data_mining_project

August 11, 2021

1 Image Processing is used to extract useful information, extract image and annalyze it

2 Application

Medical image annalysis

Artficial intelligence

Image roation and enhancement

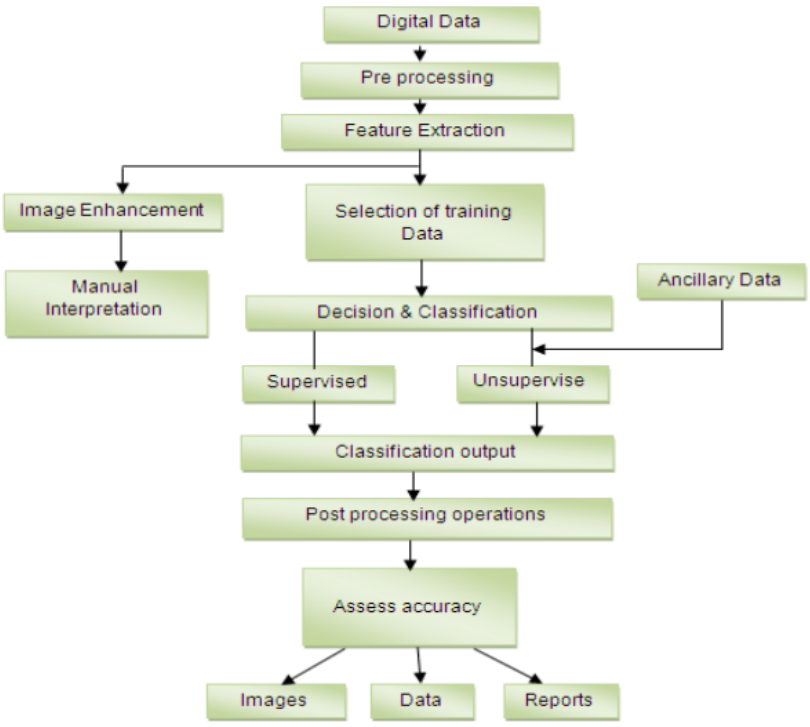
Geospatial Application

Robotic Visioning

Autonomous vehicle

[15]:
```
from IPython.display import Image
Image(filename='output/png/work_flow.png', width = 1200, height = 400)
```

[15]:



3 Purposes

Visualization

```
:Objects that are notvisible
```

Imagesharpening and restoration

```
 :A beter image
```

Image retrieval

```
:Seek for the image of interest
```

Measurement of paterns

```
:Measures various objects
```

Image Recognition

```
:Distinguish objects in an image
```

[1]:
```
# Import the modules from skimage
from skimage import data, color,io
import matplotlib.pyplot as plt
import numpy as np
from skimage.filters import try_all_threshold
# Import the filters module and sobel function
from skimage.filters import sobel

# Import the module and functionfrom
from skimage.filters import gaussian

# Load the rocket image
rocket = data.rocket()
```

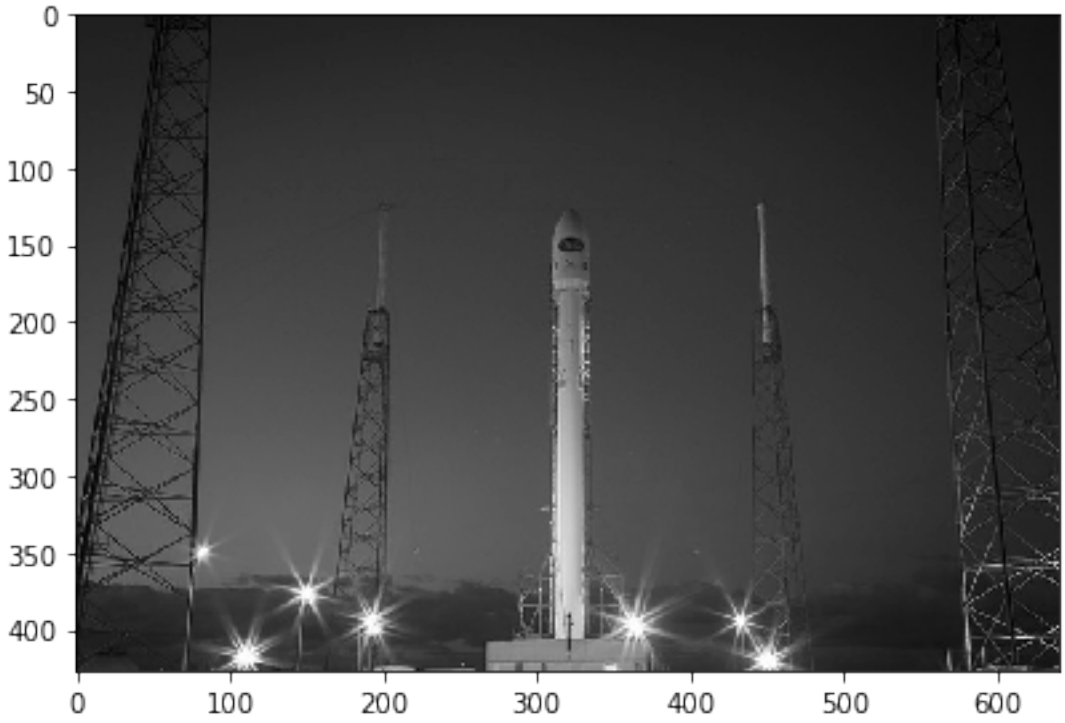
[2]:
```
# Show the original image
io.imshow(rocket)
io.show()
```

[3]:
```
rocket.shape
```

[3]:
```
(427, 640, 3)
```

[4]:
```
# Convert the image to grayscale
gray_scaled_rocket = color.rgb2grey(rocket)
# Show the grayscale image
io.imshow(gray_scaled_rocket)
io.show()
```

```
<ipython-input-4-08ca427875ca>:2: FutureWarning: rgb2grey is deprecated. It will
be removed in version 0.19.Please use rgb2gray instead.
  gray_scaled_rocket = color.rgb2grey(rocket)
```

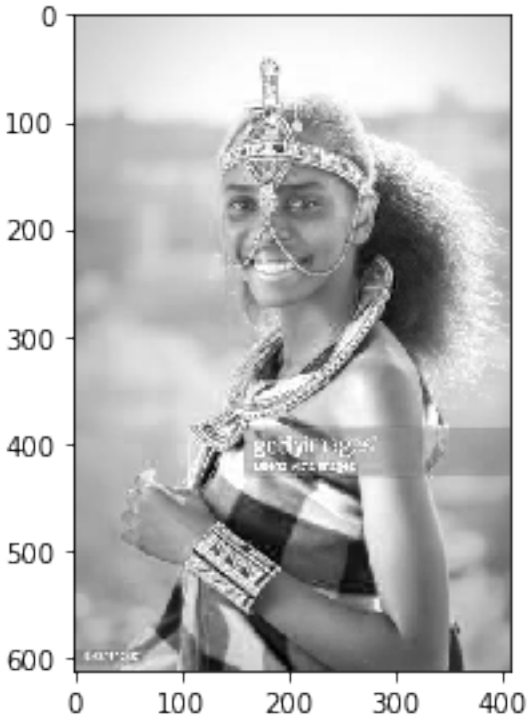


[5]:
```
# Loading the image using Matplotlib
beaty_image = plt.imread('output/png/gettyimages-870711360-612x612.jpg')
io.imshow(beaty_image)
io.show()
```

[16]:
```
type(beaty_image)
```

[16]:
```
numpy.ndarray
```

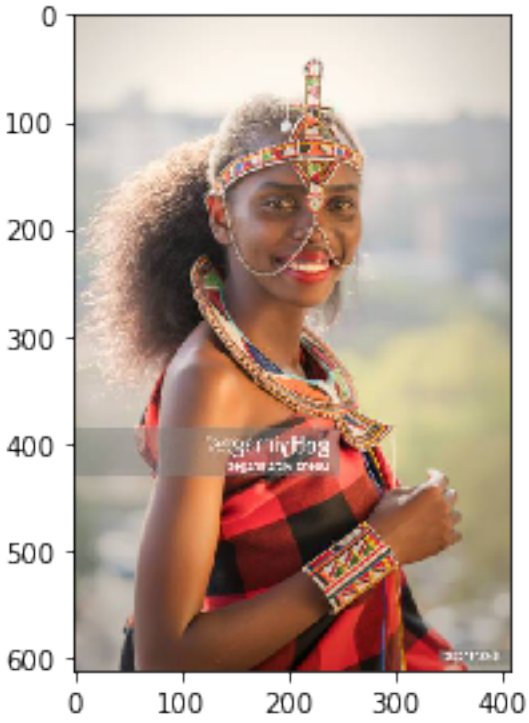
[17]:
```
# Obtaining the red values of the
imagered = beaty_image[:, :, 0]
io.imshow(imagered)
io.show()
```



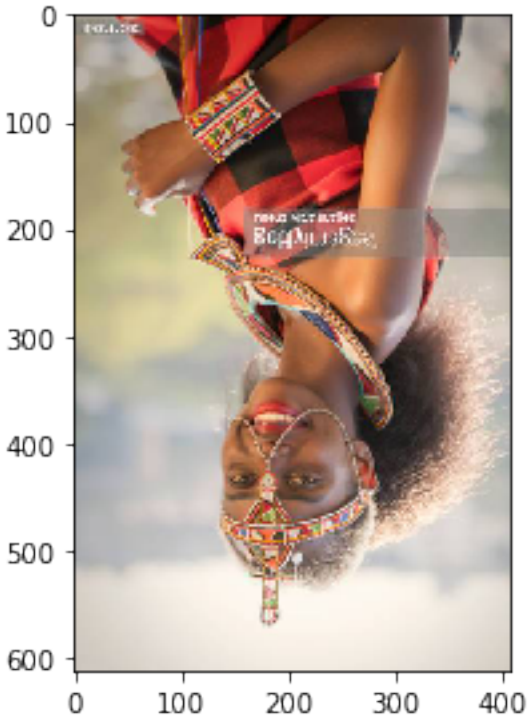
[18]:
```
# Obtaining the green values of the
imagegreen = beaty_image[:, :, 1]
io.imshow(imagegreen)
io.show()
```

[19]:
```
# Obtaining the blue values of the
imageblue = beaty_image[:, :, 2]
io.imshow(imageblue)
io.show()
```

[20]:
```
# Flip the image in left
directionhorizontally_flipped = np.fliplr(beaty_image)
io.imshow(directionhorizontally_flipped)
io.show()
```



[21]:
```
# Flip the image in left
directionhorizontally_flipped = np.flipud(beaty_image)
io.imshow(directionhorizontally_flipped)
io.show()
```



3.1 IMAGE PROCESSING IN PYTHON

Applications of histograms Analysis

Thresholding

Brightness and

contrast Equalize an image

Thresholding

Partitioning an image into:

a foreground and background By making it black and white We do so by seing each pixel to : 255(white)

if pixel > thresh value 0(black)

if pixel < threshvalue

Thresholding Simplest method of image segmentation

Isolate objects

Object detection

Face detection Etc.

[22]:
```
# Obtain the optimal threshold value
from skimage import data
img = color.rgb2gray(beaty_image)

fig, ax = try_all_threshold(img, figsize=(10, 8), verbose=False)
plt.show()
```

Original

Isodata

Li

Mean

Minimum

Otsu

Triangle

Yen

3.2 Filters Enhancing an image

Emphasize or

remove features

Smoothing Sharpening

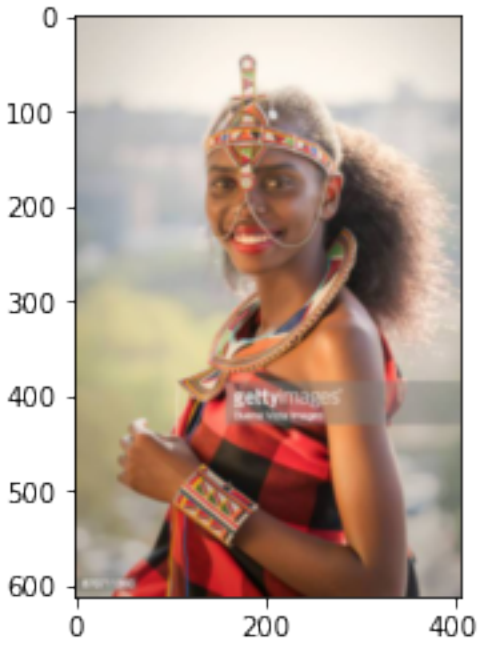
Edge detection

[23]:
```
## making curves appear
# Make the image grayscale
soaps_image_gray = color.rgb2gray(beaty_image)

# Apply edge detection filter
edge_sobel = sobel(beaty_image)

# Show original and resulting image to compare
plt.imshow(edge_sobel)
```

[23]:
```
<matplotlib.image.AxesImage at 0x7f9120250f10>
```

[24]:
```
#gaussian smoothning

# Apply edge detection
filtergaussian_image = gaussian(beaty_image, multichannel=True)
# Show original and resulting image to
plt.imshow(filtergaussian_image)
```

[24]: <matplotlib.image.AxesImage at 0x7f91203f9ac0>



3.3 CONTRAST

Enhance contrast

Contrast stretching

Histogram equalization

3.4 Types

Histogram equalization

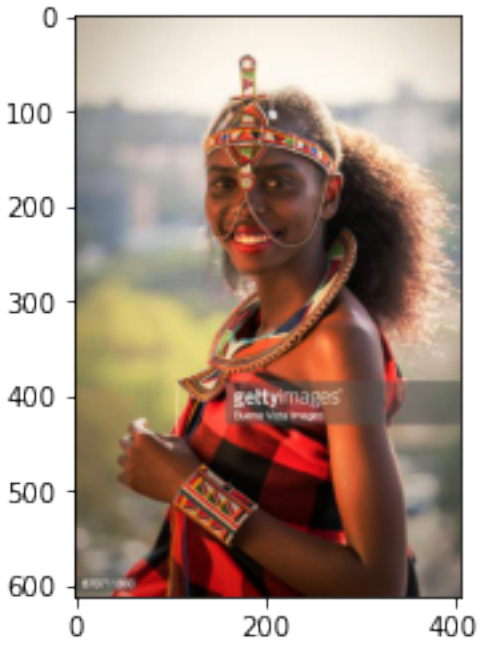
Adaptive histogram equalization

Contrast Limited Adaptive Histogram Equalization(CLAHE)

[25]:
```
from skimage import exposure
# Obtain the equalized
imageimage_eq = exposure.equalize_hist(beaty_image)
# Show original and result
plt.imshow(imageimage_eq)
```

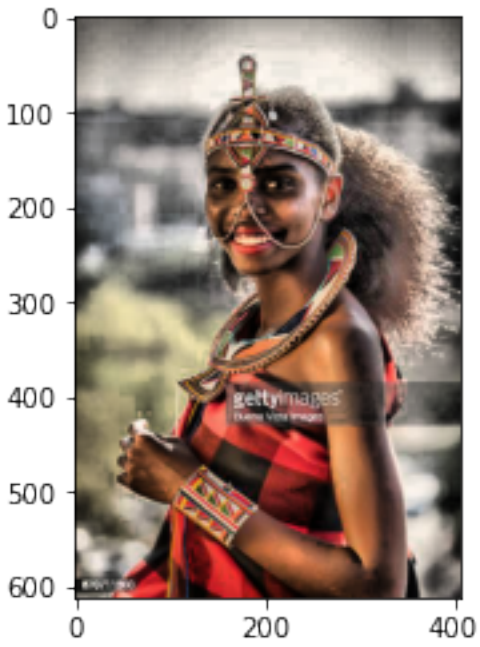
```
/opt/conda/lib/python3.8/site-packages/skimage/exposure/exposure.py:181:
UserWarning: This might be a color image. The histogram will be computed on the
flattened image. You can instead apply this function to each color channel.
  hist, bin_centers = histogram(image, nbins)
```

[25]: <matplotlib.image.AxesImage at 0x7f91204cdb20>



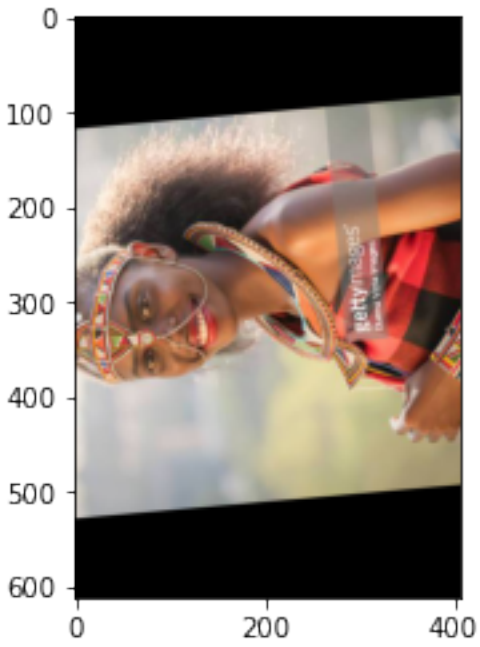
[26]:
```
# Apply the adaptive equalization on the original image
adapthist_eq_image = exposure.equalize_adapthist(beaty_image, clip_limit=0.03)
# Compare the original image to the equalized
plt.imshow(adapthist_eq_image)
```

