

data_mining_project

August 11, 2021

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

2 Application

Medical image analysis

Artificial intelligence

Image rotation and enhancement

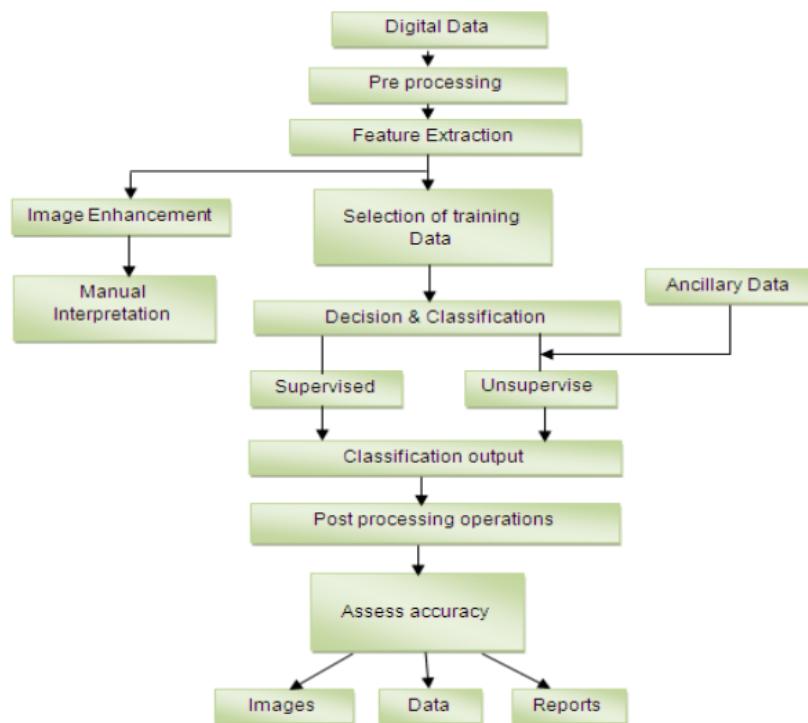
Geospatial Application

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()
```



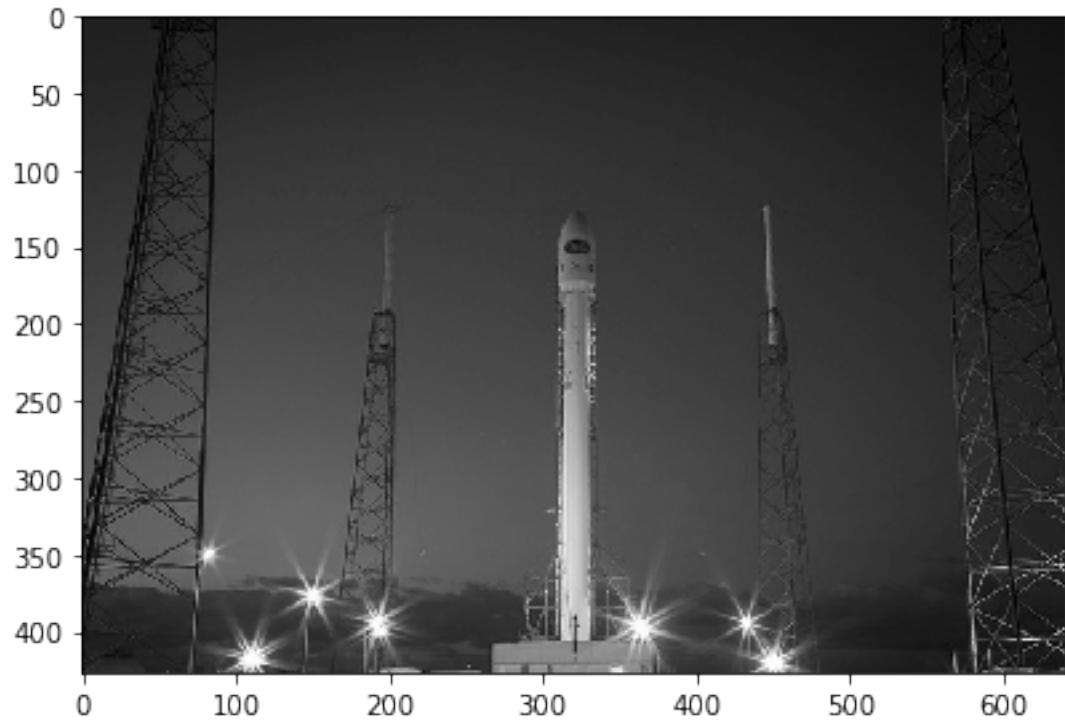
```
[3]: rocket.shape
```

```
[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)
```



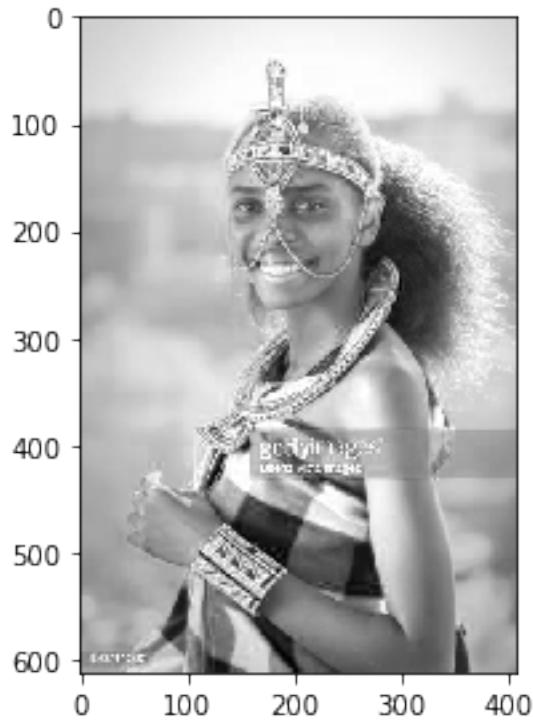
```
[5]: # Loading the image using Matplotlib
beaty_image = plt.imread('output/png/gettyimages-870711360-612x612.jpg')
io.imshow(beaty_image)
io.show()
```



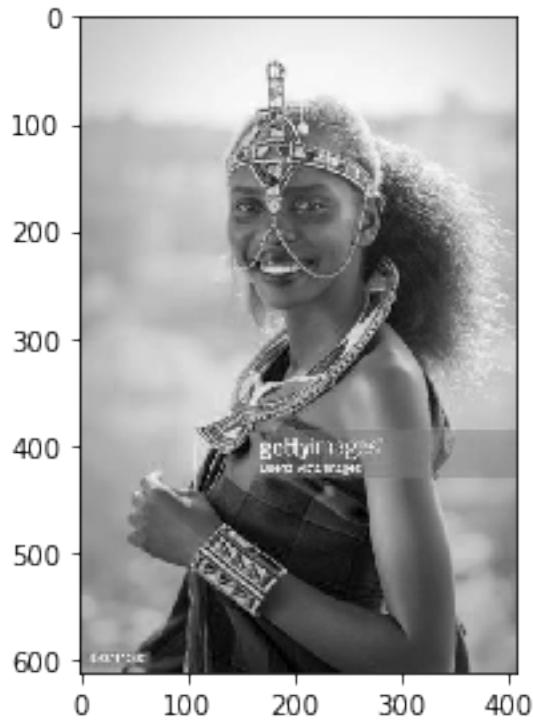
```
[16]: type(beaty_image)
```

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

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



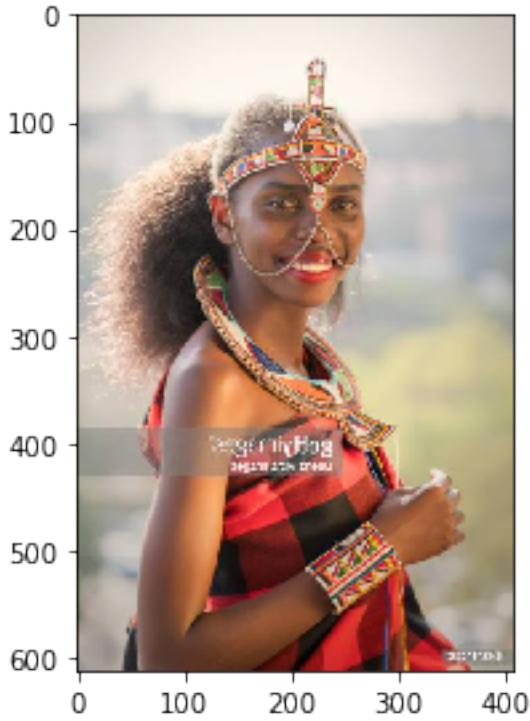
```
[18]: # Obtaining the green values of the  
imagegreen = beauty_image[:, :, 1]  
io.imshow(imagegreen)  
io.show()
```



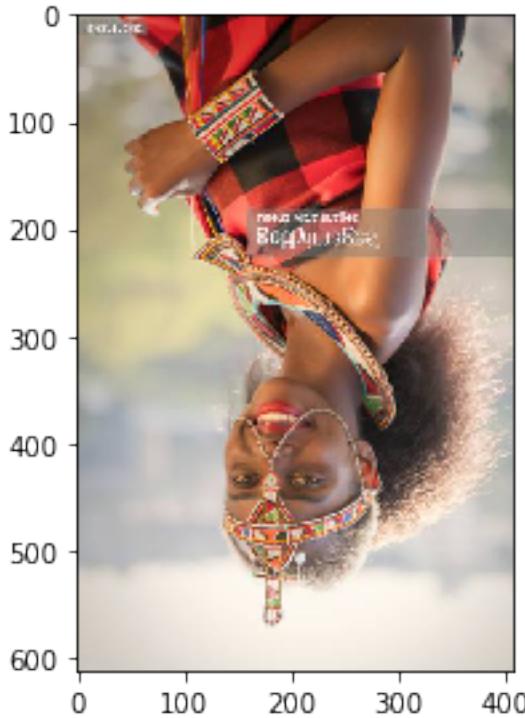
```
[19]: # Obtaining the blue values of the
imageblue = beauty_image[:, :, 2]
io.imshow(imageblue)
io.show()
```



```
[20]: # Flip the image in left
directionhorizontally_flipped = np.fliplr(beatty_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 seeing 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(beatty_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

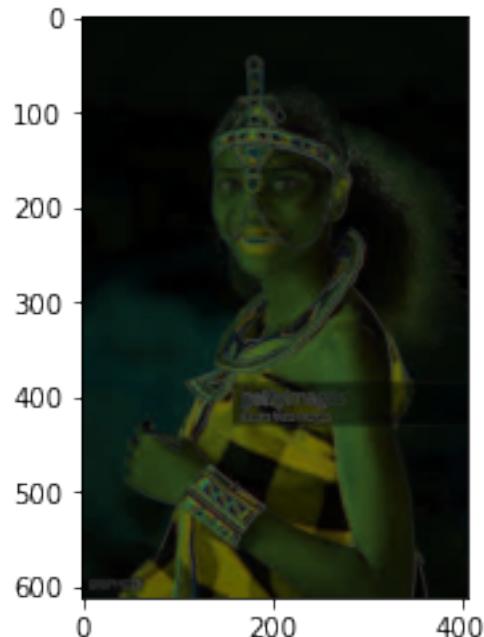
Edge detection

```
[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)
```

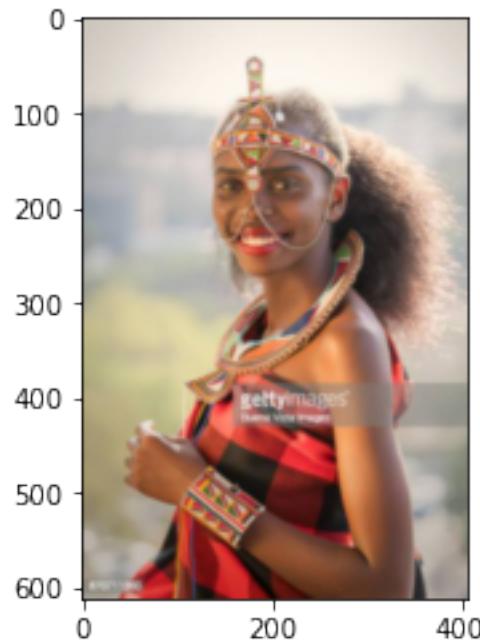
[23]: <matplotlib.image.AxesImage at 0x7f9120250f10>



```
[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

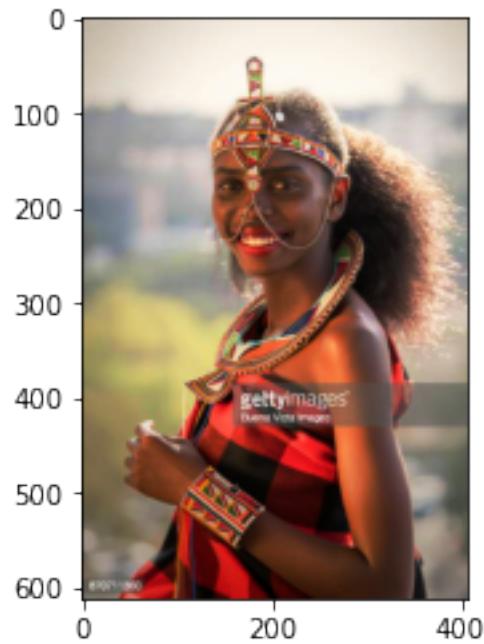
Contrast Limited Adaptive Histogram Equalization(CLAHE)

```
[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.

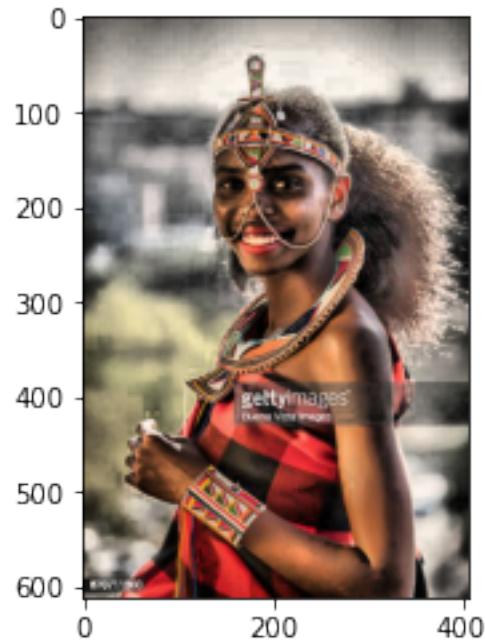
```
hist, bin_centers = histogram(image, nbins)
```

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



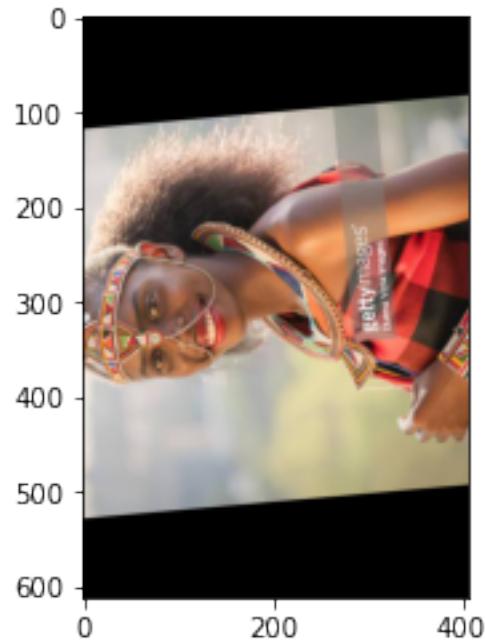
```
[26]: # Apply the adaptive equalization on the original image  
adaphist_eq_image = exposure.equalize_adapthist(beaty_image, clip_limit=0.03)  
# Compare the original image to the equalized  
plt.imshow(adaphist_eq_image)
```

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



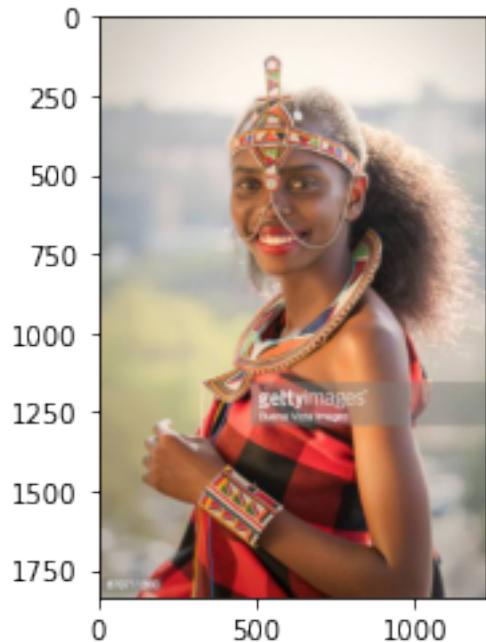
```
[27]: # Import the module and the rotate and rescale functions
from skimage.transform import rotate, rescale
# Rotate the image 90 degrees clockwise
rotated_cat_image = rotate(beaty_image, 95)
plt.imshow(rotated_cat_image)
```

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



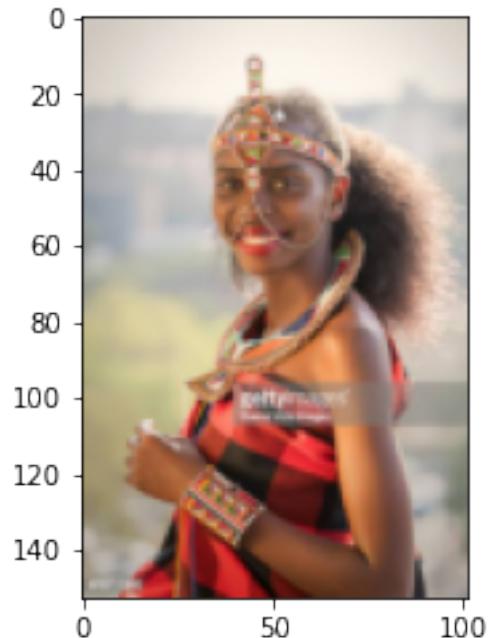
```
[28]: # Enlarge the image so it is 3 times bigger
enlarged_rocket_image = rescale(beaty_image, 3, anti_aliasing=True,
                                 multichannel=True)
# Show original and resulting image
plt.imshow(enlarged_rocket_image)
```

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



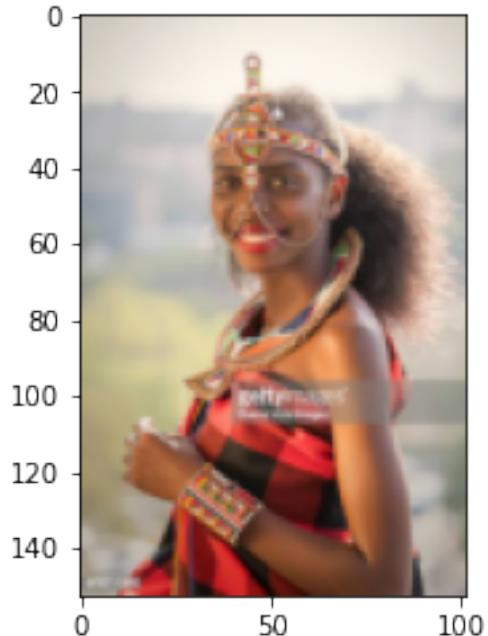
```
[29]: # shrink the image so it is 4 times smaller
enlarged_rocket_image = rescale(beaty_image, 1/4, anti_aliasing=True,
                                 multichannel=True)
# Show original and resulting image
plt.imshow(enlarged_rocket_image)
```

```
[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)
```

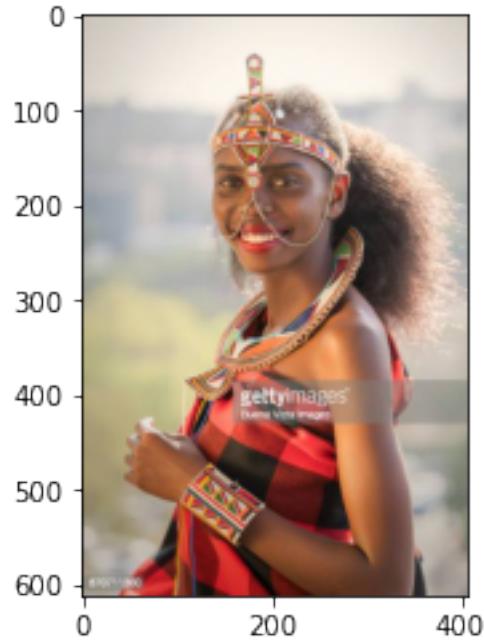
```
[30]: <matplotlib.image.AxesImage at 0x7f912073da60>
```



```
[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
# image
plt.imshow(maskrestored_image)
```

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



3.5 Image reconstruction

Fixing damaged images

Text removing Logo removing

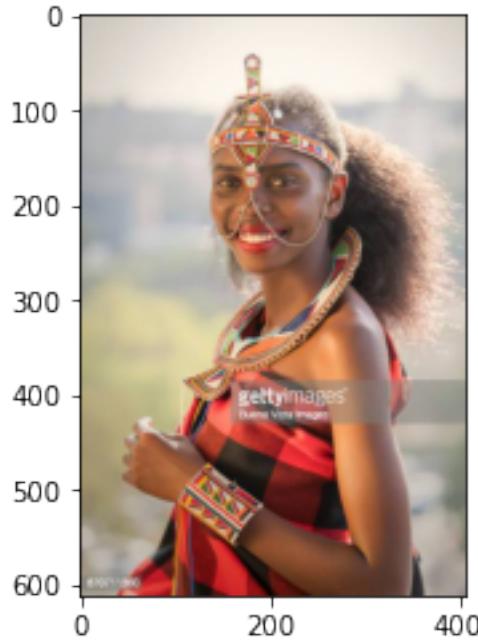
Object removing

```
[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)

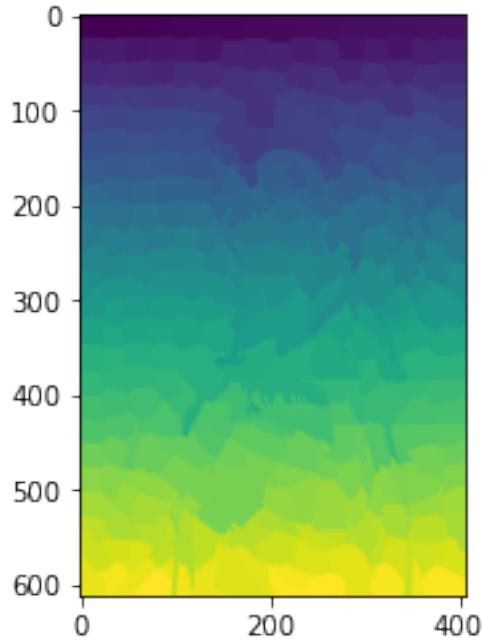
# Put segments on top of original image to compare
segmented_image = label2rgb(segments, beaty_image, kind='avg')

# 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, explicitly 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.

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

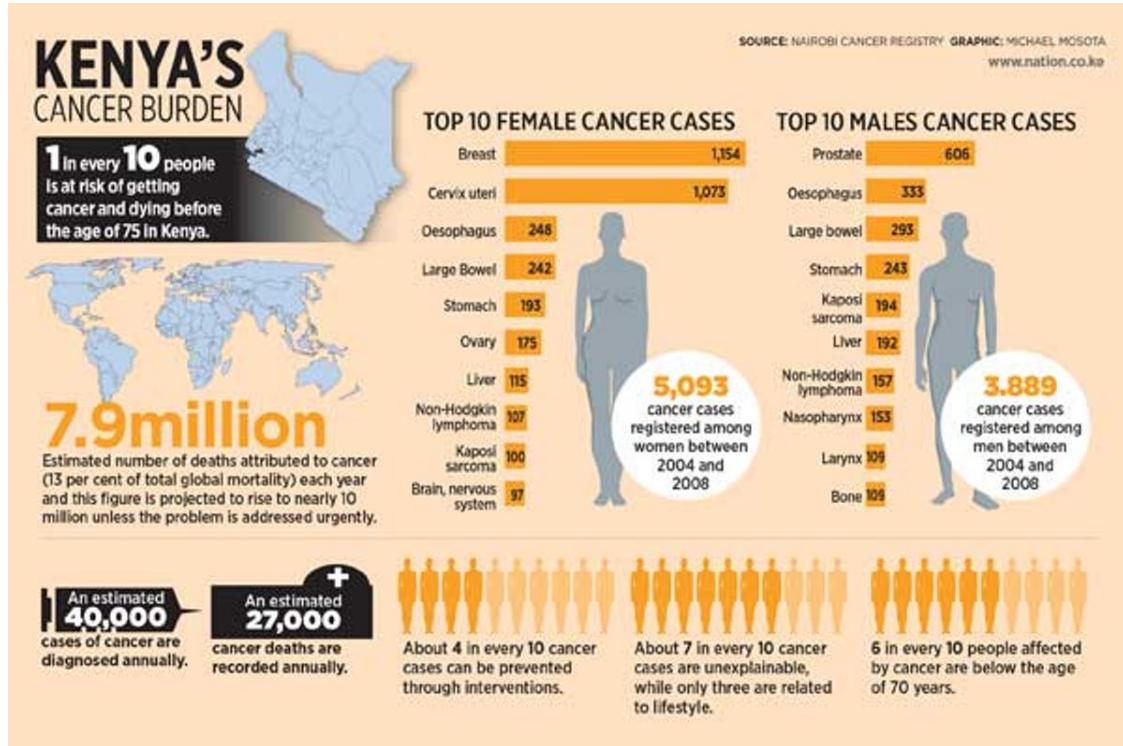
import random
from pprint import pprint
import time
import copy
from tqdm.notebook import tqdm
tqdm().pandas();5
```

```
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: uxYyYclassC.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	\
0	9125	901	1451	0	
1	9125	1551	2101	0	
2	9125	1651	301	0	
3	9125	601	1651	0	
4	9125	1701	1	0	
...		
277519	13404	2251	1901	1	
277520	13404	2201	2001	1	
277521	13404	2351	2051	1	
277522	13404	2101	2001	1	
277523	13404	1751	1801	1	

	path
0	output/archive/IDC_regular_ps50_idx5/9125/0/91...
1	output/archive/IDC_regular_ps50_idx5/9125/0/91...
2	output/archive/IDC_regular_ps50_idx5/9125/0/91...
3	output/archive/IDC_regular_ps50_idx5/9125/0/91...
4	output/archive/IDC_regular_ps50_idx5/9125/0/91...
...	...
277519	output/archive/IDC_regular_ps50_idx5/13404/1/1...
277520	output/archive/IDC_regular_ps50_idx5/13404/1/1...
277521	output/archive/IDC_regular_ps50_idx5/13404/1/1...
277522	output/archive/IDC_regular_ps50_idx5/13404/1/1...
277523	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?")

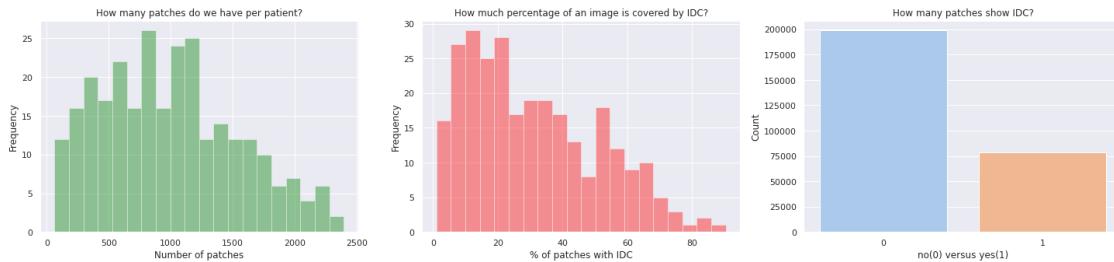
# Plotting Percentage of an image that is covered by Invasive Ductile Carcinoma
sns.distplot(cancer_perc.loc[:, 1]*100, ax=ax[1], color="red", kde=False, ↴
    bins=20)
ax[1].set_title("How much percentage of an image is covered by IDC?")
ax[1].set_ylabel("Frequency")
ax[1].set_xlabel("% of patches with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

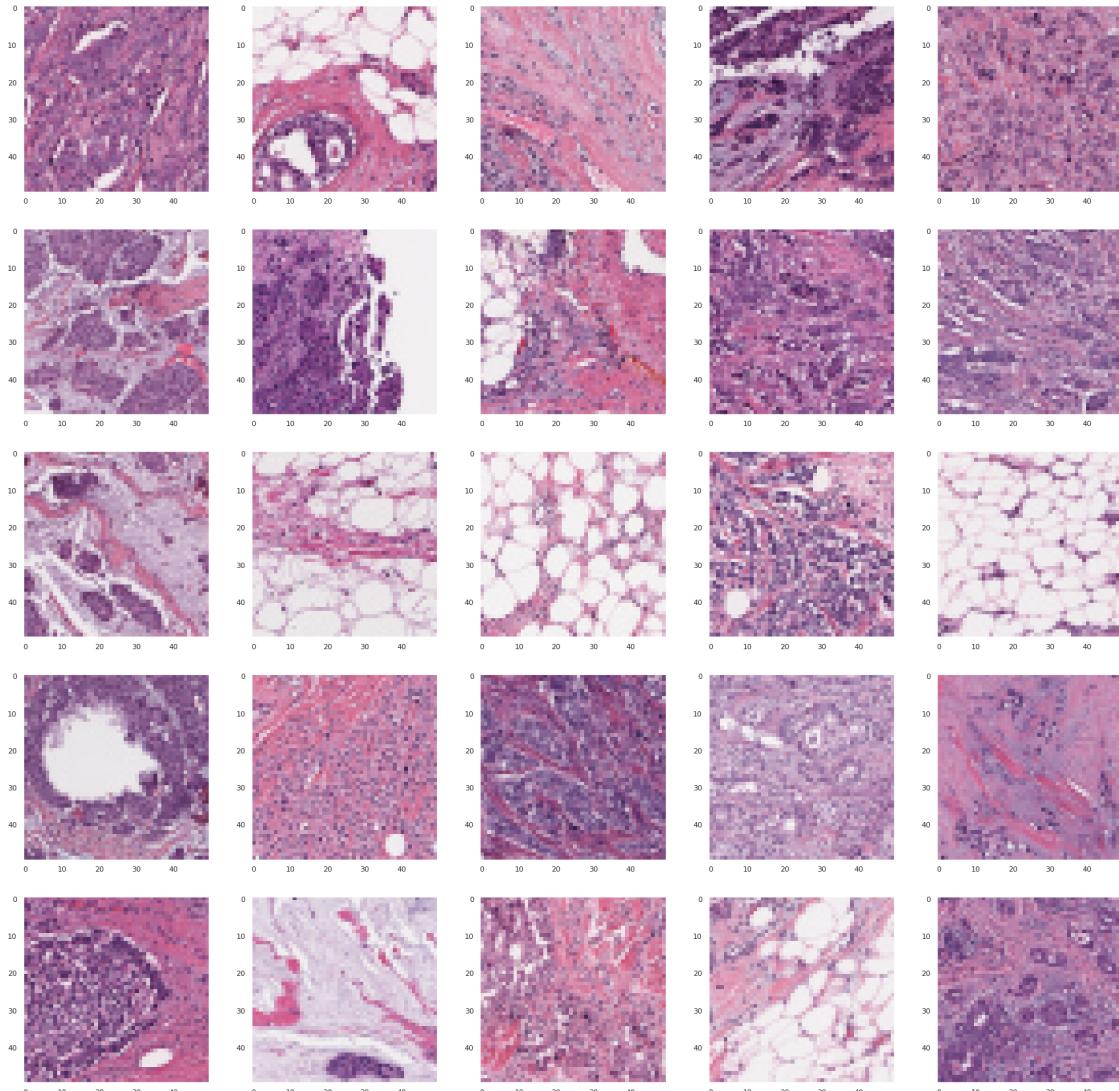
Let us now explore the visual differences between cancerous tissue cells, and healthy tissue cells. Usually partnering with a specialist is a good idea so that they can point the exact points of interest that differentiate the 2 from each other.

```
[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

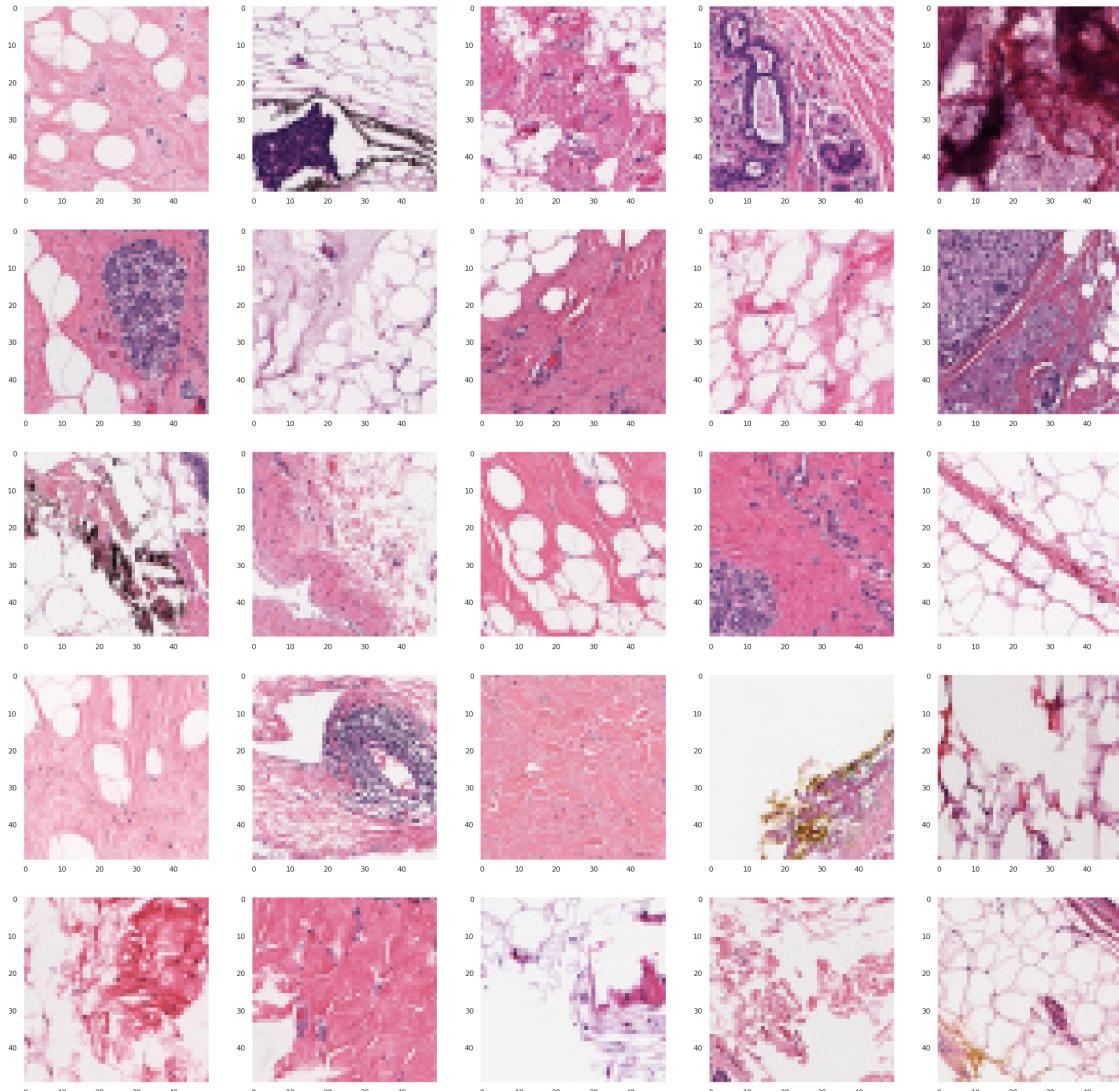
```
[16]: fig,ax = plt.subplots(n_rows,n_cols,figsize = (30,30))  
  
for row in range(n_rows):  
    for col in range(n_cols):  
        # below is a counter to cycle through the image indexes  
        idx = positive_tissue[col + n_cols*row]  
        img = io.imread(data.loc[idx, "path"])  
        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

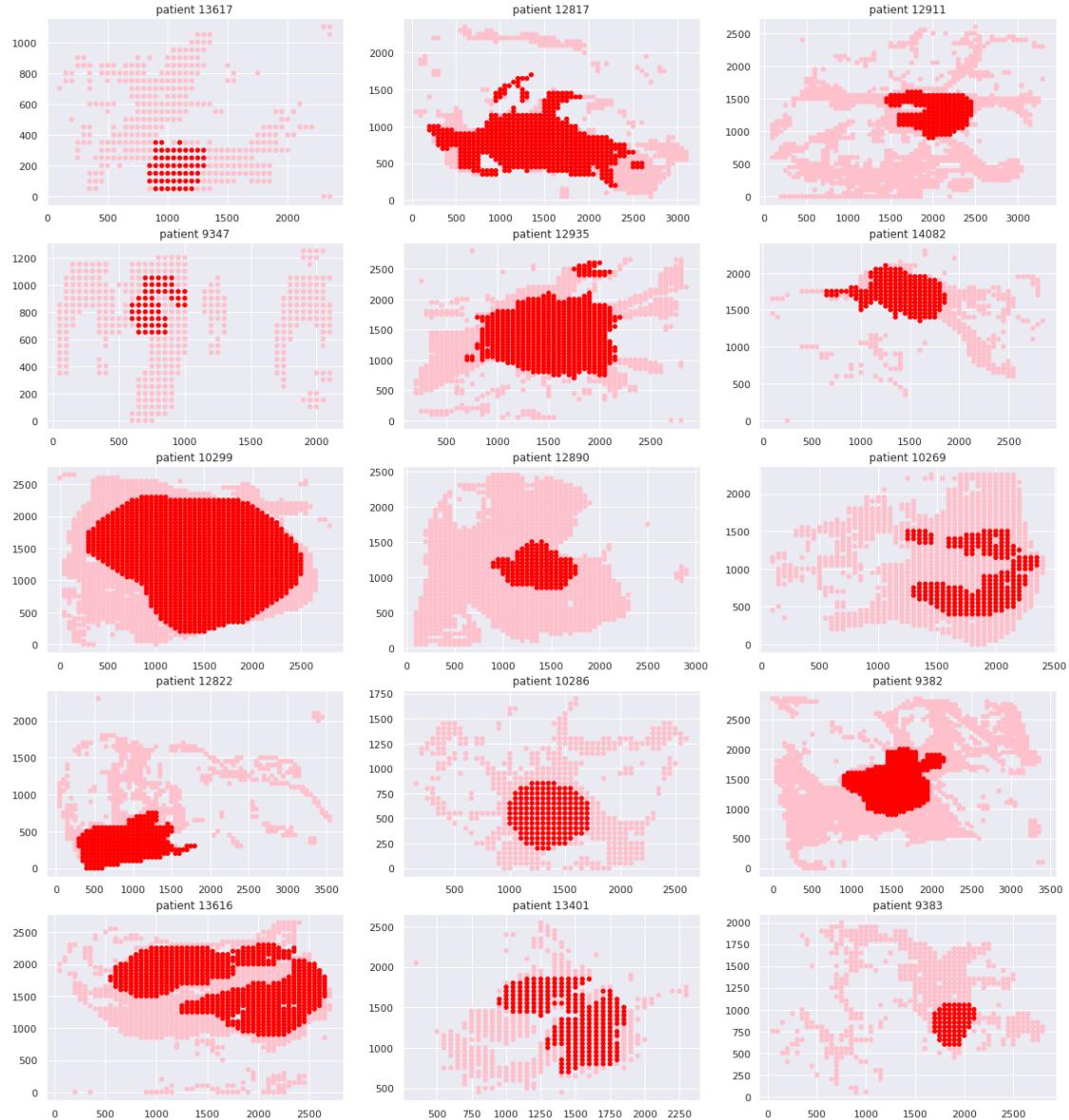
[20]: colors = ['pink', 'red']

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

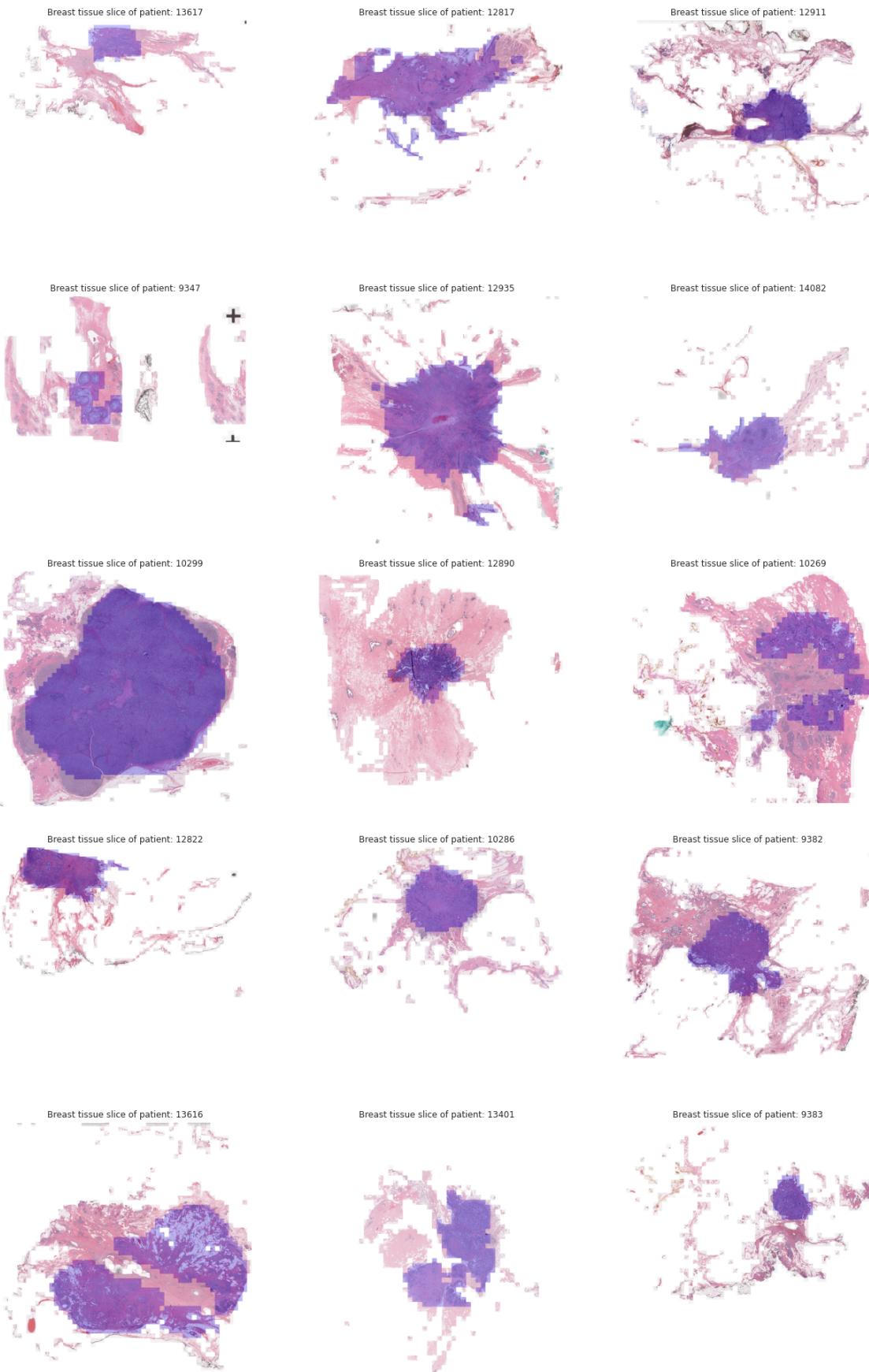
Now it's time to go one step deeper with our EDA. Instead of plotting the target values using the x-y coordinates, we now plot the images themselves on their respective x-y coordinates. This will help us visualize how the cancerous tissue looks like from a macro perspective.

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)
```



```
[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])
```

```
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 managed 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

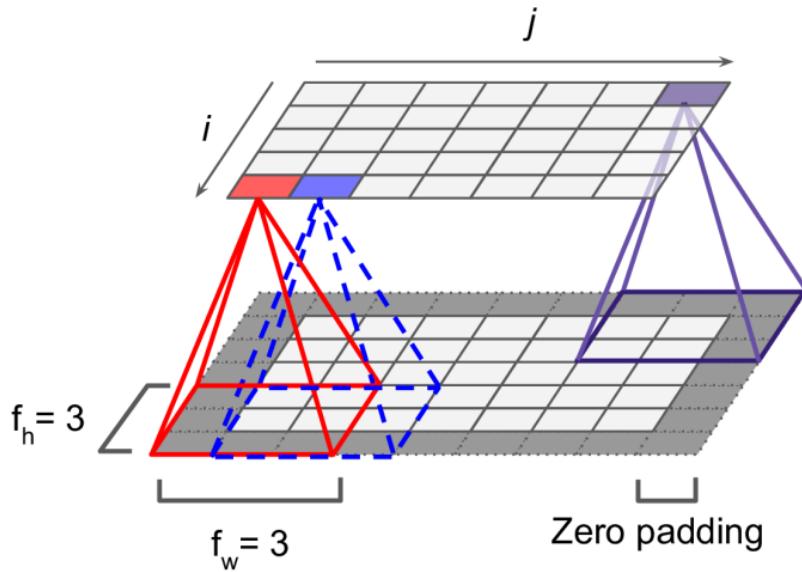
6.4.1 Convolutional Layer

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] :



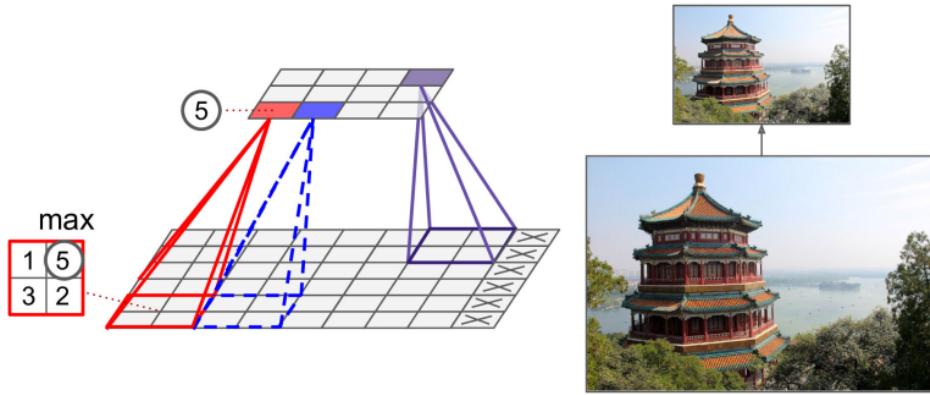
6.4.2 Pooling Layer

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 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
dropout_2 (Dropout)	(None, 6, 6, 128)	0
conv2d_3 (Conv2D)	(None, 6, 6, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(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)

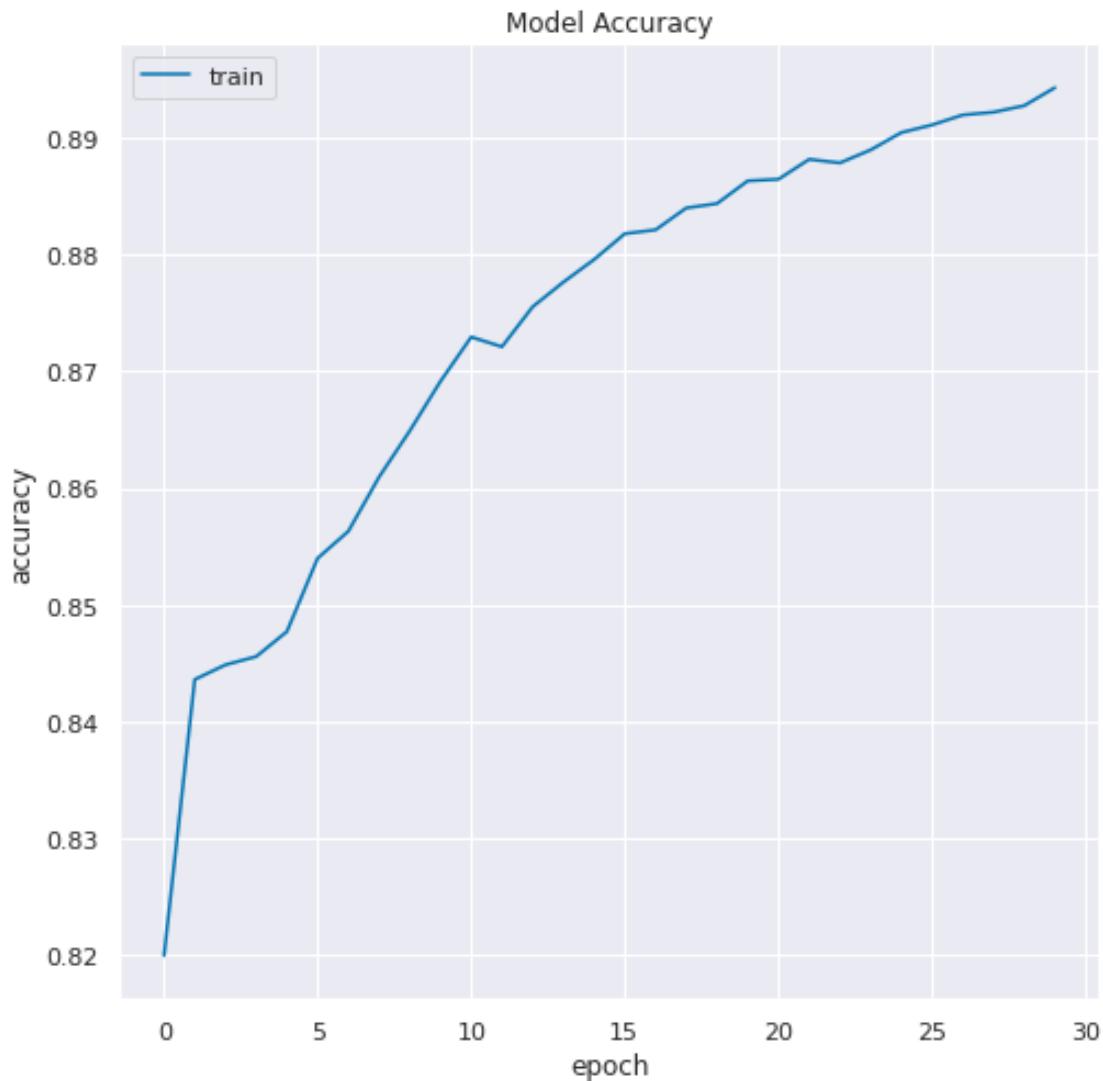
Epoch 1/30

```
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()
```

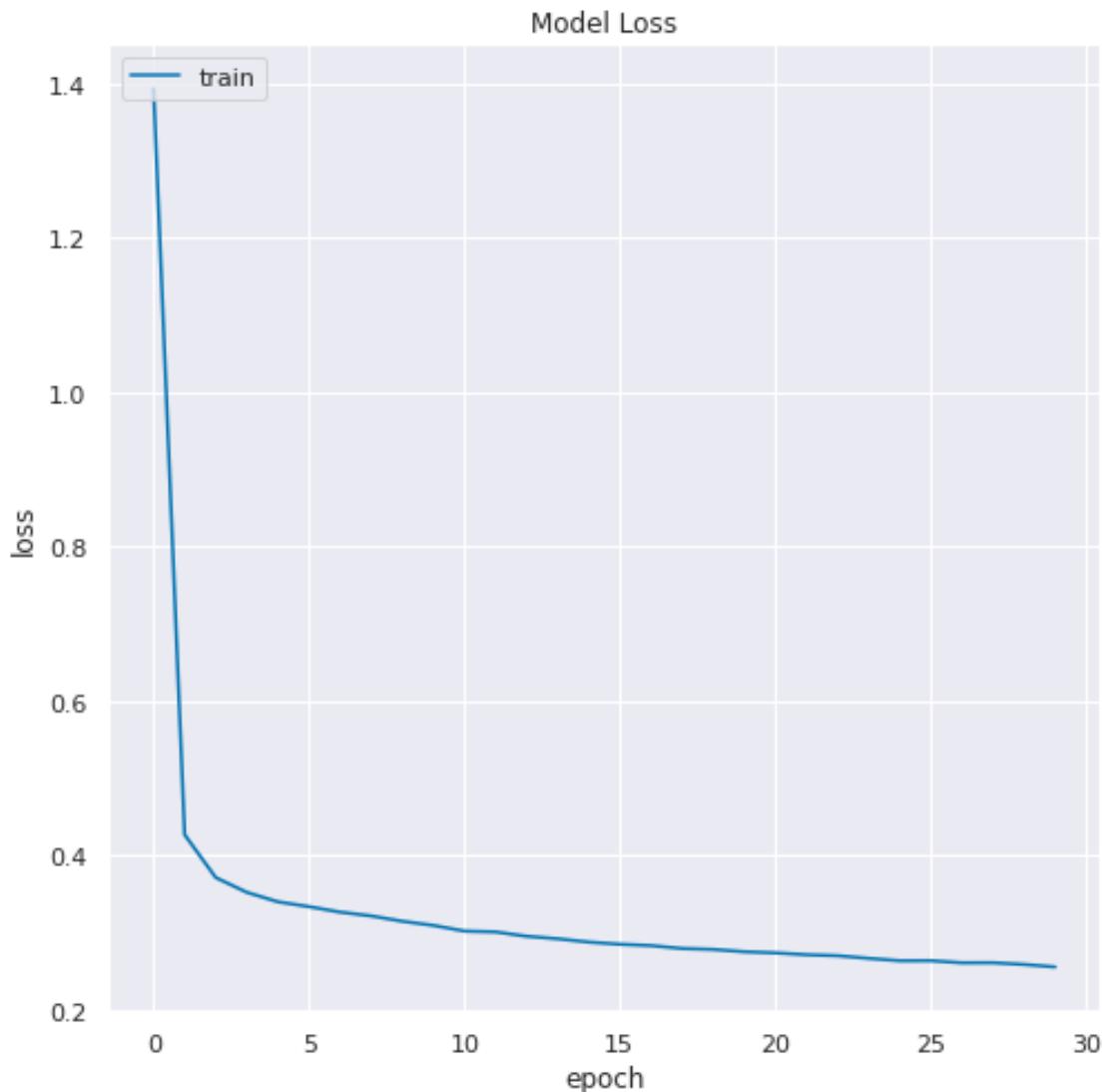


```
[39]: result = model.evaluate(X_test, y_test, batch_size = 256)
print('Test Loss, Test Accuracy :', result)
```

```
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

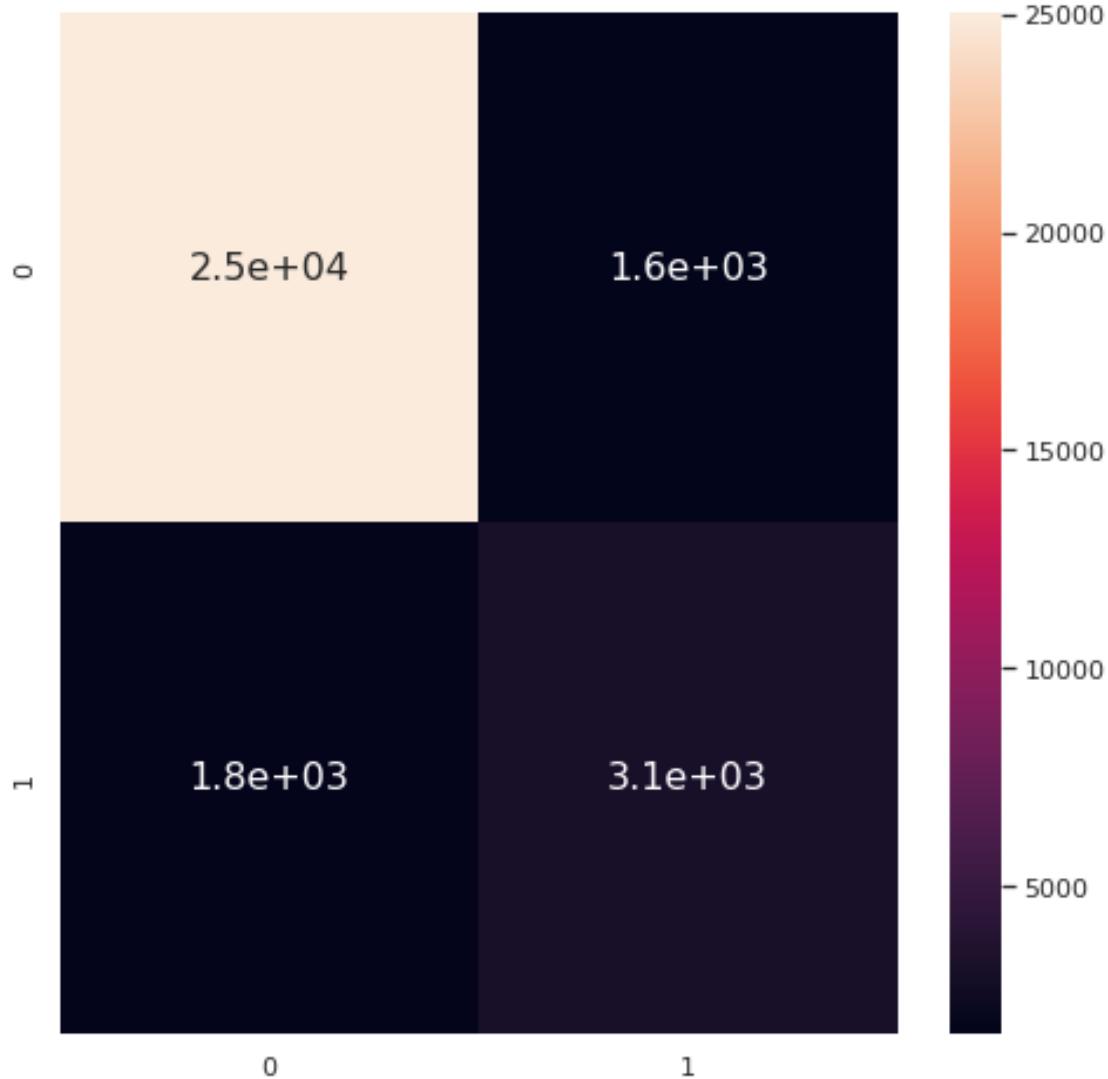
- Precision: It can be seen as a measure of a classifier's exactness. For each class, it is defined as the ratio of true positives to the sum of true and false positives. Said another way, "for all instances classified positive, what percent was correct?"
- Recall: It is a measure of the classifier's 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()
```

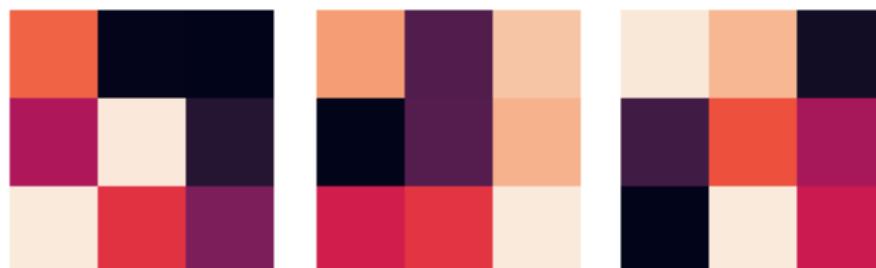
```
[45]: #Iterate thru all the layers of the model
for layer in model.layers:
    if 'conv' in layer.name:
        weights, bias= layer.get_weights()
        print(layer.name, weights.shape)

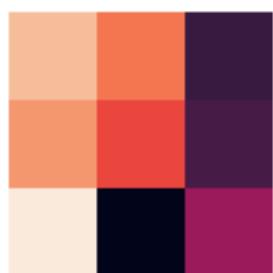
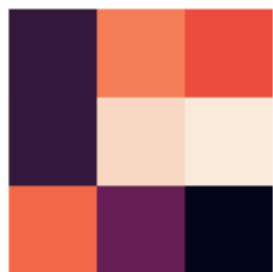
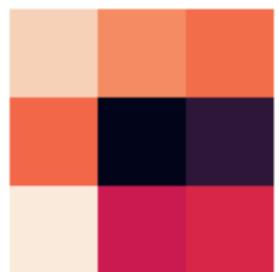
    #normalize filter values between 0 and 1 for visualization
    f_min, f_max = weights.min(), weights.max()
    filters = (weights - f_min) / (f_max - f_min)
    print(filters.shape[3])
    filter_cnt=1

    #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)
```

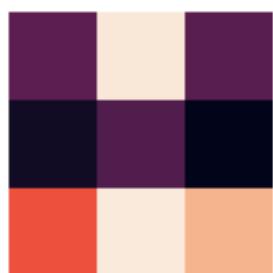
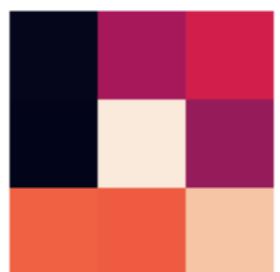
conv2d (3, 3, 3, 32)

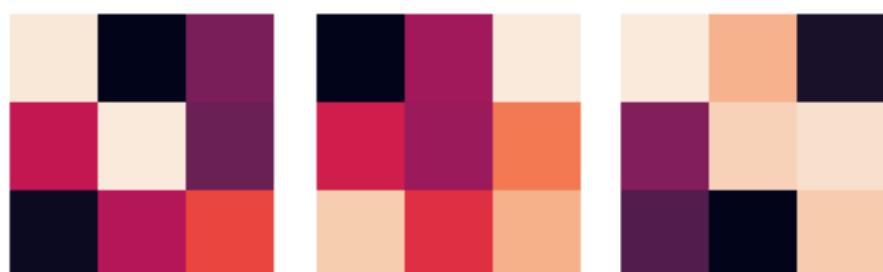
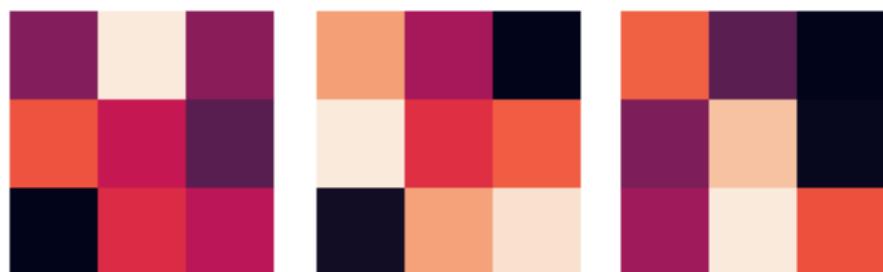
32



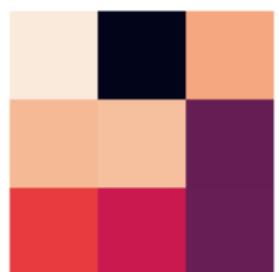


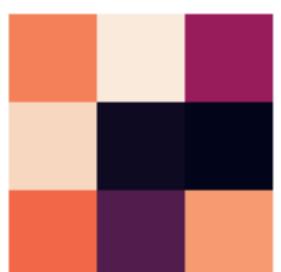
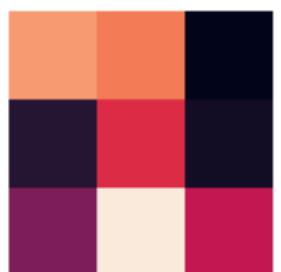
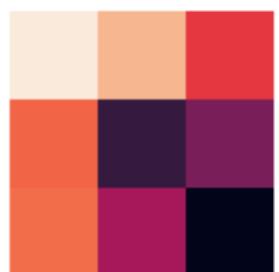
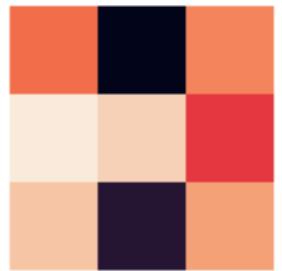














conv2d_1 (3, 3, 32, 64)
64



[WEDDING](#) [APPAREL](#) [HOMEWARE](#)

[View all posts](#)

CHAPTER 08 ■ FILE INPUT AND OUTPUT

[View Details](#) [View Details](#) [View Details](#)

DATA SCIENCE **DATA ENGINEERING** **DATA INTEGRATION**

[View Details](#) [View Details](#) [View Details](#)

ANSWER **ANSWER** **ANSWER**

Three horizontal heatmaps showing gene expression patterns across three samples. Each heatmap consists of a grid of colored squares representing different genes. The colors range from dark purple (low expression) to bright yellow (high expression). The first sample shows a distinct pattern of high expression in the first few genes. The second sample shows a more uniform distribution of high expression across the genes. The third sample shows a pattern where expression is highest in the middle genes and decreases towards the end.

1. **What is the name of the author?**
2. **What is the name of the book?**
3. **What is the name of the publisher?**
4. **What is the name of the editor?**
5. **What is the name of the illustrator?**
6. **What is the name of the designer?**
7. **What is the name of the printer?**
8. **What is the name of the distributor?**
9. **What is the name of the agent?**
10. **What is the name of the bookseller?**

1. **What is the name of the author?**
2. **What is the name of the book?**
3. **What is the name of the publisher?**
4. **What is the name of the editor?**
5. **What is the name of the translator?**
6. **What is the name of the illustrator?**
7. **What is the name of the designer?**
8. **What is the name of the printer?**
9. **What is the name of the distributor?**
10. **What is the name of the agent?**

[CONTINUAR](#) [REGISTRO](#) [ACORDAR PREÇO](#)

卷之三

[View all posts](#) [View all categories](#)

[View Details](#) [View Details](#) [View Details](#)

ANSWER: **1. 1000** 2. **10000** 3. **100000**

A horizontal strip of colorful, abstract patterns, possibly a decorative border or a sample of textile design.

[View Details](#) [Edit](#) [Delete](#)

ANSWER

[View Details](#) [View Details](#) [View Details](#)

1. **1** **2** **3**
2. **1** **2** **3**
3. **1** **2** **3**
4. **1** **2** **3**
5. **1** **2** **3**
6. **1** **2** **3**
7. **1** **2** **3**
8. **1** **2** **3**
9. **1** **2** **3**
10. **1** **2** **3**
11. **1** **2** **3**
12. **1** **2** **3**
13. **1** **2** **3**
14. **1** **2** **3**
15. **1** **2** **3**
16. **1** **2** **3**
17. **1** **2** **3**
18. **1** **2** **3**
19. **1** **2** **3**
20. **1** **2** **3**
21. **1** **2** **3**
22. **1** **2** **3**
23. **1** **2** **3**
24. **1** **2** **3**
25. **1** **2** **3**
26. **1** **2** **3**
27. **1** **2** **3**
28. **1** **2** **3**
29. **1** **2** **3**
30. **1** **2** **3**
31. **1** **2** **3**
32. **1** **2** **3**
33. **1** **2** **3**
34. **1** **2** **3**
35. **1** **2** **3**
36. **1** **2** **3**
37. **1** **2** **3**
38. **1** **2** **3**
39. **1** **2** **3**
40. **1** **2** **3**
41. **1** **2** **3**
42. **1** **2** **3**
43. **1** **2** **3**
44. **1** **2** **3**
45. **1** **2** **3**
46. **1** **2** **3**
47. **1** **2** **3**
48. **1** **2** **3**
49. **1** **2** **3**
50. **1** **2** **3**
51. **1** **2** **3**
52. **1** **2** **3**
53. **1** **2** **3**
54. **1** **2** **3**
55. **1** **2** **3**
56. **1** **2** **3**
57. **1** **2** **3**
58. **1** **2** **3**
59. **1** **2** **3**
60. **1** **2** **3**
61. **1** **2** **3**
62. **1** **2** **3**
63. **1** **2** **3**
64. **1** **2** **3**
65. **1** **2** **3**
66. **1** **2** **3**
67. **1** **2** **3**
68. **1** **2** **3**
69. **1** **2** **3**
70. **1** **2** **3**
71. **1** **2** **3**
72. **1** **2** **3**
73. **1** **2** **3**
74. **1** **2** **3**
75. **1** **2** **3**
76. **1** **2** **3**
77. **1** **2** **3**
78. **1** **2** **3**
79. **1** **2** **3**
80. **1** **2** **3**
81. **1** **2** **3**
82. **1** **2** **3**
83. **1** **2** **3**
84. **1** **2** **3**
85. **1** **2** **3**
86. **1** **2** **3**
87. **1** **2** **3**
88. **1** **2** **3**
89. **1** **2** **3**
90. **1** **2** **3**
91. **1** **2** **3**
92. **1** **2** **3**
93. **1** **2** **3**
94. **1** **2** **3**
95. **1** **2** **3**
96. **1** **2** **3**
97. **1** **2** **3**
98. **1** **2** **3**
99. **1** **2** **3**
100. **1** **2** **3**



conv2d_2 (3, 3, 64, 128)
128

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21
22	23	24
25	26	27
28	29	30
31	32	33
34	35	36
37	38	39
40	41	42
43	44	45
46	47	48
49	50	51
52	53	54
55	56	57
58	59	60
61	62	63
64	65	66
67	68	69
70	71	72
73	74	75
76	77	78
79	80	81
82	83	84
85	86	87
88	89	90
91	92	93
94	95	96
97	98	99
100	101	102
103	104	105
106	107	108
109	110	111
112	113	114
115	116	117
118	119	120
121	122	123
124	125	126
127	128	129
130	131	132
133	134	135
136	137	138
139	140	141
142	143	144
145	146	147
148	149	150
151	152	153
154	155	156
157	158	159
160	161	162
163	164	165
166	167	168
169	170	171
172	173	174
175	176	177
178	179	180
181	182	183
184	185	186
187	188	189
190	191	192
193	194	195
196	197	198
199	200	201
202	203	204
205	206	207
208	209	210
211	212	213
214	215	216
217	218	219
220	221	222
223	224	225
226	227	228
229	230	231
232	233	234
235	236	237
238	239	240
241	242	243
244	245	246
247	248	249
250	251	252
253	254	255
256	257	258
259	260	261
262	263	264
265	266	267
268	269	270
271	272	273
274	275	276
277	278	279
280	281	282
283	284	285
286	287	288
289	290	291
292	293	294
295	296	297
298	299	300
301	302	303
304	305	306
307	308	309
310	311	312
313	314	315
316	317	318
319	320	321
322	323	324
325	326	327
328	329	330
331	332	333
334	335	336
337	338	339
340	341	342
343	344	345
346	347	348
349	350	351
352	353	354
355	356	357
358	359	360
361	362	363
364	365	366
367	368	369
370	371	372
373	374	375
376	377	378
379	380	381
382	383	384
385	386	387
388	389	390
391	392	393
394	395	396
397	398	399
400	401	402
403	404	405
406	407	408
409	410	411
412	413	414
415	416	417
418	419	420
421	422	423
424	425	426
427	428	429
430	431	432
433	434	435
436	437	438
439	440	441
442	443	444
445	446	447
448	449	450
451	452	453
454	455	456
457	458	459
460	461	462
463	464	465
466	467	468
469	470	471
472	473	474
475	476	477
478	479	480
481	482	483
484	485	486
487	488	489
490	491	492
493	494	495
496	497	498
499	500	501
502	503	504
505	506	507
508	509	510
511	512	513
514	515	516
517	518	519
520	521	522
523	524	525
526	527	528
529	530	531
532	533	534
535	536	537
538	539	540
541	542	543
544	545	546
547	548	549
550	551	552
553	554	555
556	557	558
559	560	561
562	563	564
565	566	567
568	569	570
571	572	573
574	575	576
577	578	579
580	581	582
583	584	585
586	587	588
589	590	591
592	593	594
595	596	597
598	599	600
601	602	603
604	605	606
607	608	609
610	611	612
613	614	615
616	617	618
619	620	621
622	623	624
625	626	627
628	629	630
631	632	633
634	635	636
637	638	639
640	641	642
643	644	645
646	647	648
649	650	651
652	653	654
655	656	657
658	659	660
661	662	663
664	665	666
667	668	669
670	671	672
673	674	675
676	677	678
679	680	681
682	683	684
685	686	687
688	689	690
691	692	693
694	695	696
697	698	699
700	701	702
703	704	705
706	707	708
709	710	711
712	713	714
715	716	717
718	719	720
721	722	723
724	725	726
727	728	729
730	731	732
733	734	735
736	737	738
739	740	741
742	743	744
745	746	747
748	749	750
751	752	753
754	755	756
757	758	759
760	761	762
763	764	765
766	767	768
769	770	771
772	773	774
775	776	777
778	779	780
781	782	783
784	785	786
787	788	789
790	791	792
793	794	795
796	797	798
799	800	801
802	803	804
805	806	807
808	809	8010
8011	8012	8013
8014	8015	8016
8017	8018	8019
8020	8021	8022
8023	8024	8025
8026	8027	8028
8029	8030	8031
8032	8033	8034
8035	8036	8037
8038	8039	8040
8041	8042	8043
8044	8045	8046
8047	8048	8049
8050	8051	8052
8053	8054	8055
8056	8057	8058
8059	8060	8061
8062	8063	8064
8065	8066	8067
8068	8069	8070
8071	8072	8073
8074	8075	8076
8077	8078	8079
8080	8081	8082
8083	8084	8085
8086	8087	8088
8089	8090	8091
8092	8093	8094
8095	8096	8097
8098	8099	80100
80101	80102	80103
80104	80105	80106
80107	80108	80109
80110	80111	80112
80113	80114	80115
80116	80117	80118
80119	80120	80121
80122	80123	80124
80125	80126	80127
80128	80129	80130
80131	80132	80133
80134	80135	80136
80137	80138	80139
80140	80141	80142
80143	80144	80145
80146	80147	80148
80149	80150	80151
80152	80153	80154
80155	80156	80157
80158	80159	80160
80161	80162	80163
80164	80165	80166
80167	80168	80169
80170	80171	80172
80173	80174	80175
80176	80177	80178
80179	80180	80181
80182	80183	80184
80185	80186	80187
80188	80189	80190
80191	80192	80193
80194	80195	80196
80197	80198	80199
80200	80201	80202
80203	80204	80205
80206	80207	80208
80209	80210	80211
80212	80213	80214
80215	80216	80217
80218	80219	80220
80221	80222	80223
80224	80225	80226
80227	80228	80229
80230	80231	80232
80233	80234	80235
80236	80237	80238
80239	80240	80241
80242	80243	80244
80245	80246	80247
80248	80249	80250
80251	80252	80253
80254	80255	80256
80257	80258	80259
80260	80261	80262
80263	80264	80265
80266	80267	80268
80269	80270	80271
80272	80273	80274
80275	80276	80277
80278	80279	80280
80281	80282	80283
80284	80285	80286
80287	80288	80289
80290	80291	80292
80293	80294	80295
80296	80297	80298
80299	80300	80301
80302	80303	80304
80305	80306	80307
80308	80309	80310
80311	80312	80313
80314	80315	80316
80317	80318	80319
80320	80321	80322
80323	80324	80325
80326	80327	80328
80329	80330	80331
80332	80333	80334
80335	80336	80337
80338	80339	80340
80341	80342	80343
80344	80345	

1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21
22	23	24
25	26	27
28	29	30
31	32	33
34	35	36
37	38	39
40	41	42
43	44	45
46	47	48
49	50	51
52	53	54
55	56	57
58	59	60
61	62	63
64	65	66
67	68	69
70	71	72
73	74	75
76	77	78
79	80	81
82	83	84
85	86	87
88	89	90
91	92	93
94	95	96
97	98	99
100	101	102
103	104	105
106	107	108
109	110	111
112	113	114
115	116	117
118	119	120
121	122	123
124	125	126
127	128	129
130	131	132
133	134	135
136	137	138
139	140	141
142	143	144
145	146	147
148	149	150
151	152	153
154	155	156
157	158	159
160	161	162
163	164	165
166	167	168
169	170	171
172	173	174
175	176	177
178	179	180
181	182	183
184	185	186
187	188	189
190	191	192
193	194	195
196	197	198
199	200	201
202	203	204
205	206	207
208	209	210
211	212	213
214	215	216
217	218	219
220	221	222
223	224	225
226	227	228
229	230	231
232	233	234
235	236	237
238	239	240
241	242	243
244	245	246
247	248	249
250	251	252
253	254	255
256	257	258
259	260	261
262	263	264
265	266	267
268	269	270
271	272	273
274	275	276
277	278	279
280	281	282
283	284	285
286	287	288
289	290	291
292	293	294
295	296	297
298	299	300
301	302	303
304	305	306
307	308	309
310	311	312
313	314	315
316	317	318
319	320	321
322	323	324
325	326	327
328	329	330
331	332	333
334	335	336
337	338	339
340	341	342
343	344	345
346	347	348
349	350	351
352	353	354
355	356	357
358	359	360
361	362	363
364	365	366
367	368	369
370	371	372
373	374	375
376	377	378
379	380	381
382	383	384
385	386	387
388	389	390
391	392	393
394	395	396
397	398	399
400	401	402
403	404	405
406	407	408
409	410	411
412	413	414
415	416	417
418	419	420
421	422	423
424	425	426
427	428	429
430	431	432
433	434	435
436	437	438
439	440	441
442	443	444
445	446	447
448	449	450
451	452	453
454	455	456
457	458	459
460	461	462
463	464	465
466	467	468
469	470	471
472	473	474
475	476	477
478	479	480
481	482	483
484	485	486
487	488	489
490	491	492
493	494	495
496	497	498
499	500	501
502	503	504
505	506	507
508	509	510
511	512	513
514	515	516
517	518	519
520	521	522
523	524	525
526	527	528
529	530	531
532	533	534
535	536	537
538	539	540
541	542	543
544	545	546
547	548	549
550	551	552
553	554	555
556	557	558
559	560	561
562	563	564
565	566	567
568	569	570
571	572	573
574	575	576
577	578	579
580	581	582
583	584	585
586	587	588
589	590	591
592	593	594
595	596	597
598	599	600
601	602	603
604	605	606
607	608	609
610	611	612
613	614	615
616	617	618
619	620	621
622	623	624
625	626	627
628	629	630
631	632	633
634	635	636
637	638	639
640	641	642
643	644	645
646	647	648
649	650	651
652	653	654
655	656	657
658	659	660
661	662	663
664	665	666
667	668	669
670	671	672
673	674	675
676	677	678
679	680	681
682	683	684
685	686	687
688	689	690
691	692	693
694	695	696
697	698	699
700	701	702
703	704	705
706	707	708
709	710	711
712	713	714
715	716	717
718	719	720
721	722	723
724	725	726
727	728	729
730	731	732
733	734	735
736	737	738
739	740	741
742	743	744
745	746	747
748	749	750
751	752	753
754	755	756
757	758	759
760	761	762
763	764	765
766	767	768
769	770	771
772	773	774
775	776	777
778	779	780
781	782	783
784	785	786
787	788	789
790	791	792
793	794	795
796	797	798
799	800	801
802	803	804
805	806	807
808	809	8010
8011	8012	8013
8014	8015	8016
8017	8018	8019
8020	8021	8022
8023	8024	8025
8026	8027	8028
8029	8030	8031
8032	8033	8034
8035	8036	8037
8038	8039	8040
8041	8042	8043
8044	8045	8046
8047	8048	8049
8050	8051	8052
8053	8054	8055
8056	8057	8058
8059	8060	8061
8062	8063	8064
8065	8066	8067
8068	8069	8070
8071	8072	8073
8074	8075	8076
8077	8078	8079
8080	8081	8082
8083	8084	8085
8086	8087	8088
8089	8090	8091
8092	8093	8094
8095	8096	8097
8098	8099	80100
80101	80102	80103
80104	80105	80106
80107	80108	80109
80110	80111	80112
80113	80114	80115
80116	80117	80118
80119	80120	80121
80122	80123	80124
80125	80126	80127
80128	80129	80130
80131	80132	80133
80134	80135	80136
80137	80138	80139
80140	80141	80142
80143	80144	80145
80146	80147	80148
80149	80150	80151
80152	80153	80154
80155	80156	80157
80158	80159	80160
80161	80162	80163
80164	80165	80166
80167	80168	80169
80170	80171	80172
80173	80174	80175
80176	80177	80178
80179	80180	80181
80182	80183	80184
80185	80186	80187
80188	80189	80190
80191	80192	80193
80194	80195	80196
80197	80198	80199
80200	80201	80202
80203	80204	80205
80206	80207	80208
80209	80210	80211
80212	80213	80214
80215	80216	80217
80218	80219	80220
80221	80222	80223
80224	80225	80226
80227	80228	80229
80230	80231	80232
80233	80234	80235
80236	80237	80238
80239	80240	80241
80242	80243	80244
80245	80246	80247
80248	80249	80250
80251	80252	80253
80254	80255	80256
80257	80258	80259
80260	80261	80262
80263	80264	80265
80266	80267	80268
80269	80270	80271
80272	80273	80274
80275	80276	80277
80278	80279	80280
80281	80282	80283
80284	80285	80286
80287	80288	80289
80290	80291	80292
80293	80294	80295
80296	80297	80298
80299	80300	80301
80302	80303	80304
80305	80306	80307
80308	80309	80310
80311	80312	80313
80314	80315	80316
80317	80318	80319
80320	80321	80322
80323	80324	80325
80326	80327	80328
80329	80330	80331
80332	80333	80334
80335	80336	80337
80338	80339	80340
80341	80342	80343
80344	80345</	

SPANNING NUMBER	MAXIMUM SPANNING NUMBER	MINIMUM SPANNING NUMBER
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1
9	9	1
10	10	1
11	11	1
12	12	1
13	13	1
14	14	1
15	15	1
16	16	1
17	17	1
18	18	1
19	19	1
20	20	1
21	21	1
22	22	1
23	23	1
24	24	1
25	25	1
26	26	1
27	27	1
28	28	1
29	29	1
30	30	1
31	31	1
32	32	1
33	33	1
34	34	1
35	35	1
36	36	1
37	37	1
38	38	1
39	39	1
40	40	1
41	41	1
42	42	1
43	43	1
44	44	1
45	45	1
46	46	1
47	47	1
48	48	1
49	49	1
50	50	1
51	51	1
52	52	1
53	53	1
54	54	1
55	55	1
56	56	1
57	57	1
58	58	1
59	59	1
60	60	1
61	61	1
62	62	1
63	63	1
64	64	1
65	65	1
66	66	1
67	67	1
68	68	1
69	69	1
70	70	1
71	71	1
72	72	1
73	73	1
74	74	1
75	75	1
76	76	1
77	77	1
78	78	1
79	79	1
80	80	1
81	81	1
82	82	1
83	83	1
84	84	1
85	85	1
86	86	1
87	87	1
88	88	1
89	89	1
90	90	1
91	91	1
92	92	1
93	93	1
94	94	1
95	95	1
96	96	1
97	97	1
98	98	1
99	99	1
100	100	1
101	101	1
102	102	1
103	103	1
104	104	1
105	105	1
106	106	1
107	107	1
108	108	1
109	109	1
110	110	1
111	111	1
112	112	1
113	113	1
114	114	1
115	115	1
116	116	1
117	117	1
118	118	1
119	119	1
120	120	1
121	121	1
122	122	1
123	123	1
124	124	1
125	125	1
126	126	1
127	127	1
128	128	1
129	129	1
130	130	1
131	131	1
132	132	1
133	133	1
134	134	1
135	135	1
136	136	1
137	137	1
138	138	1
139	139	1
140	140	1
141	141	1
142	142	1
143	143	1
144	144	1
145	145	1
146	146	1
147	147	1
148	148	1
149	149	1
150	150	1
151	151	1
152	152	1
153	153	1
154	154	1
155	155	1
156	156	1
157	157	1
158	158	1
159	159	1
160	160	1
161	161	1
162	162	1
163	163	1
164	164	1
165	165	1
166	166	1
167	167	1
168	168	1
169	169	1
170	170	1
171	171	1
172	172	1
173	173	1
174	174	1
175	175	1
176	176	1
177	177	1
178	178	1
179	179	1
180	180	1
181	181	1
182	182	1
183	183	1
184	184	1
185	185	1
186	186	1
187	187	1
188	188	1
189	189	1
190	190	1
191	191	1
192	192	1
193	193	1
194	194	1
195	195	1
196	196	1
197	197	1
198	198	1
199	199	1
200	200	1
201	201	1
202	202	1
203	203	1
204	204	1
205	205	1
206	206	1
207	207	1
208	208	1
209	209	1
210	210	1
211	211	1
212	212	1
213	213	1
214	214	1
215	215	1
216	216	1
217	217	1
218	218	1
219	219	1
220	220	1
221	221	1
222	222	1
223	223	1
224	224	1
225	225	1
226	226	1
227	227	1
228	228	1
229	229	1
230	230	1
231	231	1
232	232	1
233	233	1
234	234	1
235	235	1
236	236	1
237	237	1
238	238	1
239	239	1
240	240	1
241	241	1
242	242	1
243	243	1
244	244	1
245	245	1
246	246	1
247	247	1
248	248	1
249	249	1
250	250	1
251	251	1
252	252	1
253	253	1
254	254	1
255	255	1
256	256	1
257	257	1
258	258	1
259	259	1
260	260	1
261	261	1
262	262	1
263	263	1
264	264	1
265	265	1
266	266	1
267	267	1
268	268	1
269	269	1
270	270	1
271	271	1
272	272	1
273	273	1
274	274	1
275	275	1
276	276	1
277	277	1
278	278	1
279	279	1
280	280	1
281	281	1
282	282	1
283	283	1
284	284	1
285	285	1
286	286	1
287	287	1
288	288	1
289	289	1
290	290	1
291	291	1
292	292	1
293	293	1
294	294	1
295	295	1
296	296	1
297	297	1
298	298	1
299	299	1
300	300	1
301	301	1
302	302	1
303	303	1
304	304	1
305	305	1
306	306	1
307	307	1
308	308	1
309	309	1
310	310	1
311	311	1
312	312	1
313	313	1
314	314	1
315	315	1
316	316	1
317	317	1
318	318	1
319	319	1
320	320	1
321	321	1
322	322	1
323	323	1
324	324	1
325	325	1
326	326	1
327	327	1
328	328	1
329	329	1
330	330	1
331	331	1
332	332	1
333	333	1
334	334	1
335	335	1
336	336	1
337	337	1
338	338	1
339	339	1
340	340	1
341	341	1
342	342	1
343	343	1
344	344	1
345	345	1
346	346	1
347	347	1
348	348	1
349	349	1
350	350	1
351	351	1
352	352	1
353	353	1
354	354	1
355	355	1
356	356	1
357	357	1
358	358	1
359	359	1
360	360	1
361	361	1
362	362	1
363	363	1
364	364	1
365	365	1
366	366	1
367	367	1
368	368	1
369	369	1
370	370	1
371	371	1
372	372	1
373	373	1
374	374	1
375	375	1
376	376	1
377	377	1
378	378	1
379	379	1
380	380	1
381	381	1
382	382	1
383	383	1
384	384	1
385	385	1
386	386	1
387	387	1
388	388	1
389	389	1
390	390	1
391	391	1
392	392	1
393	393	1
394	394	1
395	395	1
396	396	1
397	397	1
398	398	1
399	399	1
400	400	1
401	401	1
402	402	1
403	403	1
404	404	1
405	405	1
406	406	1
407	407	1
408	408	1
409	409	1
410	410	1
411	411	1
412	412	1
413	413	1
414	414	1
415	415	1
416	416	1
417	417	1
418	418	1
419	419	1
420	420	1
421	421	1
422	422	1
423	423	1
424	424	1
425	425	1
426	426	1
427	427	1
428	428	1
429	429	1
430	430	1
431	431	1
432	432	1
433	433	1
434	434	1
435	435	1
436</		



NAME	ADDRESS	PHONE NUMBER
John Doe	123 Main Street	(555) 123-4567
Jane Doe	456 Elm Street	(555) 234-5678
Bob Smith	789 Oak Street	(555) 345-6789
Susan Johnson	210 Pine Street	(555) 456-7890
David Wilson	321 Cedar Street	(555) 567-8901
Emily Davis	432 Birch Street	(555) 678-9012
Frank Miller	543 Holly Street	(555) 789-0123
Grace Lewis	654 Maple Street	(555) 890-1234
Henry Thompson	765 Chestnut Street	(555) 901-2345
Mary Parker	876 Locust Street	(555) 012-3456
Tommy Lee	987 Pine Street	(555) 123-4567
Wendy Green	109 Cedar Street	(555) 234-5678
Mike Brown	218 Birch Street	(555) 345-6789
Nancy White	327 Holly Street	(555) 456-7890
Steve Black	436 Locust Street	(555) 567-8901
Linda Gray	547 Chestnut Street	(555) 678-9012
Howard Jones	658 Pine Street	(555) 789-0123
Carolyn Parker	769 Cedar Street	(555) 890-1234
James Lee	870 Birch Street	(555) 901-2345
Patricia Green	981 Holly Street	(555) 012-3456

CONTINUOUS PREDICTION

www.scholarlypublications.com

卷之三

REFERENCES AND NOTES

www.scholastic.com

REFERENCES

www.english-test.net

[View Details](#)

ANSWER PREDICTION

REFERENCES

[View Details](#)

ANSWER

THE END

[View Details](#)

卷之三

• • • • •

www.english-test.net

[View Details](#)

Digitized by srujanika@gmail.com

[View Details](#)

www.ijerph.org

www.nature.com/scientificreports/

www.nature.com/scientificreports/

Digitized by srujanika@gmail.com



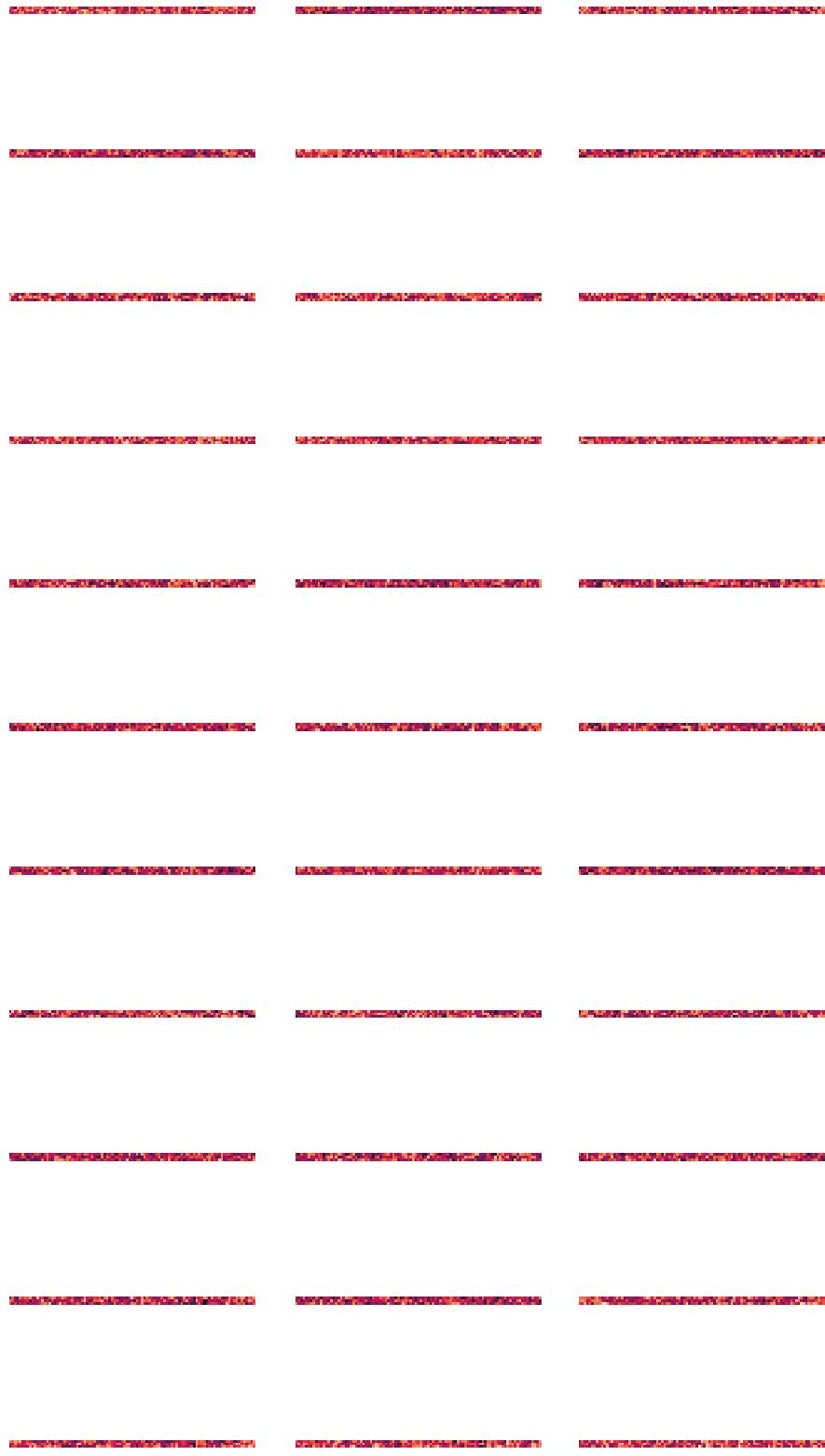
1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21
22	23	24
25	26	27
28	29	30
31	32	33
34	35	36
37	38	39
40	41	42
43	44	45
46	47	48
49	50	51
52	53	54
55	56	57
58	59	60
61	62	63
64	65	66
67	68	69
70	71	72
73	74	75
76	77	78
79	80	81
82	83	84
85	86	87
88	89	90
91	92	93
94	95	96
97	98	99
100	101	102
103	104	105
106	107	108
109	110	111
112	113	114
115	116	117
118	119	120
121	122	123
124	125	126
127	128	129
130	131	132
133	134	135
136	137	138
139	140	141
142	143	144
145	146	147
148	149	150
151	152	153
154	155	156
157	158	159
160	161	162
163	164	165
166	167	168
169	170	171
172	173	174
175	176	177
178	179	180
181	182	183
184	185	186
187	188	189
190	191	192
193	194	195
196	197	198
199	200	201
202	203	204
205	206	207
208	209	210
211	212	213
214	215	216
217	218	219
220	221	222
223	224	225
226	227	228
229	230	231
232	233	234
235	236	237
238	239	240
241	242	243
244	245	246
247	248	249
250	251	252
253	254	255
256	257	258
259	260	261
262	263	264
265	266	267
268	269	270
271	272	273
274	275	276
277	278	279
280	281	282
283	284	285
286	287	288
289	290	291
292	293	294
295	296	297
298	299	300
301	302	303
304	305	306
307	308	309
310	311	312
313	314	315
316	317	318
319	320	321
322	323	324
325	326	327
328	329	330
331	332	333
334	335	336
337	338	339
340	341	342
343	344	345
346	347	348
349	350	351
352	353	354
355	356	357
358	359	360
361	362	363
364	365	366
367	368	369
370	371	372
373	374	375
376	377	378
379	380	381
382	383	384
385	386	387
388	389	390
391	392	393
394	395	396
397	398	399
400	401	402
403	404	405
406	407	408
409	410	411
412	413	414
415	416	417
418	419	420
421	422	423
424	425	426
427	428	429
430	431	432
433	434	435
436	437	438
439	440	441
442	443	444
445	446	447
448	449	450
451	452	453
454	455	456
457	458	459
460	461	462
463	464	465
466	467	468
469	470	471
472	473	474
475	476	477
478	479	480
481	482	483
484	485	486
487	488	489
490	491	492
493	494	495
496	497	498
499	500	501
502	503	504
505	506	507
508	509	510
511	512	513
514	515	516
517	518	519
520	521	522
523	524	525
526	527	528
529	530	531
532	533	534
535	536	537
538	539	540
541	542	543
544	545	546
547	548	549
550	551	552
553	554	555
556	557	558
559	560	561
562	563	564
565	566	567
568	569	570
571	572	573
574	575	576
577	578	579
580	581	582
583	584	585
586	587	588
589	590	591
592	593	594
595	596	597
598	599	600
601	602	603
604	605	606
607	608	609
610	611	612
613	614	615
616	617	618
619	620	621
622	623	624
625	626	627
628	629	630
631	632	633
634	635	636
637	638	639
640	641	642
643	644	645
646	647	648
649	650	651
652	653	654
655	656	657
658	659	660
661	662	663
664	665	666
667	668	669
670	671	672
673	674	675
676	677	678
679	680	681
682	683	684
685	686	687
688	689	690
691	692	693
694	695	696
697	698	699
700	701	702
703	704	705
706	707	708
709	710	711
712	713	714
715	716	717
718	719	720
721	722	723
724	725	726
727	728	729
730	731	732
733	734	735
736	737	738
739	740	741
742	743	744
745	746	747
748	749	750
751	752	753
754	755	756
757	758	759
760	761	762
763	764	765
766	767	768
769	770	771
772	773	774
775	776	777
778	779	780
781	782	783
784	785	786
787	788	789
790	791	792
793	794	795
796	797	798
799	800	801
802	803	804
805	806	807
808	809	8010
8011	8012	8013
8014	8015	8016
8017	8018	8019
8020	8021	8022
8023	8024	8025
8026	8027	8028
8029	8030	8031
8032	8033	8034
8035	8036	8037
8038	8039	80310
80311	80312	80313
80314	80315	80316
80317	80318	80319
80320	80321	80322
80323	80324	80325
80326	80327	80328
80329	80330	80331
80332	80333	80334
80335	80336	80337
80338	80339	803310
803311	803312	803313
803314	803315	803316
803317	803318	803319
803320	803321	803322
803323	803324	803325
803326	803327	803328
803329	803330	803331
803332	803333	803334
803335	803336	803337
803338	803339	8033310
8033311	8033312	8033313
8033314	8033315	8033316
8033317	8033318	8033319
8033320	8033321	8033322
8033323	8033324	8033325
8033326	8033327	8033328
8033329	8033330	8033331
8033332	8033333	8033334
8033335	8033336	8033337
8033338	8033339	80333310
80333311	80333312	80333313
80333314	80333315	80333316
80333317	80333318	80333319
80333320	80333321	80333322
80333323	80333324	80333325
80333326	80333327	80333328
80333329	80333330	80333331
80333332	80333333	80333334
80333335	80333336	80333337
80333338	80333339	803333310
803333311	803333312	803333313
803333314	803333315	803333316
803333317	803333318	803333319
803333320	803333321	803333322
803333323	803333324	803333325
803333326	803333327	803333328
803333329	803333330	803333331
803333332	803333333	803333334
803333335	803333336	803333337
803333338	803333339	8033333310
8033333311	8033333312	8033333313
8033333314	8033333315	8033333316
8033333317	8033333318	8033333319
8033333320	8033333321	8033333322
8033333323	8033333324	8033333325
8033333326	8033333327	8033333328
8033333329	8033333330	8033333331
8033333332	8033333333	8033333334
8033333335	8033333336	8033333337
8033333338	8033333339	80333333310
80333333311	80333333312	80333333313
80333333314	80333333315	80333333316
80333333317	80333333318	80333333319
80333333320	80333333321	80333333322
80333333323	80333333324	80333333325
80333333326	80333333327	80333333328
80333333329	80333333330	80333333331
80333333332	80333333333	80333333334
80333333335	80333333336	80333333337
80333333338	80333333339	803333333310
803333333311	803333333312	803333333313
803333333314	803333333315	803333333316
803333333317	803333333318	803333333319
803333333320	803333333321	803333333322
803333333323	803333333324	803333333325
803333333326	803333333327	803333333328
803333333329	803333333330	803333

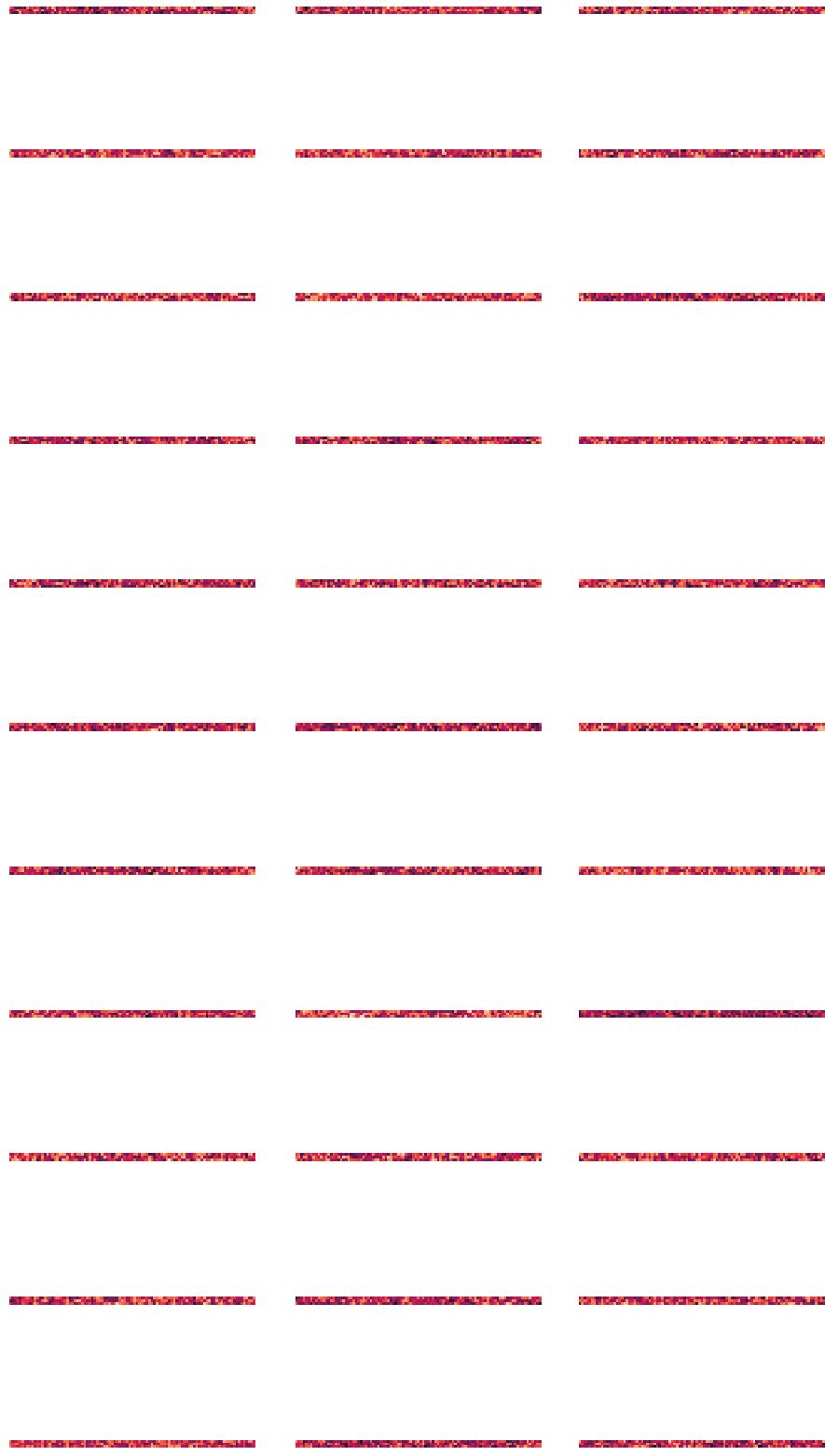
NAME	ADDRESS	PHONE NUMBER
John Doe	123 Main Street	(555) 123-4567
Jane Doe	456 Elm Street	(555) 234-5678
Bob Smith	789 Oak Street	(555) 345-6789
Susan Johnson	210 Pine Street	(555) 456-7890
David Wilson	567 Cedar Street	(555) 567-8901
Emily Davis	890 Birch Street	(555) 678-9012
Mark Green	345 Spruce Street	(555) 789-9013
Karen Brown	654 Chestnut Street	(555) 890-9014
Gregory White	987 Willow Street	(555) 901-9015

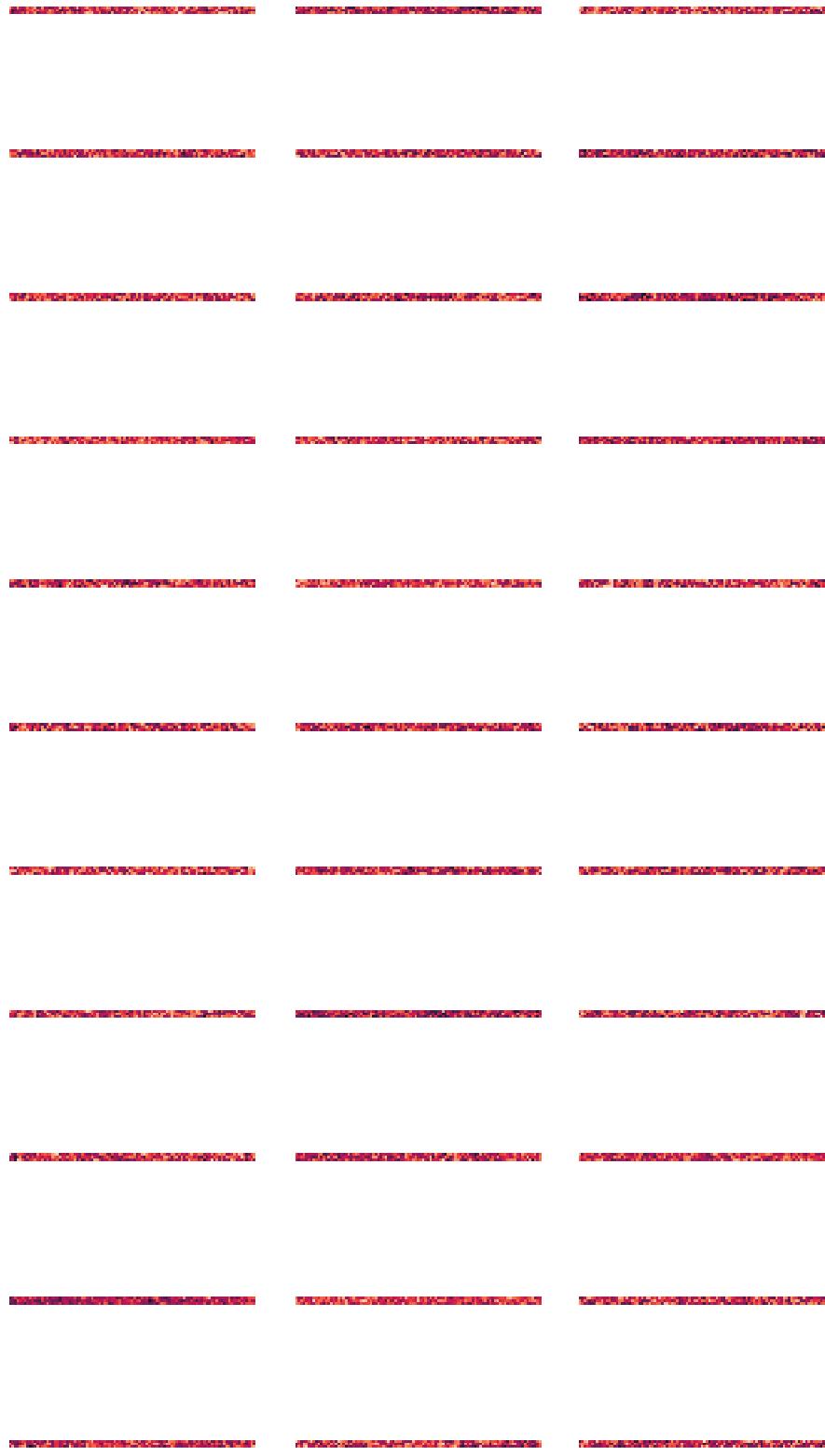


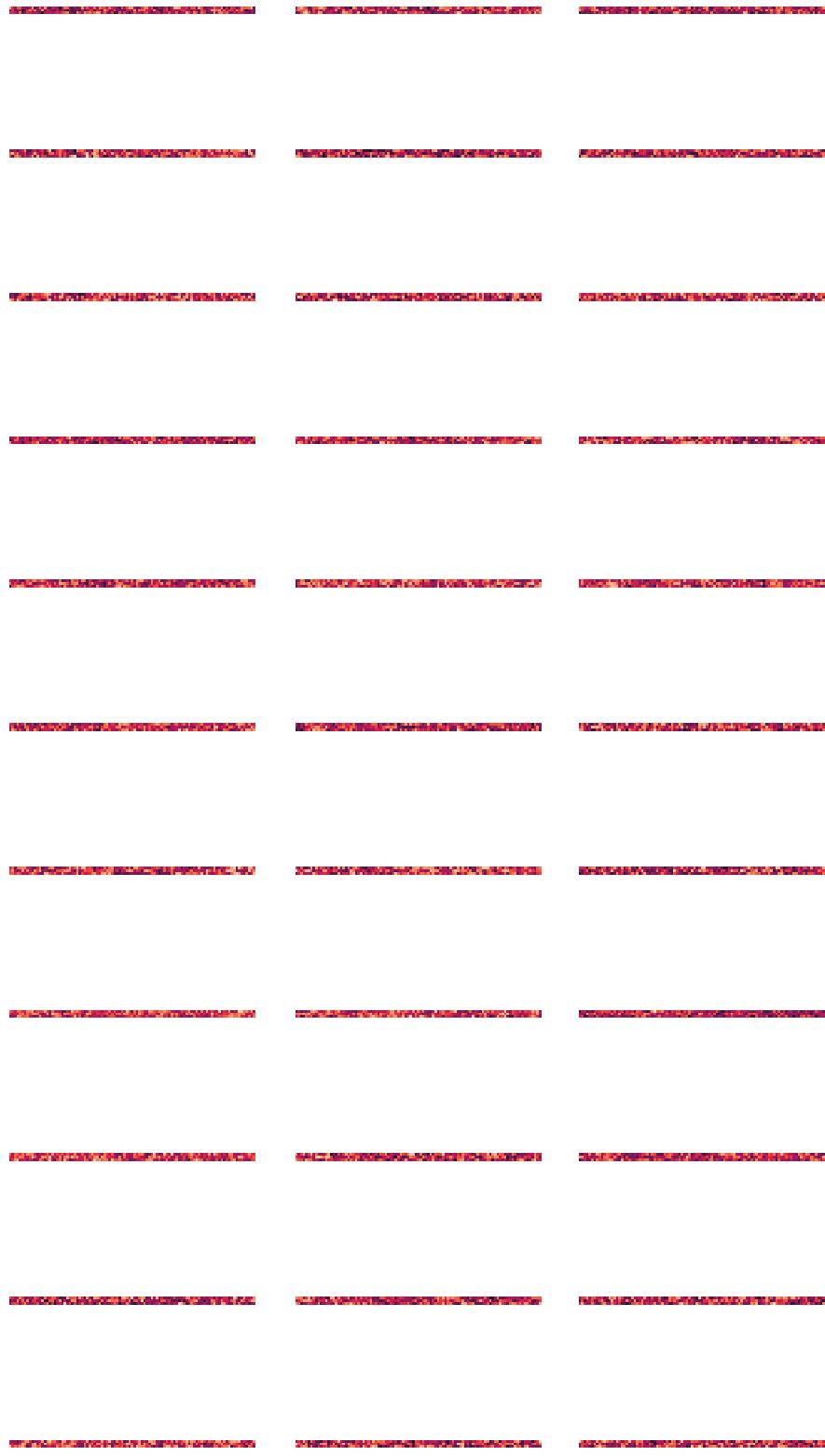
conv2d_3 (3, 3, 128, 128)

128

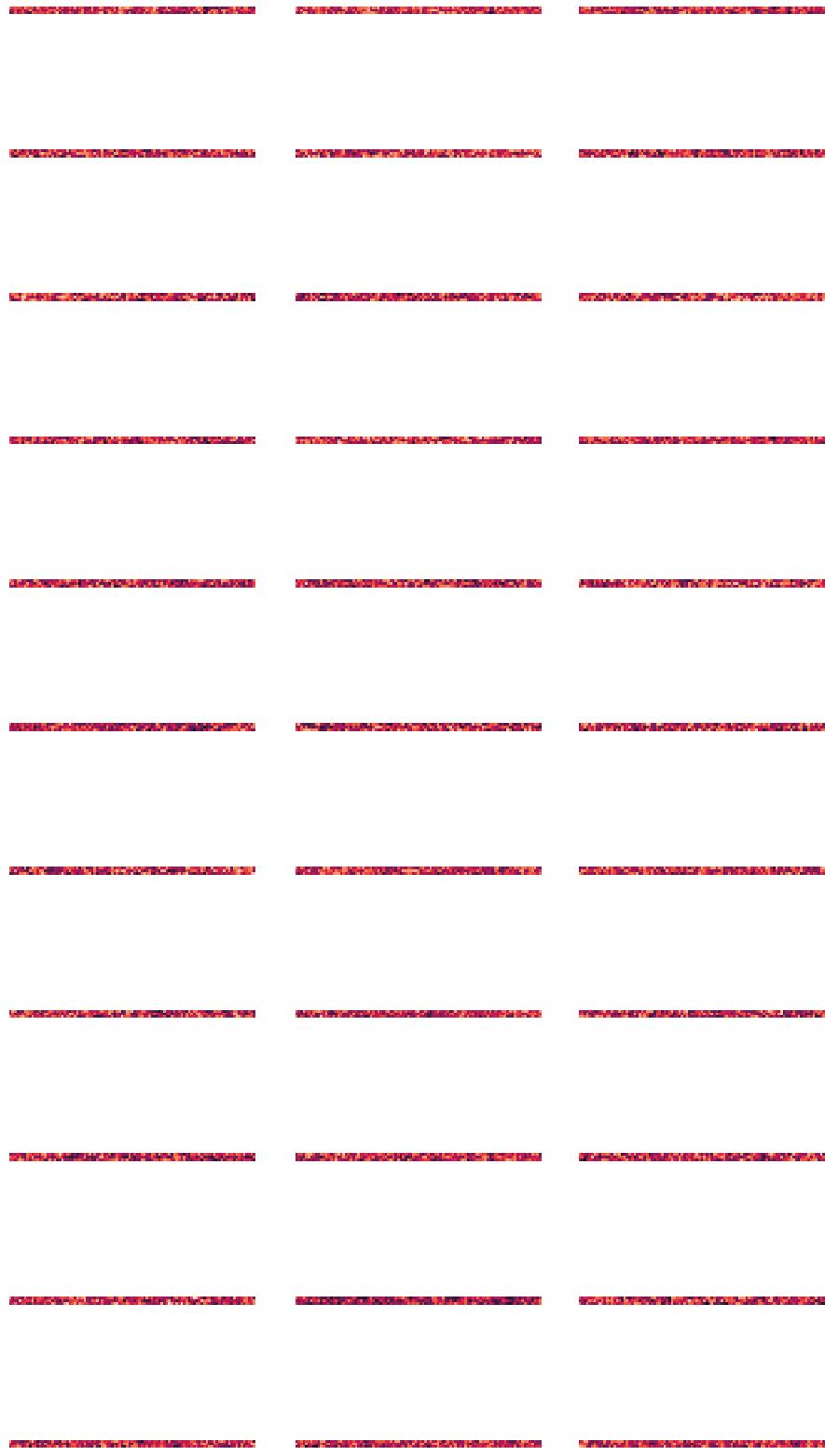


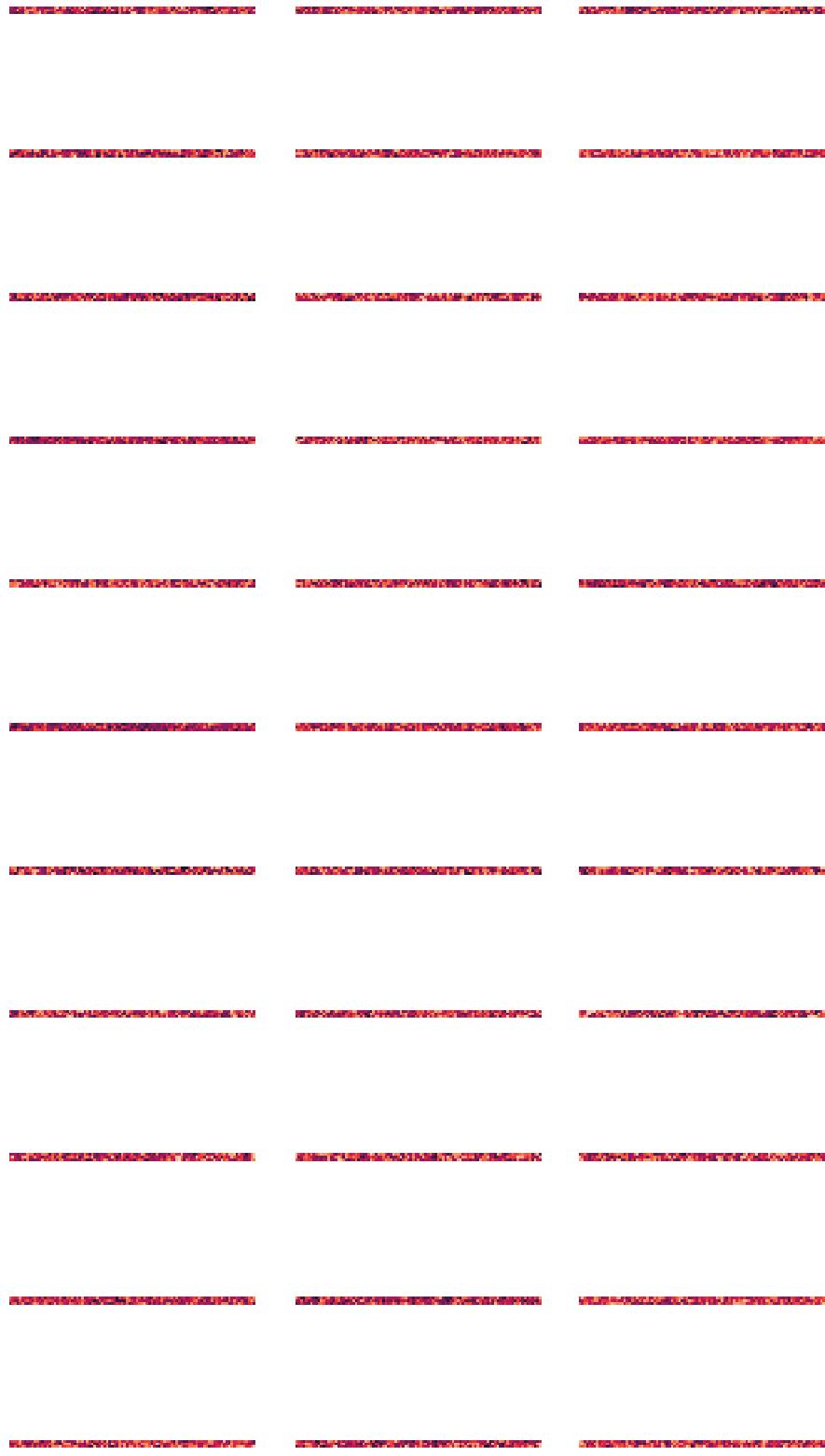




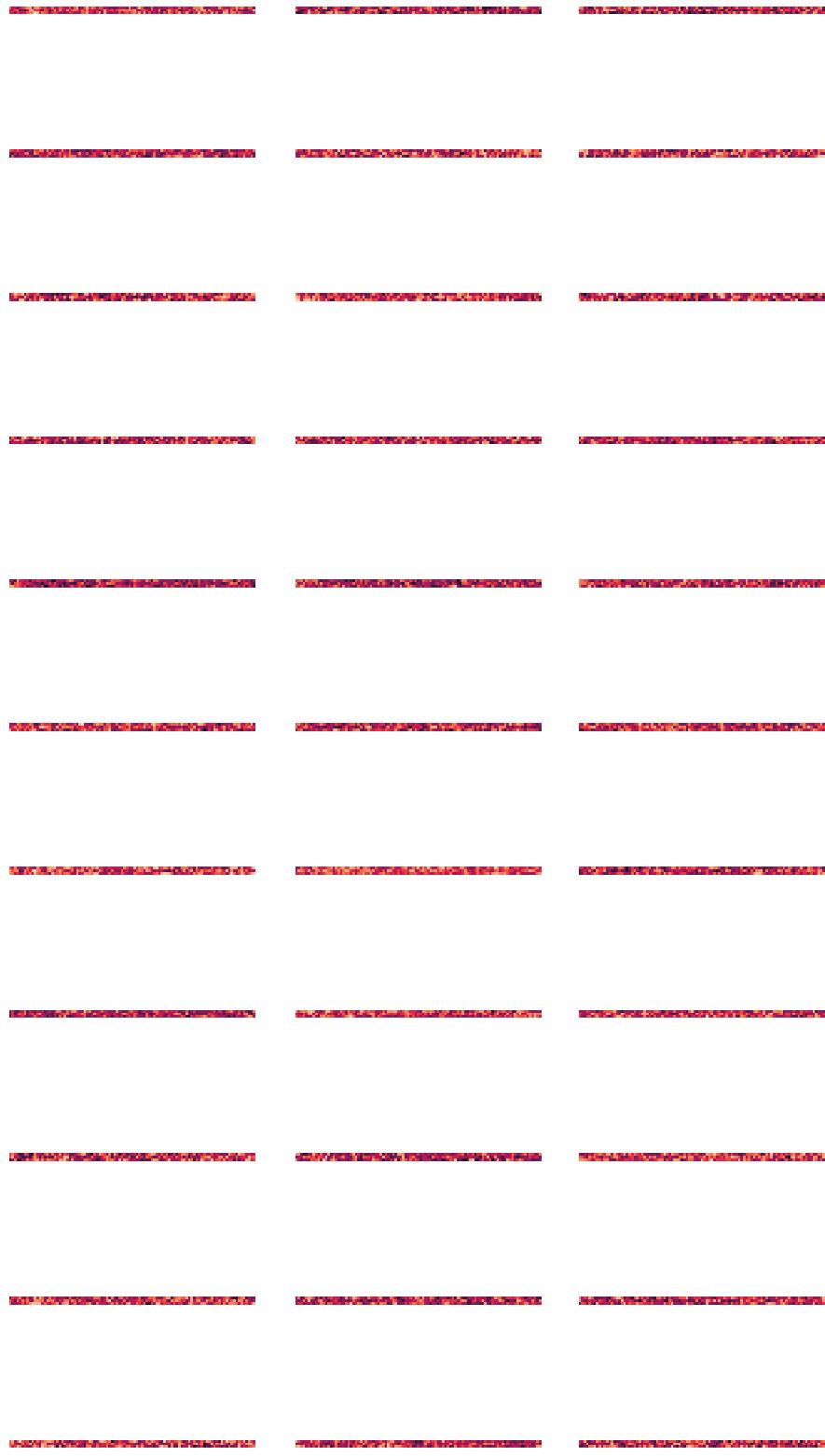


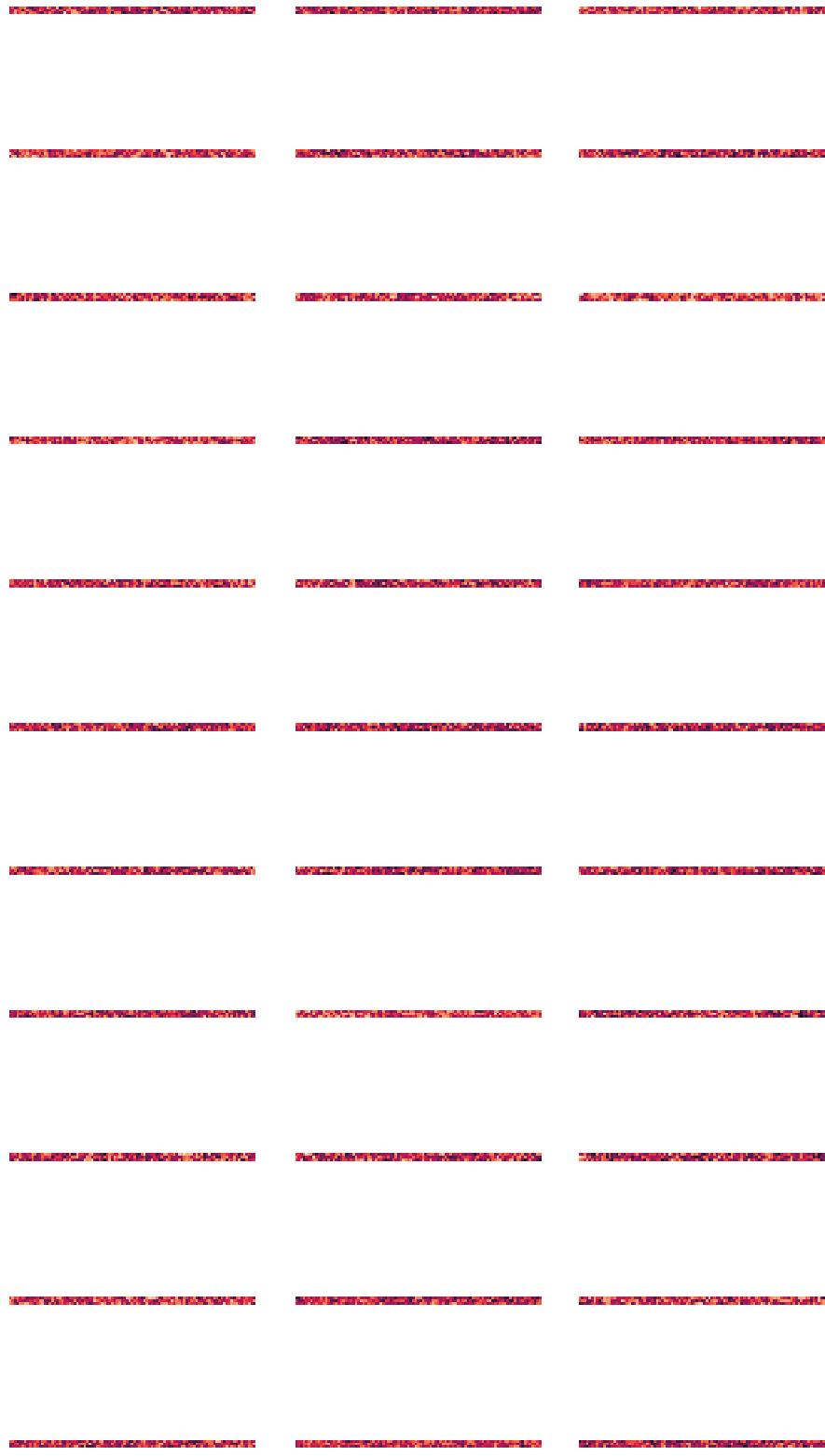














6.7 Visualizing Feature maps

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_feature_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' )

```

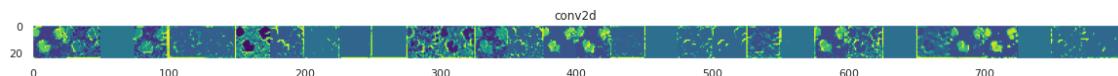
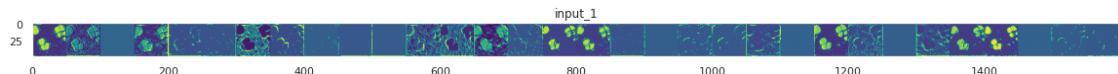
[47]: `visualize_feture_maps(model, img_path='output/archive/8863/1/
→8863_idx5_x1001_y801_class1.png') # IDC Image`

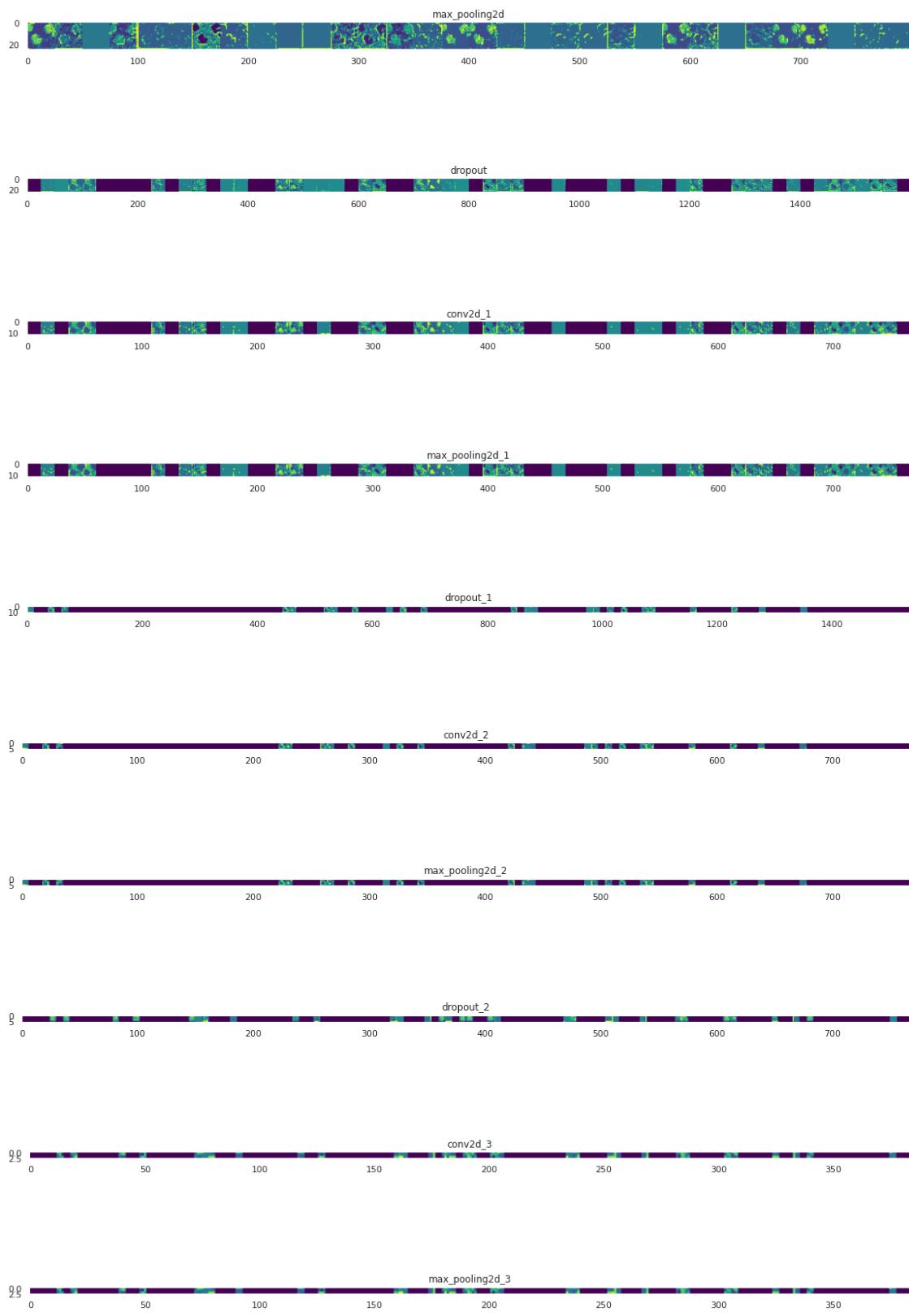
```
(1, 50, 50, 32)
(1, 25, 25, 32)
(1, 25, 25, 32)
(1, 25, 25, 64)
(1, 12, 12, 64)
(1, 12, 12, 64)
(1, 12, 12, 128)
(1, 6, 6, 128)
(1, 6, 6, 128)
```

<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)
```





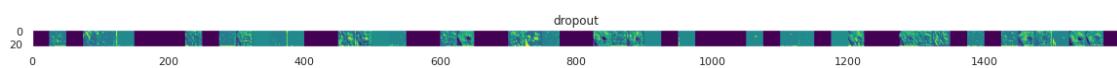
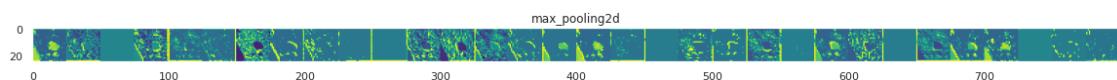
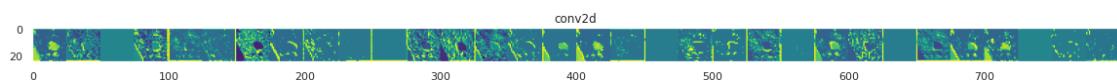
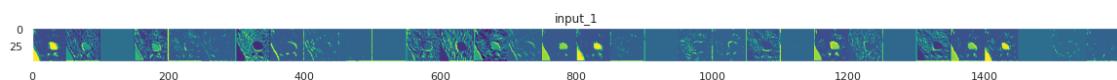
```
[48]: visualize_feture_maps(model, img_path='output/archive/8863/0/  
→8863_idx5_x101_y1301_class0.png') # Non-IDC image
```

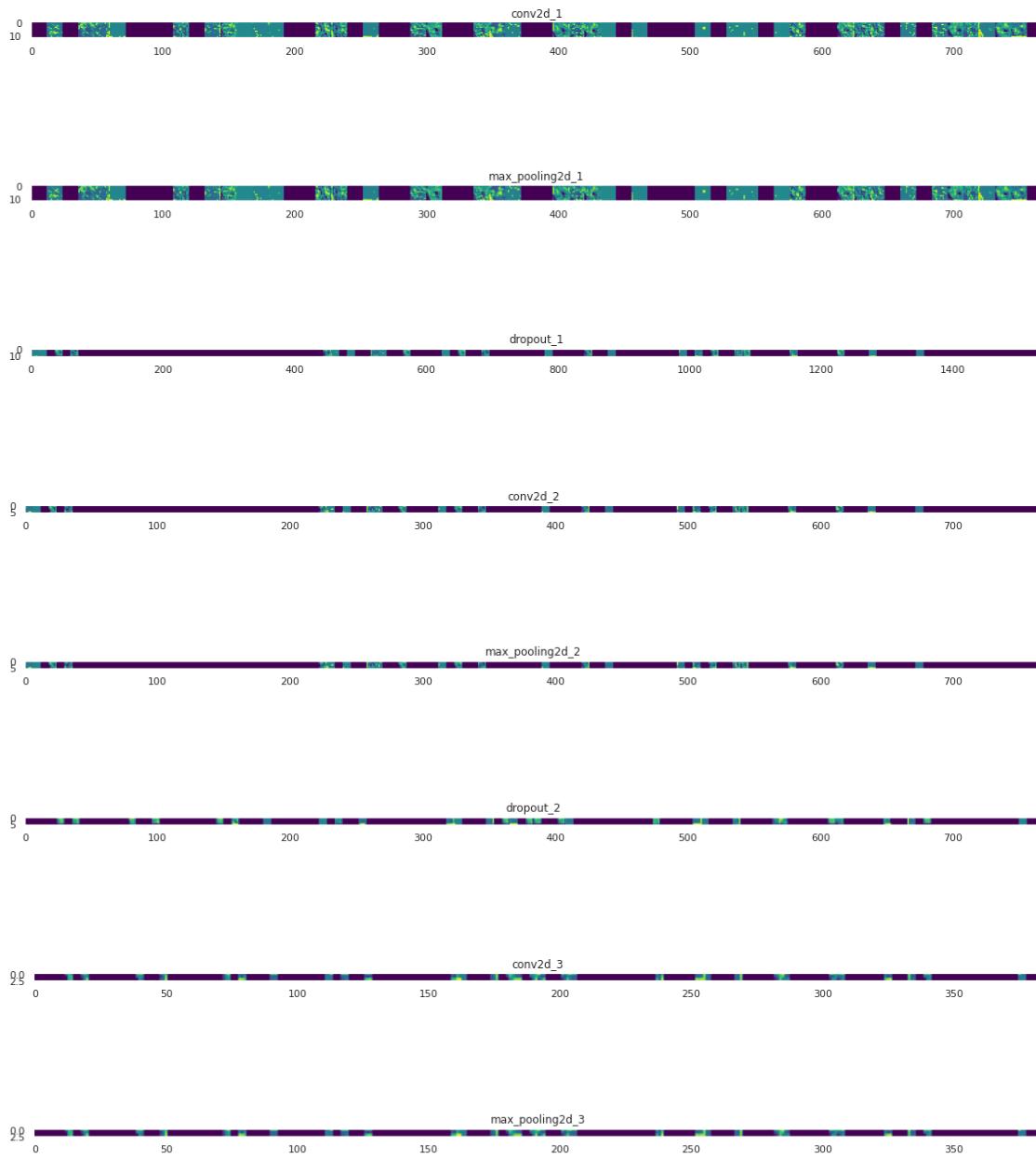
```
(1, 50, 50, 32)  
(1, 25, 25, 32)  
(1, 25, 25, 32)  
(1, 25, 25, 64)  
(1, 12, 12, 64)  
(1, 12, 12, 64)  
(1, 12, 12, 128)  
(1, 6, 6, 128)  
(1, 6, 6, 128)
```

```
<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.

[]: