 5ms/step - loss: 0.0424
- accuracy: 0.9845
20/20 [==============================] - 0s 6ms/step - loss: 0.8126 -
accuracy: 0.8301
--------------------------
Final Train Loss: 0.0424
Final Test Loss: 0.8126
--------------------------
Final Train Acc: 0.9845
Final Test Acc: 0.8301
```



- This does not seem to have make much improvement
- So far, the basic deeper ann model with earlystopping (no weights) gave us the best performance

1d. Learning Rate Modified

- Default learnin rate for adam is 0.001. Lets halve it and see how it works!

In [114]:
```
model = models.Sequential()

# Add dense layers with relu activation
model.add(layers.Dense(256, activation='relu', input_shape = (n_featur
model.add(layers.Dense(128, activation='relu'))
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 = optimizers.Adam(learning_rate=0.0005) ,
              metrics = ['accuracy'])
```

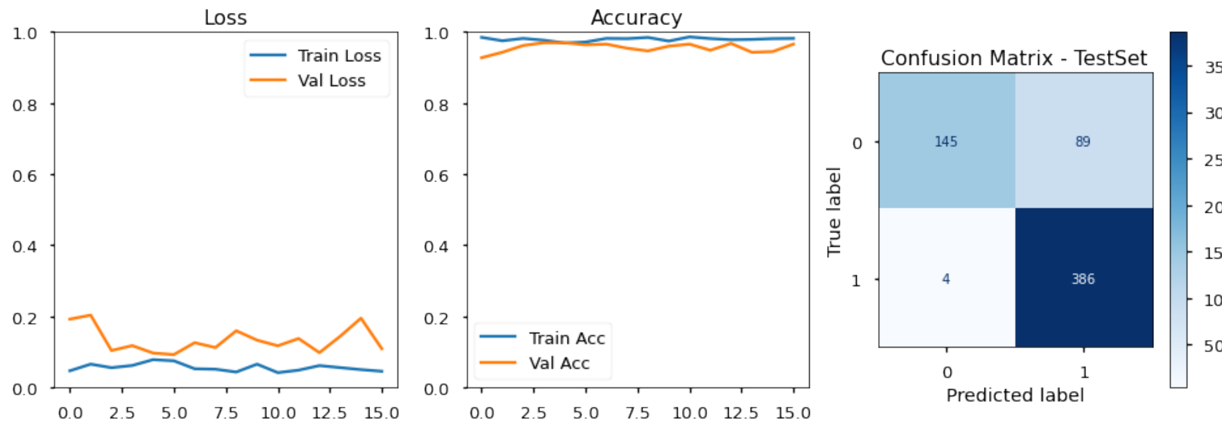
In [115]:
```
ann_lr = ann_model.fit(train_img, train_y,
                             epochs = 100, batch_size = 44,
                             verbose =1,callbacks=early_stop,
                             validation_data = (val_img, val_y))
```

```
Epoch 1/100
107/107 [==============================] - 6s 49ms/step - loss: 0.047
2 - accuracy: 0.9847 - val_loss: 0.1924 - val_accuracy: 0.9275
Epoch 2/100
107/107 [==============================] - 5s 49ms/step - loss: 0.065
9 - accuracy: 0.9751 - val_loss: 0.2036 - val_accuracy: 0.9427
Epoch 3/100
107/107 [==============================] - 5s 49ms/step - loss: 0.055
9 - accuracy: 0.9817 - val_loss: 0.1042 - val_accuracy: 0.9618
Epoch 4/100
107/107 [==============================] - 5s 48ms/step - loss: 0.062
1 - accuracy: 0.9766 - val_loss: 0.1179 - val_accuracy: 0.9695
Epoch 5/100
107/107 [==============================] - 5s 48ms/step - loss: 0.078
5 - accuracy: 0.9688 - val_loss: 0.0971 - val_accuracy: 0.9695
Epoch 6/100
107/107 [==============================] - 5s 48ms/step - loss: 0.075
3 - accuracy: 0.9715 - val_loss: 0.0927 - val_accuracy: 0.9637
Epoch 7/100
107/107 [==============================] - 5s 47ms/step - loss: 0.052
8 - accuracy: 0.9817 - val_loss: 0.1262 - val_accuracy: 0.9656
Epoch 8/100
```

```
107/107 [==============================] - 5s 47ms/step - loss: 0.051
7 - accuracy: 0.9811 - val_loss: 0.1125 - val_accuracy: 0.9542
Epoch 9/100
107/107 [==============================] - 5s 47ms/step - loss: 0.043
7 - accuracy: 0.9845 - val_loss: 0.1597 - val_accuracy: 0.9466
Epoch 10/100
107/107 [==============================] - 5s 46ms/step - loss: 0.065
8 - accuracy: 0.9743 - val_loss: 0.1334 - val_accuracy: 0.9599
Epoch 11/100
107/107 [==============================] - 5s 46ms/step - loss: 0.042
0 - accuracy: 0.9860 - val_loss: 0.1175 - val_accuracy: 0.9656
Epoch 12/100
107/107 [==============================] - 5s 46ms/step - loss: 0.049
0 - accuracy: 0.9813 - val_loss: 0.1379 - val_accuracy: 0.9485
Epoch 13/100
107/107 [==============================] - 5s 46ms/step - loss: 0.061
8 - accuracy: 0.9781 - val_loss: 0.0982 - val_accuracy: 0.9676
Epoch 14/100
107/107 [==============================] - 5s 48ms/step - loss: 0.056
1 - accuracy: 0.9790 - val_loss: 0.1442 - val_accuracy: 0.9427
Epoch 15/100
107/107 [==============================] - 5s 47ms/step - loss: 0.050
5 - accuracy: 0.9811 - val_loss: 0.1951 - val_accuracy: 0.9447
Epoch 16/100
107/107 [==============================] - 5s 46ms/step - loss: 0.045
8 - accuracy: 0.9817 - val_loss: 0.1093 - val_accuracy: 0.9656
```

In [125]:
```
plot_model_performance(ann_lr,train_img,test_img,"accuracy", "val_accu
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.62      0.76       234
           1       0.81      0.99      0.89       390

    accuracy                           0.85       624
   macro avg       0.89      0.80      0.82       624
weighted avg       0.87      0.85      0.84       624



148/148 [==============================] - 2s 10ms/step - loss: 0.036
6 - accuracy: 0.9881
20/20 [==============================] - 0s 6ms/step - loss: 0.5738 -
accuracy: 0.8510
--------------------------
Final Train Loss: 0.0366
Final Test Loss: 0.5738
--------------------------
Final Train Acc: 0.9881
Final Test Acc: 0.851
```



2. Convolutional NN Model

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

- The **convolution layer** is the core building block of the CNN. It carries the main portion of the network's computational load. This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field.
- The **pooling layer** replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights.
- Neurons in the *fully connected layer* have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect. The FC layer helps to map the representation between the input and the output.

2a. Baseline CNN

- 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.

In [60]:
```
print(train_images.shape)
print(val_images.shape)
```

```
(4708, 128, 128, 3)
(524, 128, 128, 3)
```

In [61]:
```
cnn_model = Sequential()

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

# Flatten
cnn_model.add(Flatten())

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

# Compile model
cnn_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', me

cnn_model.summary()
```

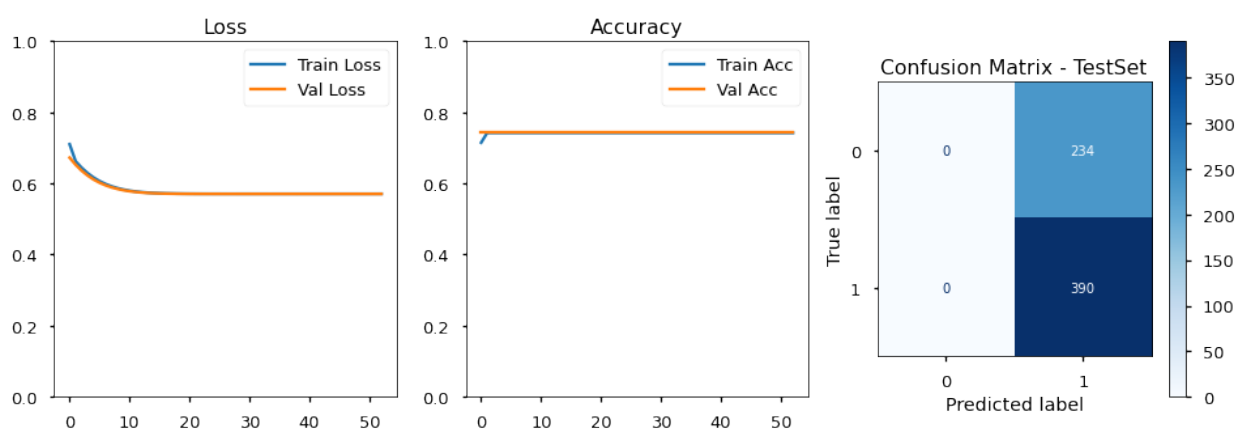
```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 8)       224
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 8)         0
_________________________________________________________________
flatten (Flatten)            (None, 31752)             0
_________________________________________________________________
dense_35 (Dense)             (None, 8)                 254024
_________________________________________________________________
dense_36 (Dense)             (None, 1)                 9
=================================================================
Total params: 254,257
Trainable params: 254,257
Non-trainable params: 0
_________________________________________________________________
```

In [62]:
```
baseline_cnn_model = cnn_model.fit(train_images, train_y,
                                   epochs = 100, batch_size = 44,
                                   verbose =1,callbacks=[early_stop],
                                   validation_data = (val_images, val_y))
```

```
Epoch 33/100
107/107 [==============================] - 5s 50ms/step - loss: 0.568
5 - acc: 0.7444 - val_loss: 0.5706 - val_acc: 0.7424
Epoch 34/100
107/107 [==============================] - 5s 50ms/step - loss: 0.574
9 - acc: 0.7383 - val_loss: 0.5706 - val_acc: 0.7424
Epoch 35/100
107/107 [==============================] - 5s 50ms/step - loss: 0.573
7 - acc: 0.7394 - val_loss: 0.5706 - val_acc: 0.7424
Epoch 36/100
107/107 [==============================] - 5s 50ms/step - loss: 0.581
2 - acc: 0.7323 - val_loss: 0.5706 - val_acc: 0.7424
Epoch 37/100
107/107 [==============================] - 5s 48ms/step - loss: 0.571
2 - acc: 0.7418 - val_loss: 0.5706 - val_acc: 0.7424
Epoch 38/100
107/107 [==============================] - 5s 48ms/step - loss: 0.564
5 - acc: 0.7481 - val_loss: 0.5706 - val_acc: 0.7424
Epoch 39/100
107/107 [==============================] - 5s 49ms/step - loss: 0.570
```

In [63]:
```
plot_model_performance(baseline_cnn_model,train_images,test_images, 'a
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       234
           1       0.62      1.00      0.77       390

    accuracy                           0.62       624
   macro avg       0.31      0.50      0.38       624
weighted avg       0.39      0.62      0.48       624



148/148 [==============================] - 1s 7ms/step - loss: 0.5708
- acc: 0.7421
20/20 [==============================] - 0s 8ms/step - loss: 0.6947 -
acc: 0.6250
-------------------------
Final Train Loss: 0.5708
Final Test Loss: 0.6947
-------------------------
Final Train Acc: 0.7421
Final Test Acc: 0.625
```



- This did not perform well either mis-labeling all the pneumonia cases

2b. Deeper CNN model with more layers

- An output layer with 1 neuron making the predictions.
- We will alternate convolutional and pooling layers
- Larger number of parameters in the later layers which will help to detect more abstract patterns
- Add some final dense layers to add a classifier to the convolutional base

In [64]:
```
cnn_model = Sequential()

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

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

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

# Flatten
cnn_model.add(Flatten())

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

# Compile model
cnn_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', me

cnn_model.summary()
```

```
Model: "sequential_9"
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 63, 63, 32)	0
conv2d_2 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 30, 30, 64)	0

```
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_37 (Dense)             (None, 128)               3211392
_________________________________________________________________
dense_38 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_39 (Dense)             (None, 1)                 65
=================================================================
Total params: 3,312,961
Trainable params: 3,312,961
Non-trainable params: 0
_________________________________________________________________
```

In [65]:

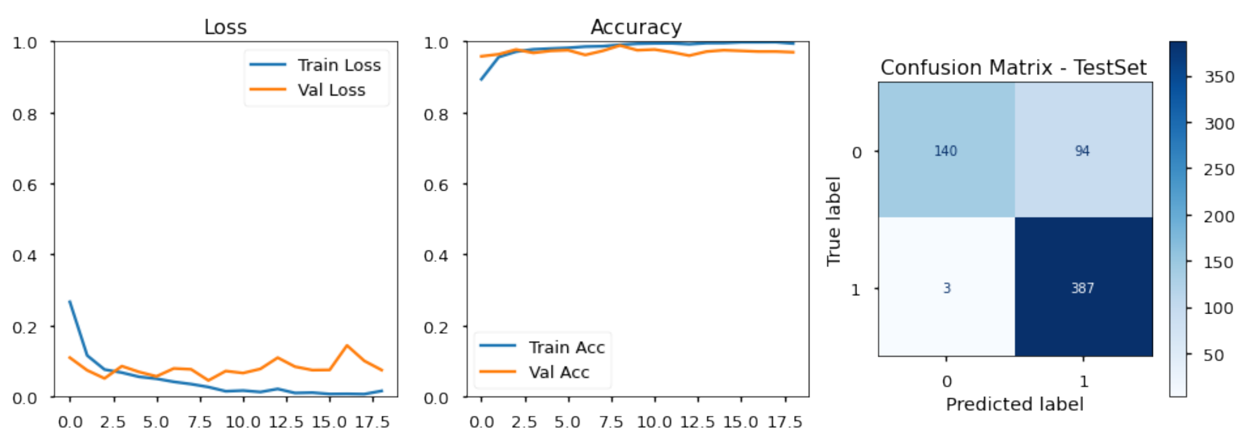
```
deper_cnn_model = cnn_model.fit(train_images, train_y,
                                epochs = 100, batch_size = 44,
                                verbose =1,callbacks=[early_stop],
                                validation_data = (val_images, val_y))
```

```
Epoch 1/100
107/107 [==============================] - 34s 312ms/step - loss: 0.4
303 - acc: 0.8282 - val_loss: 0.1097 - val_acc: 0.9580
Epoch 2/100
107/107 [==============================] - 33s 305ms/step - loss: 0.1
339 - acc: 0.9493 - val_loss: 0.0741 - val_acc: 0.9637
Epoch 3/100
107/107 [==============================] - 32s 301ms/step - loss: 0.0
787 - acc: 0.9678 - val_loss: 0.0513 - val_acc: 0.9771
Epoch 4/100
107/107 [==============================] - 32s 304ms/step - loss: 0.0
684 - acc: 0.9762 - val_loss: 0.0858 - val_acc: 0.9676
Epoch 5/100
107/107 [==============================] - 32s 303ms/step - loss: 0.0
617 - acc: 0.9764 - val_loss: 0.0694 - val_acc: 0.9733
Epoch 6/100
107/107 [==============================] - 32s 303ms/step - loss: 0.0
505 - acc: 0.9821 - val_loss: 0.0570 - val_acc: 0.9752
Epoch 7/100
107/107 [==============================] - 32s 303ms/step - loss: 0.0
430 - acc: 0.9850 - val_loss: 0.0793 - val_acc: 0.9618
Epoch 8/100
107/107 [==============================] - 32s 299ms/step - loss: 0.0
305 - acc: 0.9878 - val_loss: 0.0769 - val_acc: 0.9733
Epoch 9/100
107/107 [==============================] - 33s 308ms/step - loss: 0.0
203 - acc: 0.9924 - val_loss: 0.0455 - val_acc: 0.9885
```

```
Epoch 10/100
107/107 [==============================] - 32s 303ms/step - loss: 0.0
139 - acc: 0.9953 - val_loss: 0.0720 - val_acc: 0.9752
Epoch 11/100
107/107 [==============================] - 33s 305ms/step - loss: 0.0
163 - acc: 0.9944 - val_loss: 0.0663 - val_acc: 0.9771
Epoch 12/100
107/107 [==============================] - 32s 300ms/step - loss: 0.0
104 - acc: 0.9963 - val_loss: 0.0780 - val_acc: 0.9695
Epoch 13/100
107/107 [==============================] - 32s 303ms/step - loss: 0.0
202 - acc: 0.9910 - val_loss: 0.1096 - val_acc: 0.9599
Epoch 14/100
107/107 [==============================] - 32s 300ms/step - loss: 0.0
093 - acc: 0.9966 - val_loss: 0.0843 - val_acc: 0.9714
Epoch 15/100
107/107 [==============================] - 32s 301ms/step - loss: 0.0
054 - acc: 0.9983 - val_loss: 0.0747 - val_acc: 0.9752
Epoch 16/100
107/107 [==============================] - 32s 300ms/step - loss: 0.0
077 - acc: 0.9972 - val_loss: 0.0752 - val_acc: 0.9733
Epoch 17/100
107/107 [==============================] - 32s 304ms/step - loss: 0.0
066 - acc: 0.9979 - val_loss: 0.1441 - val_acc: 0.9714
Epoch 18/100
107/107 [==============================] - 32s 298ms/step - loss: 0.0
085 - acc: 0.9972 - val_loss: 0.1006 - val_acc: 0.9714
Epoch 19/100
107/107 [==============================] - 32s 301ms/step - loss: 0.0
077 - acc: 0.9977 - val_loss: 0.0749 - val_acc: 0.9695
```

In [66]:
```
plot_model_performance(deeper_cnn_model,train_images,test_images, 'acc
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.60      0.74       234
           1       0.80      0.99      0.89       390

    accuracy                           0.84       624
   macro avg       0.89      0.80      0.82       624
weighted avg       0.87      0.84      0.83       624


148/148 [==============================] - 8s 51ms/step - loss: 0.014
8 - acc: 0.9953
20/20 [==============================] - 1s 53ms/step - loss: 0.7067
- acc: 0.8446
--------------------------
Final Train Loss: 0.0148
Final Test Loss: 0.7067
--------------------------
Final Train Acc: 0.9953
Final Test Acc: 0.8446
```



- Even though the peneumonia prediction rate is 99%, only 3 cases are mis-labeled, but the normal cases are mis-labeled 40% of the time.
- There may be some overfitting as test and train accuracy differ by 15%.
- Let's implement dropout and see if that helps with overfitting

In [70]:
```
#cnn_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', m
#deeper_cnn_model2 = cnn_model.fit(train_images, train_y,
#                              epochs = 100, batch_size = 44,
#                              verbose =1,callbacks=[early_stopping],
#                              validation_data = (val_images, val_y))
```

In [69]:
```
#plot_model_performance(deeper_cnn_model2,train_images,test_images, 'a
```

CNN with Dropout regularization

In [71]:
```
cnn_model = Sequential()

# 1st Convolution and Pooling and dropout
cnn_model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128,
cnn_model.add(MaxPool2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.3)) # regularization, turn off 40% of the neur

# 2nd Convolution and Pooling
cnn_model.add(Conv2D(64, (3, 3), activation="relu"))
cnn_model.add(MaxPool2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.3)) # regularization

# 3rd Convolution and Pooling
cnn_model.add(Conv2D(128, (3, 3), activation="relu"))
cnn_model.add(MaxPool2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.3)) # regularization

# Flatten
cnn_model.add(Flatten())

cnn_model.add(Dense(activation = 'relu', units = 128)) # inner layer
cnn_model.add(Dropout(0.1)) # regularization
cnn_model.add(Dense(activation = 'relu', units = 64)) # inner layer
cnn_model.add(Dropout(0.1)) # regularization
cnn_model.add(Dense(activation = 'sigmoid', units = 1)) # output layer

cnn_model.save("Dropout_CNN")

# Compile model
cnn_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', me

cnn_model.summary()
```

```
INFO:tensorflow:Assets written to: Dropout_CNN/assets
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_8 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
flatten_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_40 (Dense)             (None, 128)               3211392
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_41 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_10 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_42 (Dense)             (None, 1)                 65
=================================================================
Total params: 3,312,961
Trainable params: 3,312,961
Non-trainable params: 0
_________________________________________________________________
```

In [72]:

```
dropout_cnn_model = cnn_model.fit(train_images, train_y,
                                  epochs = 100, batch_size = 44,
                                  verbose =1,callbacks=[early_stop],
                                  validation_data = (val_images, val_y))
```

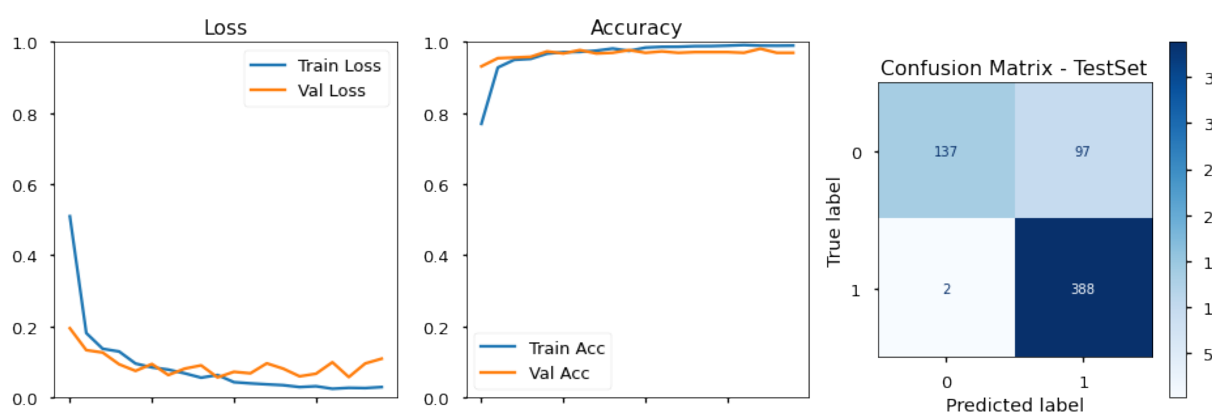
```
Epoch 1/100
107/107 [==============================] - 38s 345ms/step - loss: 0.6
392 - acc: 0.7139 - val_loss: 0.1949 - val_acc: 0.9313
Epoch 2/100
107/107 [==============================] - 37s 349ms/step - loss: 0.1
946 - acc: 0.9174 - val_loss: 0.1334 - val_acc: 0.9542
Epoch 3/100
107/107 [==============================] - 36s 340ms/step - loss: 0.1
467 - acc: 0.9434 - val_loss: 0.1266 - val_acc: 0.9561
```

```
Epoch 4/100
107/107 [==============================] - 37s 343ms/step - loss: 0.1
303 - acc: 0.9541 - val_loss: 0.0936 - val_acc: 0.9580
Epoch 5/100
107/107 [==============================] - 38s 351ms/step - loss: 0.1
006 - acc: 0.9667 - val_loss: 0.0750 - val_acc: 0.9733
Epoch 6/100
107/107 [==============================] - 38s 352ms/step - loss: 0.0
742 - acc: 0.9735 - val_loss: 0.0943 - val_acc: 0.9676
Epoch 7/100
107/107 [==============================] - 37s 345ms/step - loss: 0.0
696 - acc: 0.9748 - val_loss: 0.0634 - val_acc: 0.9771
Epoch 8/100
107/107 [==============================] - 36s 336ms/step - loss: 0.0
626 - acc: 0.9790 - val_loss: 0.0814 - val_acc: 0.9676
Epoch 9/100
107/107 [==============================] - 38s 353ms/step - loss: 0.0
537 - acc: 0.9795 - val_loss: 0.0906 - val_acc: 0.9695
Epoch 10/100
107/107 [==============================] - 36s 338ms/step - loss: 0.0
620 - acc: 0.9738 - val_loss: 0.0567 - val_acc: 0.9771
Epoch 11/100
107/107 [==============================] - 35s 330ms/step - loss: 0.0
560 - acc: 0.9792 - val_loss: 0.0725 - val_acc: 0.9695
Epoch 12/100
107/107 [==============================] - 36s 332ms/step - loss: 0.0
387 - acc: 0.9871 - val_loss: 0.0682 - val_acc: 0.9733
Epoch 13/100
107/107 [==============================] - 35s 327ms/step - loss: 0.0
402 - acc: 0.9861 - val_loss: 0.0963 - val_acc: 0.9695
Epoch 14/100
107/107 [==============================] - 36s 333ms/step - loss: 0.0
310 - acc: 0.9902 - val_loss: 0.0813 - val_acc: 0.9714
Epoch 15/100
107/107 [==============================] - 36s 332ms/step - loss: 0.0
316 - acc: 0.9874 - val_loss: 0.0598 - val_acc: 0.9714
Epoch 16/100
107/107 [==============================] - 35s 332ms/step - loss: 0.0
227 - acc: 0.9919 - val_loss: 0.0670 - val_acc: 0.9714
Epoch 17/100
107/107 [==============================] - 35s 332ms/step - loss: 0.0
262 - acc: 0.9895 - val_loss: 0.0994 - val_acc: 0.9695
Epoch 18/100
107/107 [==============================] - 35s 329ms/step - loss: 0.0
231 - acc: 0.9915 - val_loss: 0.0576 - val_acc: 0.9809
Epoch 19/100
107/107 [==============================] - 36s 332ms/step - loss: 0.0
228 - acc: 0.9919 - val_loss: 0.0959 - val_acc: 0.9695
Epoch 20/100
107/107 [==============================] - 36s 338ms/step - loss: 0.0
```

```
364 - acc: 0.9859 - val_loss: 0.1092 - val_acc: 0.9695
```

In [73]:
```
plot_model_performance(dropout_cnn_model,train_images,test_images, 'ac
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.59      0.73       234
           1       0.80      0.99      0.89       390

    accuracy                           0.84       624
   macro avg       0.89      0.79      0.81       624
weighted avg       0.87      0.84      0.83       624



148/148 [==============================] - 8s 53ms/step - loss: 0.029
1 - acc: 0.9904
20/20 [==============================] - 1s 56ms/step - loss: 0.6597
- acc: 0.8413
--------------------------
Final Train Loss: 0.0291
Final Test Loss: 0.6597
--------------------------
Final Train Acc: 0.9904
Final Test Acc: 0.8413
```



- We do not see any significant improvemement in this version of model.

3. Transfer Learning Approach: Models using pre-trained modules

- There are various pre-trained models that are now a days used for image classification. These as a part of transfer learning approach(https://keras.io/guides/transfer_learning/ (https://keras.io/guides/transfer_learning/)). Shown below is the list of most commonly used pre-trained modules:
  - Resnet18, Resnet34, Resnet50 and Resnet101
  - VGG16 and VGG19
  - EfficientNet
  - Inception
  - Xception

Here is the list of steps that we will follow to use these models

- Instantiate a base model and load pre-trained weights into it.
- Freeze all layers in the base model by setting trainable = False.
- Change the image size from the input layer so we can use the model on our images: (128, 128, 3)
- Create a new model on top of the output of one (or several) layers from the base model.
- Add a global pooling layer = 'avg' rather than flattening the image.
- Train this new model on our dataset.

In this exercise we will use Resnet101 and Xception

3a. Xception

In [74]:
```
from keras.applications import Xception
```

In [75]:
```
Xception().summary()
```

```
k5_sepconv2_act[0][0]

__________________________________________________________________________________________________
block5_sepconv2_bn (BatchNormal (None, 19, 19, 728)  2912        bloc
k5_sepconv2[0][0]

__________________________________________________________________________________________________
block5_sepconv3_act (Activation (None, 19, 19, 728)  0           bloc
k5_sepconv2_bn[0][0]

__________________________________________________________________________________________________
block5_sepconv3 (SeparableConv2 (None, 19, 19, 728)  536536      bloc
k5_sepconv3_act[0][0]

__________________________________________________________________________________________________
block5_sepconv3_bn (BatchNormal (None, 19, 19, 728)  2912        bloc
k5_sepconv3[0][0]

__________________________________________________________________________________________________
add 3 (Add)                     (None. 19. 19. 728)  0           bloc
```

In [76]:
```
base_model = keras.applications.Xception(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(128, 128, 3),
    include_top=False)  # Do not include the ImageNet classifier at th
base_model.trainable = False
base_model.summary()
```

```
13[0][0]

__________________________________________________________________________________________________
block4_pool (MaxPooling2D)      (None, 8, 8, 728)    0           bloc
k4_sepconv2_bn[0][0]

__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 8, 8, 728)    2912        conv
2d_13[0][0]

__________________________________________________________________________________________________
add_14 (Add)                    (None, 8, 8, 728)    0           bloc
k4_pool[0][0]
                                                                 batc
h_normalization_6[0][0]

__________________________________________________________________________________________________
block5_sepconv1_act (Activation (None, 8, 8, 728)    0           add_
14[0][0]

__________________________________________________________________________________________________
```

In [77]:
```
inputs = keras.Input(shape=(128, 128, 3))
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as y
# learn in a few paragraphs.
x = base_model(inputs, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = keras.layers.GlobalAveragePooling2D()(x)
# A Dense classifier with a single unit (binary classification)
outputs = keras.layers.Dense(1,activation = "sigmoid")(x)
transfer_model = keras.Model(inputs, outputs)

# Add the fully connected layers
#transfer_model.add(Dense(1, activation = "sigmoid"))

transfer_model.summary()
transfer_model.save("XceptionD");
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
xception (Functional)        (None, 4, 4, 2048)        20861480
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense_43 (Dense)             (None, 1)                 2049
=================================================================
Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480
_________________________________________________________________
INFO:tensorflow:Assets written to: XceptionD/assets
```

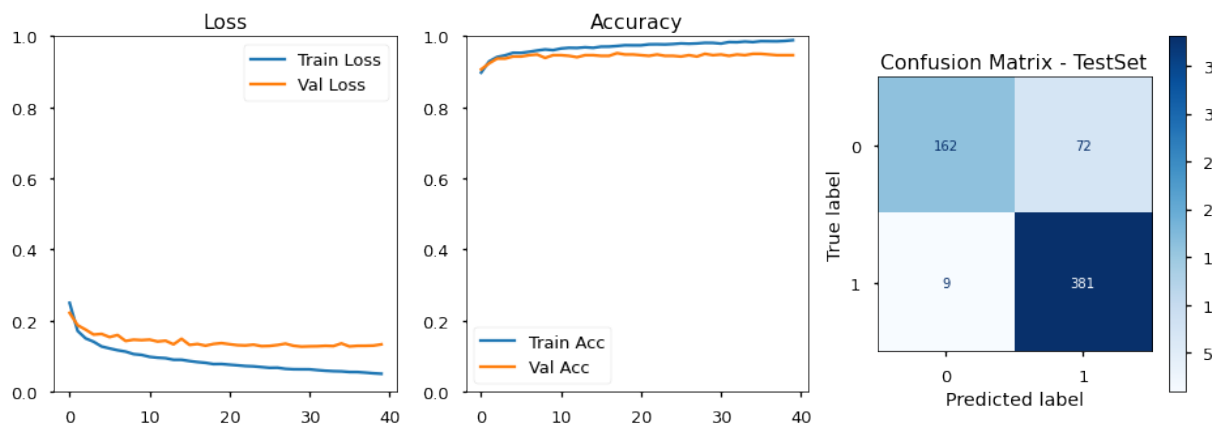
In [78]:
```
transfer_model.compile(optimizer = "adam", loss = "binary_crossentropy
```

In [79]:
```
xception_model = transfer_model.fit(train_images, train_y,
                                    epochs = 100, batch_size = 44,
                                    verbose =1,callbacks=[early_stop],
                                    validation_data = (val_images, val_y))
```

```
Epoch 34/100
107/107 [==============================] - 52s 487ms/step - loss: 0.0
500 - accuracy: 0.9892 - val_loss: 0.1287 - val_accuracy: 0.9466
Epoch 35/100
107/107 [==============================] - 52s 490ms/step - loss: 0.0
583 - accuracy: 0.9839 - val_loss: 0.1362 - val_accuracy: 0.9504
Epoch 36/100
107/107 [==============================] - 53s 493ms/step - loss: 0.0
582 - accuracy: 0.9826 - val_loss: 0.1277 - val_accuracy: 0.9504
Epoch 37/100
107/107 [==============================] - 53s 494ms/step - loss: 0.0
531 - accuracy: 0.9894 - val_loss: 0.1293 - val_accuracy: 0.9485
Epoch 38/100
107/107 [==============================] - 53s 493ms/step - loss: 0.0
497 - accuracy: 0.9879 - val_loss: 0.1294 - val_accuracy: 0.9466
Epoch 39/100
107/107 [==============================] - 52s 489ms/step - loss: 0.0
492 - accuracy: 0.9880 - val_loss: 0.1300 - val_accuracy: 0.9466
Epoch 40/100
107/107 [==============================] - 52s 490ms/step - loss: 0.0
```

In [80]:
```
plot_model_performance(xception_model,train_images,test_images, 'accur
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.69      0.80       234
           1       0.84      0.98      0.90       390

    accuracy                           0.87       624
   macro avg       0.89      0.83      0.85       624
weighted avg       0.88      0.87      0.86       624



148/148 [==============================] - 46s 310ms/step - loss: 0.0
581 - accuracy: 0.9843
20/20 [==============================] - 6s 296ms/step - loss: 0.3529
- accuracy: 0.8702
--------------------------
Final Train Loss: 0.0581
Final Test Loss: 0.3529
--------------------------
Final Train Acc: 0.9843
Final Test Acc: 0.8702
```



In [81]:

```
inputs = keras.Input(shape=(128, 128, 3))
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as y
# learn in a few paragraphs.
x = base_model(inputs, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = keras.layers.GlobalAveragePooling2D()(x)
# A Dense classifier with a single unit (binary classification)
outputs = keras.layers.Dense(1,activation = "sigmoid")(x)
transfer_model = keras.Model(inputs, outputs)

model = Sequential()
model.add(transfer_model)

# 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("XceptionD_deep");
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
model_1 (Functional)        (None, 1)                 20863529

dense_45 (Dense)            (None, 128)               256

dropout_11 (Dropout)        (None, 128)               0

dense_46 (Dense)            (None, 64)                8256

dropout_12 (Dropout)        (None, 64)                0

dense_47 (Dense)            (None, 1)                 65

=================================================================
Total params: 20,872,106
Trainable params: 10,626
Non-trainable params: 20,861,480
_________________________________________________________________
INFO:tensorflow:Assets written to: XceptionD_deep/assets
```

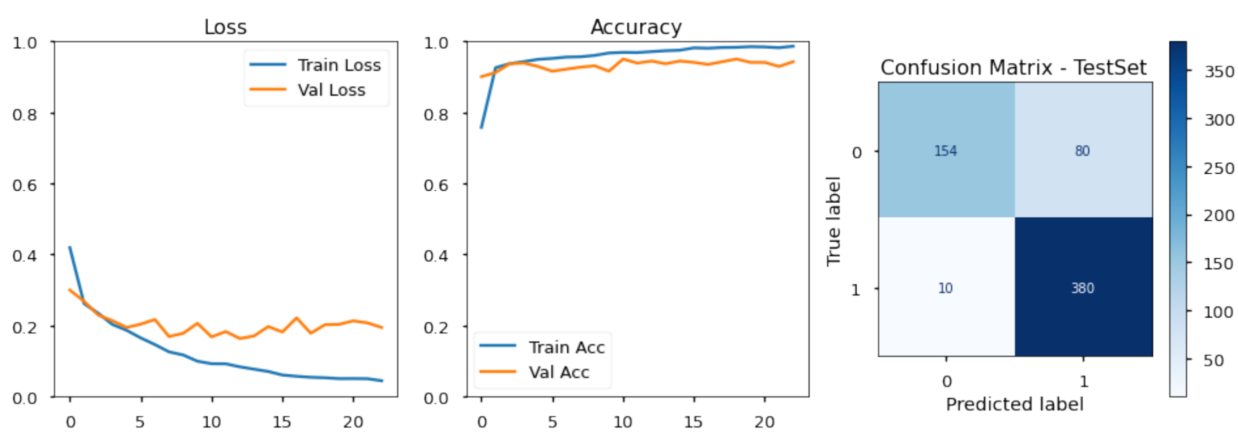
In [82]:
```
model.compile(optimizer = "adam", loss = "binary_crossentropy", metric
```

In [83]:
```
xception_model_deep = model.fit(train_images, train_y,
                                 epochs = 100, batch_size = 44,
                                 verbose =1,callbacks=[early_stop],
                                 validation_data = (val_images, val_y))
```

```
107/107 [==============================] - 53s 493ms/step - loss: 0.0
542 - accuracy: 0.9822 - val_loss: 0.2216 - val_accuracy: 0.9351
Epoch 18/100
107/107 [==============================] - 53s 496ms/step - loss: 0.0
507 - accuracy: 0.9848 - val_loss: 0.1785 - val_accuracy: 0.9427
Epoch 19/100
107/107 [==============================] - 53s 498ms/step - loss: 0.0
511 - accuracy: 0.9849 - val_loss: 0.2024 - val_accuracy: 0.9504
Epoch 20/100
107/107 [==============================] - 53s 495ms/step - loss: 0.0
496 - accuracy: 0.9851 - val_loss: 0.2035 - val_accuracy: 0.9408
Epoch 21/100
107/107 [==============================] - 54s 501ms/step - loss: 0.0
425 - accuracy: 0.9884 - val_loss: 0.2133 - val_accuracy: 0.9408
Epoch 22/100
107/107 [==============================] - 54s 500ms/step - loss: 0.0
474 - accuracy: 0.9828 - val_loss: 0.2076 - val_accuracy: 0.9294
Epoch 23/100
107/107 [==============================] - 53s 497ms/step - loss: 0.0
460 - accuracy: 0.9857 - val_loss: 0.1947 - val_accuracy: 0.9427
```

In [84]:
```
plot_model_performance(xception_model_deep,train_images,test_images, '
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.66      0.77       234
           1       0.83      0.97      0.89       390

    accuracy                           0.86       624
   macro avg       0.88      0.82      0.83       624
weighted avg       0.87      0.86      0.85       624


148/148 [==============================] - 47s 316ms/step - loss: 0.0
610 - accuracy: 0.9807
20/20 [==============================] - 6s 317ms/step - loss: 0.5213
- accuracy: 0.8558
--------------------------
Final Train Loss: 0.061
Final Test Loss: 0.5213
--------------------------
Final Train Acc: 0.9807
Final Test Acc: 0.8558
```



3b. RESNET101

In [85]:
```
from keras.applications.resnet import ResNet101
ResNet101().summary()
```

```
______________________________
conv2_block1_2_conv (Conv2D)    (None, 56, 56, 64)   36928       conv
2_block1_1_relu[0][0]

__________________________________________________________________
______________________________
conv2_block1_2_bn (BatchNormali (None, 56, 56, 64)   256         conv
2_block1_2_conv[0][0]

__________________________________________________________________
______________________________
conv2_block1_2_relu (Activation (None, 56, 56, 64)   0           conv
2_block1_2_bn[0][0]

__________________________________________________________________
______________________________
conv2_block1_0_conv (Conv2D)    (None, 56, 56, 256)  16640       pool
1_pool[0][0]

__________________________________________________________________
______________________________
conv2_block1_3_conv (Conv2D)    (None, 56, 56, 256)  16640       conv
2_block1_2_relu[0][0]
```

In [86]:
```
resnet_model = keras.applications.ResNet101(weights = "imagenet",
                                            input_shape = (128, 128, 3
                                            pooling="avg", include_top
                                            classes = 2)

resnet_model.trainable = False
resnet_model.summary()
```

```
2_block1_2_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_2_relu (Activation (None, 32, 32, 64)   0           conv
2_block1_2_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_0_conv (Conv2D)    (None, 32, 32, 256)  16640       pool
1_pool[0][0]
__________________________________________________________________________________________________
conv2_block1_3_conv (Conv2D)    (None, 32, 32, 256)  16640       conv
2_block1_2_relu[0][0]
__________________________________________________________________________________________________
conv2_block1_0_bn (BatchNormali (None, 32, 32, 256)  1024        conv
2_block1_0_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_3_bn (BatchNormali (None, 32, 32, 256)  1024        conv
```

In [87]:
```
RESmodel = Sequential()
RESmodel.add(resnet_model)

# Add the fully connected layers
RESmodel.add(Dense(1, activation = "sigmoid"))

RESmodel.summary()
RESmodel.save("RESNET101");
RESmodel.compile(optimizer = "adam", loss = "binary_crossentropy", met
```

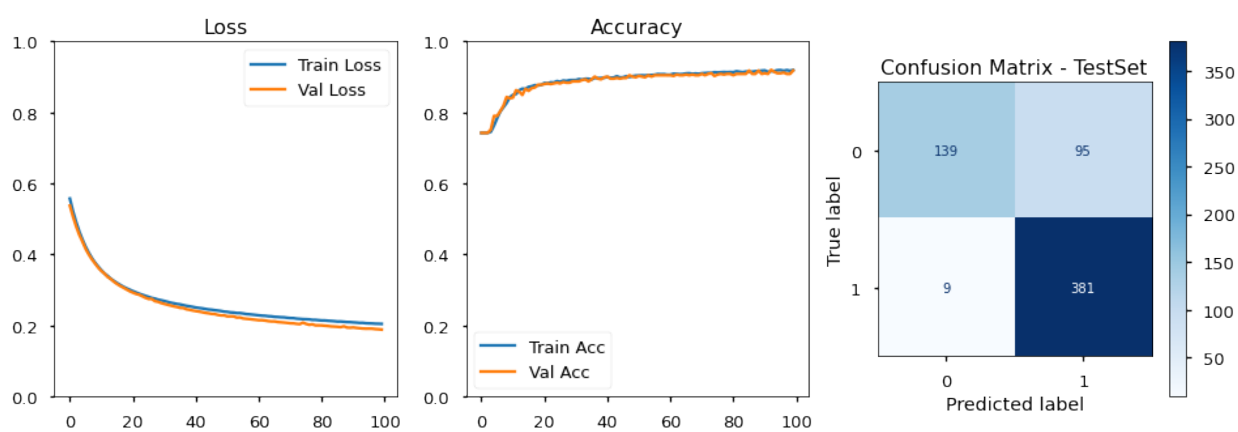
```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
resnet101 (Functional)      (None, 2048)              42658176

_________________________________________________________________
dense_48 (Dense)            (None, 1)                 2049
=================================================================
Total params: 42,660,225
Trainable params: 2,049
Non-trainable params: 42,658,176
_________________________________________________________________
INFO:tensorflow:Assets written to: RESNET101/assets
```

In [88]:
```
resnet101_model = RESmodel.fit(train_images, train_y,
                               epochs = 100, batch_size = 44,
                               verbose =1,callbacks=[early_stop],
                               validation_data = (val_images, val_y))
```

```
107/107 [==============================] - 118s 1s/step - loss: 0.208
7 - accuracy: 0.9160 - val_loss: 0.1926 - val_accuracy: 0.9198
Epoch 94/100
107/107 [==============================] - 117s 1s/step - loss: 0.200
1 - accuracy: 0.9242 - val_loss: 0.1917 - val_accuracy: 0.9141
Epoch 95/100
107/107 [==============================] - 115s 1s/step - loss: 0.211
2 - accuracy: 0.9213 - val_loss: 0.1913 - val_accuracy: 0.9103
Epoch 96/100
107/107 [==============================] - 116s 1s/step - loss: 0.197
9 - accuracy: 0.9221 - val_loss: 0.1915 - val_accuracy: 0.9103
Epoch 97/100
107/107 [==============================] - 121s 1s/step - loss: 0.199
8 - accuracy: 0.9180 - val_loss: 0.1912 - val_accuracy: 0.9122
Epoch 98/100
107/107 [==============================] - 116s 1s/step - loss: 0.209
8 - accuracy: 0.9174 - val_loss: 0.1902 - val_accuracy: 0.9084
Epoch 99/100
107/107 [==============================] - 115s 1s/step - loss: 0.197
2 - accuracy: 0.9192 - val_loss: 0.1896 - val_accuracy: 0.9084
```

In [89]:
```
plot_model_performance(resnet101_model,train_images,test_images, 'accu
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.59      0.73       234
           1       0.80      0.98      0.88       390

    accuracy                           0.83       624
   macro avg       0.87      0.79      0.80       624
weighted avg       0.85      0.83      0.82       624



148/148 [==============================] - 109s 734ms/step - loss: 0.
2038 - accuracy: 0.9182
20/20 [==============================] - 14s 695ms/step - loss: 0.378
2 - accuracy: 0.8333
--------------------------
Final Train Loss: 0.2038
Final Test Loss: 0.3782
--------------------------
Final Train Acc: 0.9182
Final Test Acc: 0.8333
```



- Again we have better identification for pneumonia cases but normal cases get mis-labelled 40% of the time.
- I would like to experiment data augmentation and see if it helps improve model performance.
- The simplest Xception model performed better in the category of transfer learning models, so I would only use for Data Augmentation studies

4. Data Augmentation

In [90]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.3,
                                   zoom_range=0.1,
                                   horizontal_flip=False)
```

In [91]:
```
# Since we have 4708 images in train set, I set a batch size of 214(di
#will need 22 steps in our model training to go over all 4708 images
train_generator = train_datagen.flow_from_directory(
        train_folder,
        target_size=(128, 128),
        batch_size = 214, #128
        class_mode='binary')
# get all the data in the directory split/validation  and reshape them
# we have 524 images, so choosing 131 (divisor of 524) and we will hav
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory
        val_folder,
        target_size=(128, 128),
        batch_size = 131,#20
        class_mode='binary')

# get all the data in the directory split/train , and reshape them
# I keep it at 624 same as the number of images in test dataset
test_generator = ImageDataGenerator(rescale=1./255).flow_from_director
        test_folder,
        target_size=(128, 128),
        batch_size = 624, #156#20
        class_mode='binary')
```

```
Found 4708 images belonging to 2 classes.
Found 524 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

In [148]:
```
Xtest_aug, ytest_aug = next(test_generator)
Xtrain_aug, ytrain_aug = next(train_generator)
Xval_aug, yval_aug = next(val_generator)
```

In [149]:
```
Xtrain_aug_v = Xtrain_aug.reshape(Xtrain_aug.shape[0], -1)
Xtest_aug_v  = Xtest_aug.reshape(Xtest_aug.shape[0], -1)
Xval_aug_v   = Xval_aug.reshape(Xval_aug.shape[0], -1)
```

In [151]:
```
Xtrain_aug_v.shape
```

Out[151]:
```
(214, 49152)
```

In [152]:
```
ytrain_aug_v = np.reshape(ytrain_aug, (ytrain_aug.shape[0],1))
ytest_aug_v = np.reshape(ytest_aug, (ytest_aug.shape[0],1))
yval_aug_v = np.reshape(yval_aug, (yval_aug.shape[0],1))
```

In [94]:
```
inputs = keras.Input(shape=(128, 128, 3))
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as y
# learn in a few paragraphs.
x = base_model(inputs, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = keras.layers.GlobalAveragePooling2D()(x)
# A Dense classifier with a single unit (binary classification)
outputs = keras.layers.Dense(1,activation = "sigmoid")(x)
transfer_model = keras.Model(inputs, outputs)

# Add the fully connected layers
#transfer_model.add(Dense(1, activation = "sigmoid"))

transfer_model.summary()
#transfer_model.save("XceptionD");
transfer_model.compile(optimizer = "adam", loss = "binary_crossentropy
```

```
Model: "model_2"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_7 (InputLayer)        [(None, 128, 128, 3)]     0
_________________________________________________________________
xception (Functional)       (None, 4, 4, 2048)        20861480
_________________________________________________________________
global_average_pooling2d_2 ( (None, 2048)             0
_________________________________________________________________
dense_49 (Dense)            (None, 1)                 2049
=================================================================
Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480
_________________________________________________________________
```

In [95]:
```
xception_aug = transfer_model.fit(train_generator,
                              steps_per_epoch=22, #25
                              epochs=100,
                              validation_data=val_generator,
                              validation_steps=4, callbacks=[early_sto
```

```
22/22 [==============================] - 82s 4s/step - loss: 0.1638 -
accuracy: 0.9337 - val_loss: 0.1942 - val_accuracy: 0.9370
Epoch 26/100
22/22 [==============================] - 82s 4s/step - loss: 0.1598 -
accuracy: 0.9348 - val_loss: 0.2315 - val_accuracy: 0.9141
Epoch 27/100
22/22 [==============================] - 82s 4s/step - loss: 0.1607 -
accuracy: 0.9368 - val_loss: 0.2078 - val_accuracy: 0.9294
Epoch 28/100
22/22 [==============================] - 82s 4s/step - loss: 0.1478 -
accuracy: 0.9455 - val_loss: 0.2337 - val_accuracy: 0.9179
Epoch 29/100
22/22 [==============================] - 83s 4s/step - loss: 0.1661 -
accuracy: 0.9405 - val_loss: 0.1866 - val_accuracy: 0.9351
Epoch 30/100
22/22 [==============================] - 85s 4s/step - loss: 0.1692 -
accuracy: 0.9320 - val_loss: 0.1962 - val_accuracy: 0.9294
Epoch 31/100
22/22 [==============================] - 86s 4s/step - loss: 0.1603 -
accuracy: 0.9383 - val_loss: 0.1990 - val_accuracy: 0.9332
```

In [96]:
```
#test_loss, test_acc = xception_aug.model.evaluate(test_generator, ste
test_loss, test_acc = xception_aug.model.evaluate(Xtest_aug, ytest_aug
print('test acc:', test_acc)
```

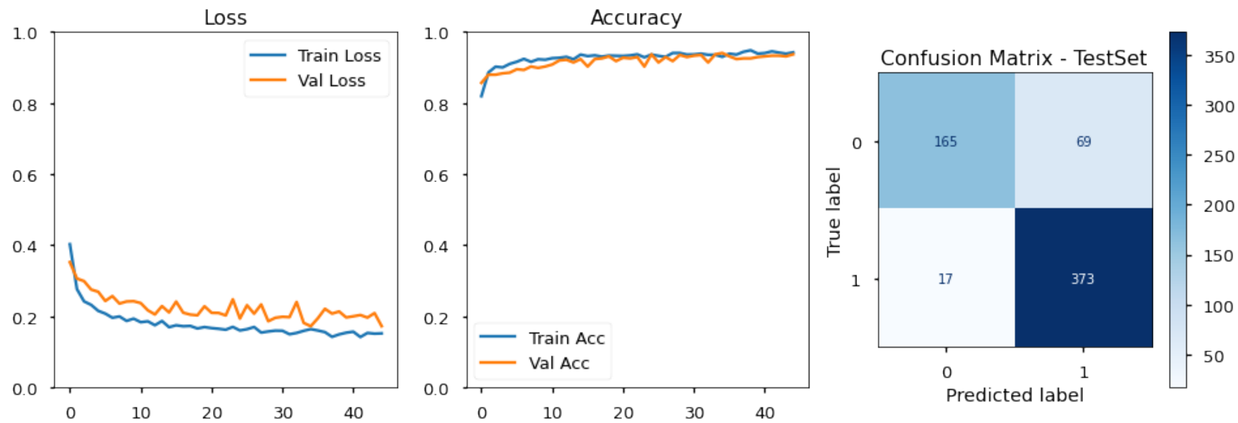
```
20/20 [==============================] - 7s 314ms/step - loss: 0.2888
- accuracy: 0.8622
test acc: 0.8621794581413269
```

In [105]:

```
 Model evaluation function for augmented data
ef aug_model_performance(Model, Xtrain, Xtest, Acc, Val_acc):

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

       # Diplay train and validation loss and accuracy:
       fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16,5))
       ax1.plot(Model.history['loss'])
       ax1.plot(Model.history['val_loss'])
       ax1.set_title("Loss")
       ax1.legend(labels = ['Train Loss', 'Val Loss'])
       ax1.set_ylim(0,1)
       ax2.plot(Model.history[Acc])
       ax2.plot(Model.history[Val_acc])
       ax2.legend(labels = ['Train Acc', 'Val Acc'])
       ax2.set_title('Accuracy')
       ax2.set_ylim(0,1)

       # Output (probability) predictions for the test set
       y_hat_test = Model.model.predict(Xtest_aug)
       y_pred = np.rint(y_hat_test).astype(np.int) # Round elements of
       y_true = ytest_aug.astype(np.int)

       # Generate a confusion matrix displaying the predictive accuracy
       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
       print('Classification Report:')
       print(classification_report(y_true, y_pred))
       print('\n')

       # Print final train and test loss and accuracy:
       train_loss, train_acc = Model.model.evaluate(Xtrain, ytrain_aug
       test_loss, test_acc = Model.model.evaluate(Xtest, ytest_aug);
       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')
```

In [98]:
```
aug_model_performance(xception_aug,Xtrain_aug,Xtest_aug, 'accuracy','v
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.71      0.79       234
           1       0.84      0.96      0.90       390

    accuracy                           0.86       624
   macro avg       0.88      0.83      0.84       624
weighted avg       0.87      0.86      0.86       624



7/7 [==============================] - 2s 307ms/step - loss: 0.1682 - 
accuracy: 0.9393
20/20 [==============================] - 6s 315ms/step - loss: 0.2888
- accuracy: 0.8622
--------------------------
Final Train Loss: 0.1682
Final Test Loss: 0.2888
--------------------------
Final Train Acc: 0.9393
Final Test Acc: 0.8622
```



In [99]:
```
#plot_model_performance(xception_aug,train_images,test_images, 'accura
```

In [100]:
```

```

```
inputs = keras.Input(shape=(128, 128, 3))
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning
x = base_model(inputs, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = keras.layers.GlobalAveragePooling2D()(x)
# A Dense classifier with a single unit (binary classification)
outputs = keras.layers.Dense(1,activation = "sigmoid")(x)
transfer_model = keras.Model(inputs, outputs)

model = Sequential()
model.add(transfer_model)
#transfer_model.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("XceptionD_deep_aug");
```

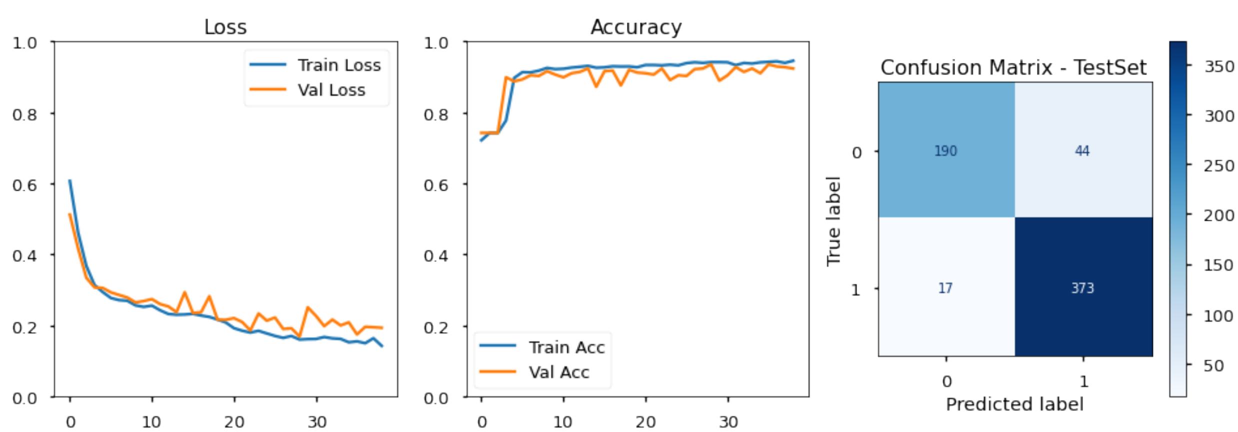
```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
model_3 (Functional)        (None, 1)                 20863529
_________________________________________________________________
dense_51 (Dense)            (None, 128)               256
_________________________________________________________________
dropout_13 (Dropout)        (None, 128)               0
_________________________________________________________________
dense_52 (Dense)            (None, 64)                8256
_________________________________________________________________
dropout_14 (Dropout)        (None, 64)                0
_________________________________________________________________
dense_53 (Dense)            (None, 1)                 65
=================================================================
Total params: 20,872,106
Trainable params: 10,626
Non-trainable params: 20,861,480
_________________________________________________________________
INFO:tensorflow:Assets written to: XceptionD_deep_aug/assets
```

In [101]:
```
model.compile(optimizer = "adam", loss = "binary_crossentropy", metric
xception_aug_deep = model.fit(train_generator,
                              steps_per_epoch=22,
                              epochs=100,
                              validation_data=val_generator,
                              validation_steps=4, callbacks=[early_stc
```

```
Epoch 33/100
22/22 [==============================] - 83s 4s/step - loss: 0.1642 -
accuracy: 0.9379 - val_loss: 0.2167 - val_accuracy: 0.9141
Epoch 34/100
22/22 [==============================] - 82s 4s/step - loss: 0.1553 -
accuracy: 0.9403 - val_loss: 0.2005 - val_accuracy: 0.9237
Epoch 35/100
22/22 [==============================] - 80s 4s/step - loss: 0.1551 -
accuracy: 0.9402 - val_loss: 0.2091 - val_accuracy: 0.9103
Epoch 36/100
22/22 [==============================] - 80s 4s/step - loss: 0.1506 -
accuracy: 0.9464 - val_loss: 0.1751 - val_accuracy: 0.9351
Epoch 37/100
22/22 [==============================] - 80s 4s/step - loss: 0.1460 -
accuracy: 0.9485 - val_loss: 0.1965 - val_accuracy: 0.9294
Epoch 38/100
22/22 [==============================] - 80s 4s/step - loss: 0.1556 -
accuracy: 0.9442 - val_loss: 0.1953 - val_accuracy: 0.9275
Epoch 39/100
22/22 [==============================] - 80s 4s/step - loss: 0.1410 -
```

In [102]:
```
plot_model_performance(xception_aug_deep,train_images,test_images, 'ac
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.81      0.86       234
           1       0.89      0.96      0.92       390

    accuracy                           0.90       624
   macro avg       0.91      0.88      0.89       624
weighted avg       0.90      0.90      0.90       624


148/148 [==============================] - 49s 326ms/step - loss: 0.1
630 - accuracy: 0.9363
20/20 [==============================] - 6s 305ms/step - loss: 0.2715
- accuracy: 0.9022
--------------------------
Final Train Loss: 0.163
Final Test Loss: 0.2715
--------------------------
Final Train Acc: 0.9363
Final Test Acc: 0.9022
```



- The model performance has improved and test and train accuracies are much closer indicating no overfitting

CNN model with Augmented data:

- For completeness/curosity lets check how the CNN deeper model performs on the augmented data

In [192]:
```
cnn_model = Sequential()

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

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

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

# Flatten
cnn_model.add(Flatten())

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

# Compile model
cnn_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', me
#cnn_model.summary()
cnn_model.save("CNN_DEEP_AUG");
```

```
INFO:tensorflow:Assets written to: CNN_DEEP_AUG/assets
```

In [122]:
```
aug_cnn_deep = cnn_model.fit(train_generator,
                              steps_per_epoch=22,
                              epochs=100,
                              validation_data=val_generator,
                              validation_steps=4, callbacks=[early_stc
```

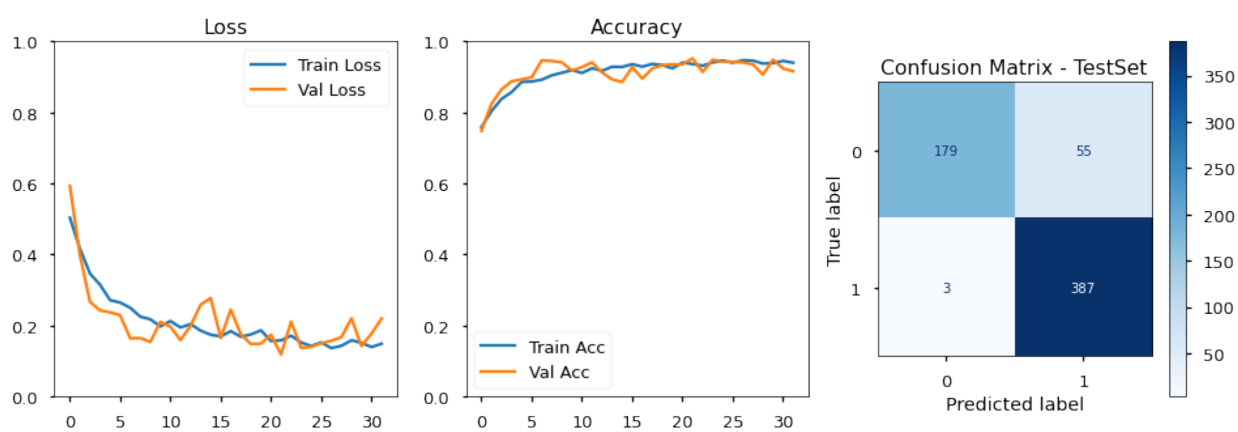
```
Epoch 1/100
22/22 [==============================] - 73s 3s/step - loss: 0.5034 -
acc: 0.7583 - val_loss: 0.5933 - val_acc: 0.7481
Epoch 2/100
22/22 [==============================] - 72s 3s/step - loss: 0.4165 -
acc: 0.8037 - val_loss: 0.3982 - val_acc: 0.8244
Epoch 3/100
```

```
22/22 [==============================] - 71s 3s/step - loss: 0.3463 -
acc: 0.8379 - val_loss: 0.2668 - val_acc: 0.8645
Epoch 4/100
22/22 [==============================] - 71s 3s/step - loss: 0.3149 -
acc: 0.8571 - val_loss: 0.2428 - val_acc: 0.8874
Epoch 5/100
22/22 [==============================] - 72s 3s/step - loss: 0.2715 -
acc: 0.8857 - val_loss: 0.2373 - val_acc: 0.8931
Epoch 6/100
22/22 [==============================] - 73s 3s/step - loss: 0.2648 -
acc: 0.8870 - val_loss: 0.2291 - val_acc: 0.8989
Epoch 7/100
22/22 [==============================] - 71s 3s/step - loss: 0.2501 -
acc: 0.8921 - val_loss: 0.1643 - val_acc: 0.9466
Epoch 8/100
22/22 [==============================] - 70s 3s/step - loss: 0.2252 -
acc: 0.9044 - val_loss: 0.1646 - val_acc: 0.9447
Epoch 9/100
22/22 [==============================] - 70s 3s/step - loss: 0.2177 -
acc: 0.9114 - val_loss: 0.1540 - val_acc: 0.9408
Epoch 10/100
22/22 [==============================] - 71s 3s/step - loss: 0.1976 -
acc: 0.9191 - val_loss: 0.2103 - val_acc: 0.9179
Epoch 11/100
22/22 [==============================] - 71s 3s/step - loss: 0.2127 -
acc: 0.9110 - val_loss: 0.1968 - val_acc: 0.9275
Epoch 12/100
22/22 [==============================] - 71s 3s/step - loss: 0.1946 -
acc: 0.9242 - val_loss: 0.1592 - val_acc: 0.9408
Epoch 13/100
22/22 [==============================] - 70s 3s/step - loss: 0.2043 -
acc: 0.9172 - val_loss: 0.1992 - val_acc: 0.9122
Epoch 14/100
22/22 [==============================] - 70s 3s/step - loss: 0.1857 -
acc: 0.9284 - val_loss: 0.2586 - val_acc: 0.8931
Epoch 15/100
22/22 [==============================] - 70s 3s/step - loss: 0.1741 -
acc: 0.9282 - val_loss: 0.2776 - val_acc: 0.8855
Epoch 16/100
22/22 [==============================] - 70s 3s/step - loss: 0.1695 -
acc: 0.9356 - val_loss: 0.1652 - val_acc: 0.9275
Epoch 17/100
22/22 [==============================] - 72s 3s/step - loss: 0.1843 -
acc: 0.9288 - val_loss: 0.2452 - val_acc: 0.8950
Epoch 18/100
22/22 [==============================] - 71s 3s/step - loss: 0.1686 -
acc: 0.9365 - val_loss: 0.1756 - val_acc: 0.9237
Epoch 19/100
22/22 [==============================] - 71s 3s/step - loss: 0.1750 -
acc: 0.9331 - val_loss: 0.1481 - val_acc: 0.9332
```

```
Epoch 20/100
22/22 [==============================] - 70s 3s/step - loss: 0.1867 -
acc: 0.9242 - val_loss: 0.1492 - val_acc: 0.9351
Epoch 21/100
22/22 [==============================] - 71s 3s/step - loss: 0.1563 -
acc: 0.9403 - val_loss: 0.1742 - val_acc: 0.9351
Epoch 22/100
22/22 [==============================] - 70s 3s/step - loss: 0.1584 -
acc: 0.9363 - val_loss: 0.1187 - val_acc: 0.9523
Epoch 23/100
22/22 [==============================] - 71s 3s/step - loss: 0.1716 -
acc: 0.9316 - val_loss: 0.2107 - val_acc: 0.9141
Epoch 24/100
22/22 [==============================] - 73s 3s/step - loss: 0.1522 -
acc: 0.9407 - val_loss: 0.1372 - val_acc: 0.9485
Epoch 25/100
22/22 [==============================] - 72s 3s/step - loss: 0.1424 -
acc: 0.9458 - val_loss: 0.1401 - val_acc: 0.9427
Epoch 26/100
22/22 [==============================] - 72s 3s/step - loss: 0.1527 -
acc: 0.9401 - val_loss: 0.1503 - val_acc: 0.9427
Epoch 27/100
22/22 [==============================] - 71s 3s/step - loss: 0.1365 -
acc: 0.9469 - val_loss: 0.1571 - val_acc: 0.9408
Epoch 28/100
22/22 [==============================] - 74s 3s/step - loss: 0.1431 -
acc: 0.9456 - val_loss: 0.1673 - val_acc: 0.9351
Epoch 29/100
22/22 [==============================] - 72s 3s/step - loss: 0.1586 -
acc: 0.9376 - val_loss: 0.2202 - val_acc: 0.9065
Epoch 30/100
22/22 [==============================] - 72s 3s/step - loss: 0.1513 -
acc: 0.9390 - val_loss: 0.1423 - val_acc: 0.9485
Epoch 31/100
22/22 [==============================] - 74s 3s/step - loss: 0.1397 -
acc: 0.9450 - val_loss: 0.1777 - val_acc: 0.9237
Epoch 32/100
22/22 [==============================] - 71s 3s/step - loss: 0.1490 -
acc: 0.9399 - val_loss: 0.2196 - val_acc: 0.9160
```

In [124]:
```
plot_model_performance(aug_cnn_deep,train_images,test_images, 'acc','v
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.76      0.86       234
           1       0.88      0.99      0.93       390

    accuracy                           0.91       624
   macro avg       0.93      0.88      0.90       624
weighted avg       0.92      0.91      0.90       624



148/148 [==============================] - 9s 63ms/step - loss: 0.106
6 - acc: 0.9571
20/20 [==============================] - 1s 59ms/step - loss: 0.2481
- acc: 0.9071
--------------------------
Final Train Loss: 0.1066
Final Test Loss: 0.2481
--------------------------
Final Train Acc: 0.9571
Final Test Acc: 0.9071
```



In [205]:
```
cnn_model.save("aug_cnn_deep.h5")
```

In [183]:
```
#aug_cnn_deepR = cnn_model.fit(train_generator,
#                              steps_per_epoch=22,
#                              epochs=100,
#                              validation_data=val_generator,
#                              validation_steps=4, callbacks=[early_st
```

In [182]:
```
#plot_model_performance(aug_cnn_deepR,train_images,test_images, 'acc',
```

- The model with augmentated data has performed better as compared to the same model that was trained on data.

Two Best Models

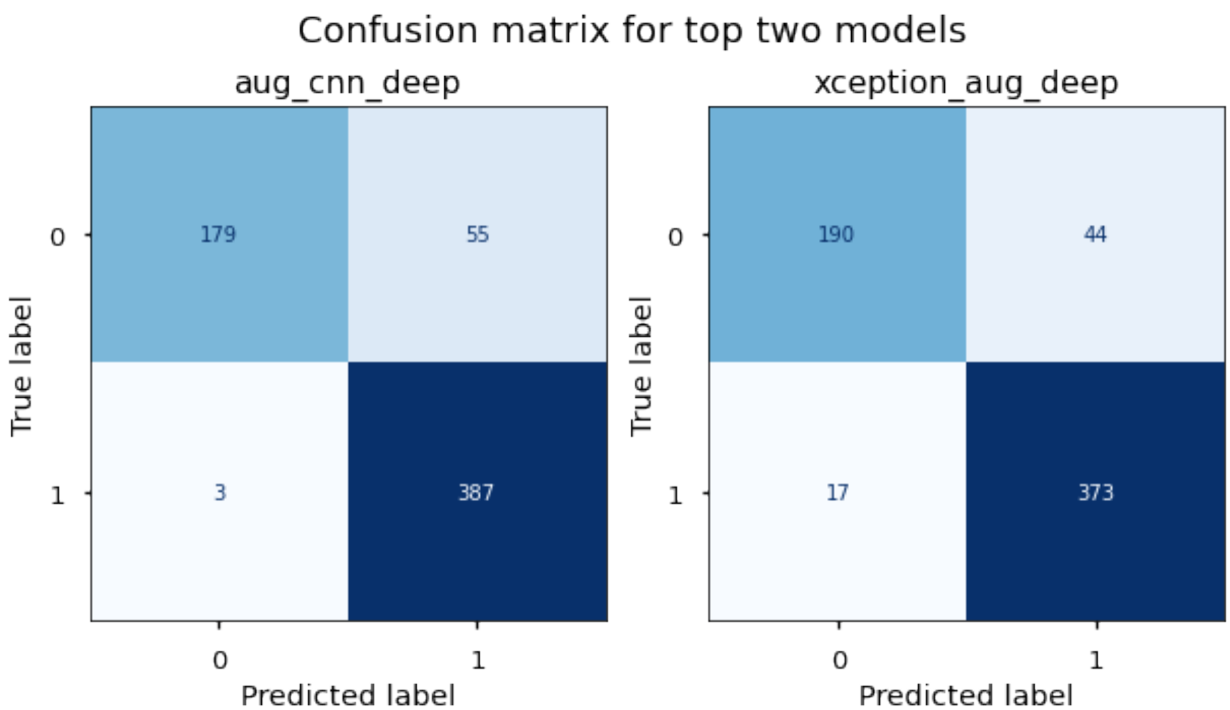
- We want the models that that have high recall for pneumonia as we do not want pneumonia cases to be mis-labeled. Keeping this in mind, the CNN deeper model on augmented data set gives the best results (99% recall rate), with only 3 pneumonia cases as mis-labeled. The recall for normal cases is about 76% in this case
- The second best model following the same metrics is Xception model again on augmented dataset. I chose this model because it has somewhat better recall for normal cases (81%) with a slight decrease in recall for pneumonia cases (96%).

In [170]:
```
train_acc_cnn_aug = np.round(aug_cnn_deep.model.evaluate(train_images,
val_acc_cnn_aug = np.round(aug_cnn_deep.model.evaluate(val_images, val
test_acc_cnn_aug = np.round(aug_cnn_deep.model.evaluate(test_images, t

train_acc_xception_aug = np.round(xception_aug_deep.model.evaluate(tra
val_acc_xcseption_aug = np.round(xception_aug_deep.model.evaluate(val_
test_acc_cnn_aug = np.round(xception_aug_deep.model.evaluate(test_imag
```

```
148/148 [==============================] - 8s 54ms/step - loss: 0.106
6 - acc: 0.9571
17/17 [==============================] - 1s 55ms/step - loss: 0.1187
- acc: 0.9523
20/20 [==============================] - 1s 54ms/step - loss: 0.2481
- acc: 0.9071
148/148 [==============================] - 50s 336ms/step - loss: 0.1
630 - accuracy: 0.9363
17/17 [==============================] - 6s 325ms/step - loss: 0.1692
- accuracy: 0.9351
20/20 [==============================] - 7s 346ms/step - loss: 0.2715
- accuracy: 0.9022
```

In [171]:
```
with plt.style.context('seaborn-talk'):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,4))

    ax1.plot(aug_cnn_deep.history['acc'])
    ax1.plot(aug_cnn_deep.history['val_acc'])
    ax1.set_title('Augmented CNN Model Accuracy')
    ax1.legend(labels = [f'train_acc: {train_acc_cnn_aug}', f'val_acc:
    ax1.set_ylim([0.50, 1])
    ax1.set_xlim([0, 35])

    ax2.plot(xception_aug_deep.history['accuracy'])
    ax2.plot(xception_aug_deep.history['val_accuracy'])
    ax2.set_title('Xception with Addel Layers Accuracy')
    ax2.legend(labels = [f'train_acc: {train_acc_xception_aug}', f'val
    ax2.set_ylim([0.50, 1])
    ax2.set_xlim([0, 35])



plt.savefig('./images/_top_two_models_comp.pdf', dpi=300, bbox_inches=
```

In [172]:
```
‹t('seaborn-talk'):

ıbplots(1, 2, figsize=(10,5))
ıfusion matrix for top two models \n", fontsize=18)

ıodelname in zip(axs.ravel(),[aug_cnn_deep,xception_aug_deep],['CNN Dee
 result.model.predict(test_images)
ːint(y_hat_test).astype(np.int)
ː_y.astype(np.int)
ın_matrix(y_true, y_pred) # normalize = 'true'
ːionMatrixDisplay(confusion_matrix=cm)
ıp = "Blues", ax=ax, colorbar =False)
 modelname)
ː/TopModels_CM.pdf', dpi=300, bbox_inches='tight')
```



In [103]:
```
#model_train = keras.models.load_model("XceptionD_deep_aug")
```

In [255]:
```
#!pip install lime
```

In [245]:
```
import lime
from lime import lime_tabular
```

In [253]:
```
#explainer = lime_tabular.LimeTabularExplainer(
#    training_data=np.array(train_img),
#    feature_names=train_y.tolist(),
#    class_names=['Normal', 'Pneumonia'],
#    mode='classification'
#)
```

In [254]:
```
#train_y.tolist()
#train_img
```

Feature Extraction

Visualizing a Layer

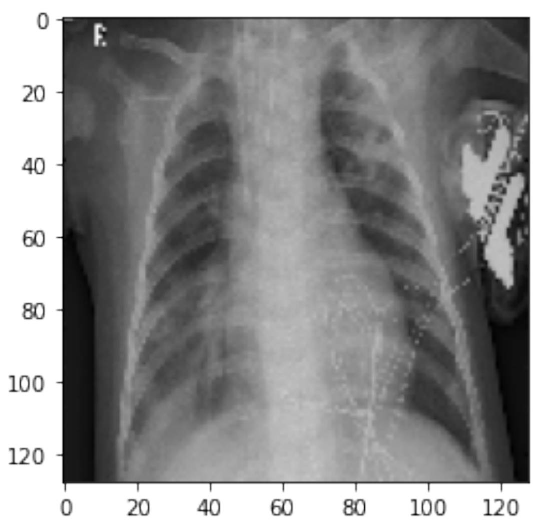
- In order to get a better sense of what representations our CNN is learning under the hood, we will visualize the feature maps generated during training.
- CNNs work by applying a filter successively over an image. This transformation creates a new representation of the image which we call a feature map.

In [206]:
```
best_model = keras.models.load_model("aug_cnn_deep.h5")
best_model.summary()
```

```
Model: "sequential_29"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_21 (Conv2D)           (None, 126, 126, 32)      896       
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 63, 63, 32)        0         
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 61, 61, 64)        18496     
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 30, 30, 64)        0         
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 28, 28, 128)       73856     
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 14, 14, 128)       0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 25088)             0         
_________________________________________________________________
dense_139 (Dense)            (None, 128)               3211392   
_________________________________________________________________
dense_140 (Dense)            (None, 64)                8256      
_________________________________________________________________
dense_141 (Dense)            (None, 1)                 65        
=================================================================
Total params: 3,312,961
Trainable params: 3,312,961
Non-trainable params: 0
_________________________________________________________________
```

In [378]:
```
# Lets just Visulaize one image
# Display the image
filename = 'train/PNEUMONIA/BACTERIA-8705009-0002.jpeg'
img = image.load_img(filename, target_size=(128, 128))
img_tensor = image.img_to_array(img)

# reshape the image into tensor to be able to use with the CNN archite
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()
```

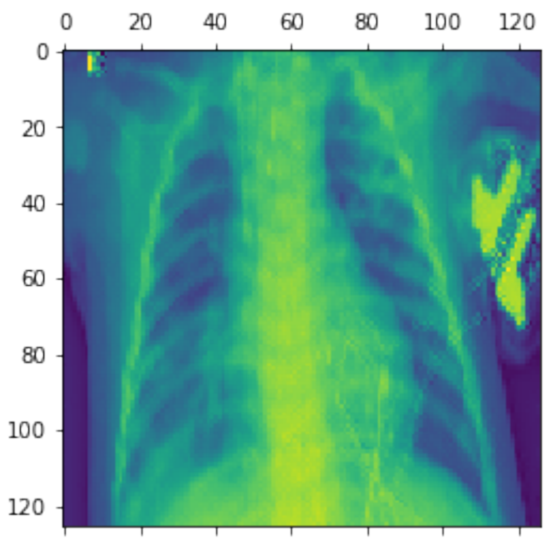
```
(1, 128, 128, 3)
```



- Now lets visulaize the forst activation layer and see the third channel

In [380]:
```
layer_outputs = [layer.output for layer in best_model.layers[:6]]
# Rather then a model with a single output, we are going to make a moc
activation_model = models.Model(inputs=best_model.input, outputs=layer
activations = activation_model(img_tensor)

first_layer_activation = activations[0]
print(first_layer_activation.shape)

# We slice the third channel and preview the results
plt.matshow(first_layer_activation[0, :, :, 3], cmap='viridis')
plt.show()
```

```
(1, 126, 126, 32)
```



In [381]:
```
layer_outputs
```

Out[381]:
```
[<KerasTensor: shape=(None, 126, 126, 32) dtype=float32 (created by l
ayer 'conv2d_21')>,
 <KerasTensor: shape=(None, 63, 63, 32) dtype=float32 (created by lay
er 'max_pooling2d_13')>,
 <KerasTensor: shape=(None, 61, 61, 64) dtype=float32 (created by lay
er 'conv2d_22')>,
 <KerasTensor: shape=(None, 30, 30, 64) dtype=float32 (created by lay
er 'max_pooling2d_14')>,
 <KerasTensor: shape=(None, 28, 28, 128) dtype=float32 (created by la
yer 'conv2d_23')>,
 <KerasTensor: shape=(None, 14, 14, 128) dtype=float32 (created by la
yer 'max_pooling2d_15')>]
```

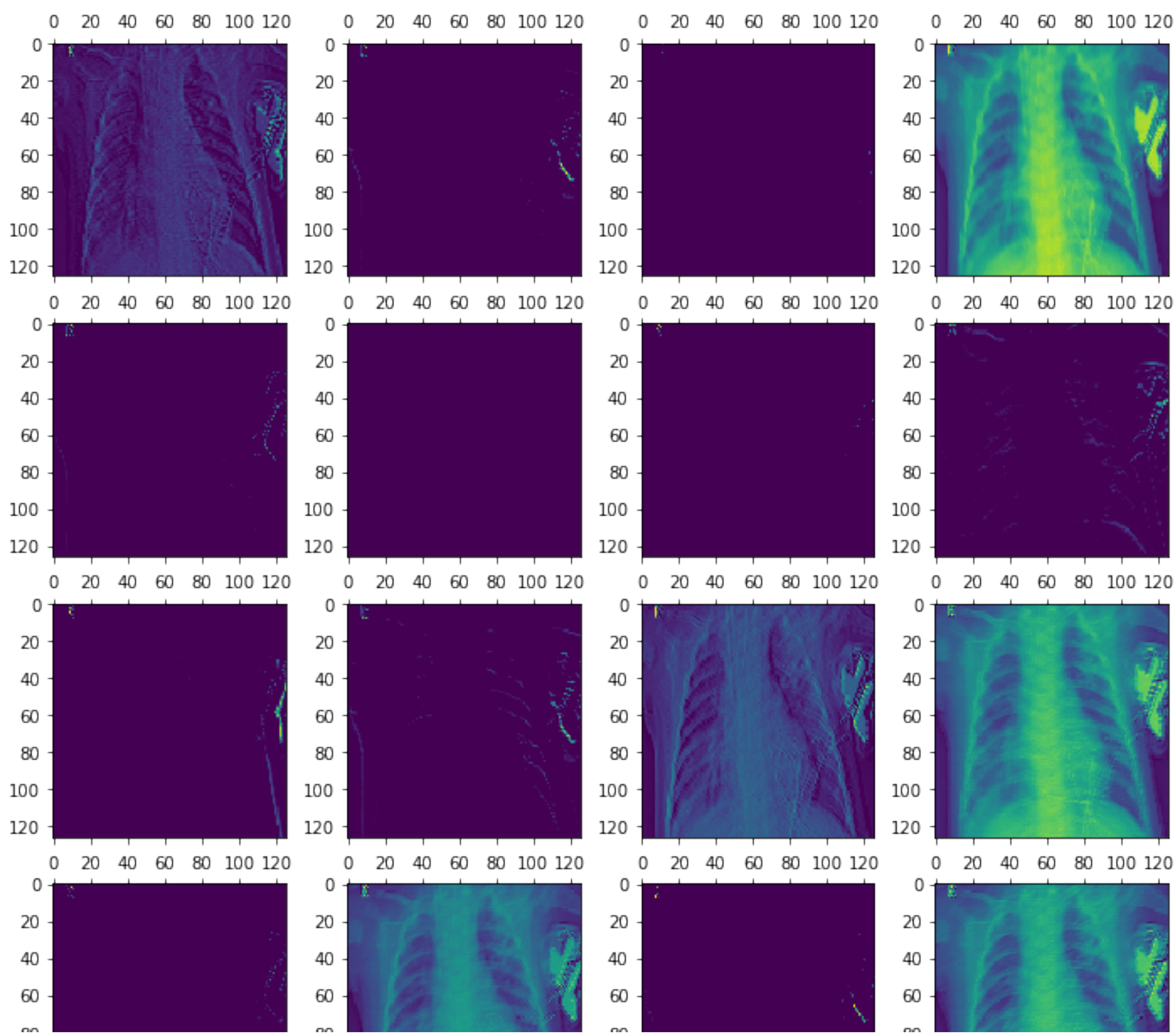
In [382]:
```
best_model.input
```

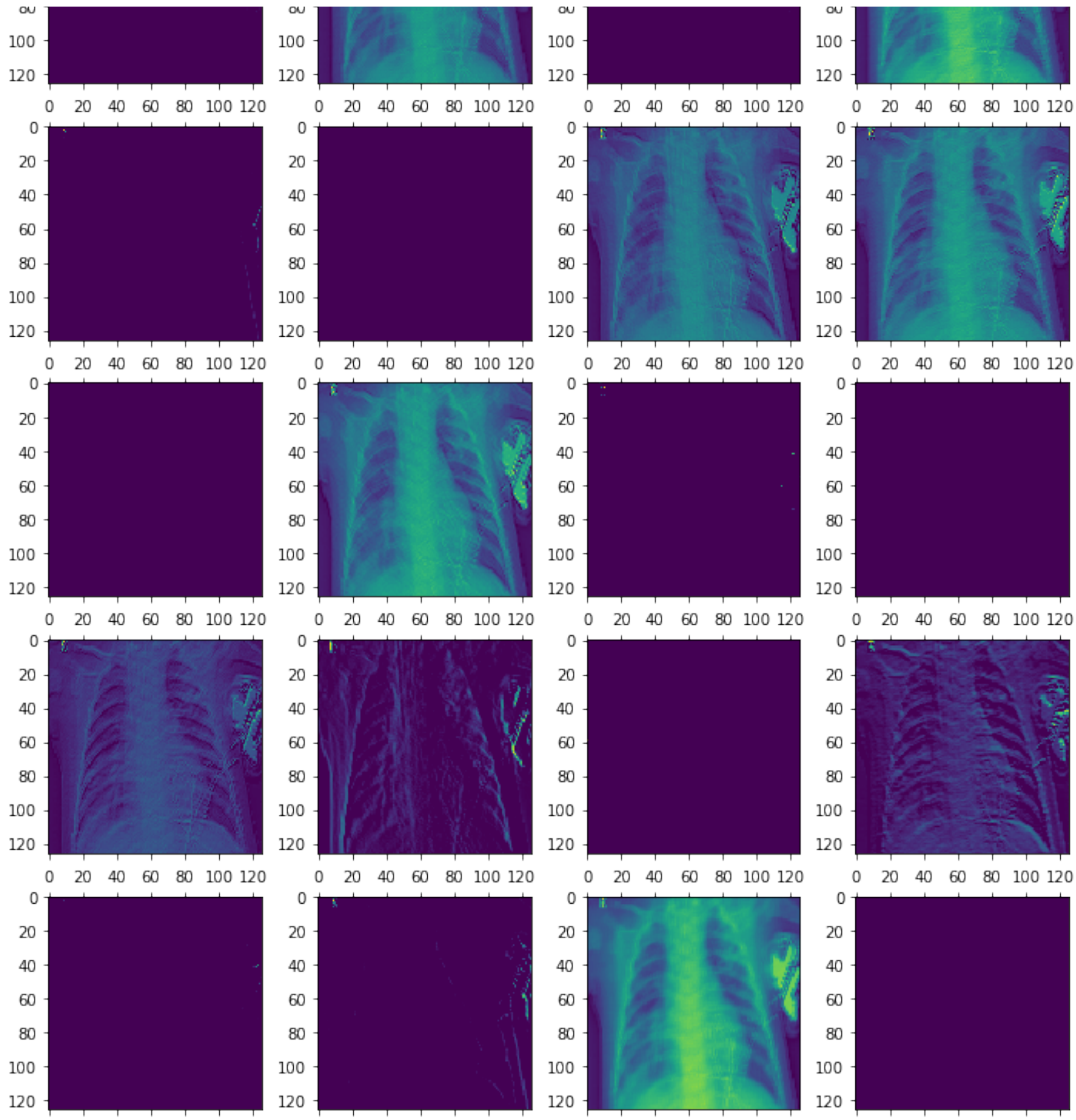
Out[382]:
```
<KerasTensor: shape=(None, 128, 128, 3) dtype=float32 (created by lay
er 'conv2d_21_input')>
```

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

- The initial three layers output feature maps that have 32 channels each.

In [384]:
```
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]
    ax.matshow(first_layer_activation[0, :, :, i], cmap='viridis')
```

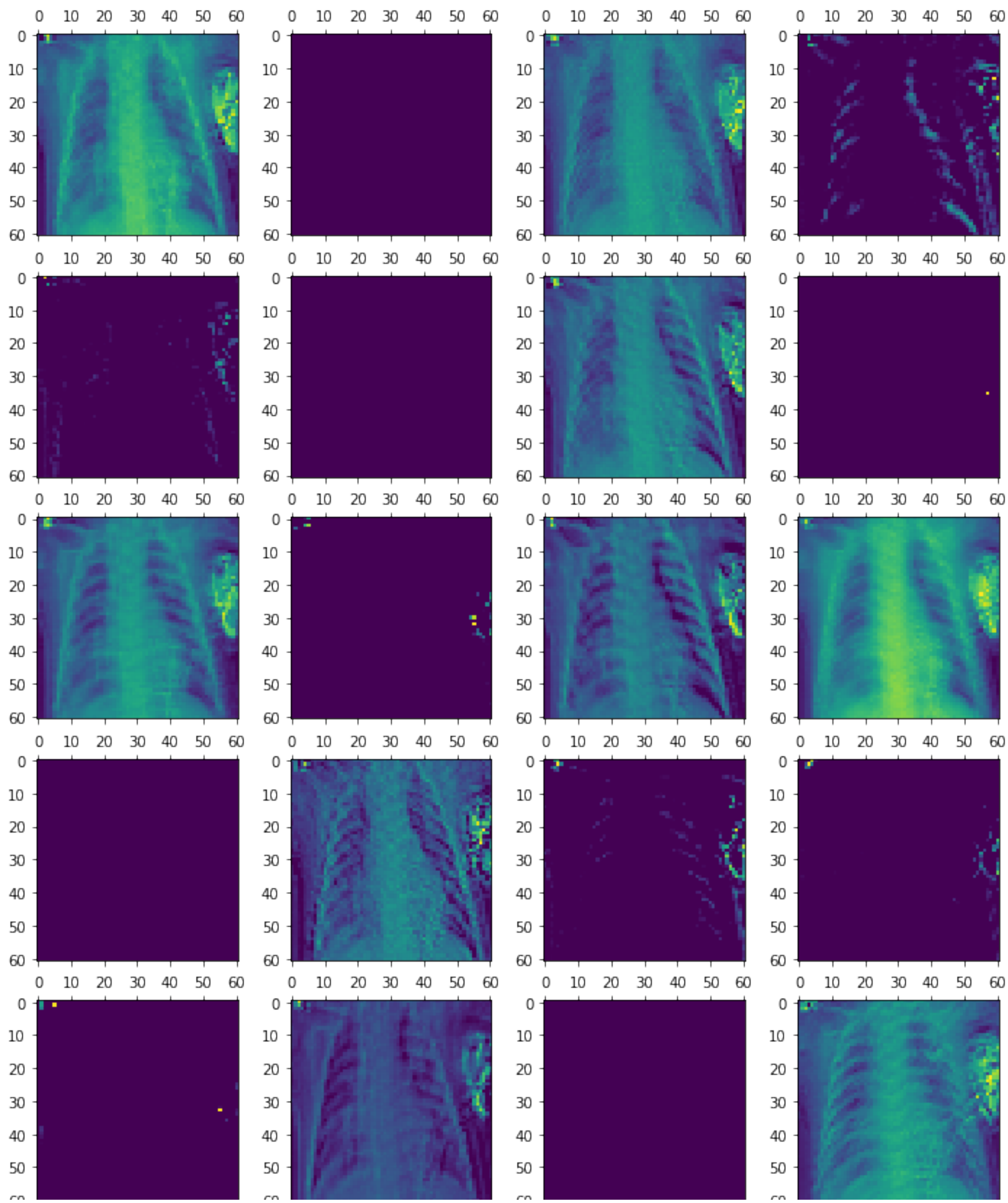


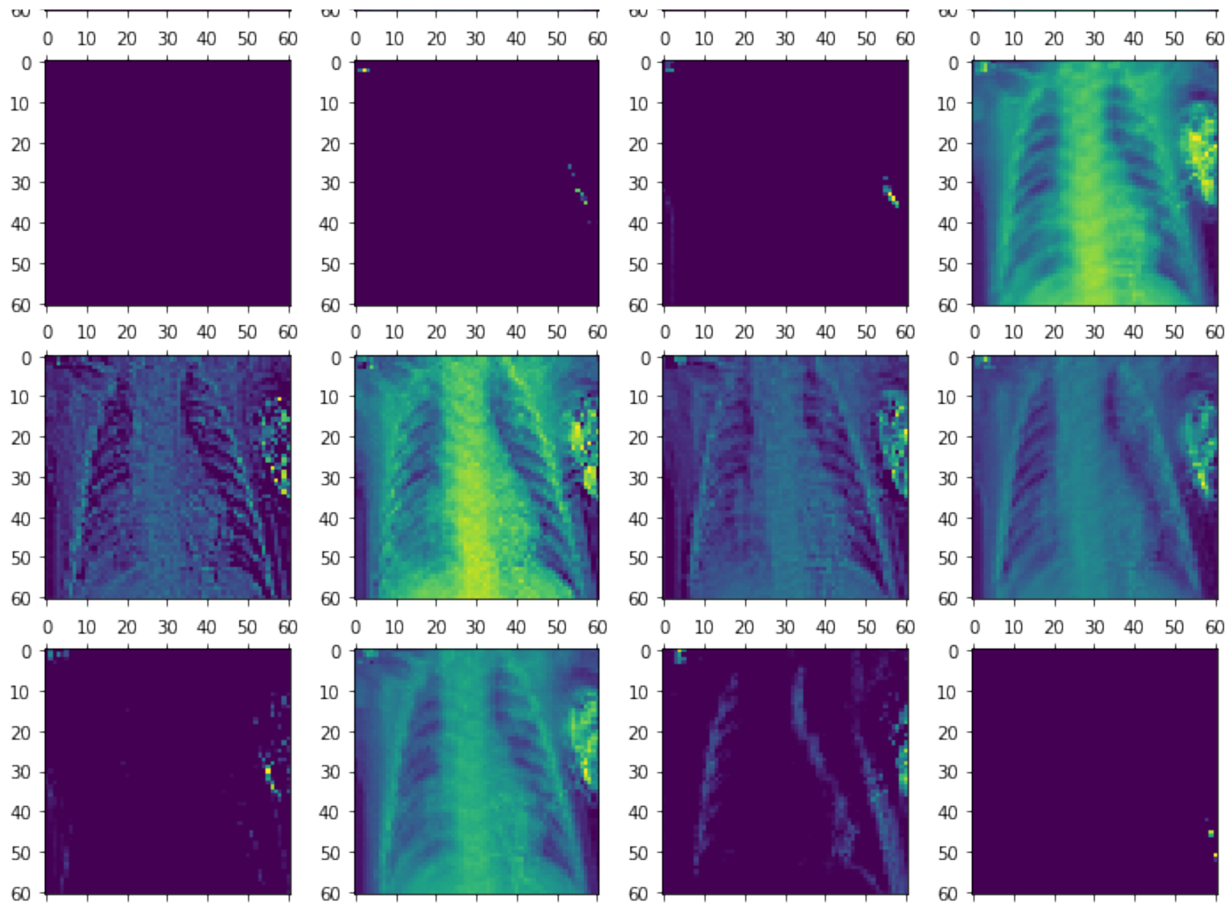


Lets visulaize the third activation layer

In [390]:

```
#second_layer_activation = activations[1]
#print(second_layer_activation.shape)
fig, axes = plt.subplots(8, 4, figsize=(12,24))
for i in range(32):
    row = i//4
    column = i%4
    ax = axes[row, column]
    second_layer_activation = activations[2]
    ax.matshow(second_layer_activation[0, :, :, i], cmap='viridis')
```





- One can see how the later activation layers capture abstract pattern while the first layer captures mode deeper patterns

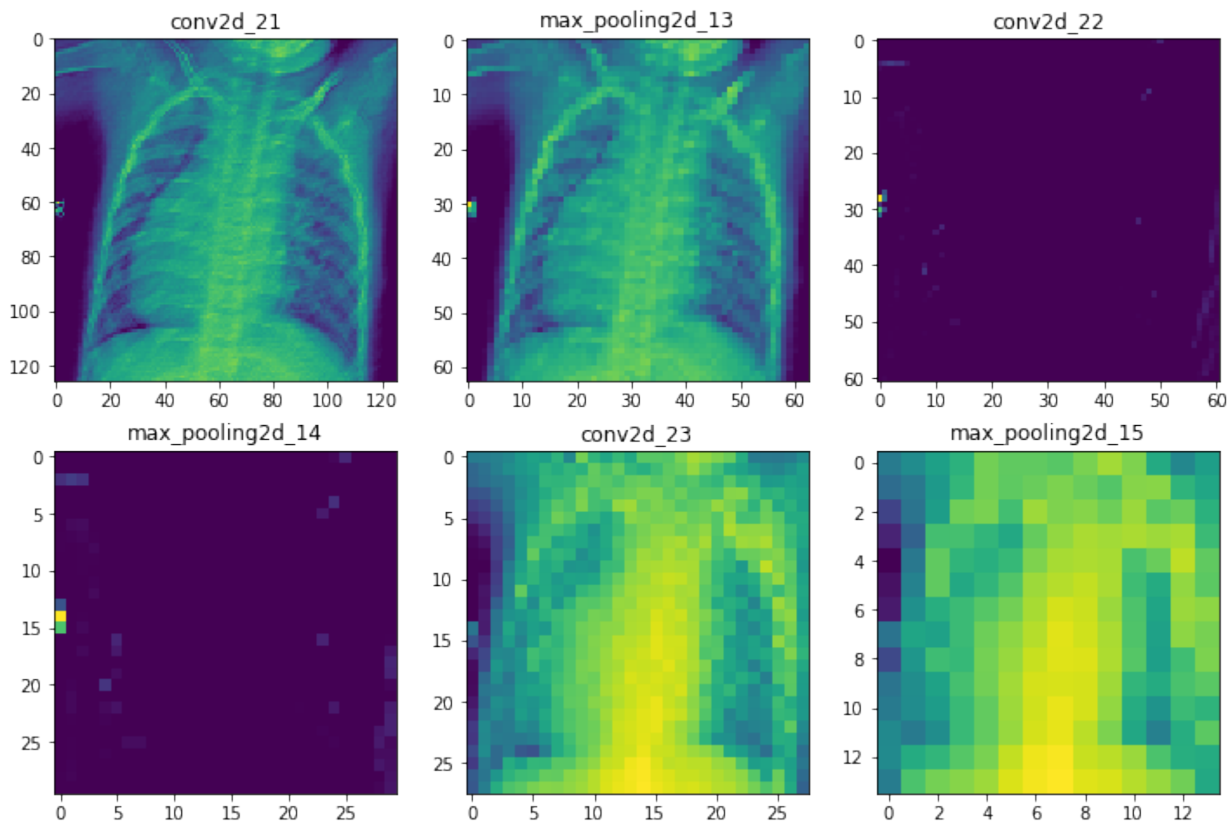
Visualize a single channel for each of the activation layers:

- Below we are looking at 15th channel fro each activation layer

In [251]:

```
fig, axes = plt.subplots(2,3, figsize=(12,8))

layer_names = []
for layer in best_model.layers[:6]:
    layer_names.append(layer.name)
    print(layer.name)

for i in range(6):
    row = i//3
    column = i%3
    ax = axes[row, column]
    cur_layer = activations1[i]
    ax.matshow(cur_layer[0, :, :, 15], cmap='viridis')
    ax.xaxis.set_ticks_position('bottom')
    ax.set_title(layer_names[i])
```

```
conv2d_21
max_pooling2d_13
conv2d_22
max_pooling2d_14
conv2d_23
max_pooling2d_15
```



Model Visulaziations using LIME

- LIME, the acronym for local interpretable model-agnostic explanations, is a technique that approximates any black box machine learning model with a local, interpretable model to explain each individual prediction (https://arxiv.org/abs/1602.04938 (https://arxiv.org/abs/1602.04938)) https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/ (https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/).
- Lets visulazise one image using LIME and see how the given model has made demarcations to label the image

In [258]:
```
#img_path = 'train/PNEUMONIA/BACTERIA-8705009-0002.jpeg'
#img = image.load_img(img_path, target_size=(128, 128))
#img_tensor = image.img_to_array(img)

# reshape the image into tensor to be able to use with the CNN archite
#img_tensor = np.expand_dims(img_tensor, axis=0)
#img_tensor /= 255.

#explanation = explainer.explain_instance(x_img, best_model.predict,
#                                          hide_color=0, num_samples=1
```

- Functions needed for LIME
- First function returns the output from best model (CNN Augmented)
- Second function returns the output from second-best model (Xception Augmented)

In [391]:
```
def predict_fn(image):
    image = ((image.astype(float))/255)
    image = image.reshape((-1,128,128,3))
    return aug_cnn_deep.model(image)
```

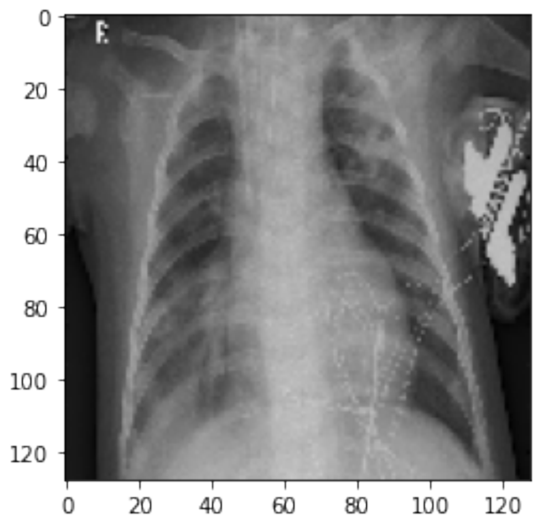
In [392]:
```
def predict_fn2(image):
    image = ((image.astype(float))/255)
    image = image.reshape((-1,128,128,3))
    return xception_aug_deep.model(image)
```

In [393]:
```
from lime import lime_image
explainer = lime_image.LimeImageExplainer()
```

- Lets see the original image as it is !

In [394]:
```
plt.imshow(img_tensor[0])
```

Out[394]: <matplotlib.image.AxesImage at 0x7fa31fdecf10>



In [395]:
```
np.double( predict_fn(img_tensor[0]))
```

Out[395]: 0.9930863380432129

In [396]:
```
np.double( predict_fn2(img_tensor[0]))
```

Out[396]: 0.9442552328109741

- Lime explanation functions

In [397]:
```
explanation_0 = explainer.explain_instance(np.double(img_tensor[0]), p
```

100% 1000/1000 [00:27<00:00, 36.02it/s]

In [398]:
```
explanation_1 = explainer.explain_instance(np.double(img_tensor[0]), p
```

100% 1000/1000 [00:22<00:00, 43.65it/s]

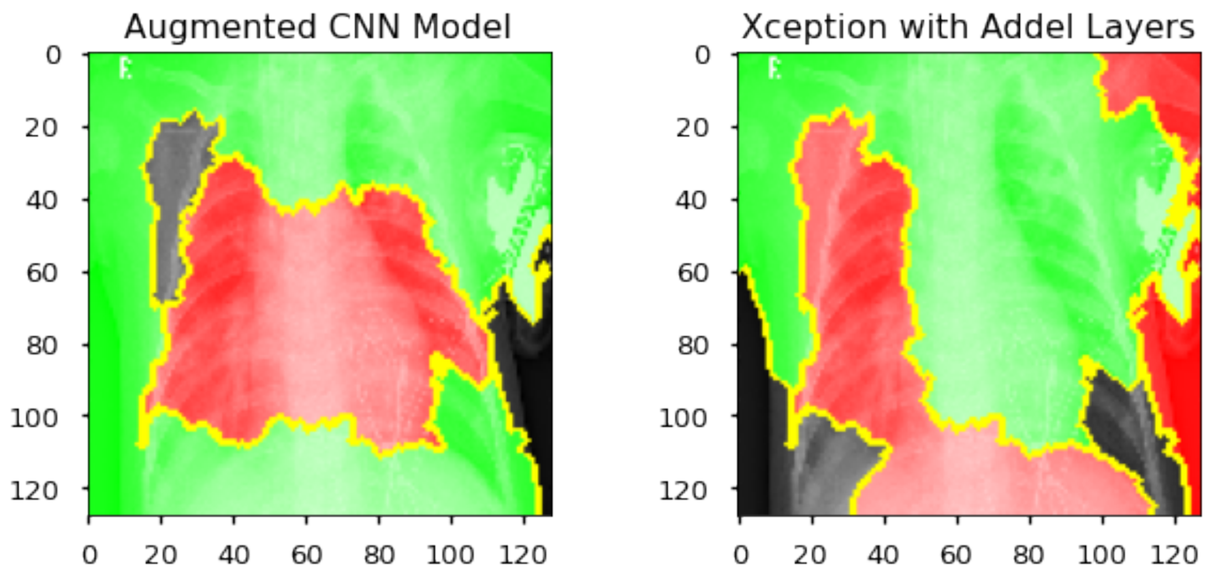
Lets plot the mask boundaries for two models

*Towards and against - green and red respectively.

In [399]:
```
from skimage.segmentation import mark_boundaries

temp, mask = explanation_0.get_image_and_mask(explanation_0.top_labels
#plt.imshow(mark_boundaries(temp, mask))
```

In [400]:
```
temp1, mask1 = explanation_1.get_image_and_mask(explanation_1.top_labe
#plt.imshow(mark_boundaries(temp1, mask1))
```

In [401]:
```
with plt.style.context('seaborn-talk'):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,4))

    ax1.imshow(mark_boundaries(temp, mask))
    ax1.set_title('Augmented CNN Model')

    ax2.imshow(mark_boundaries(temp1, mask1))
    ax2.set_title('Xception with Addel Layers')
```

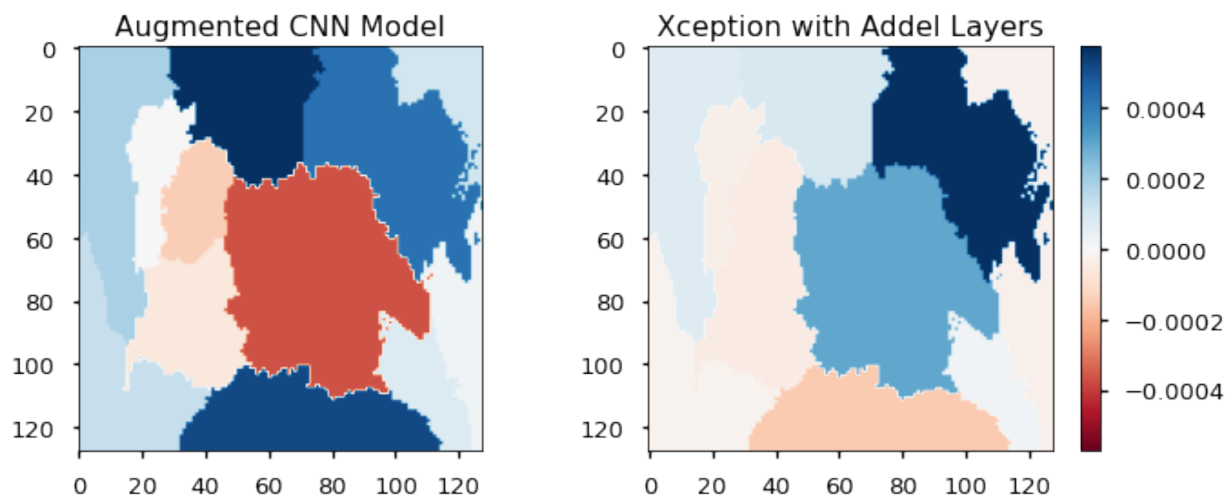


- The two models work differently by defining boundaries somewhat differently!

Lets now look at the Heatmap - The more blue it is, the higher positive impact!

In [402]:
```
ind =  explanation_0.top_labels[0]

#Map each explanation weight to the corresponding superpixel
dict_heatmap = dict(explanation_0.local_exp[ind])
heatmap = np.vectorize(dict_heatmap.get)(explanation_0.segments)

ind1 =  explanation_1.top_labels[0]
dict_heatmap1 = dict(explanation_1.local_exp[ind1])
heatmap1 = np.vectorize(dict_heatmap1.get)(explanation_1.segments)
#Plot. The visualization makes more sense if a symmetrical colorbar is

#plt.imshow(heatmap, cmap = 'RdBu', vmin  = -heatmap.max(), vmax = hea
#plt.colorbar()
```

In [403]:
```
with plt.style.context('seaborn-talk'):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,4),sharey=False)

    s1 = ax1.imshow(heatmap, cmap = 'RdBu', vmin  = -heatmap.max(), vm
    ax1.set_title('Augmented CNN Model')
    #fig.colorbar(s1, ax=ax1,orientation='vertical')
    s2 =ax2.imshow(heatmap1, cmap = 'RdBu', vmin  = -heatmap1.max(), v
    ax2.set_title('Xception with Addel Layers')
    fig.colorbar(s2, ax=ax2,orientation='vertical')
    plt.tight_layout()
    plt.savefig('./images/Topmodels_lime_comp.pdf', dpi=300, bbox_inch
```



- One can see how the two models has slight variations in defining the importance for various sections of a given image

Recommendations

- Neural network are a useful tool to aid the healthcare professional in stream-lining the diagnosing process when classifying x-ray images. This will allow for a quicker return time and greater patient satisfaction.
- Detecting people that have pneumonia as early as possible is very important for early intervention. Using neural networks for x-ray image clasiification will significantly reduce the waiting times for patients to hear back from radiologist and their treatment recommendations.
- The Radiologist will have reduction in his work-load and there will be a mechanism to provide continued care for patients when the radiologist is out on sick leave or has to take vacations

Further Improvements

- We probably need to implememt weights in the training to take into account class imbalance using oversampling techniques which could improve performance.
- Cropping images to remove unnecessary captured details such as R may bring some improvement to models