[26]: <matplotlib.image.AxesImage at 0x7f91205d2a00>



[27]:
```
# Import the module and the rotate and rescale functions
from skimage.transform import rotate, rescale
# Rotate the image 90 degrees clockwise
rotated_cat_image = rotate(beaty_image, 95)
plt.imshow(rotated_cat_image)
```

[27]: <matplotlib.image.AxesImage at 0x7f912063c5b0>

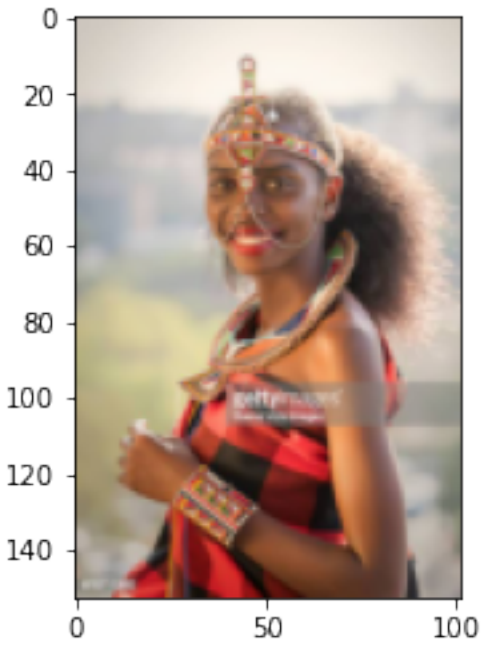


[28]:
```
# Enlarge the image so it is 3 times bigger
enlarged_rocket_image = rescale(beaty_image, 3, anti_aliasing=True,␣
↪multichannel=True)
# Show original and resulting image
plt.imshow(enlarged_rocket_image)
```

[28]: <matplotlib.image.AxesImage at 0x7f9121fa3bb0>

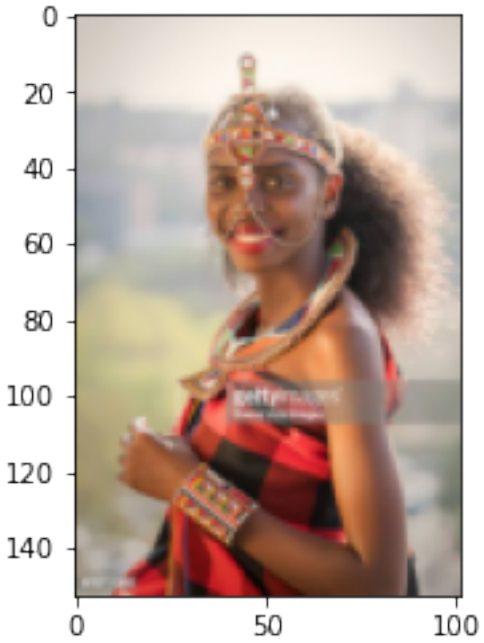
[29]:
```
# shrink the image so it is 4 times smaller
enlarged_rocket_image = rescale(beaty_image, 1/4, anti_aliasing=True,␣
 ↪multichannel=True)
# Show original and resulting image
plt.imshow(enlarged_rocket_image)
```

[29]: <matplotlib.image.AxesImage at 0x7f9120213eb0>



[30]:
```
# shrink the image so it is 4 times smaller
enlarged_rocket_image = rescale(beaty_image, 1/4, anti_aliasing=True,␣
↪multichannel=True)
# Show original and resulting image
plt.imshow(enlarged_rocket_image)
```

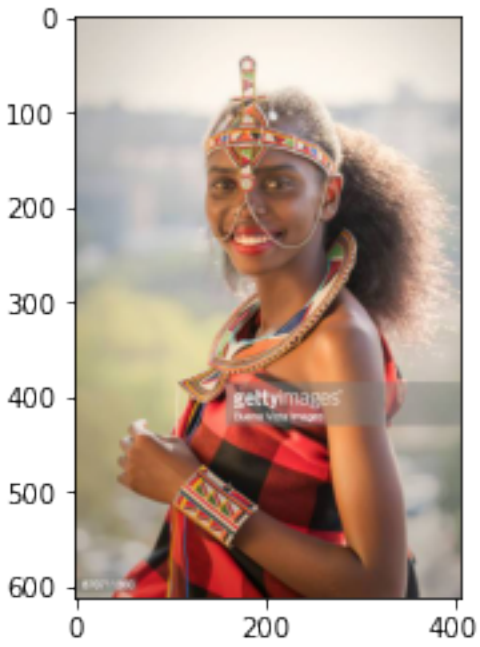
[30]: <matplotlib.image.AxesImage at 0x7f912073da60>



[31]:
```
def get_mask(image):
    ''' Creates mask with three defect regions '''
    mask = np.zeros(image.shape[:-1])
    mask[101:106, 0:240] = 1
    mask[152:154, 0:60] = 1
    mask[153:155, 60:100] = 1
    mask[154:156, 100:120] = 1
    mask[155:156, 120:140] = 1
    mask[212:217, 0:150] = 1
    mask[217:222, 150:256] = 1
    return mask
```

[32]:
```
from skimage.restoration import inpaint
# Obtain the mask
mask = get_mask(beaty_image)
# Apply inpainting to the damaged image using the
maskrestored_image = inpaint.
 ↪inpaint_biharmonic(beaty_image,mask,multichannel=True)# Show the resulting␣
 ↪imageshow_image(restored_image)
plt.imshow(maskrestored_image)
```

[32]:
```
<matplotlib.image.AxesImage at 0x7f91201af790>
```



3.5 Image reconstruction

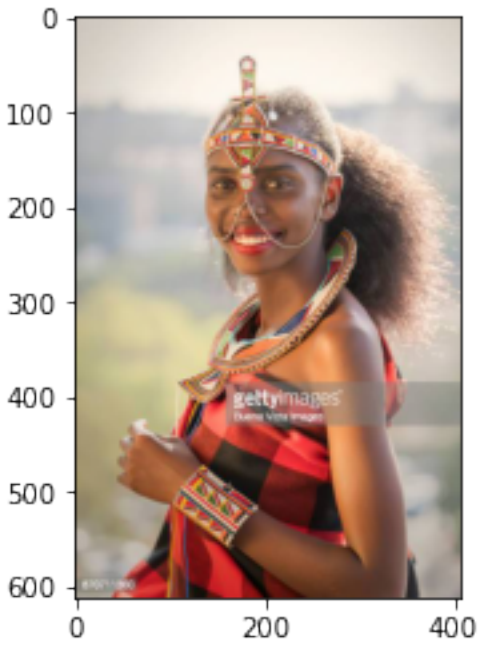
Fixing damaged images

Text removing Logo removing

Object removing

[33]:
```
# Initialize the mask
mask = np.zeros(beaty_image.shape[:-1])
# Set the pixels where the logo is to 1
mask[180:290, 400:400] = 1

# Apply inpainting to remove the logo
image_logo_removed = inpaint.inpaint_biharmonic(beaty_image, mask,␣
 ↪multichannel=True)

# Show the original and logo removed images
plt.imshow(maskrestored_image)
```

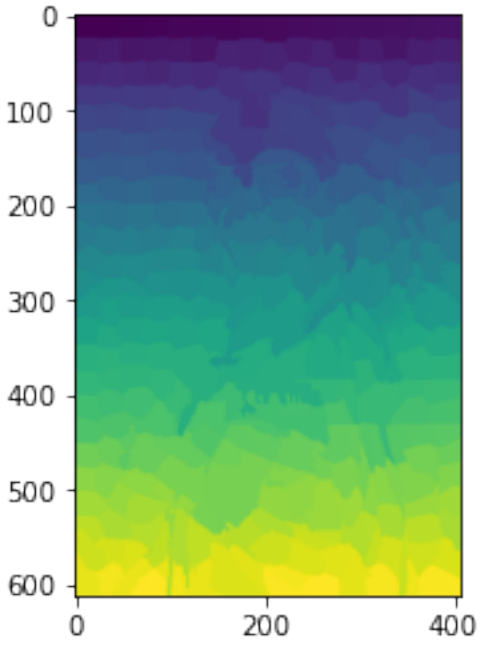
[33]:
```
<matplotlib.image.AxesImage at 0x7f91205b9bb0>
```



[34]:
```
# Import the slic function from segmentation module
from skimage.segmentation import slic

# Import the label2rgb function from color module
from skimage.color import label2rgb

# Obtain the segmentation with 400 regions
segments = slic(beaty_image, n_segments= 400)

# Put segments on top of original image to compare
segmented_image = label2rgb(segments, beaty_image, kind='avg')

# Show the segmented image
plt.imshow(segments)
```

```
<ipython-input-34-12015b8b96f5>:8: FutureWarning: skimage.measure.label's
indexing starts from 0. In future version it will start from 1. To disable this
warning, explicitely set the `start_label` parameter to 1.
  segments = slic(beaty_image, n_segments= 400)
<ipython-input-34-12015b8b96f5>:11: FutureWarning: The new recommended value for
bg_label is 0. Until version 0.19, the default bg_label value is -1. From
version 0.19, the bg_label default value will be 0. To avoid this warning,
please explicitly set bg_label value.
  segmented_image = label2rgb(segments, beaty_image, kind='avg')
```

[34]:
```
<matplotlib.image.AxesImage at 0x7f91200e76a0>
```



Edges,

Corners

Face detections

4 Breast Cancer Classification

Breast cancer is a common cancer in women, and one of the major causes of death among women around the world. Invasive ductal carcinoma (IDC) is the most widespread type of breast cancer with about 80% of all diagnosed cases. IDC is cancer that began growing in a milk duct and has invaded the fibrous or fatty tissue of the breast outside of the duct. Early accurate diagnosis plays an important role in choosing the right treatment plan and improving survival rate among the patients. In recent years, efforts have been made to predict and detect all types of cancers by employing artificial intelligence. An appropriate dataset is the first essential step to achieve such a goal. This paper introduces a histopathological microscopy image dataset of 922 images related to 124 patients with IDC. The dataset has been published and is accessible through the web at: http://databiox.com. The distinctive feature of this dataset as compared to similar ones is that it contains an equal number of specimens from each of three grades of IDC, which leads to approximately 50 specimens for each grade.

<img src="output/png/lobules_and_ducts_of_the_breast.jpg" width="800" height="600"">

Fig.1. Lobules and ducts of the breast.

4.1 Introduction

In Kenya, cancer is the third leading cause of death after infectious and cardiovascular diseases. From 2012 to 2018, the annual incidence of cancer increased from 37,000 to 47,887 new cases.7 During the same period, annual cancer mortality rose almost 16%, from 28,500 to 32,987 cancer-related deaths. The number of new cancer cases is expected to rise by more than 120% over the next 2 decades.

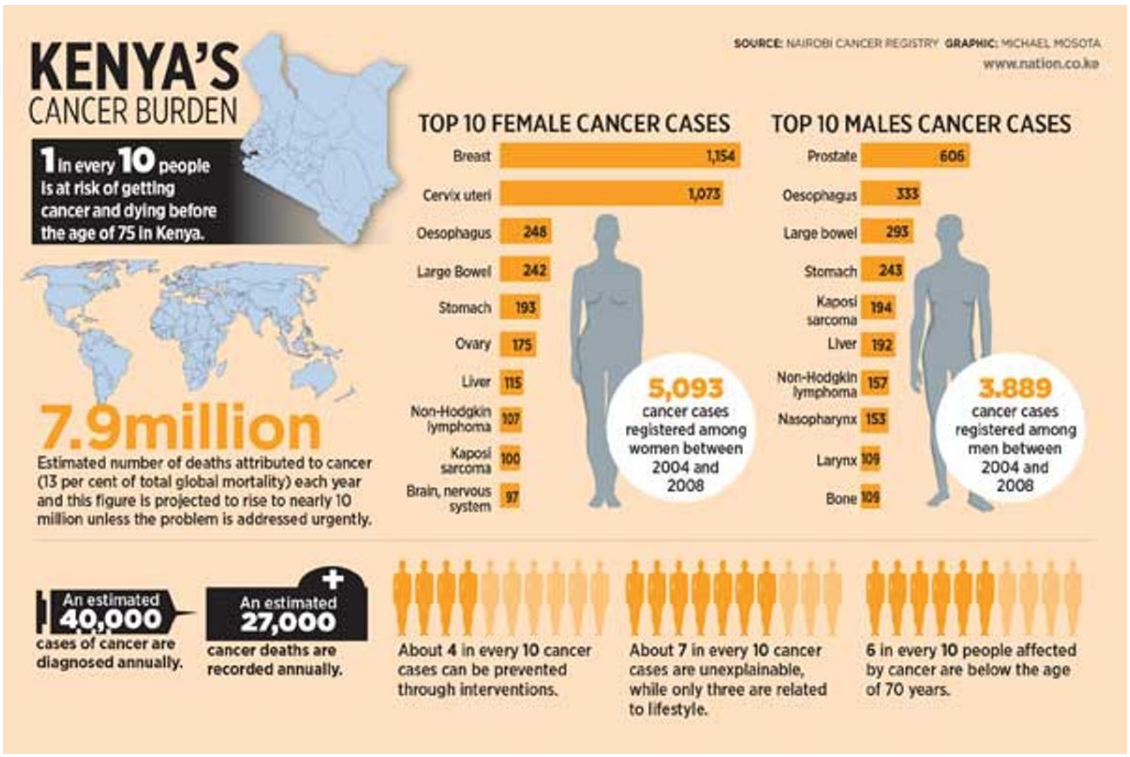
[1]:
```
# data processing and linear algebra

import numpy as np
import pandas as pd
from numba import njit

#Data Visualiation
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline
import seaborn as sns
sns.set()
import cv2 as cv

# Machine learning alogorithm
from sklearn import svm
from sklearn.model_selection import train_test_split, StratifiedKFold
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
from keras.optimizers import Adam
from keras import Input, Model
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪plot_confusion_matrix
import keras


from glob import glob
from skimage import io
from os import listdir
import pickle
import random

import random
from pprint import pprint
import time
import copy
from tqdm.notebook import tqdm
tqdm().pandas();5
```

```
print('import complete')
```

```
HBox(children=(HTML(value=''), FloatProgress(value=1.0, bar_style='info',␣
↪layout=Layout(width='20px'), max=1.0…

import complete

/opt/conda/lib/python3.8/site-packages/tqdm/std.py:703: FutureWarning: The Panel
class is removed from pandas. Accessing it from the top-level namespace will
also be removed in the next version
  from pandas import Panel
```

[2]:
```
Image(filename='output/png/stats.jpg', width = 1000, height = 300)
```

[2]:



4.2 About the data

The original dataset consisted of 162 whole mount slide images of Breast Cancer (BCa) specimens scanned at 40x. From that, 277,524 patches of size 50 x 50 were extracted (198,738 IDC negative and 78,786 IDC positive). Each patch's file name is of the format: uxXyYclassC.png — > example 10253idx5x1351y1101class0.png . Where u is the patient ID (10253idx5), X is the x-coordinate of where this patch was cropped from, Y is the y-coordinate of where this patch was cropped from, and C indicates the class where 0 is non-IDC and 1 is IDC.

4.3 Data preparation

[3]:
```
data = listdir("output/archive/IDC_regular_ps50_idx5")
len(data)
```

[3]: 279

[4]:
```
file = listdir("output/archive")
len(file)
```

[4]: 280

We have a total number of about 280 sub-folders, let's take a peak into the folder and try and understand what those sub-folders are.

Each subfolder seems to be the ID of the corresponding patient

[5]:
```
patient_file = listdir("output/archive/IDC_regular_ps50_idx5/13689")
len(patient_file)
```

[5]: 2

Each file has 2 sub-folders, labeled 1 and 0

1. Folder 0: Non-Invasive Ductal Carcinoma (IDC)
2. Folder 1 : Invasive Ductal Carcinoma (IDC)

[6]:
```
zero = listdir('output/archive/IDC_regular_ps50_idx5/13689/0')
one = listdir('output/archive/IDC_regular_ps50_idx5/13689/1')
print(f"The number of NON IDC in Id 13689 is : {len(zero)}")
print(f"The number of IDC in Id 13689 is : {len(one)}")
```

```
The number of NON IDC in Id 13689 is : 510
The number of IDC in Id 13689 is : 76
```

5 What do we know about our data?

1. Now that we have a good understanding of the file structure let's try and understand how much data we are about to process.
2. How many patients do we have?
   - It seems that we have a total number of 280 patients.
   - This sample size is relatively small therefore we have to be careful not to overfit our model. We need to implement our model in such a way that it maximizes generalization.
3. Each patient has a batch of patches that were extracted, therefore the total number of patches is likely much greater than 280.

6 NEXT STEP

1. How many patches do we have in total?
2. Which of them are IDC patches and which are Non-IDC?
    - In order to train our model we need to feed our model each patch individually, therefore each patch will act as an input.
    - The snippet below loops through the entire file structure and extracts the total number of crops for each of the classes.

[7]:
```
train = 'output/archive/IDC_regular_ps50_idx5/'
patient_ids = listdir(train)
```

[8]:
```
class_0_total = 0
class_1_total = 0
```

[9]:
```
for patient_id in patient_ids:
    class_0_files = listdir(train + patient_id + '/0')
    class_1_files = listdir(train + patient_id + '/1')

    class_0_total += len(class_0_files)
    class_1_total += len(class_1_files)

total_images = class_0_total + class_1_total

print(f'Number of patches in Class 0: {class_0_total}')
print(f'Number of patches in Class 1: {class_1_total}')
print(f'Total number of patches: {total_images}')
```

```
Number of patches in Class 0: 198738
Number of patches in Class 1: 78786
Total number of patches: 277524
```

Storing the image_path, patient_id, target and x & y coordinates

[10]:
```
columns = ["patient_id",'x','y',"target","path"]
data_rows = []
i = 0
iss = 0
isss = 0

# note that we loop through the classes after looping through the
# patient ids so that we avoid splitting our data into [all class 0 then all␣
 ↪class 1]
for patient_id in patient_ids:
    for c in [0,1]:
        class_path = train + patient_id + '/' + str(c) + '/'
        imgs = listdir(class_path)
```

```
        # Extracting Image Paths
        img_paths = [class_path + img + '/' for img in imgs]

        # Extracting Image Coordinates
        img_coords = [img.split('_',4)[2:4] for img in imgs]
        x_coords = [int(coords[0][1:]) for coords in img_coords]
        y_coords = [int(coords[1][1:]) for coords in img_coords]

        for (path,x,y) in zip(img_paths,x_coords,y_coords):
            values = [patient_id,x,y,c,path]
            data_rows.append({k:v for (k,v) in zip(columns,values)})
```

[11]:
```
# We create a new dataframe using the list of dicts that we generated above
data = pd.DataFrame(data_rows)
print(data.shape)
data
```

```
(277524, 5)
```

[11]:

	patient_id	x	y	target	path
0	9125	901	1451	0	output/archive/IDC_regular_ps50_idx5/9125/0/91…
1	9125	1551	2101	0	output/archive/IDC_regular_ps50_idx5/9125/0/91…
2	9125	1651	301	0	output/archive/IDC_regular_ps50_idx5/9125/0/91…
3	9125	601	1651	0	output/archive/IDC_regular_ps50_idx5/9125/0/91…
4	9125	1701	1	0	output/archive/IDC_regular_ps50_idx5/9125/0/91…
…	…	…	…	…	…
277519	13404	2251	1901	1	output/archive/IDC_regular_ps50_idx5/13404/1/1…
277520	13404	2201	2001	1	output/archive/IDC_regular_ps50_idx5/13404/1/1…
277521	13404	2351	2051	1	output/archive/IDC_regular_ps50_idx5/13404/1/1…
277522	13404	2101	2001	1	output/archive/IDC_regular_ps50_idx5/13404/1/1…
277523	13404	1751	1801	1	output/archive/IDC_regular_ps50_idx5/13404/1/1…

```
[277524 rows x 5 columns]
```

6.1 Data Analysis

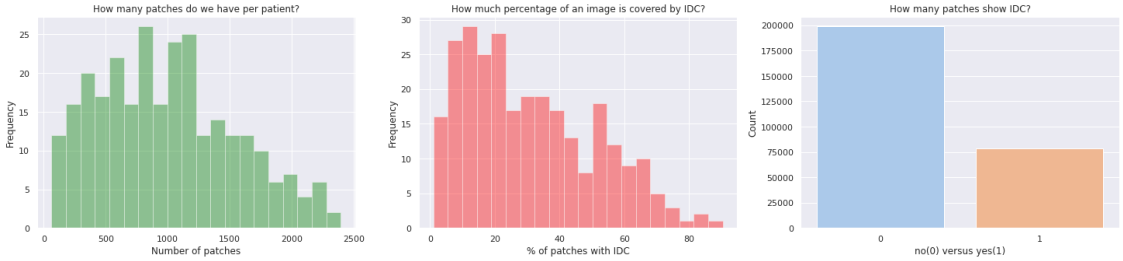
Now that we have set up our data, let's create a visual summary to help us draw some insights from our data.

[12]:
```
#default theme
sns.set(context='notebook', style='darkgrid', palette='colorblind',␣
 ↪font='sans-serif', font_scale=1, rc=None)
matplotlib.rcParams['figure.figsize'] =[8,8]
matplotlib.rcParams.update({'font.size': 15})
matplotlib.rcParams['font.family'] = 'sans-serif'
```

[13]:
```
cancer_perc = data.groupby("patient_id").target.value_counts()/ data.
 ↪groupby("patient_id").target.size()
cancer_perc = cancer_perc.unstack()

fig, ax = plt.subplots(1,3,figsize=(25,5))

# Plotting Frequency of Patches per Patient
sns.distplot(data.groupby("patient_id").size(), ax=ax[0], color="green",␣
 ↪kde=False, bins=20)
ax[0].set_xlabel("Number of patches")
ax[0].set_ylabel("Frequency")
ax[0].set_title("How many patches do we have per patient?")

# Plotting Percentage of an image that is covered by Invasive Ductile Carcinoma
sns.distplot(cancer_perc.loc[:, 1]*100, ax=ax[1], color="red", kde=False,␣
 ↪bins=20)
ax[1].set_title("How much percentage of an image is covered by IDC?")
ax[1].set_ylabel("Frequency")
ax[1].set_xlabel("% of patches with IDC")

# Plotting number of patches that show IDC
sns.countplot(data.target, palette='pastel', ax=ax[2]);
ax[2].set_ylabel("Count")
ax[2].set_xlabel("no(0) versus yes(1)")
ax[2].set_title("How many patches show IDC?");
```

```
/opt/conda/lib/python3.8/site-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

```
/opt/conda/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



6.1.1 Healthy Tissue Patches Vs Cancerous Tissue Patches

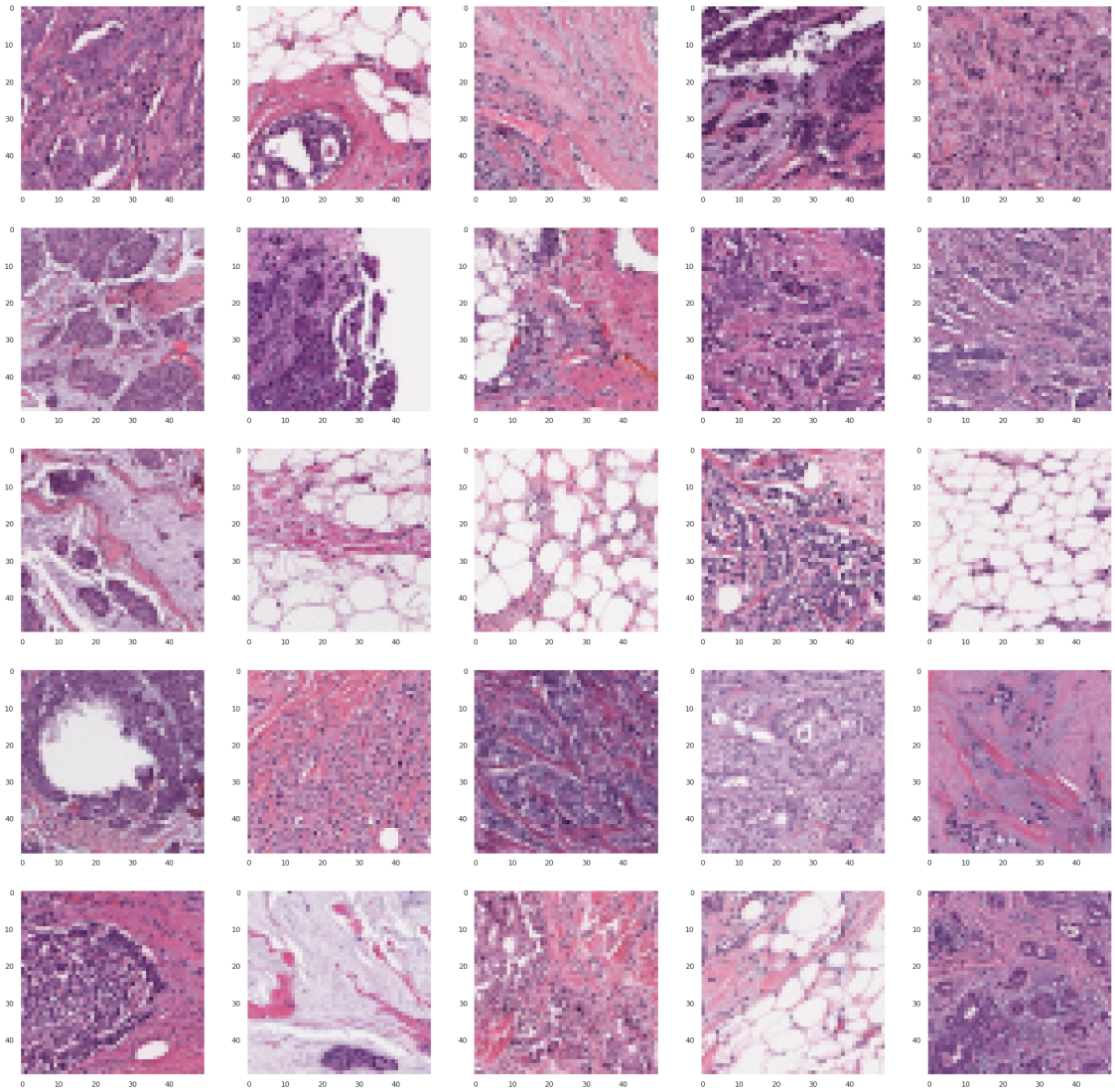
Let us now explore the visual differences between cancerous tissue cells, and healthy tissue cells. Usually partnering with a specialist is a good idea so that they can point the exact points of interest that differentiate the 2 from each other.

[14]:
```
positive_tissue = np.random.choice(data[data.target==1].index.values, size=100,␣
↪replace=False)
negative_tissue = np.random.choice(data[data.target==0].index.values, size=100,␣
↪replace=False)
```

[15]:
```
n_rows = 5
n_cols = 5
```

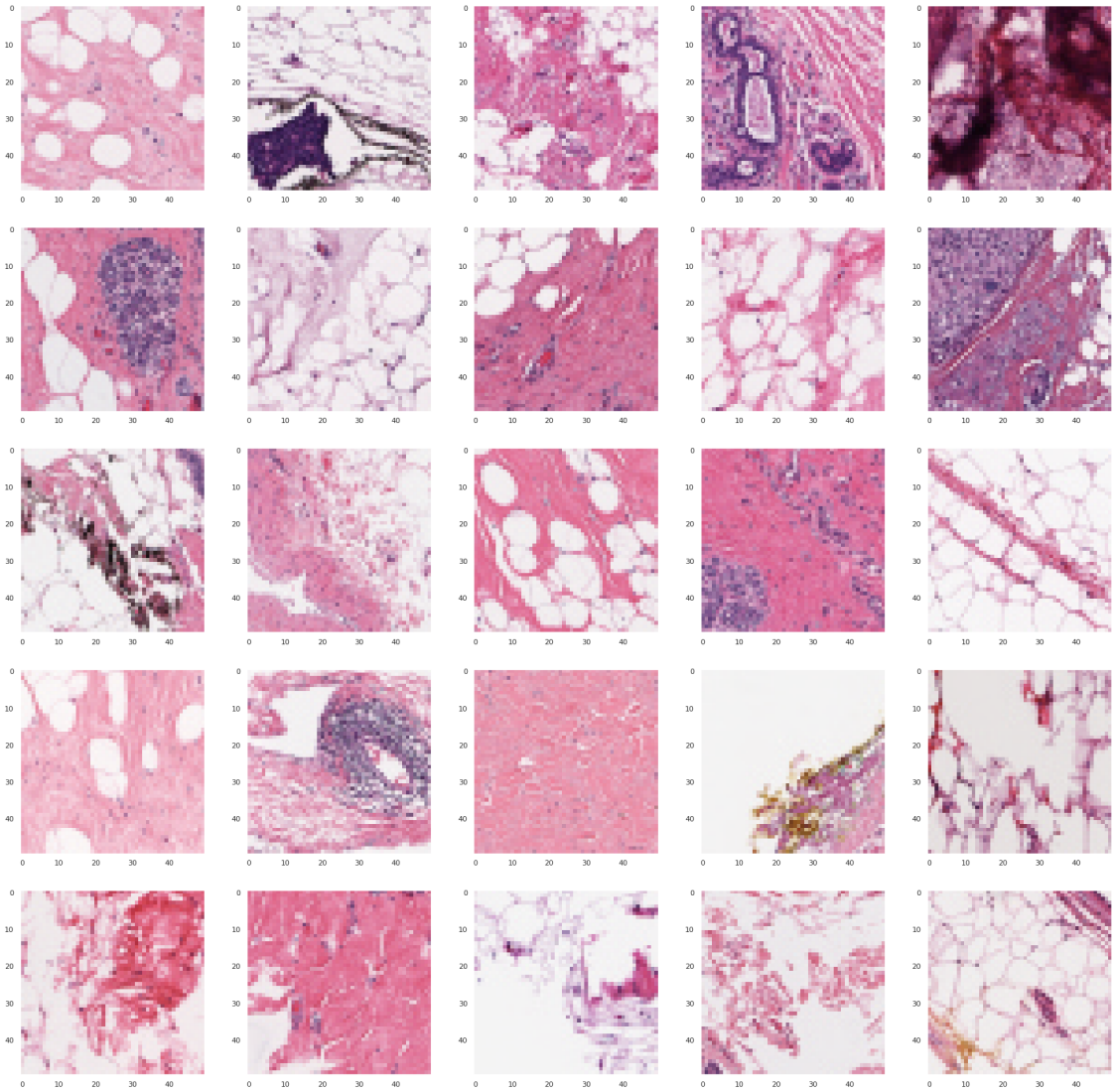
6.1.2 A) Cancerous Patches

[16]:
```
fig,ax = plt.subplots(n_rows,n_cols,figsize = (30,30))

for row in range(n_rows):
    for col in range(n_cols):
        # below is a counter to cycle through the image indexes
        idx = positive_tissue[col + n_cols*row]
        img = io.imread(data.loc[idx, "path"])
        ax[row,col].imshow(img[:,:,:])
        ax[row,col].grid(False)
```



6.1.3 B) Non-Cancerous Patches

[17]:
```
fig,ax = plt.subplots(n_rows,n_cols,figsize = (30,30))

for row in range(n_rows):
    for col in range(n_cols):
        # below is a counter to cycle through the image indices
        idx = negative_tissue[col + n_cols*row]
        img = io.imread(data.loc[idx, "path"])
        ax[row,col].imshow(img[:,:,:])
        ax[row,col].grid(False)
```



6.1.4 Visualizing the Breast Tissue

Earlier we extracted the coordinates of the cropped tissue cells, we can use those coordinates to reconstruct the whole breast tissue of the patient. This way we can explore how the diseased tissue looks when compared to the healthy tissue.

We can also explore the most common places that the cancer tends to occur in. It would be interesting to plot a heatmap of the most common areas where the cancer appears.

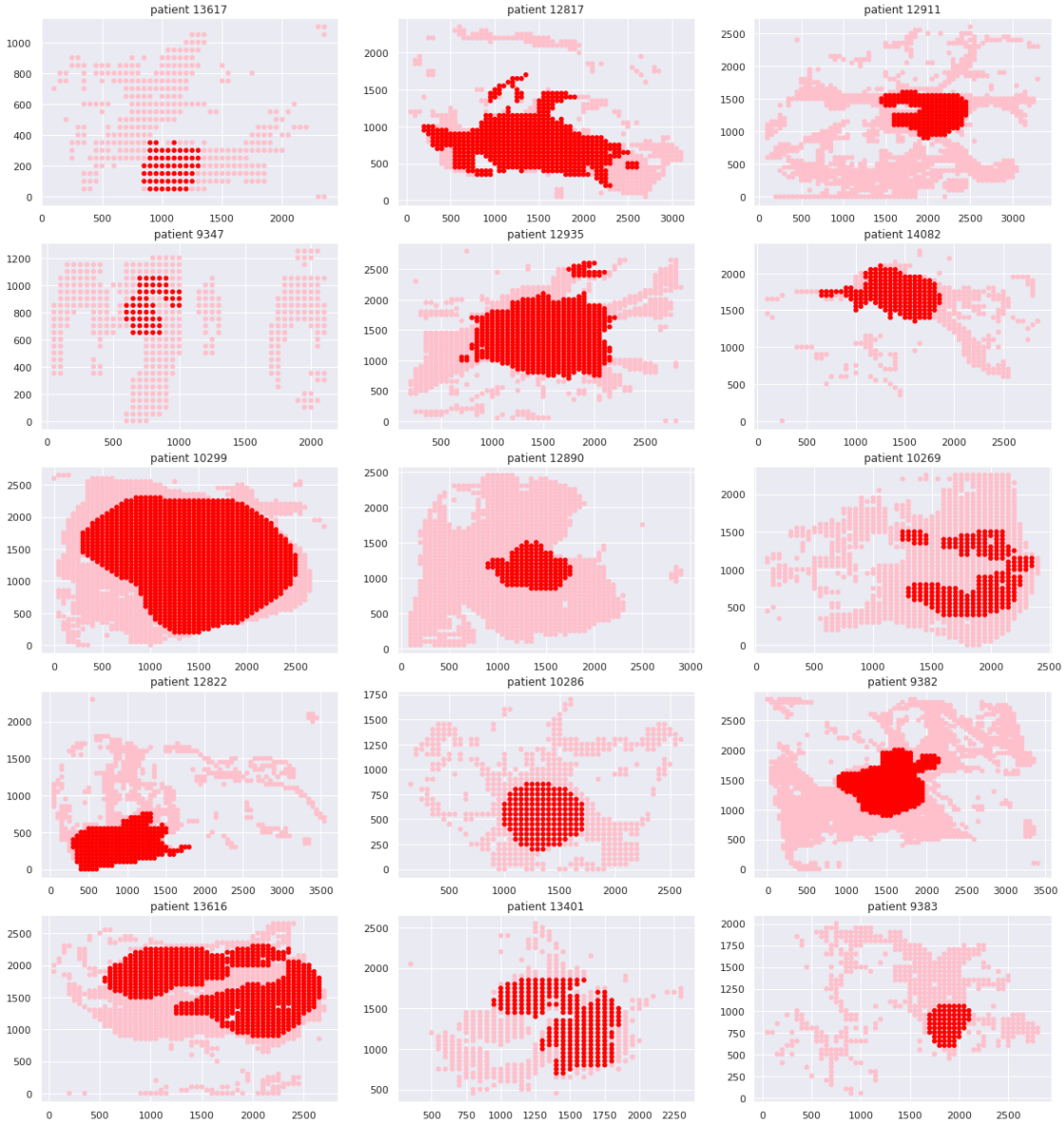
If position of the crop has significance then perhaps we can use it as an input feature for our model.

To simplify things we will create a function that slices our existing dataframe and retrieves the values associated with a patient id.

[18]:
```
def get_patient_df(patient_id):
    return data.loc[data['patient_id']== patient_id,:]
```

[19]:
```
n_rows = 5
n_cols = 3
n_imgs = n_rows*n_cols
```

[20]:
```
colors = ['pink', 'red']

fig, ax = plt.subplots(n_rows,n_cols,figsize=(20, 22))

patient_ids = np.random.choice( data.patient_id.unique(), size=n_imgs,␣
 ↪replace=False)

for row in range(n_rows):
    for col in range(n_cols):
        patient_id = patient_ids[col + n_cols*row]
        patient_df = get_patient_df(patient_id)

        ax[row,col].scatter(patient_df.x.values, \
                            patient_df.y.values, \
                            c=patient_df.target.values,\
                            cmap=ListedColormap(colors), s=20)
        ax[row,col].set_title("patient " + patient_id)
```



6.1.5 Insights

1. Sometimes we don't have the full tissue information. It seems that tissue patches have been discarded or lost during preparation.
2. Cancerous Tissue tends to appear in clusters rather than, being dispersed all over the place.

6.1.6 Repatching the Actual Breast Tissue Image

Now it's time to go one step deeper with our EDA. Instead of plotting the target values using the x-y coordinates, we now plot the images themselves on their respective x-y coordinates. This will help us visualize how the cancerous tissue looks like from a macro perspective.

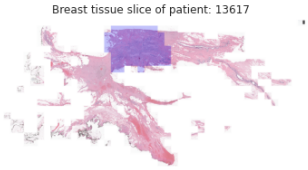
uint8 is used unsigned 8 bit integer. And that is the range of pixel. We can't have pixel value more than 2^8 -1. Therefore, for images uint8 type is used. Whereas double is used to handle very big numbers.

[21]:
```
def visualise_breast_tissue(patient_id, df = data,pred = False, crop_dimension␣
↪= [50,50]):
    # Plotting Settings
    plt.xticks([])
    plt.yticks([])
    # Get patient dataframe
    p_df = get_patient_df(patient_id)
    # Get the dimensions of the breast tissue image
    max_coord = np.max((*p_df.x,*p_df.y))
    # Allocate an array to fill image pixels in,use uint8 type as you don't␣
↪need an int over 255
    grid = 255*np.ones(shape = (max_coord + crop_dimension[0], max_coord +␣
↪crop_dimension[1], 3)).astype(np.uint8)
    mask = 255*np.ones(shape = (max_coord + crop_dimension[0], max_coord +␣
↪crop_dimension[1], 3)).astype(np.uint8)
    # Replace array values with values of the image
    for x,y,target,path in zip(p_df['x'],p_df['y'],p_df['target'],p_df['path']):
        try:
            img = io.imread(path)
            # Replace array values with cropped image values
            grid[y:y+crop_dimension[1],x:x+crop_dimension[0]] = img
            # Check if target is cancerous or not
            if target != 0:
                # If the target is cancerous then, replace array values with␣
↪the color blue
                mask[y:y+crop_dimension[1],x:x+crop_dimension[0]] = [0,0,255]
        except: pass
    # if prediction is not specifies then show the image normally
    if pred == False:
        io.imshow(grid)
        img = grid
    # if prediction is specified then apply a mask to the areas that contain␣
↪predicted cancerous cells
    else:
        # Specify the desired alpha value
        alpha = 0.78
        # This is step is very important, adding 2 numpy arrays sets the values␣
↪to float64, which is why convert them back to uint8
        img = (mask * (1.0 - alpha) + grid * alpha).astype('uint8')
        io.imshow(img)
    return img
```

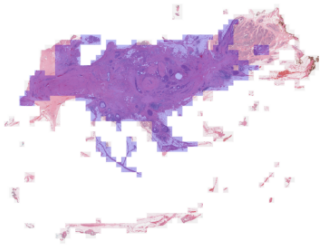
[22]:
```
n_rows = 5
n_cols = 3
n_imgs = n_rows*n_cols
```

[23]:
```
fig, ax = plt.subplots(n_rows,n_cols,figsize=(20, 27))

for row in range(n_rows):
    for col in range(n_cols):
        p_id = patient_ids[col + n_cols*row]

        img = visualise_breast_tissue(p_id, pred = True)
        ax[row,col].grid(False)
        ax[row,col].set_xticks([])
        ax[row,col].set_yticks([])
        ax[row,col].set_title("Breast tissue slice of patient: " + p_id)
        ax[row,col].imshow(img)
```

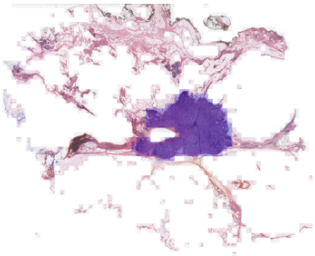
Breast tissue slice of patient: 13617



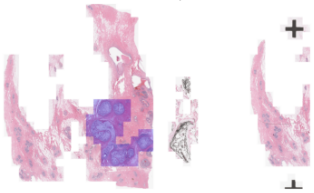
Breast tissue slice of patient: 12817



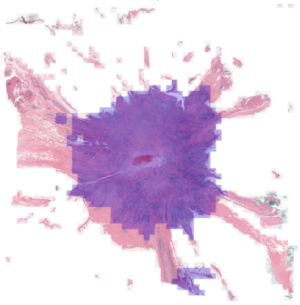
Breast tissue slice of patient: 12911



Breast tissue slice of patient: 9347

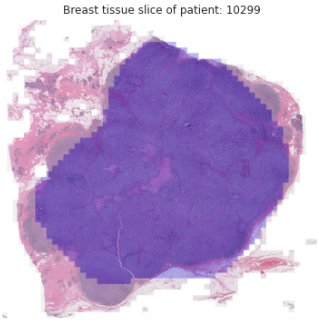


Breast tissue slice of patient: 12935

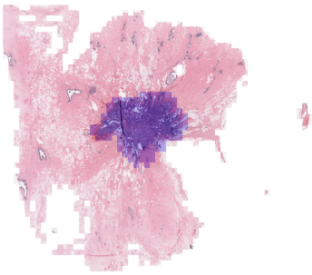


Breast tissue slice of patient: 14082

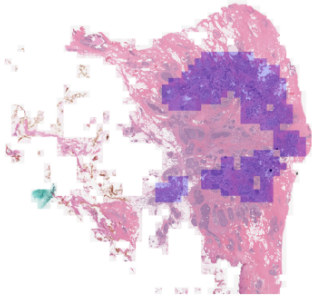
Breast tissue slice of patient: 10299



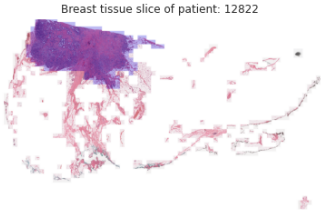
Breast tissue slice of patient: 12890



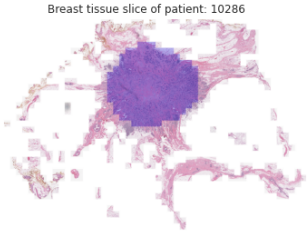
Breast tissue slice of patient: 10269



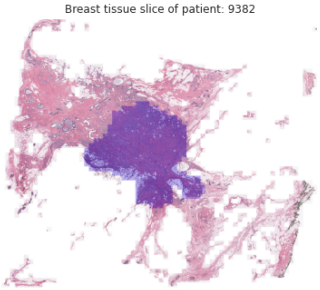
Breast tissue slice of patient: 12822



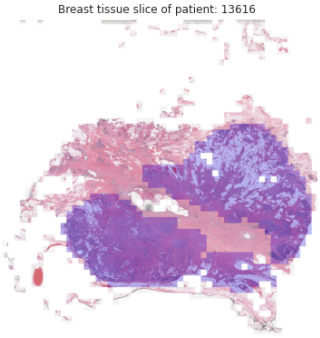
Breast tissue slice of patient: 10286



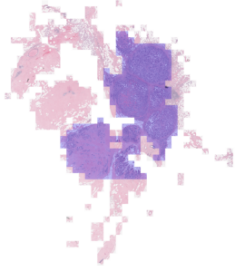
Breast tissue slice of patient: 9382



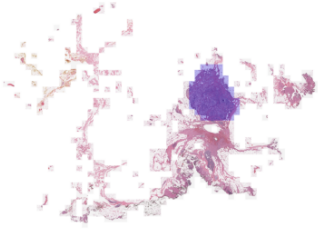
Breast tissue slice of patient: 13616



Breast tissue slice of patient: 13401



Breast tissue slice of patient: 9383



[24]:
```
# Get the file dir
breast_img = glob('output/archive/IDC_regular_ps50_idx5/**/*.png', recursive =␣
↪True)
```

[25]:
```
non_img = []
can_img = []

for img in breast_img:
    if img[-5] == '0' :
        non_img.append(img)

    elif img[-5] == '1' :
        can_img.append(img)
```

[26]:
```
non_num = len(non_img)
can_num = len(can_img)

total_img_num = non_num + can_num

print('Number of Images in IDC (-): {}' .format(non_num))
print('Number of Images in IDC (+) : {}' .format(can_num))
print('Total Number of Images : {}' .format(total_img_num))
```

```
Number of Images in IDC (-): 198738
Number of Images in IDC (+) : 78786
Total Number of Images : 277524
```

6.2 Oversampling

[27]:
```
#can_img = can_img * 2
```

[28]:
```
some_non_img = random.sample(non_img, len(can_img))
some_can_img = random.sample(can_img, len(can_img))

non_img_arr = []
can_img_arr = []

for img in some_non_img:
    n_img = cv.imread(img, cv.IMREAD_COLOR)
    n_img_size = cv.resize(n_img, (50, 50), interpolation = cv.INTER_LINEAR)
    non_img_arr.append([n_img_size, 0])

for img in some_can_img:
    c_img = cv.imread(img, cv.IMREAD_COLOR)
    c_img_size = cv.resize(c_img, (50, 50), interpolation = cv.INTER_LINEAR)
```

```
    can_img_arr.append([c_img_size, 1])
```

[29]:
```
X = []
y = []

breast_img_arr = np.concatenate((non_img_arr, can_img_arr))
random.shuffle(breast_img_arr)

for feature, label in breast_img_arr:
    X.append(feature)
    y.append(label)

X = np.array(X)
y = np.array(y)

print('X shape : {}' .format(X.shape))
```

```
X shape : (157572, 50, 50, 3)
```

6.3 Data Scaling

[30]:
```
#X = X/255
```

[31]:
```
X_train, X_predict, y_train, y_true = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 128)

rate = 0.8
num = int(X_train.shape[0] * rate)

X_test = X_train[num:]
X_train = X_train[:num]

y_test = y_train[num:]
y_train = y_train[:num]

y_train = to_categorical(y_train, 2)
y_test = to_categorical(y_test, 2)
y_true = to_categorical(y_true, 2)

print('X_train shape : {}' .format(X_train.shape))
print('X_test shape : {}' .format(X_test.shape))
print('X_predict shape : {}' .format(X_predict.shape))
print('y_train shape : {}' .format(y_train.shape))
print('y_test shape : {}' .format(y_test.shape))
print('y_true shape : {}' .format(y_true.shape))
```

```
X_train shape : (100845, 50, 50, 3)
```

```
X_test shape : (25212, 50, 50, 3)
X_predict shape : (31515, 50, 50, 3)
y_train shape : (100845, 2)
y_test shape : (25212, 2)
y_true shape : (31515, 2)
```

6.4 Convolutional neural networks (CNN)

CNN emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s. According to Aurelien, CNNs have man- aged to achieve superhuman performance on some complex visual tasks. Some of their application include: * Power image search services * Self-driving cars * Automatic video classification systems

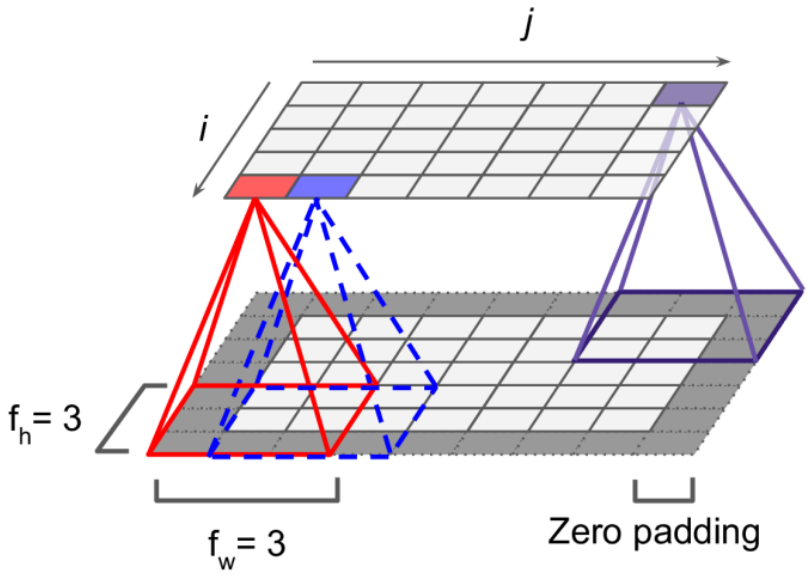
6.4.1 Convolutional Layer

Convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network.

Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.

[32]:
```
Image(filename='output/png/conv.png', width = 1500, height = 700)
```

[32]:

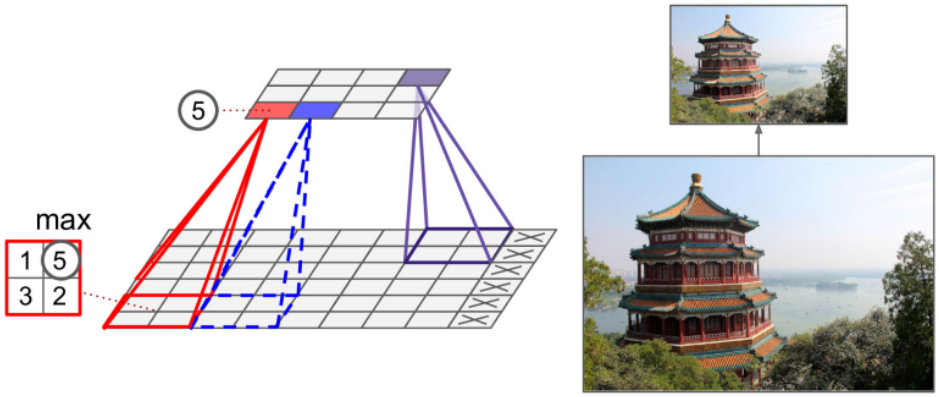


6.4.2 Pooling Layer

Their goal is to subsample (i.e., shrink) the input image in order to reduce: * The computational load * The memory usage * The number of parameters (thereby limiting the risk of overfitting).

[33]:
```
Image(filename='output/png/maxpool.png', width = 1500, height = 700)
```

[33]:



[34]:
```
def get_model(inp_shape = (50, 50, 3)):
    inp = Input(inp_shape)
    m = Conv2D(32, (3, 3) , padding="same", activation='relu')(inp)
    m = MaxPooling2D(2)(m)
    m = Dropout(0.25)(m)

    m = Conv2D(64, (3, 3), padding="same", activation='relu')(m)
    m = MaxPooling2D(2)(m)
    m = Dropout(0.25)(m)

    m = Conv2D(128, (3, 3), padding="same", activation='relu')(m)
    m = MaxPooling2D(2)(m)
    m = Dropout(0.25)(m)

    m = Conv2D(128, (3, 3), padding="same", activation='relu')(m)
    m = MaxPooling2D(2)(m)
    m = Dropout(0.25)(m)

    m = Flatten()(m)
    m = Dense(128, activation = "relu")(m)

    m = Dropout(0.5)(m)
    out = Dense(2, activation = "sigmoid")(m)

    model = Model(inp, out)
    model.compile(optimizer=Adam(learning_rate = 0.0001),␣
 ↪loss="binary_crossentropy", metrics = ['accuracy'])
    return model
```

[35]:
```
model = get_model()
model.summary()
```

```
Model: "model"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 50, 50, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 50, 50, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 25, 25, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 128)         147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 3, 3, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 128)               147584
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 388,674
Trainable params: 388,674
Non-trainable params: 0
_________________________________________________________________
```

[36]:
```
callbacks = [keras.callbacks.EarlyStopping(monitor="loss", patience=3)]
```

[37]:
```
history = model.fit(X_train, y_train, validation_data = (X_test, y_test),␣
↪epochs = 30, batch_size = 256, callbacks=callbacks)
```

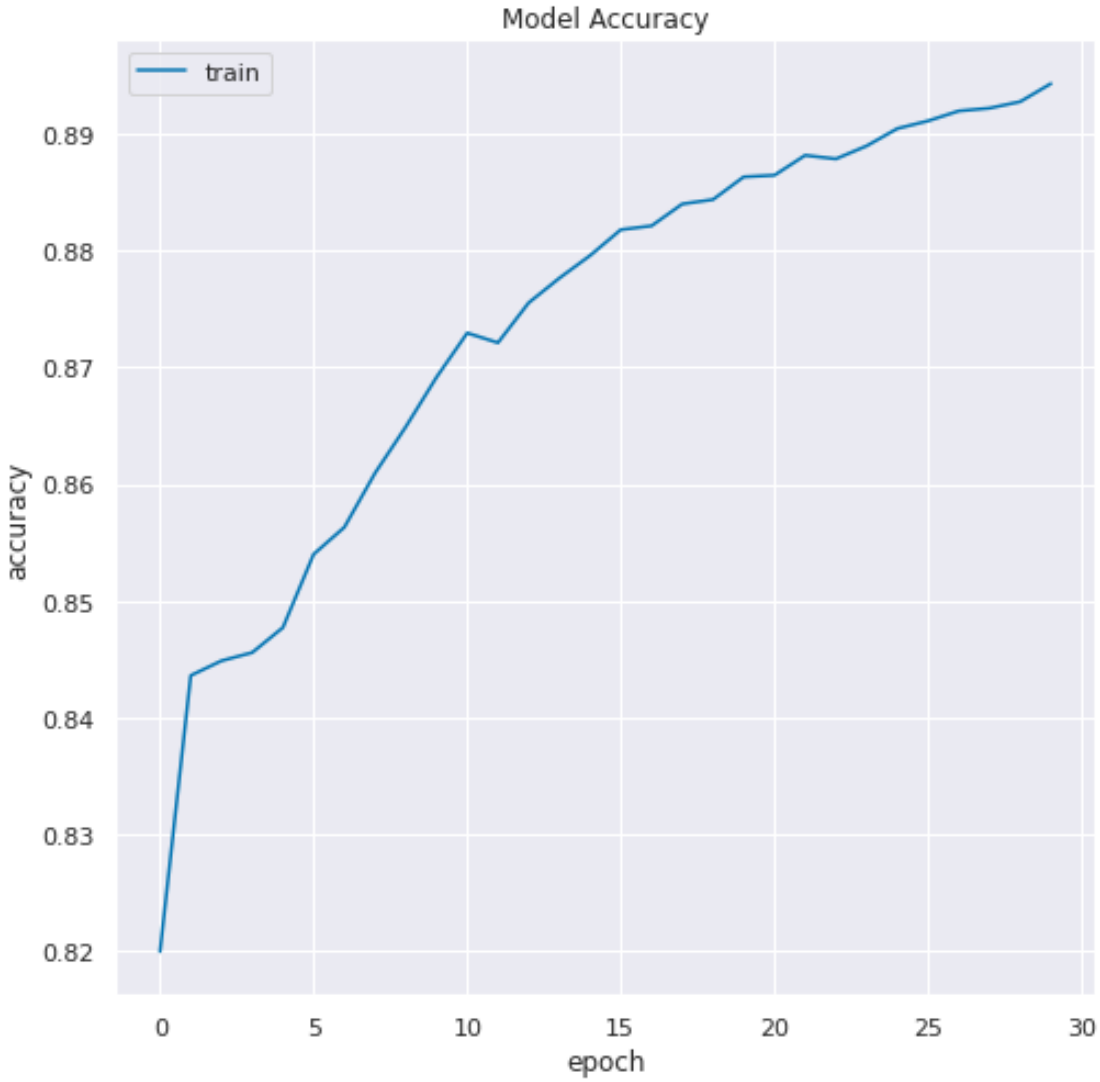
```
Epoch 1/30
```

```
394/394 [==============================] - 161s 406ms/step - loss: 3.7837 -
accuracy: 0.7981 - val_loss: 0.5581 - val_accuracy: 0.8522
Epoch 2/30
394/394 [==============================] - 164s 417ms/step - loss: 0.4463 -
accuracy: 0.8431 - val_loss: 0.4708 - val_accuracy: 0.8502
Epoch 3/30
394/394 [==============================] - 165s 418ms/step - loss: 0.3761 -
accuracy: 0.8458 - val_loss: 0.5701 - val_accuracy: 0.8515
Epoch 4/30
394/394 [==============================] - 163s 415ms/step - loss: 0.3551 -
accuracy: 0.8467 - val_loss: 0.5612 - val_accuracy: 0.8516
Epoch 5/30
394/394 [==============================] - 163s 414ms/step - loss: 0.3429 -
accuracy: 0.8466 - val_loss: 0.5668 - val_accuracy: 0.8576
Epoch 6/30
394/394 [==============================] - 166s 421ms/step - loss: 0.3335 -
accuracy: 0.8526 - val_loss: 0.4998 - val_accuracy: 0.8641
Epoch 7/30
394/394 [==============================] - 166s 421ms/step - loss: 0.3270 -
accuracy: 0.8583 - val_loss: 0.5151 - val_accuracy: 0.8566
Epoch 8/30
394/394 [==============================] - 168s 426ms/step - loss: 0.3241 -
accuracy: 0.8603 - val_loss: 0.5108 - val_accuracy: 0.8428
Epoch 9/30
394/394 [==============================] - 156s 396ms/step - loss: 0.3168 -
accuracy: 0.8636 - val_loss: 0.4632 - val_accuracy: 0.8299
Epoch 10/30
394/394 [==============================] - 152s 386ms/step - loss: 0.3112 -
accuracy: 0.8689 - val_loss: 0.4546 - val_accuracy: 0.8124
Epoch 11/30
394/394 [==============================] - 154s 392ms/step - loss: 0.3038 -
accuracy: 0.8730 - val_loss: 0.4414 - val_accuracy: 0.8303
Epoch 12/30
394/394 [==============================] - 152s 386ms/step - loss: 0.2987 -
accuracy: 0.8734 - val_loss: 0.4176 - val_accuracy: 0.8404
Epoch 13/30
394/394 [==============================] - 143s 364ms/step - loss: 0.2970 -
accuracy: 0.8753 - val_loss: 0.3969 - val_accuracy: 0.8466
Epoch 14/30
394/394 [==============================] - 130s 330ms/step - loss: 0.2954 -
accuracy: 0.8750 - val_loss: 0.3379 - val_accuracy: 0.8685
Epoch 15/30
394/394 [==============================] - 130s 330ms/step - loss: 0.2880 -
accuracy: 0.8799 - val_loss: 0.3604 - val_accuracy: 0.8691
Epoch 16/30
394/394 [==============================] - 132s 334ms/step - loss: 0.2850 -
accuracy: 0.8810 - val_loss: 0.3122 - val_accuracy: 0.8849
Epoch 17/30
```

```
394/394 [==============================] - 133s 337ms/step - loss: 0.2858 -
accuracy: 0.8806 - val_loss: 0.3360 - val_accuracy: 0.8812
Epoch 18/30
394/394 [==============================] - 142s 360ms/step - loss: 0.2809 -
accuracy: 0.8833 - val_loss: 0.3626 - val_accuracy: 0.8720
Epoch 19/30
394/394 [==============================] - 136s 346ms/step - loss: 0.2819 -
accuracy: 0.8830 - val_loss: 0.3371 - val_accuracy: 0.8778
Epoch 20/30
394/394 [==============================] - 130s 331ms/step - loss: 0.2759 -
accuracy: 0.8862 - val_loss: 0.3221 - val_accuracy: 0.8883
Epoch 21/30
394/394 [==============================] - 129s 327ms/step - loss: 0.2737 -
accuracy: 0.8866 - val_loss: 0.3369 - val_accuracy: 0.8766
Epoch 22/30
394/394 [==============================] - 129s 329ms/step - loss: 0.2721 -
accuracy: 0.8886 - val_loss: 0.3290 - val_accuracy: 0.8904
Epoch 23/30
394/394 [==============================] - 129s 328ms/step - loss: 0.2740 -
accuracy: 0.8867 - val_loss: 0.3283 - val_accuracy: 0.8845
Epoch 24/30
394/394 [==============================] - 129s 328ms/step - loss: 0.2664 -
accuracy: 0.8891 - val_loss: 0.3004 - val_accuracy: 0.8940
Epoch 25/30
394/394 [==============================] - 129s 328ms/step - loss: 0.2633 -
accuracy: 0.8915 - val_loss: 0.3011 - val_accuracy: 0.8956
Epoch 26/30
394/394 [==============================] - 129s 327ms/step - loss: 0.2615 -
accuracy: 0.8930 - val_loss: 0.3025 - val_accuracy: 0.8927
Epoch 27/30
394/394 [==============================] - 129s 326ms/step - loss: 0.2599 -
accuracy: 0.8925 - val_loss: 0.2943 - val_accuracy: 0.8969
Epoch 28/30
394/394 [==============================] - 128s 325ms/step - loss: 0.2588 -
accuracy: 0.8933 - val_loss: 0.3141 - val_accuracy: 0.8939
Epoch 29/30
394/394 [==============================] - 128s 325ms/step - loss: 0.2591 -
accuracy: 0.8923 - val_loss: 0.2958 - val_accuracy: 0.8981
Epoch 30/30
394/394 [==============================] - 128s 326ms/step - loss: 0.2584 -
accuracy: 0.8937 - val_loss: 0.2959 - val_accuracy: 0.8945
```

[38]:
```
plt.plot(history.history['accuracy'])
plt.title('Model Accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(['train', 'test'], loc='upper left')
```

```
plt.show()
```

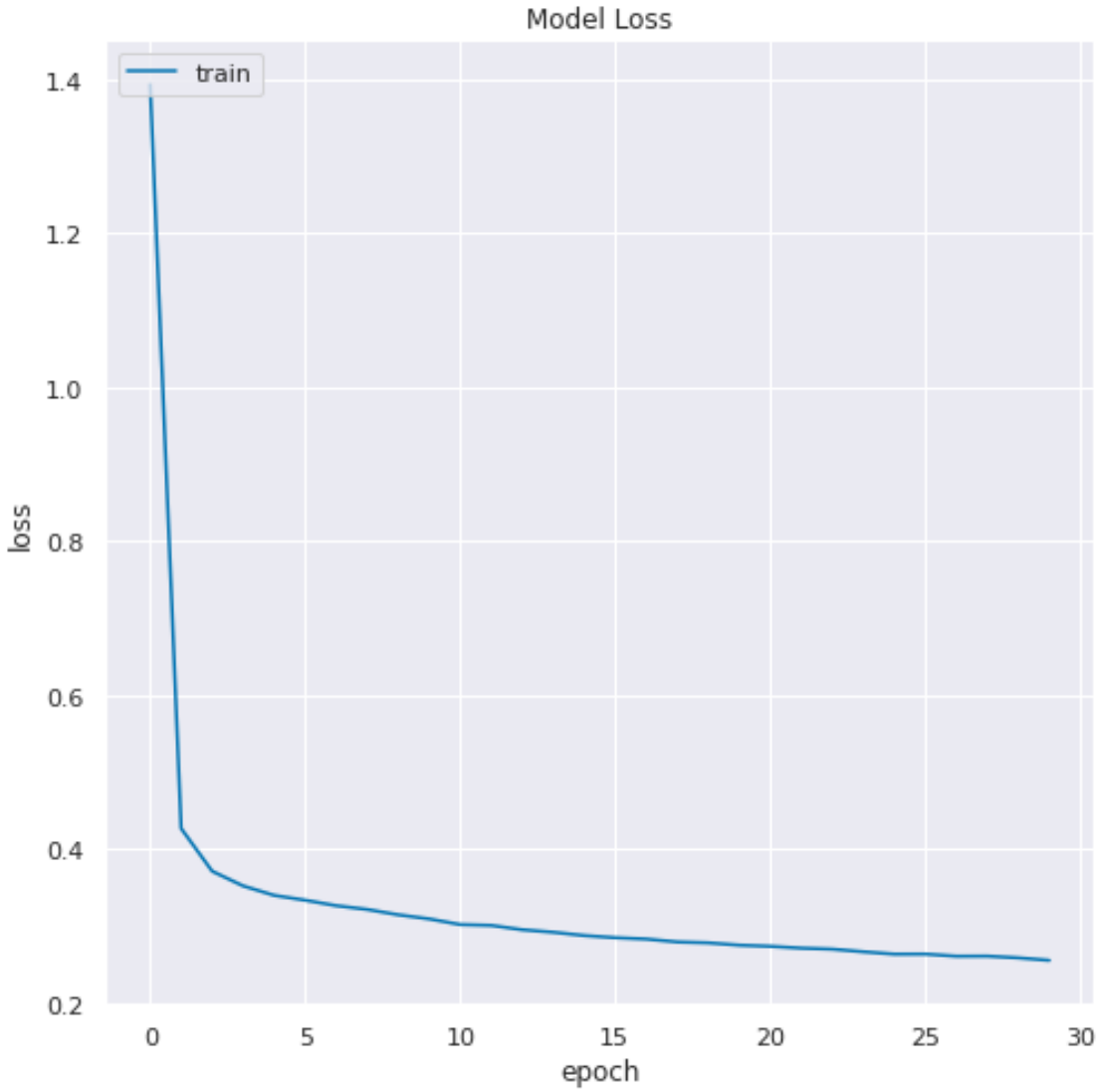


[39]:
```
result = model.evaluate(X_test, y_test, batch_size = 256)
print('Test Loss, Test Accuracy :', result)
```

```
99/99 [==============================] - 9s 88ms/step - loss: 0.2959 - accuracy:
0.8945
Test Loss, Test Accuracy : [0.2959146201610565, 0.8944550156593323]
```

[40]:
```
plt.plot(history.history['loss'])
plt.title('Model Loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'], loc='upper left')
```

```
plt.show()
```



[41]:
```
y_pred = model.predict(X_predict)

true = 0
for i in range(X_predict.shape[0]):
    if(np.argmax(y_pred[i]) == np.argmax(y_true[i])):
        true = true + 1

pre_accuracy = 100 * float(true/X_predict.shape[0])
print('Predict Accuracy: {}' .format(pre_accuracy))
```

```
Predict Accuracy: 89.28129462160875
```

6.5 Model Evaluation

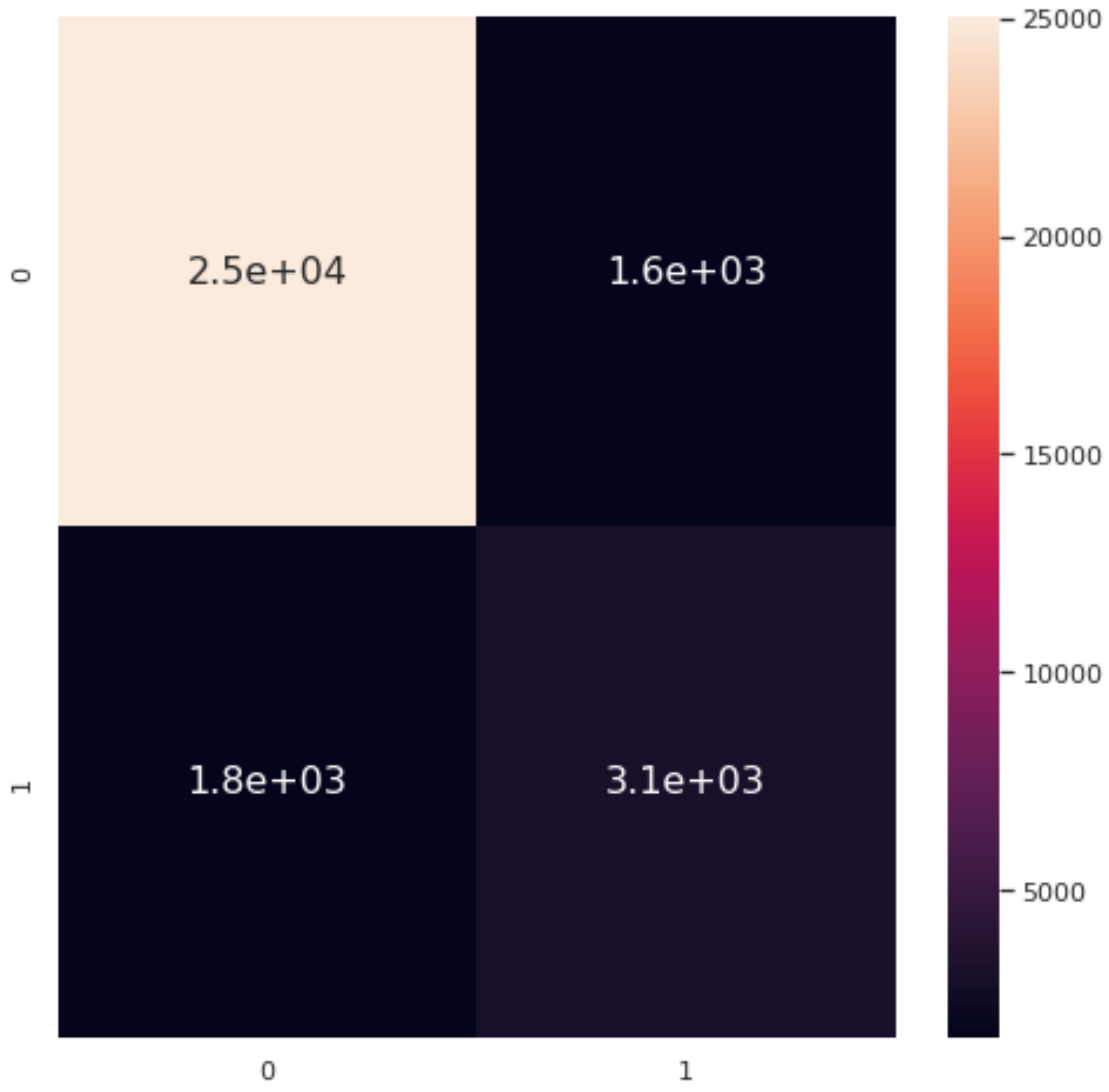
- Precision: It can be seen as a measure of a classifier's exactness. For each class, it is defined as the ratio of true positives to the sum of true and false positives. Said another way, "for all instances classified positive, what percent was correct?"
- Recall: It is a measure of the classifier's completeness; the ability of a classifier to correctly find all positive instances. For each class, it is defined as the ratio of true positives to the sum of true positives and false negatives. Said another way, "for all instances that were actually positive, what percent was classified correctly?"
- f1 score: The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy.
- support: Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

[42]:
```
print(classification_report(np.argmax(y_true, axis=1), np.argmax(y_pred,␣
↪axis=1)))
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.94     26620
           1       0.66      0.64      0.65      4895

    accuracy                           0.89     31515
   macro avg       0.80      0.79      0.79     31515
weighted avg       0.89      0.89      0.89     31515
```

[43]:
```
mat = confusion_matrix(np.argmax(y_true, axis=1), np.argmax(y_pred, axis=1))
sns.heatmap(mat, annot=True, annot_kws={"size": 16})
```

[43]: <AxesSubplot:>



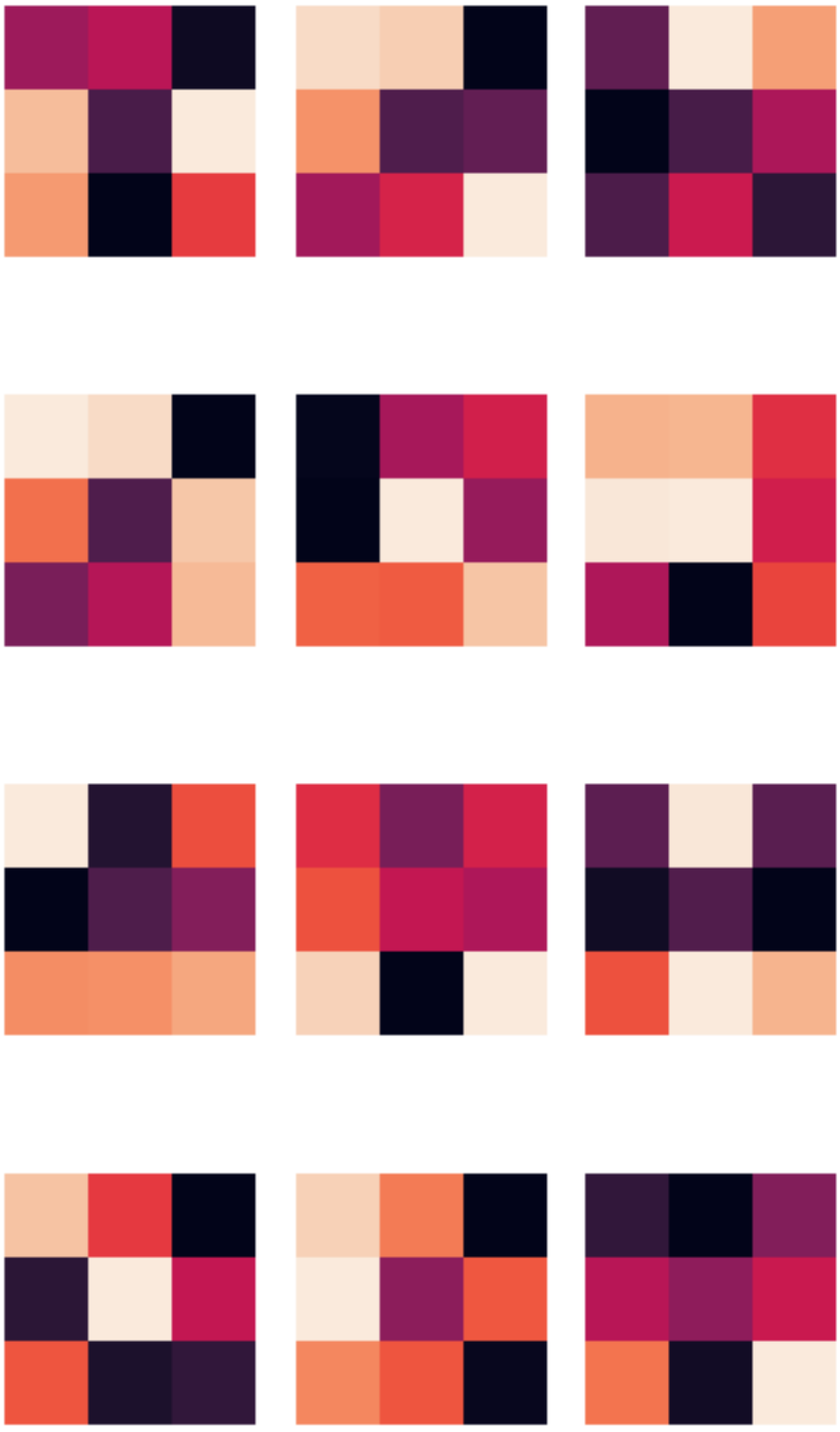
6.6 Visualizing Filters

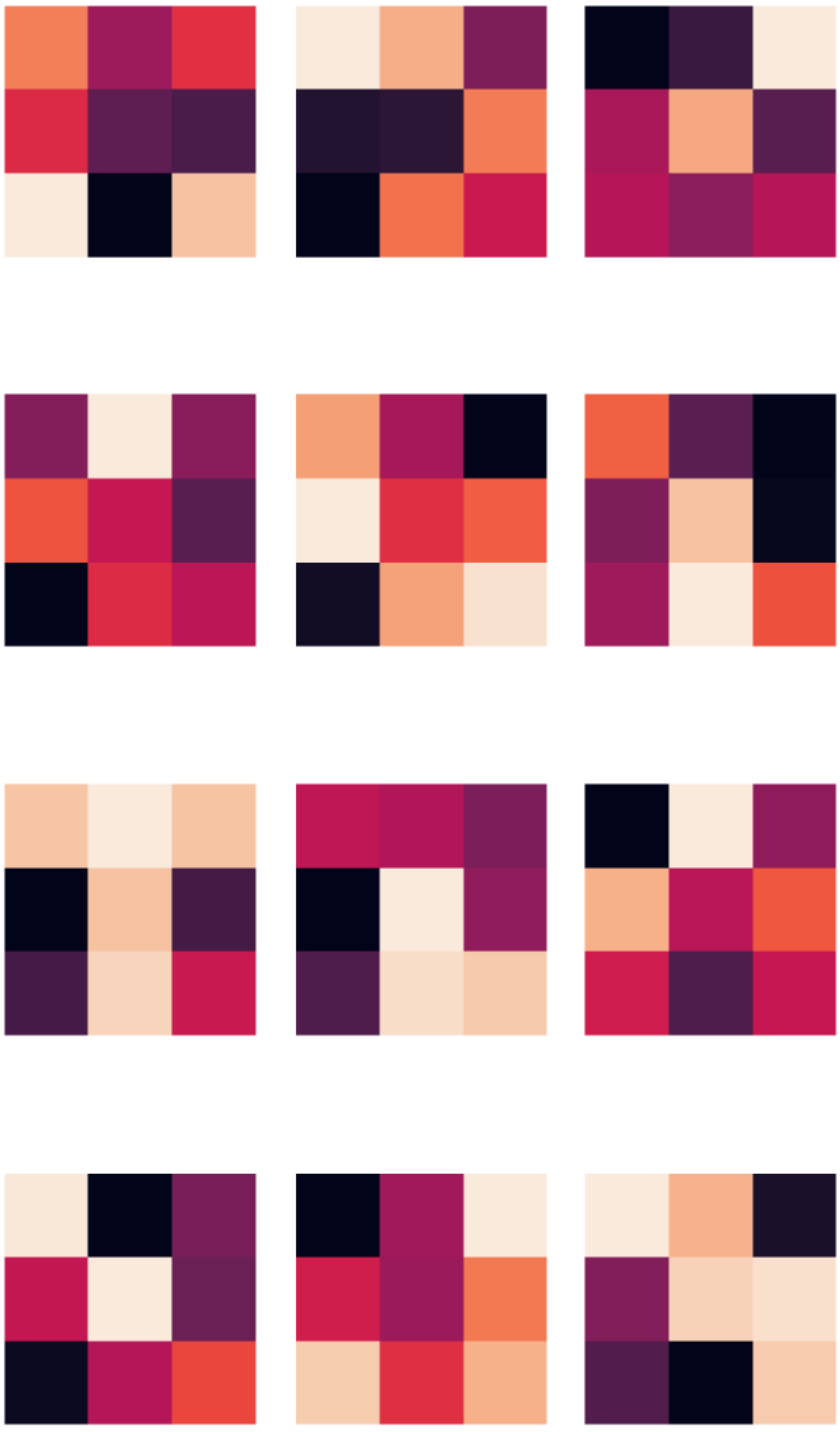
CNN uses learned filters to convolve the feature maps from the previous layer. Filters are two-dimensional weights and these weights have a spatial relationship with each other.

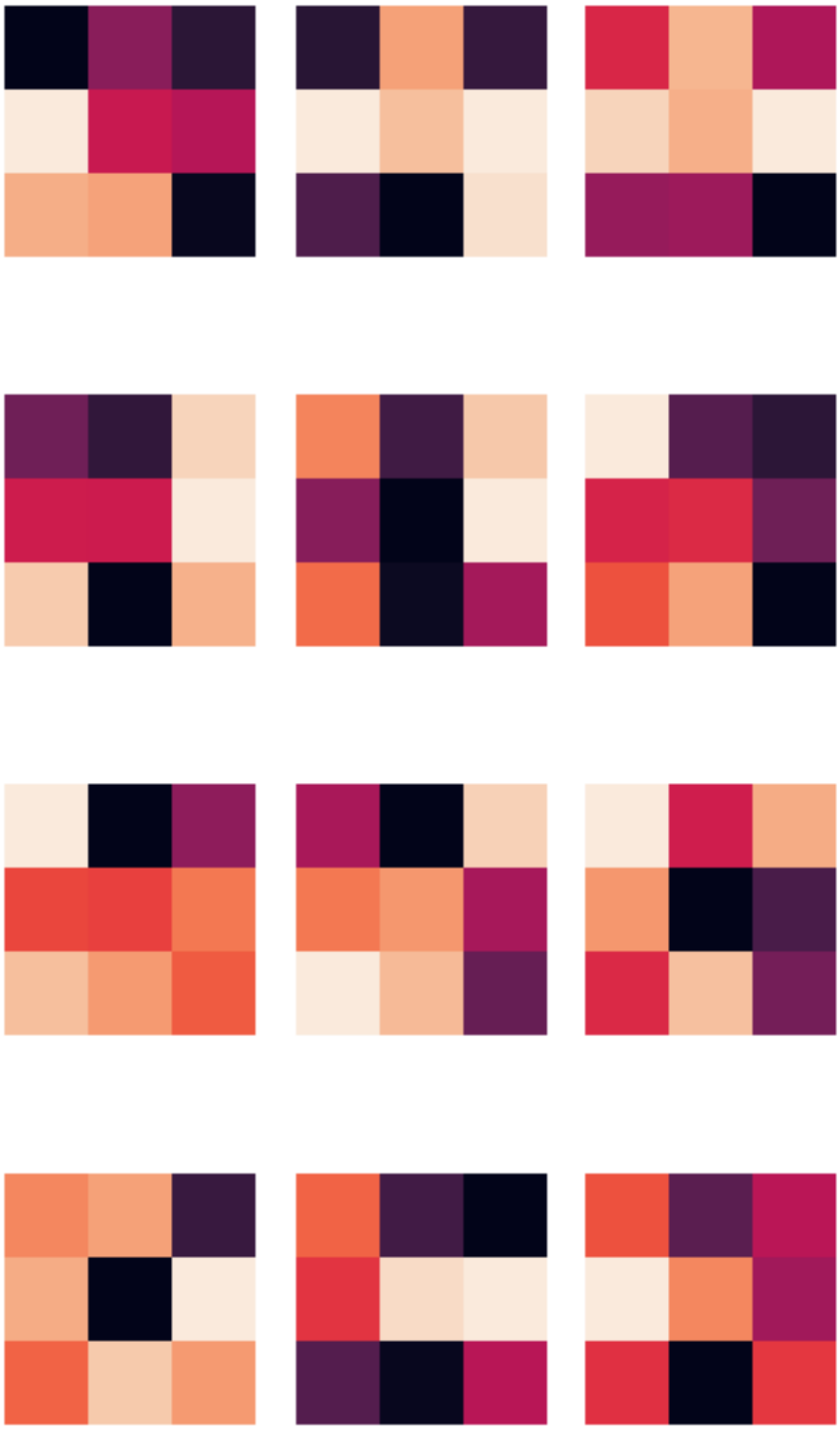
[44]:
```
def plot_filters(conv_filter):
    fig, axes = plt.subplots(1, 3, figsize=(5,5))
    axes = axes.flatten()
    for img, ax in zip( conv_filter, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

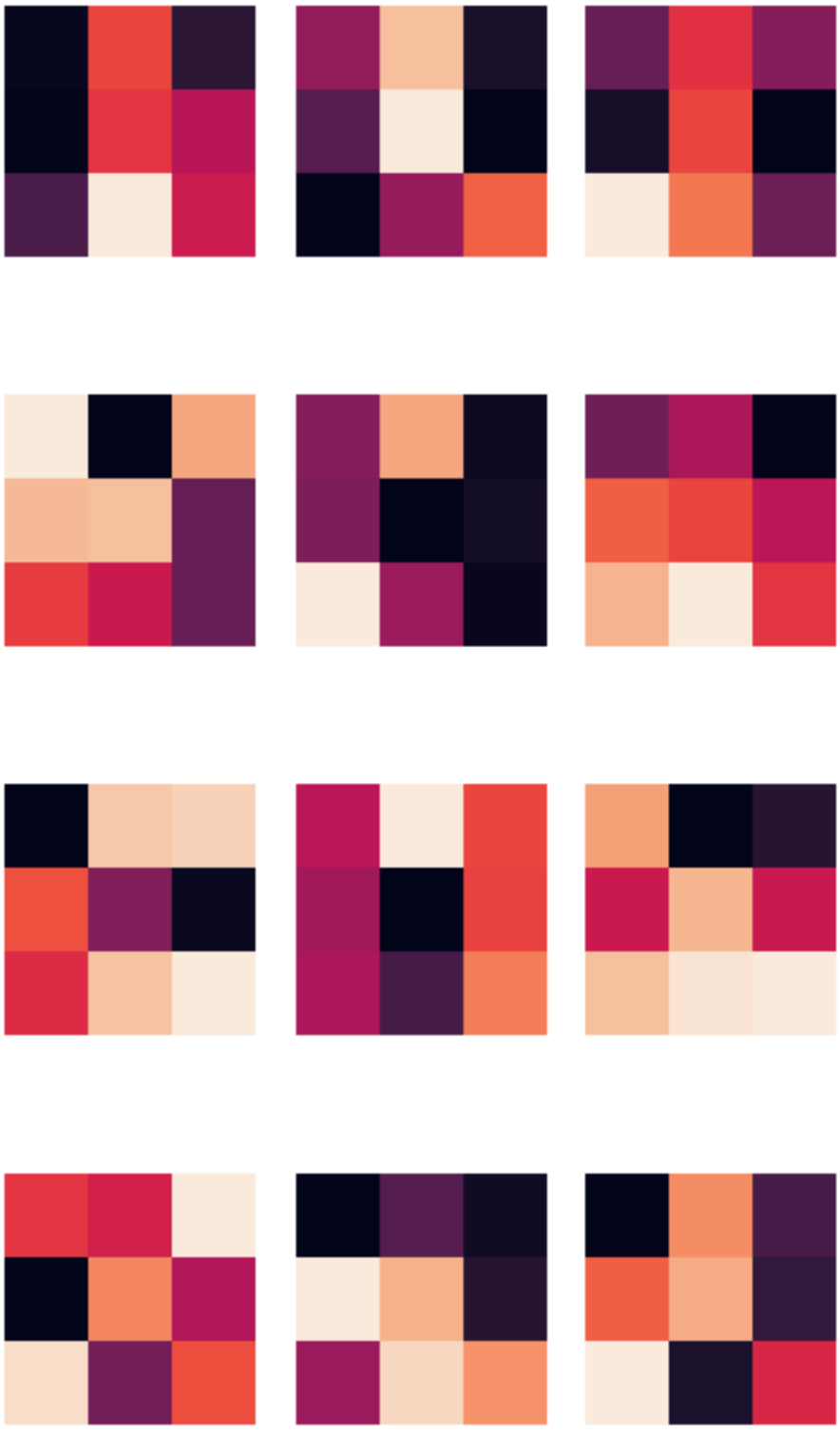
[45]:
```
#Iterate thru all the layers of the model
for layer in model.layers:
    if 'conv' in layer.name:
        weights, bias= layer.get_weights()
        print(layer.name, weights.shape)

        #normalize filter values between  0 and 1 for visualization
        f_min, f_max = weights.min(), weights.max()
        filters = (weights - f_min) / (f_max - f_min)
        print(filters.shape[3])
        filter_cnt=1

        #plotting all the filters
        for i in range(filters.shape[3]):
            #get the filters
            filt=filters[:,:,:, i]
            #plotting each of the channel, color image RGB channels
            plot_filters(filt)
```

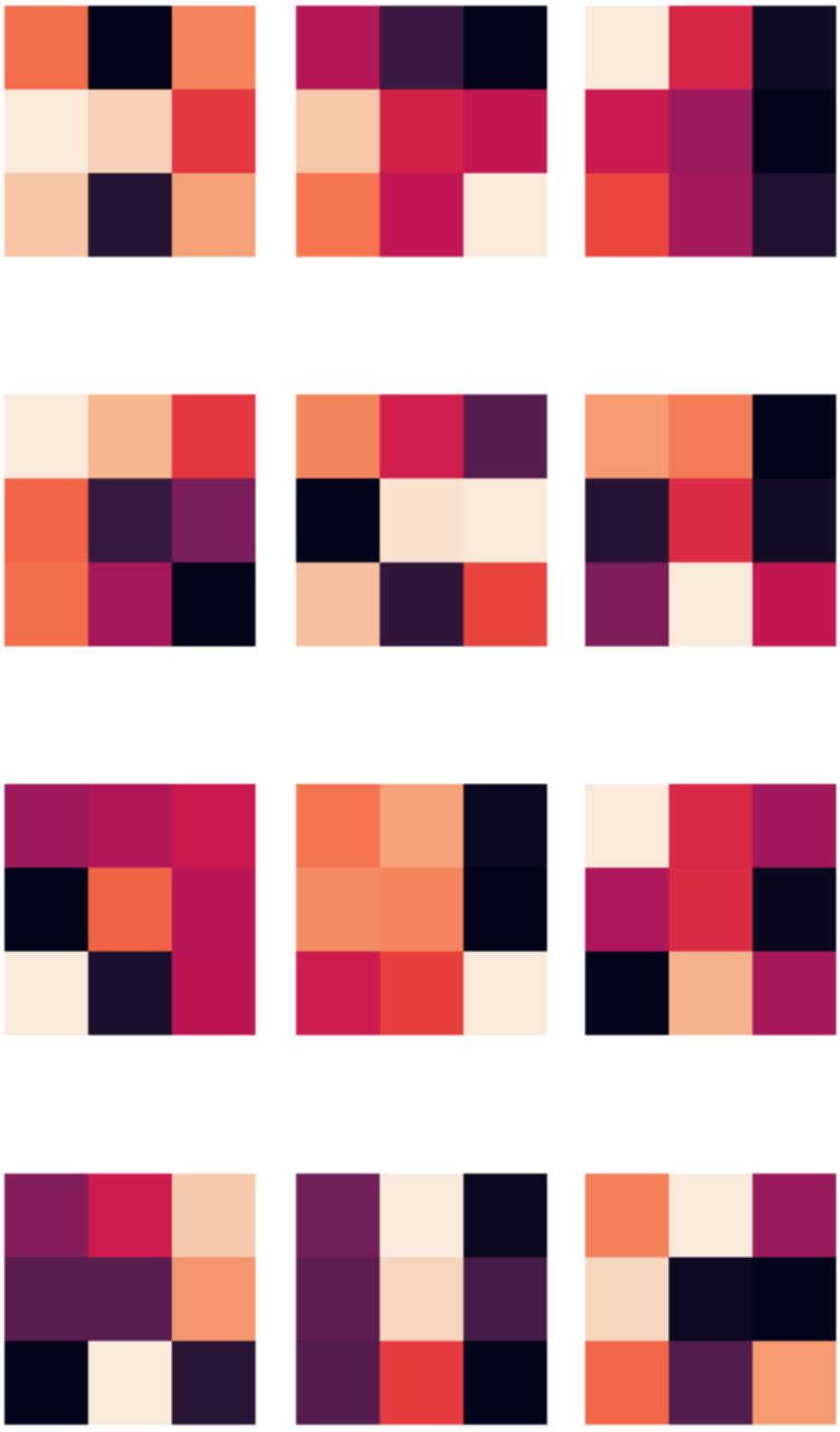
```
conv2d (3, 3, 3, 32)
32
```







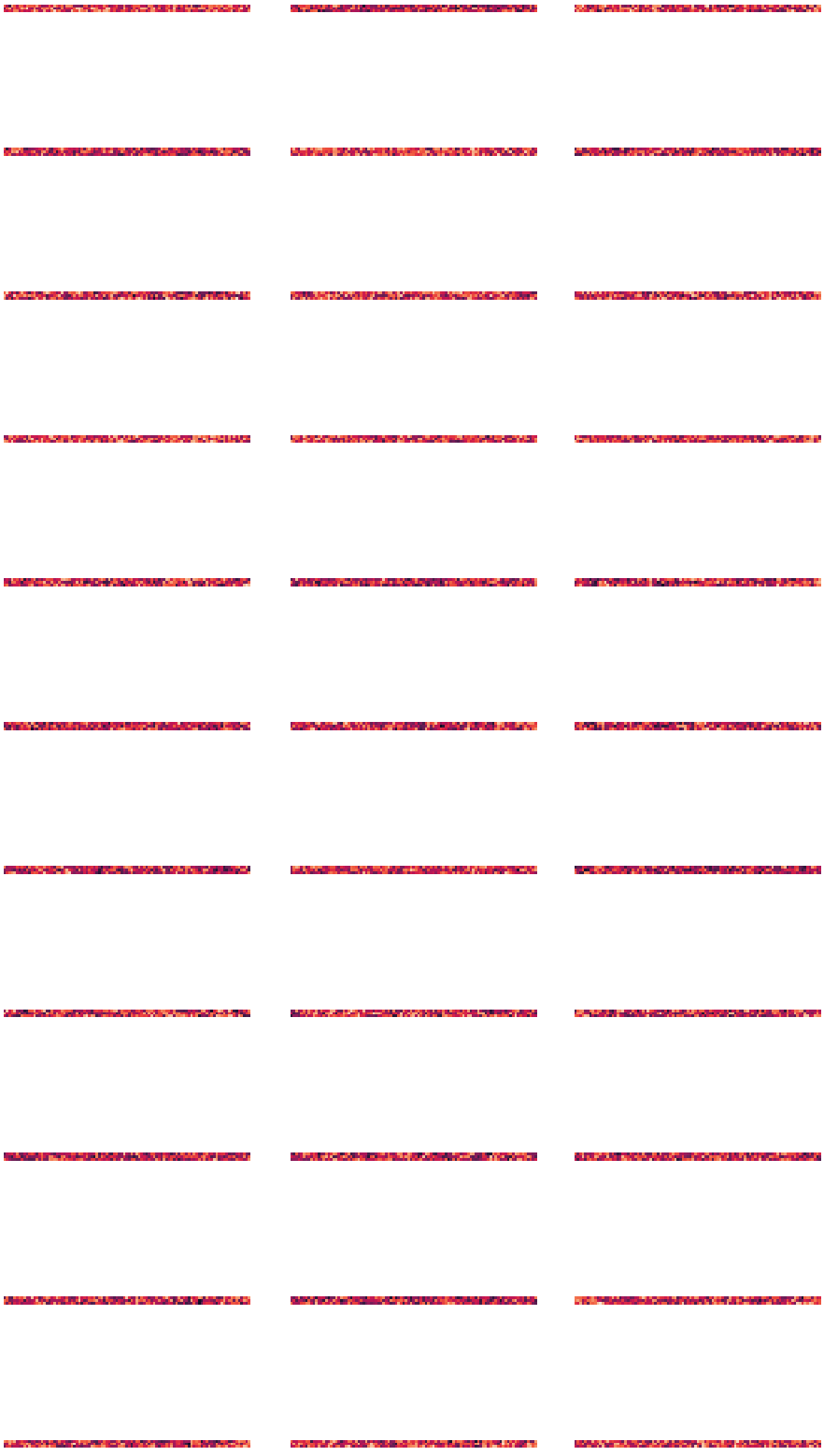


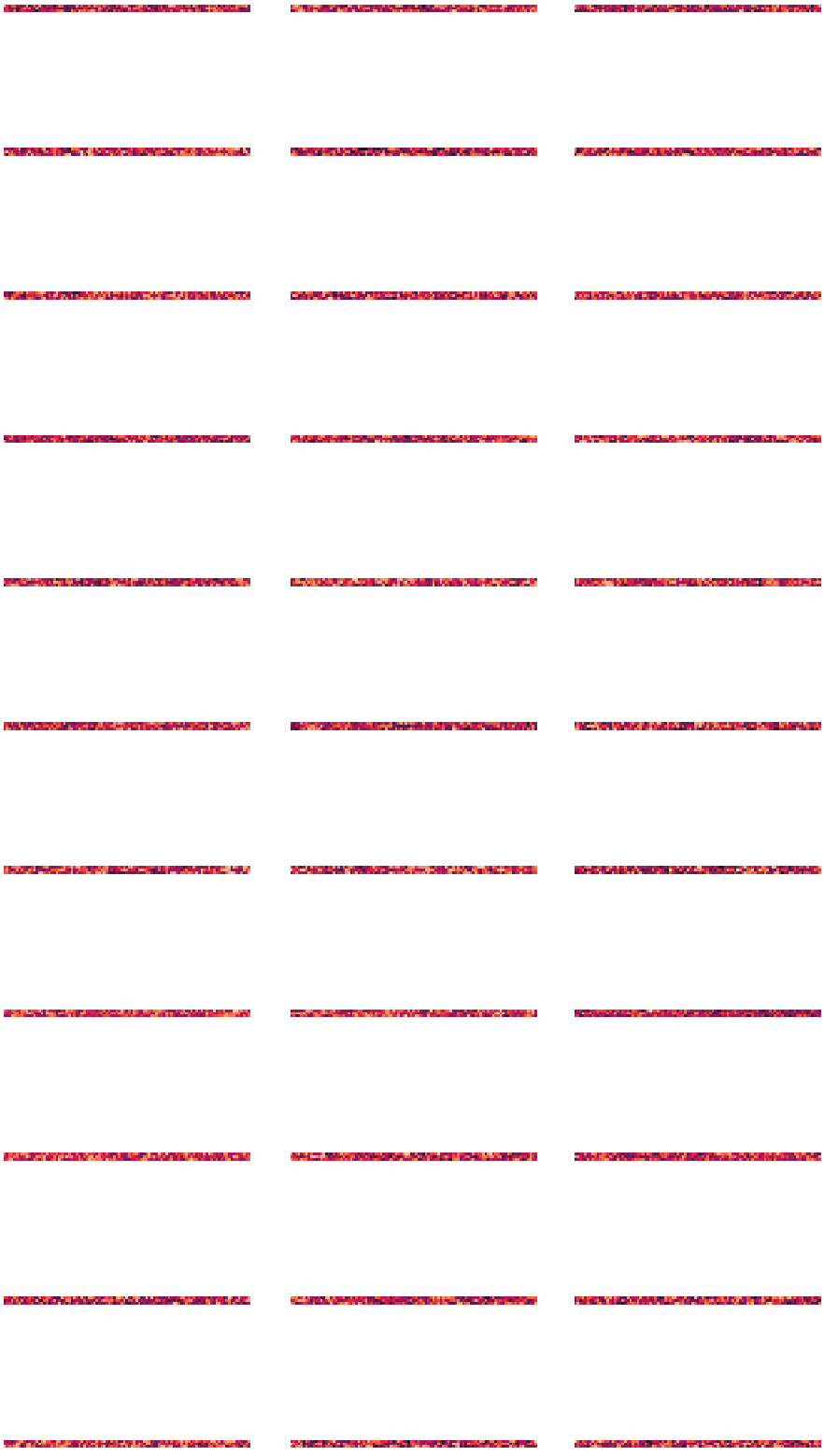


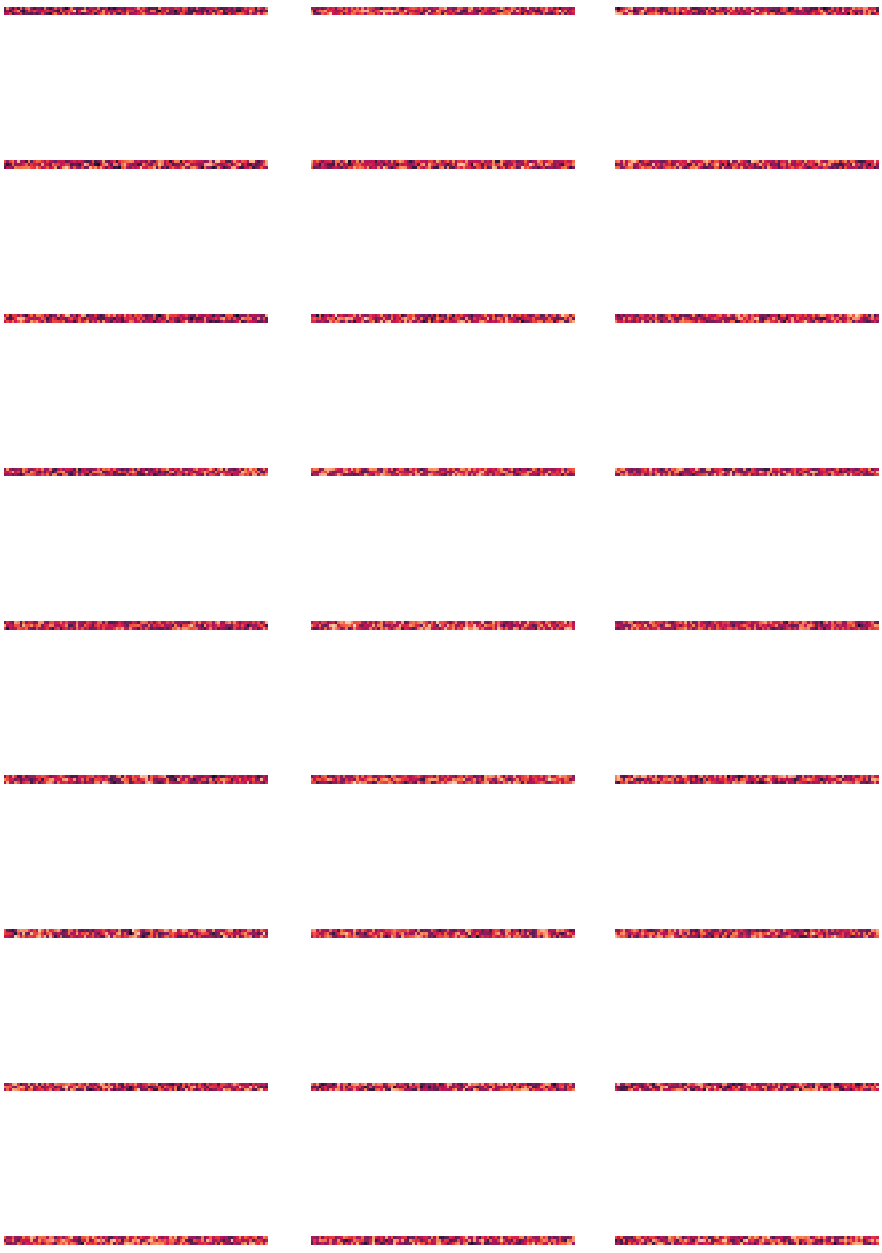
```
conv2d_1 (3, 3, 32, 64)
64
```

```
conv2d_2 (3, 3, 64, 128)
128
```

conv2d_3 (3, 3, 128, 128)
128







6.7 Visualizing Feature maps

Feature maps are generated by applying filters to the input image or the feature map output of the prior layers. Feature map visualization will provide insight into the internal representations for specific input for each of the Convolutional layers in the model.

[46]:
```
def visualize_feture_maps(model, img_path='output/archive/8863/1/
↪8863_idx5_x1001_y801_class1.png'):
    # Define a new Model, Input= image
    # Output= intermediate representations for all layers in the
    # previous model after the first.
    successive_outputs = [layer.output for layer in model.layers[1:]]
    # visualization_model = Model(img_input, successive_outputs)
    visualization_model = keras.models.Model(inputs = model.input, outputs =␣
↪successive_outputs)
    # Read the input image
    x = cv.imread(img_path, cv.IMREAD_COLOR)
    x = cv.resize(x, (50, 50), interpolation = cv.INTER_LINEAR)
    x = x/255
    x = np.expand_dims(x, axis=0)
    # Let's run our image through our network, thus obtaining all
    # intermediate representations for this image.
    successive_feature_maps = visualization_model.predict(x)
    # These are the names of the layers, so can have them as part of our plot
    layer_names = [layer.name for layer in model.layers]
    # -----------------------------------------------------------------------
    # Now let's display our representations
    # -----------------------------------------------------------------------
    for layer_name, feature_map in zip(layer_names, successive_feature_maps):
      print(feature_map.shape)
      if len(feature_map.shape) == 4:

        #----------------------------------------
        # Just do this for the conv / maxpool layers, not the fully-connected␣
↪layers
        #----------------------------------------
        n_features = feature_map.shape[-1]  # number of features in the feature␣
↪map
        size       = feature_map.shape[ 1]  # feature map shape (1, size, size,␣
↪n_features)

        # We will tile our images in this matrix
        display_grid = np.zeros((size, size * n_features))

        #-------------------------------------------------
        # Postprocess the feature to be visually palatable
        #-------------------------------------------------
        for i in range(n_features):
          x  = feature_map[0, :, :, i]
          x -= x.mean()
          x /= x.std ()
          x *=  64
          x += 128
```

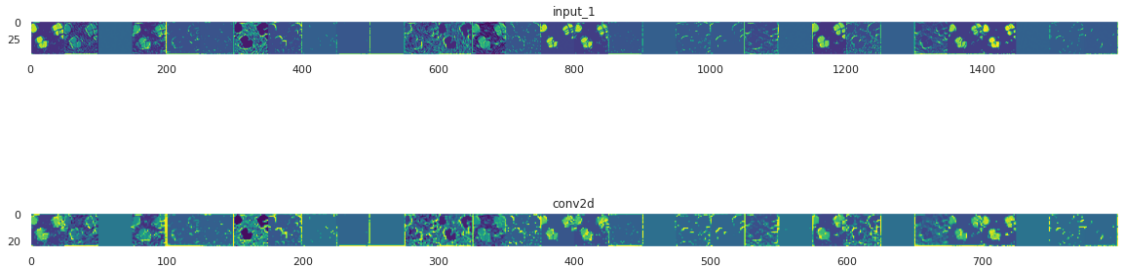
```
            x  = np.clip(x, 0, 255).astype('uint8')
            display_grid[:, i * size : (i + 1) * size] = x # Tile each filter␣
↪into a horizontal grid

        #-----------------
        # Display the grid
        #-----------------

        scale = 20. / n_features
        plt.figure( figsize=(scale * n_features, scale) )
        plt.title ( layer_name )
        plt.grid  ( False )
        plt.imshow( display_grid, aspect='auto', cmap='viridis' )
```

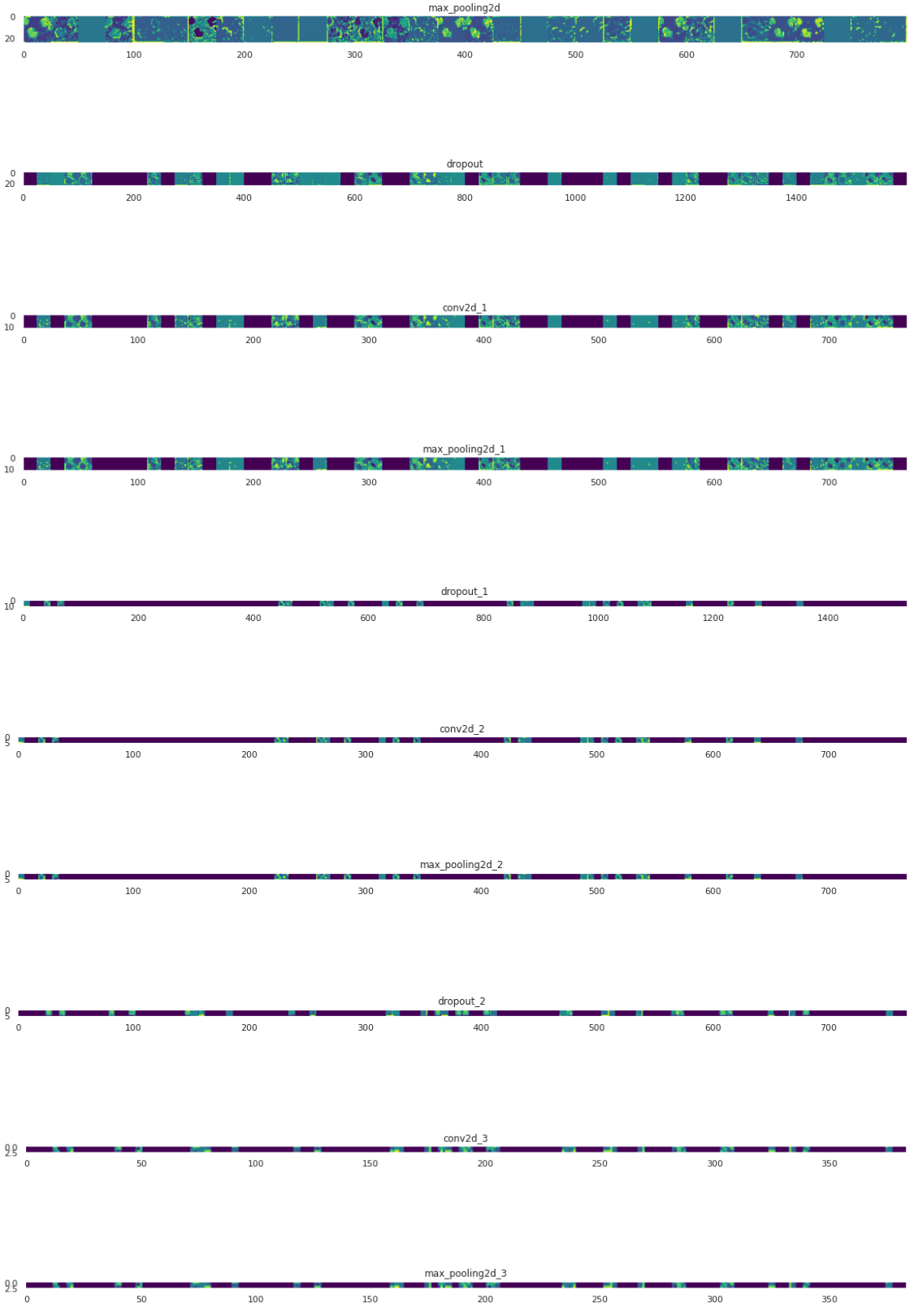
[47]:
```
visualize_feture_maps(model, img_path='output/archive/8863/1/
↪8863_idx5_x1001_y801_class1.png') # IDC Image
```

```
(1, 50, 50, 32)
(1, 25, 25, 32)
(1, 25, 25, 32)
(1, 25, 25, 64)
(1, 12, 12, 64)
(1, 12, 12, 64)
(1, 12, 12, 128)
(1, 6, 6, 128)
(1, 6, 6, 128)
```

```
<ipython-input-46-6add9b0551df>:40: RuntimeWarning: invalid value encountered in
true_divide
  x /= x.std ()
```

```
(1, 6, 6, 128)
(1, 3, 3, 128)
(1, 3, 3, 128)
(1, 1152)
(1, 128)
(1, 128)
(1, 2)
```



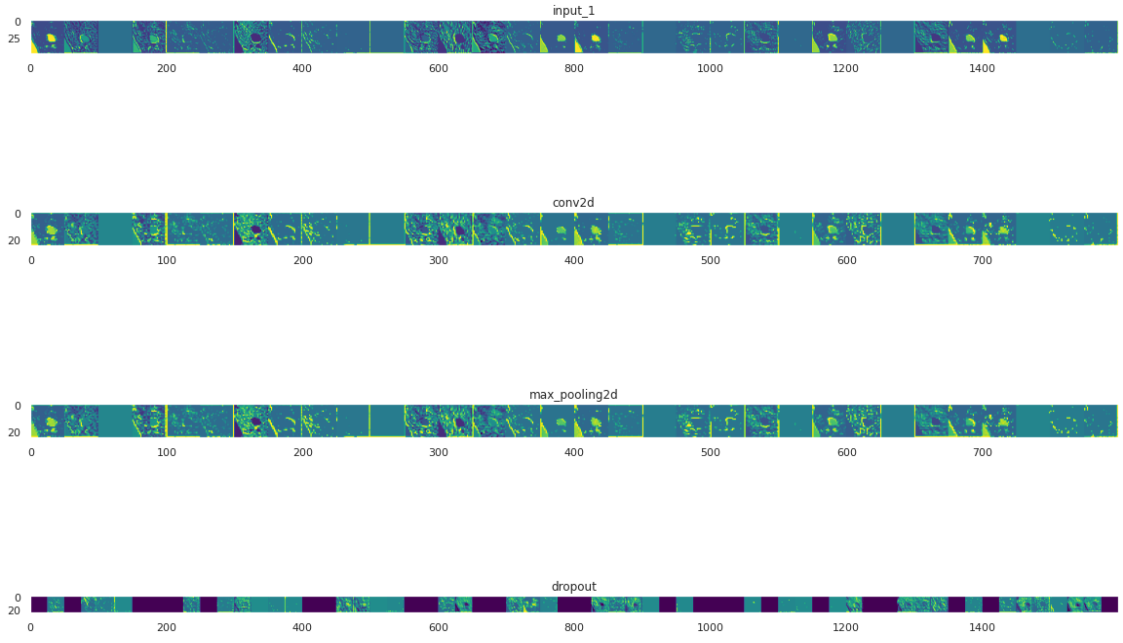

max_pooling2d
dropout
conv2d_1
max_pooling2d_1
dropout_1
conv2d_2
max_pooling2d_2
dropout_2
conv2d_3
max_pooling2d_3

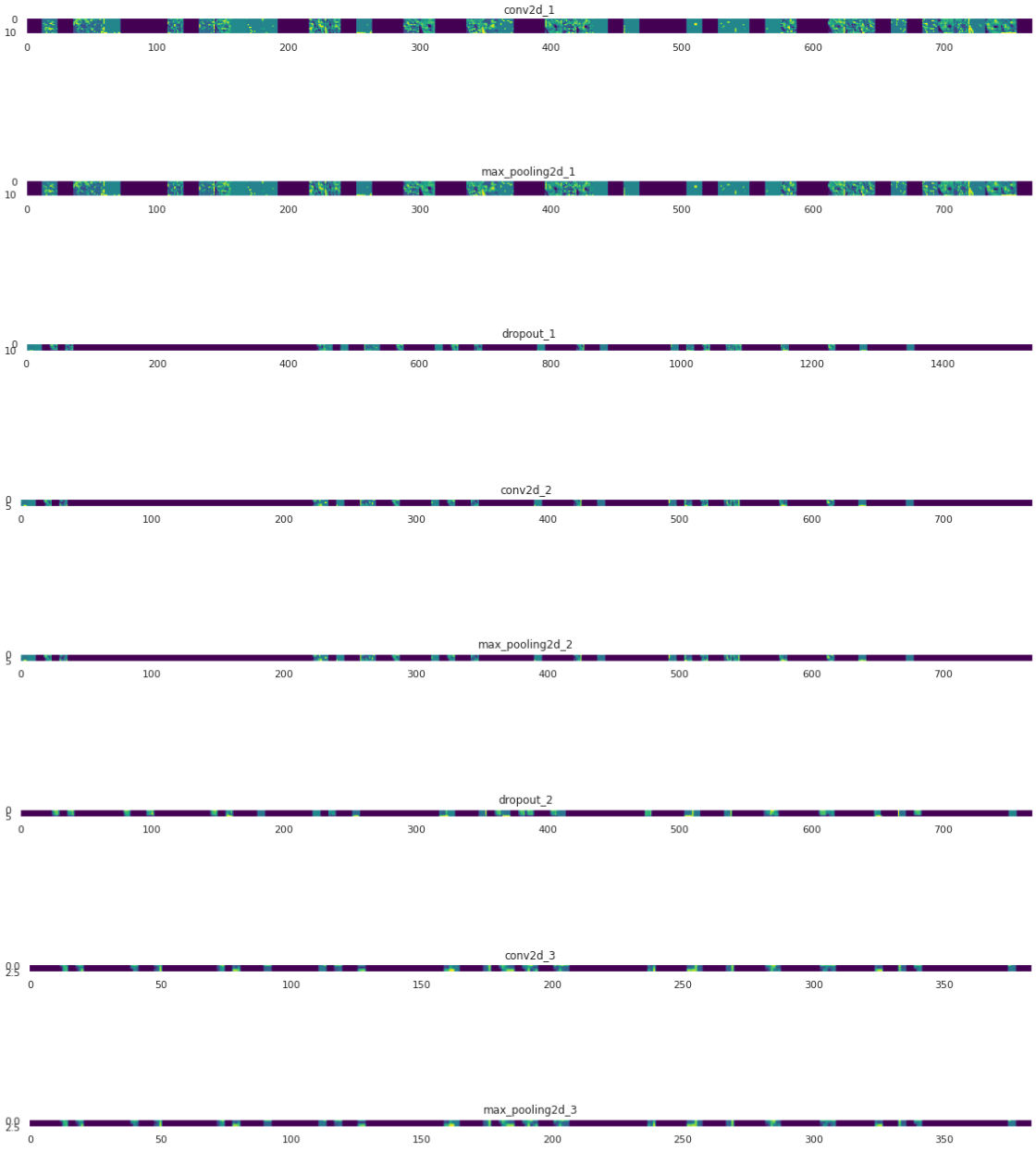
[48]:
```
visualize_feture_maps(model, img_path='output/archive/8863/0/
↪8863_idx5_x101_y1301_class0.png') # NOn-IDC image
```

```
(1, 50, 50, 32)
(1, 25, 25, 32)
(1, 25, 25, 32)
(1, 25, 25, 64)
(1, 12, 12, 64)
(1, 12, 12, 64)
(1, 12, 12, 128)
(1, 6, 6, 128)
(1, 6, 6, 128)
```

```
<ipython-input-46-6add9b0551df>:40: RuntimeWarning: invalid value encountered in
true_divide
  x /= x.std ()
```

```
(1, 6, 6, 128)
(1, 3, 3, 128)
(1, 3, 3, 128)
(1, 1152)
(1, 128)
(1, 128)
(1, 2)
```





6.8 References

Cancer on the Global Stage: Incidence and Cancer-Related Mortality in Kenya - The ASCO Post. (2021). Retrieved 10 August 2021, from https://ascopost.com/issues/february-25-2021/cancer-on-the-global-stage-incidence-and-cancer-related-mortality-in-kenya/

Feng-ying Cui, Li-jun Zou and Bei Song, "Edge feature extraction based on digital image processing techniques," 2008 IEEE International Conference on Automation and Logistics, 2008, pp. 2320-2324, doi: 10.1109/ICAL.2008.4636554

Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.

G. Kumar and P. K. Bhatia, "A Detailed Review of Feature Extraction in Image Processing Systems," 2014 Fourth International Conference on Advanced Computing & Communication Technologies, 2014, pp. 5-12, doi: 10.1109/ACCT.2014.74.

Khoon, L. L., & Chuin, L. S. (2016). A survey of medical image processing tools. International Journal of Software Engineering and Computer Systems (IJSECS), 2(1), 10-27.

Kunaver, Matevž & Tasic, Jurij. (2005). Image feature extraction - an overview. Journal of Crystal Growth - J CRYST GROWTH. 183 - 186. 10.1109/EURCON.2005.1629889.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., … & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. Medical image analysis, 42, 60-88.

Tiwari, H. (2017). A REVIEW PAPER ON MEDICAL IMAGE PROCESSING. International Journal of Research-GRANTHAALAYAH, 5(4RACSIT), 21-29.

[ ]: