# Data Science Tools Project Report

# Team Members

| | |
|---|---|
| Mariam Amgad Abdelhalim | 20221424642 |
| Mariem Osama Abdellatif | 20221462958 |
| Mariem Salah Abdelrahman | 20221311472 |
| Manal Taha Sabry Zaki | 20221457169 |
| Ahmed Mohamed Mohamed Omar | 20221453829 |
| Omnya Nasr AbdelAty Hassan | 20221457157 |
| Mariam Ali Hassan Mousa | 20221462924 |
| Huda Ahmed Elsayed Hafez | 20221442691 |
| Salma Afifi Ali Mohammed | 20221453713 |

# 1. project idea

***Predicting Breast Cancer Survival Rates and kinds***

Objective: The objective of this project is to develop a machine learning model that can predict the survival rates and kinds of breast cancer patients based on various features.

# 2. what our code do !

This code imports various libraries and modules for data analysis, visualization, machine learning, and deep learning. It then reads in a CSV file containing DICOM data and image directories. It performs data cleaning, preprocessing, exploratory data analysis, image visualization, feature selection using chi-squared and mutual information techniques, model training using CNN and SVM algorithms, evaluation of model performance using accuracy score and classification report metrics, comparison of models using bar plots, fine-tuning of the CNN model using transfer learning with pre-trained weights, visualization of images from each model's predictions for comparison purposes.

Additionally, it demonstrates the application of feature selection techniques such as Recursive Feature Elimination (RFE), Sequential Feature Selection (SFS), Principal Component Analysis (PCA), and K-Means Clustering for dimensionality reduction and clustering analysis.

# 1. Data preprocessing

1

Data Loading and Cleaning:

- Describe how the data was loaded and cleaned. Highlight the necessity of dropping irrelevant columns and handling missing values for analysis.

2

Data Exploration:

- Explain the significance of visualizations in understanding the distribution and relationships within the breast cancer dataset.

3

Image Processing:

- Elaborate on the image-related preprocessing steps, showcasing how grayscale conversions and image loading were performed.

4

Machine Learning Preparation:

- Mention the imported classifiers and their potential usage in model training and evaluation.

5

Data Transformation:

- Discuss the categorical conversions and renaming operations to improve data understanding.

6

Insights via Visualizations:

- Explain how visualizations aid in gaining insights into different attributes of breast cancer data.

# Libraries Imported

```
import os
from os import listdir
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns

import cv2
from matplotlib.image import imread

import tensorflow as tf
from keras.utils.np_utils import to_categorical
from keras.preprocessing import image
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix


import glob
import PIL
import random

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.decomposition import PCA
from sklearn.decomposition import IncrementalPCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision score, recall score, f1 score
```

The code begins by importing various libraries and packages necessary for data manipulation, visualization, machine learning, and image processing.

# Loading and Exploring Data

```
dicom_data = pd.read_csv(r"D:\Semster 5\Data Science Tools\Project\archive\csv\dicom_info.csv")
image_dir = r"D:\Semster 5\Data Science Tools\Project\archive\jpeg"
```

**Loads Data:** Reads information about DICOM (medical imaging) from a CSV file.
**Defines Image Directory:** Specifies the location where the related image files are stored.

```
dicom_data.head()
```

| | file_path | image_path | AccessionNumber | BitsAllocated | BitsStored | BodyPartExamined | Columr |
|---|---|---|---|---|---|---|---|
| 0 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.12930... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.129308... | NaN | 16 | 16 | BREAST | 3 |
| 1 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.24838... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.248386... | NaN | 16 | 16 | BREAST | 352 |
| 2 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.26721... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.267213... | NaN | 16 | 16 | BREAST | 154 |
| 3 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.38118... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.381187... | NaN | 16 | 16 | BREAST | 5 |
| 4 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.38118... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.381187... | NaN | 8 | 8 | Left Breast | 310 |

5 rows × 38 columns

**Shows Initial Data:** Displays a preview of the first few rows of the loaded data to get a quick look at what it contains.

```
dicom_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10237 entries, 0 to 10236
Data columns (total 38 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   file_path                               10237 non-null  object
 1   image_path                              10237 non-null  object
 2   AccessionNumber                         0 non-null      float64
 3   BitsAllocated                           10237 non-null  int64
 4   BitsStored                              10237 non-null  int64
 5   BodyPartExamined                        10237 non-null  object
 6   Columns                                 10237 non-null  int64
 7   ContentDate                             10237 non-null  int64
 8   ContentTime                             10237 non-null  float64
 9   ConversionType                          10237 non-null  object
 10  HighBit                                 10237 non-null  int64
 11  InstanceNumber                          10237 non-null  int64
 12  LargestImagePixelValue                  10237 non-null  int64
 13  Laterality                              9671 non-null   object
 14  Modality                                10237 non-null  object
 15  PatientBirthDate                        0 non-null      float64
 16  PatientID                               10237 non-null  object
 17  PatientName                             10237 non-null  object
 18  PatientOrientation                      10237 non-null  object
 19  PatientSex                              0 non-null      float64
 20  PhotometricInterpretation               10237 non-null  object
 21  PixelRepresentation                     10237 non-null  int64
 22  ReferringPhysicianName                  0 non-null      float64
 23  Rows                                    10237 non-null  int64
 24  SOPClassUID                             10237 non-null  object
 25  SOPInstanceUID                          10237 non-null  object
 26  SamplesPerPixel                         10237 non-null  int64
 27  SecondaryCaptureDeviceManufacturer      10237 non-null  object
 28  SecondaryCaptureDeviceManufacturerModelName  10237 non-null  object
 29  SeriesDescription                       9671 non-null   object
 30  SeriesInstanceUID                       10237 non-null  object
 31  SeriesNumber                            10237 non-null  int64
 32  SmallestImagePixelValue                 10237 non-null  int64
 33  SpecificCharacterSet                    10237 non-null  object
 34  StudyDate                               9671 non-null   float64
 35  StudyID                                 10237 non-null  object
 36  StudyInstanceUID                        10237 non-null  object
 37  StudyTime                               9671 non-null   float64
dtypes: float64(7), int64(12), object(19)
memory usage: 3.0+ MB
```

**Provides Data Summary:** Gives a summary of the data's structure, including the number of entries, column names, data types, and any missing values.

# Filtering Data

```
cropped_images = dicom_data[dicom_data.SeriesDescription == 'cropped images'].image_path
cropped_images.head()
```

```
0     CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.129308...
3     CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.381187...
6     CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.153339...
7     CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.178994...
10    CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.411833...
Name: image_path, dtype: object
```

- It filters the dicom_data DataFrame to extract information about images labeled with the SeriesDescription 'cropped images'.
- The variable cropped_images stores the paths to these cropped images.

# Path Transformation

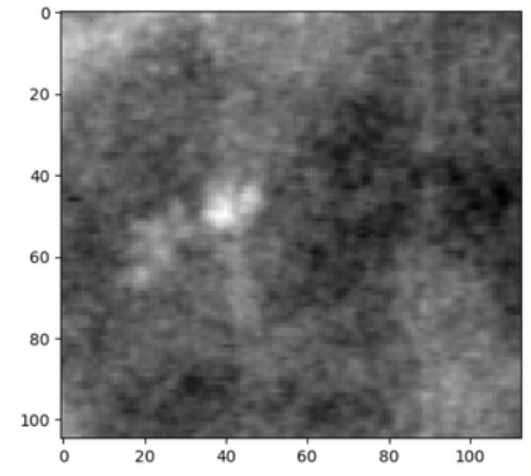
```
cropped_images = cropped_images.apply(lambda x: x.replace('CBIS-DDSM/jpeg', image_dir))
cropped_images.head()
```

```
0     D:\Semster 5\Data Science Tools\Project\archiv...
3     D:\Semster 5\Data Science Tools\Project\archiv...
6     D:\Semster 5\Data Science Tools\Project\archiv...
7     D:\Semster 5\Data Science Tools\Project\archiv...
10    D:\Semster 5\Data Science Tools\Project\archiv...
Name: image_path, dtype: object
```

- Replaces part of the image paths (specifically, 'CBIS-DDSM/jpeg') with the image_dir path previously defined.
- This step adjusts the paths to point to the actual location of the image files in the directory specified by image_dir.

# Displaying Images

```
for file in cropped_images[0:10]:
    cropped_images_show = PIL.Image.open(file)
    gray_img= cropped_images_show.convert("L")
    plt.imshow(gray_img, cmap='gray')
```




- Iterates through the first ten image paths in cropped_images.
- Opens each image using PIL (Python Imaging Library) and converts them to grayscale (convert("L")).
- Displays the grayscale versions of these images using Matplotlib (plt.imshow).

# Filtering Data for 'Full Mammogram Images'

```
full_mammogram_images = dicom_data[dicom_data.SeriesDescription == 'full mammogram images'].image_path
full_mammogram_images.head()
```

```
1     CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.248386...
2     CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.267213...
11    CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.210396...
12    CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.749566...
15    CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.987658...
Name: image_path, dtype: object
```

- It filters the dicom_data DataFrame to extract information about images labeled as 'full mammogram images'.
- The variable full_mammogram_images stores the paths to these full mammogram images.

# Path Transformation

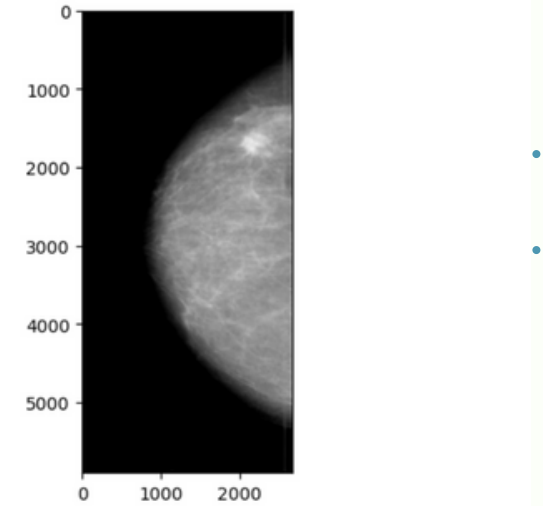
```
full_mammogram_images = full_mammogram_images.apply(lambda x: x.replace('CBIS-DDSM/jpeg', image_dir))
full_mammogram_images.head()
```

```
1     D:\Semster 5\Data Science Tools\Project\archiv...
2     D:\Semster 5\Data Science Tools\Project\archiv...
11    D:\Semster 5\Data Science Tools\Project\archiv...
12    D:\Semster 5\Data Science Tools\Project\archiv...
15    D:\Semster 5\Data Science Tools\Project\archiv...
Name: image_path, dtype: object
```

- Replaces part of the image paths (specifically, 'CBIS-DDSM/jpeg') with the image_dir path previously defined.
- This step adjusts the paths to point to the actual location of the image files in the directory specified by image_dir.

# Displaying Images

```
for file  in full_mammogram_images[0:10]:
    full_mammogram_images_show = PIL.Image.open(file)
    gray_img= full_mammogram_images_show.convert("L")
    plt.imshow(gray_img, cmap='gray')
```




- Replaces part of the image paths (specifically, 'CBIS-DDSM/jpeg') with the image_dir path previously defined.
- This step adjusts the paths to point to the actual location of the image files in the directory specified by image_dir.

# Filtering Data for 'ROI Mask Images'

```
ROI_mask_images = dicom_data[dicom_data.SeriesDescription == 'ROI mask images'].image_path
ROI_mask_images.head()
```

```
5     CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.153339...
8     CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.178994...
9     CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.411833...
14    CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.236373...
20    CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.357008...
Name: image_path, dtype: object
```

- Filters the dicom_data DataFrame to extract information about images labeled as 'ROI mask images'.
- The variable ROI_mask_images stores the paths to these ROI mask images.

# Path Transformation

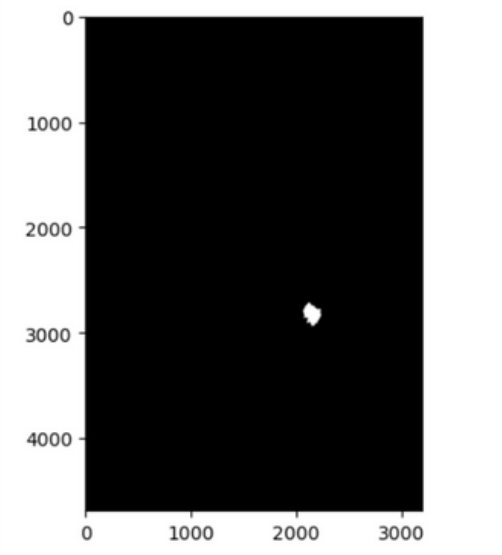
```
ROI_mask_images = ROI_mask_images.apply(lambda x: x.replace('CBIS-DDSM/jpeg', image_dir))
ROI_mask_images.head()
```

```
5     D:\Semster 5\Data Science Tools\Project\archiv...
8     D:\Semster 5\Data Science Tools\Project\archiv...
9     D:\Semster 5\Data Science Tools\Project\archiv...
14    D:\Semster 5\Data Science Tools\Project\archiv...
20    D:\Semster 5\Data Science Tools\Project\archiv...
Name: image_path, dtype: object
```

- Replaces part of the image paths (specifically, 'CBIS-DDSM/jpeg') with the image_dir path previously defined.
- This step adjusts the paths to point to the actual location of the image files in the directory specified by image_dir.

# Displaying Images

```
for file  in ROI_mask_images[0:10]:
    ROI_mask_images_show = PIL.Image.open(file)
    gray_img= ROI_mask_images_show.convert("L")
    plt.imshow(gray_img, cmap='gray')
```




- Iterates through the first ten image paths in ROI_mask_images.
- Opens each image using PIL and converts them to grayscale (convert("L")).
- Displays the grayscale versions of these images using Matplotlib (plt.imshow).

# Loading and Displaying DataFrames

```
calc_case_df = pd.read_csv(r"D:\Semster 5\Data Science Tools\Project\archive\csv\calc_case_description_train_set.csv")
calc_case_df.head(5)
```

| | patient_id | breast density | left or right breast | image view | abnormality id | abnormality type | calc type | calc distribution | assessment | pathology | subtlety | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | P_00005 | 3 | RIGHT | CC | 1 | calcification | AMORPHOUS | CLUSTERED | 3 | MALIGNANT | 3 | Training_P_0 |
| 1 | P_00005 | 3 | RIGHT | MLO | 1 | calcification | AMORPHOUS | CLUSTERED | 3 | MALIGNANT | 3 | Training_P_0 |
| 2 | P_00007 | 4 | LEFT | CC | 1 | calcification | PLEOMORPHIC | LINEAR | 4 | BENIGN | 4 | Training_P_( |
| 3 | P_00007 | 4 | LEFT | MLO | 1 | calcification | PLEOMORPHIC | LINEAR | 4 | BENIGN | 4 | Training_P_0 |
| 4 | P_00008 | 1 | LEFT | CC | 1 | calcification | NaN | REGIONAL | 2 | BENIGN_WITHOUT_CALLBACK | 3 | Training_P_( |

```
mass_case_df = pd.read_csv(r"D:\Semster 5\Data Science Tools\Project\archive\csv\mass_case_description_train_set.csv")
mass_case_df.head(5)
```

| | patient_id | breast_density | left or right breast | image view | abnormality id | abnormality type | mass shape | mass margins | assessment | pathology | subtlety | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | P_00001 | 3 | LEFT | CC | 1 | mass | IRREGULAR-ARCHITECTURAL_DISTORTION | SPICULATED | 4 | MALIGNANT | 4 | Tr |
| 1 | P_00001 | 3 | LEFT | MLO | 1 | mass | IRREGULAR-ARCHITECTURAL_DISTORTION | SPICULATED | 4 | MALIGNANT | 4 | Tra |
| 2 | P_00004 | 3 | LEFT | CC | 1 | mass | ARCHITECTURAL_DISTORTION | ILL_DEFINED | 4 | BENIGN | 3 | Tr |
| 3 | P_00004 | 3 | LEFT | MLO | 1 | mass | ARCHITECTURAL_DISTORTION | ILL_DEFINED | 4 | BENIGN | 3 | Tra |
| 4 | P_00004 | 3 | RIGHT | MLO | 1 | mass | OVAL | CIRCUMSCRIBED | 4 | BENIGN | 5 | Tra |

- Loads two CSV files, calc_case_description_train_set.csv and mass_case_description_train_set.csv, into DataFrames named calc_case_df and mass_case_df respectively.
- Displays the first five rows of each DataFrame using .head() to provide an initial overview of the data.

# Copying and Cleaning Data

```
dicom_cleaned_data = dicom_data.copy()
dicom_cleaned_data.head()
```

| | file_path | image_path | AccessionNumber | BitsAllocated | BitsStored | BodyPartExamined | Columr |
|---|---|---|---|---|---|---|---|
| 0 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.12930... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.129308... | NaN | 16 | 16 | BREAST | 3 |
| 1 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.24838... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.248386... | NaN | 16 | 16 | BREAST | 352 |
| 2 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.26721... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.267213... | NaN | 16 | 16 | BREAST | 154 |
| 3 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.38118... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.381187... | NaN | 16 | 16 | BREAST | 1 |
| 4 | CBIS-DDSM/dicom/1.3.6.1.4.1.9590.100.1.2.38118... | CBIS-DDSM/jpeg/1.3.6.1.4.1.9590.100.1.2.381187... | NaN | 8 | 8 | Left Breast | 310 |

5 rows × 38 columns

```
dicom_cleaned_data.drop(['PatientBirthDate','AccessionNumber','Columns','ContentDate','ContentTime','PatientSex','PatientBirthDat
                         'ReferringPhysicianName','Rows','SOPClassUID','SOPInstanceUID',
                         'StudyDate','StudyID','StudyInstanceUID','StudyTime','InstanceNumber','SeriesInst
```

- Creates a copy of the dicom_data DataFrame named dicom_cleaned_data.
- Removes specific columns (as indicated in the list passed to drop) from the copied DataFrame.

```
dicom_cleaned_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10237 entries, 0 to 10236
Data columns (total 21 columns):
 #   Column                                     Non-Null Count  Dtype
---  ------                                     --------------  -----
 0   file_path                                  10237 non-null  object
 1   image_path                                 10237 non-null  object
 2   BitsAllocated                              10237 non-null  int64
 3   BitsStored                                 10237 non-null  int64
 4   BodyPartExamined                           10237 non-null  object
 5   ConversionType                             10237 non-null  object
 6   HighBit                                    10237 non-null  int64
 7   LargestImagePixelValue                     10237 non-null  int64
 8   Laterality                                 9671 non-null   object
 9   Modality                                   10237 non-null  object
 10  PatientID                                  10237 non-null  object
 11  PatientName                                10237 non-null  object
 12  PatientOrientation                         10237 non-null  object
 13  PhotometricInterpretation                  10237 non-null  object
 14  PixelRepresentation                        10237 non-null  int64
 15  SamplesPerPixel                            10237 non-null  int64
 16  SecondaryCaptureDeviceManufacturer         10237 non-null  object
 17  SecondaryCaptureDeviceManufacturerModelName  10237 non-null  object
 18  SeriesDescription                          9671 non-null   object
 19  SmallestImagePixelValue                    10237 non-null  int64
 20  SpecificCharacterSet                       10237 non-null  object
dtypes: int64(7), object(14)
memory usage: 1.6+ MB
```

```
dicom_cleaned_data.isna().sum()
```

```
file_path                                      0
image_path                                     0
BitsAllocated                                  0
BitsStored                                     0
BodyPartExamined                               0
ConversionType                                 0
HighBit                                        0
LargestImagePixelValue                         0
Laterality                                   566
Modality                                       0
PatientID                                      0
PatientName                                    0
PatientOrientation                             0
PhotometricInterpretation                      0
PixelRepresentation                            0
SamplesPerPixel                                0
SecondaryCaptureDeviceManufacturer             0
SecondaryCaptureDeviceManufacturerModelName    0
SeriesDescription                            566
SmallestImagePixelValue                        0
SpecificCharacterSet                           0
dtype: int64
```

```
dicom_cleaned_data['SeriesDescription'].fillna(method = 'bfill', axis = 0, inplace=True)
```

```
dicom_cleaned_data['Laterality'].fillna(method = 'bfill', axis = 0, inplace=True)
```


```
dicom_cleaned_data.isna().sum()
```


```
file_path                                      0
image_path                                     0
BitsAllocated                                  0
BitsStored                                     0
BodyPartExamined                               0
ConversionType                                 0
HighBit                                        0
LargestImagePixelValue                         0
Laterality                                     0
Modality                                       0
PatientID                                      0
PatientName                                    0
PatientOrientation                             0
PhotometricInterpretation                      0
PixelRepresentation                            0
SamplesPerPixel                                0
SecondaryCaptureDeviceManufacturer             0
SecondaryCaptureDeviceManufacturerModelName    0
SeriesDescription                              0
SmallestImagePixelValue                        0
SpecificCharacterSet                           0
dtype: int64
```

- Provides information about the structure of the cleaned DataFrame using info() and checks for missing values using isna().sum().
- Fills missing values in 'SeriesDescription' and 'Laterality' columns with the values from the next row (bfill method).

# Data Cleaning and Transformation

- Creates copies of calc_case_df and mass_case_df DataFrames as Data_cleaning_1 and Data_cleaning_2 respectively.
- Renames specific columns in both DataFrames using rename(columns={...}).
- Checks for missing values using isna().sum() and handles missing values in certain columns using the bfill method to fill them with values from the next row.

```
Data_cleaning_1 = calc_case_df.copy()
Data_cleaning_1 = Data_cleaning_1.rename(columns={'calc type':'calc_type'})
Data_cleaning_1 = Data_cleaning_1.rename(columns={'calc distribution':'calc_distribution'})
Data_cleaning_1 = Data_cleaning_1.rename(columns={'image view':'image_view'})
Data_cleaning_1 = Data_cleaning_1.rename(columns={'left or right breast':'left_or_right_breast'})
Data_cleaning_1 = Data_cleaning_1.rename(columns={'breast density':'breast_density'})
Data_cleaning_1 = Data_cleaning_1.rename(columns={'abnormality type':'abnormality_type'})
Data_cleaning_1['pathology'] = Data_cleaning_1['pathology'].astype('category')
Data_cleaning_1['calc_type'] = Data_cleaning_1['calc_type'].astype('category')
Data_cleaning_1['calc_distribution'] = Data_cleaning_1['calc_distribution'].astype('category')
Data_cleaning_1['abnormality_type'] = Data_cleaning_1['abnormality_type'].astype('category')
Data_cleaning_1['image_view'] = Data_cleaning_1['image_view'].astype('category')
Data_cleaning_1['left_or_right_breast'] = Data_cleaning_1['left_or_right_breast'].astype('category')
Data_cleaning_1.isna().sum()
```

```
patient_id                0
breast_density            0
left_or_right_breast      0
image_view                0
abnormality id            0
abnormality_type          0
calc_type                20
calc_distribution       376
assessment                0
pathology                 0
subtlety                  0
image file path           0
cropped image file path   0
ROI mask file path        0
dtype: int64
```

```
Data_cleaning_1['calc_type'].fillna(method = 'bfill', axis = 0, inplace=True)
Data_cleaning_1['calc_distribution'].fillna(method = 'bfill', axis = 0, inplace=True)
Data_cleaning_1.isna().sum()
```

```
patient_id                0
breast_density            0
left_or_right_breast      0
image_view                0
abnormality id            0
abnormality_type          0
calc_type                 0
calc_distribution         0
assessment                0
pathology                 0
subtlety                  0
image file path           0
cropped image file path   0
ROI mask file path        0
dtype: int64
```

```
Data_cleaning_2 = mass_case_df.copy()
Data_cleaning_2 = Data_cleaning_2.rename(columns={'mass shape':'mass_shape'})
Data_cleaning_2 = Data_cleaning_2.rename(columns={'left or right breast':'left_or_right_breast'})
Data_cleaning_2 = Data_cleaning_2.rename(columns={'mass margins':'mass_margins'})
Data_cleaning_2 = Data_cleaning_2.rename(columns={'image view':'image_view'})
Data_cleaning_2 = Data_cleaning_2.rename(columns={'abnormality type':'abnormality_type'})
Data_cleaning_2['left_or_right_breast'] = Data_cleaning_2['left_or_right_breast'].astype('category')
Data_cleaning_2['image_view'] = Data_cleaning_2['image_view'].astype('category')
Data_cleaning_2['mass_margins'] = Data_cleaning_2['mass_margins'].astype('category')
Data_cleaning_2['mass_shape'] = Data_cleaning_2['mass_shape'].astype('category')
Data_cleaning_2['abnormality_type'] = Data_cleaning_2['abnormality_type'].astype('category')
Data_cleaning_2['pathology'] = Data_cleaning_2['pathology'].astype('category')
Data_cleaning_2.isna().sum()
```

```
patient_id                0
breast_density            0
left_or_right_breast      0
image_view                0
abnormality id            0
abnormality_type          0
mass_shape                4
mass_margins             43
assessment                0
pathology                 0
subtlety                  0
image file path           0
cropped image file path   0
ROI mask file path        0
dtype: int64
```

```
Data_cleaning_2['mass_shape'].fillna(method = 'bfill', axis = 0, inplace=True)
Data_cleaning_2['mass_margins'].fillna(method = 'bfill', axis = 0, inplace=True)
Data_cleaning_2.isna().sum()
```

```
patient_id                0
breast_density            0
left_or_right_breast      0
image_view                0
abnormality id            0
abnormality_type          0
mass_shape                0
mass_margins              0
assessment                0
pathology                 0
subtlety                  0
image file path           0
cropped image file path   0
ROI mask file path        0
dtype: int64
```

# Loading Additional DataFrames

```
mass_train = pd.read_csv(r"D:\Semster 5\Data Science Tools\Project\archive\csv\mass_case_description_train_set.csv")
mass_test = pd.read_csv(r"D:\Semster 5\Data Science Tools\Project\archive\csv\mass_case_description_test_set.csv")
mass_train.head()
```

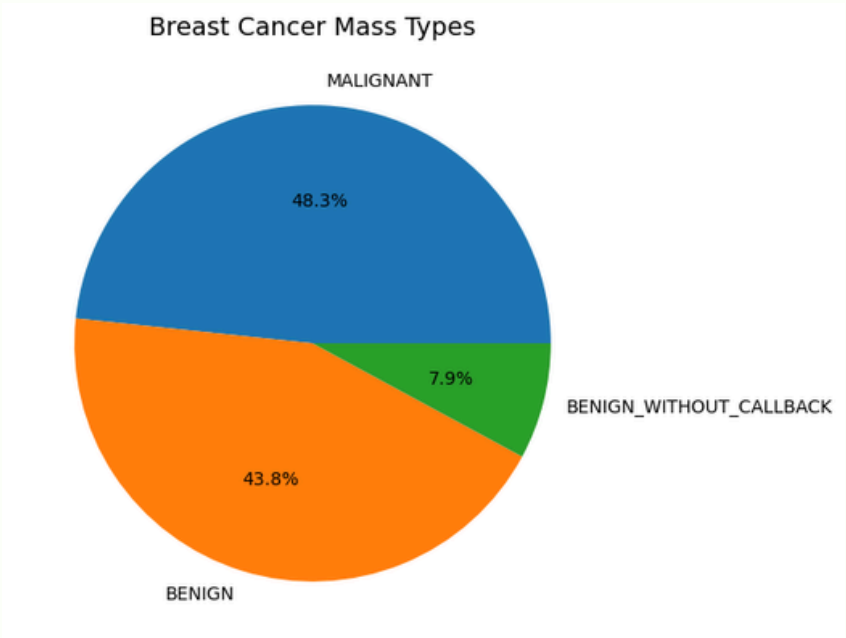
| | patient_id | breast_density | left or right breast | image view | abnormality id | abnormality type | mass shape | mass margins | assessment | pathology | subtlety | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | P_00001 | 3 | LEFT | CC | 1 | mass | IRREGULAR-ARCHITECTURAL_DISTORTION | SPICULATED | 4 | MALIGNANT | 4 | Tri |
| 1 | P_00001 | 3 | LEFT | MLO | 1 | mass | IRREGULAR-ARCHITECTURAL_DISTORTION | SPICULATED | 4 | MALIGNANT | 4 | Tra |
| 2 | P_00004 | 3 | LEFT | CC | 1 | mass | ARCHITECTURAL_DISTORTION | ILL_DEFINED | 4 | BENIGN | 3 | Tri |
| 3 | P_00004 | 3 | LEFT | MLO | 1 | mass | ARCHITECTURAL_DISTORTION | ILL_DEFINED | 4 | BENIGN | 3 | Tra |
| 4 | P_00004 | 3 | RIGHT | MLO | 1 | mass | OVAL | CIRCUMSCRIBED | 4 | BENIGN | 5 | Tra |

- Reads two additional CSV files, mass_case_description_train_set.csv and mass_case_description_test_set.csv, into DataFrames named mass_train and mass_test respectively.
- Displays the first few rows of the mass_train DataFrame using .head().

# Pie Chart: Pathology Distributions

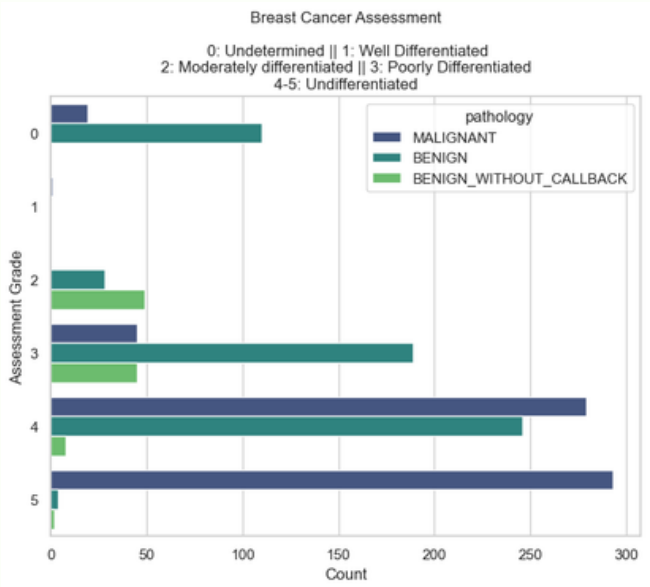
```
# pathology distributions
value = mass_train['pathology'].value_counts()
plt.figure(figsize=(8,6))

plt.pie(value, labels=value.index, autopct='%1.1f%%')
plt.title('Breast Cancer Mass Types', fontsize=14)
plt.show()
```

- Displays a pie chart showing the distribution of different types of breast cancer pathologies in the dataset.



# Count Plot: Breast Assessment Types

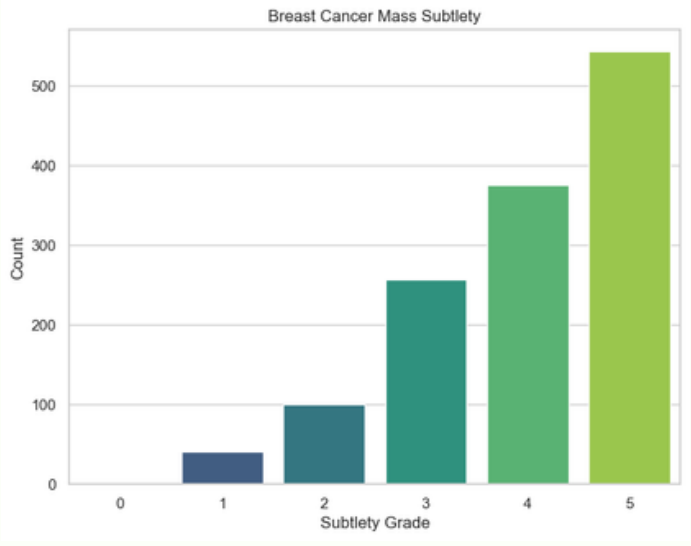
```
# examine breast assessment types
sns.set(style="whitegrid")

plt.figure(figsize=(8,6))

sns.countplot(data=mass_train, y='assessment', hue='pathology', palette='viridis')
plt.title('Breast Cancer Assessment\n\n 0: Undetermined || 1: Well Differentiated\n2: Moderately differentiated || 3: Poorly Diff
          fontsize=12)
plt.xlabel('Count')
plt.ylabel('Assessment Grade')
plt.show()
```




- Utilizes a count plot to visualize the distribution of different assessment grades for breast cancer, categorized by pathology type.

# Count Plot: Cancer Subtlety

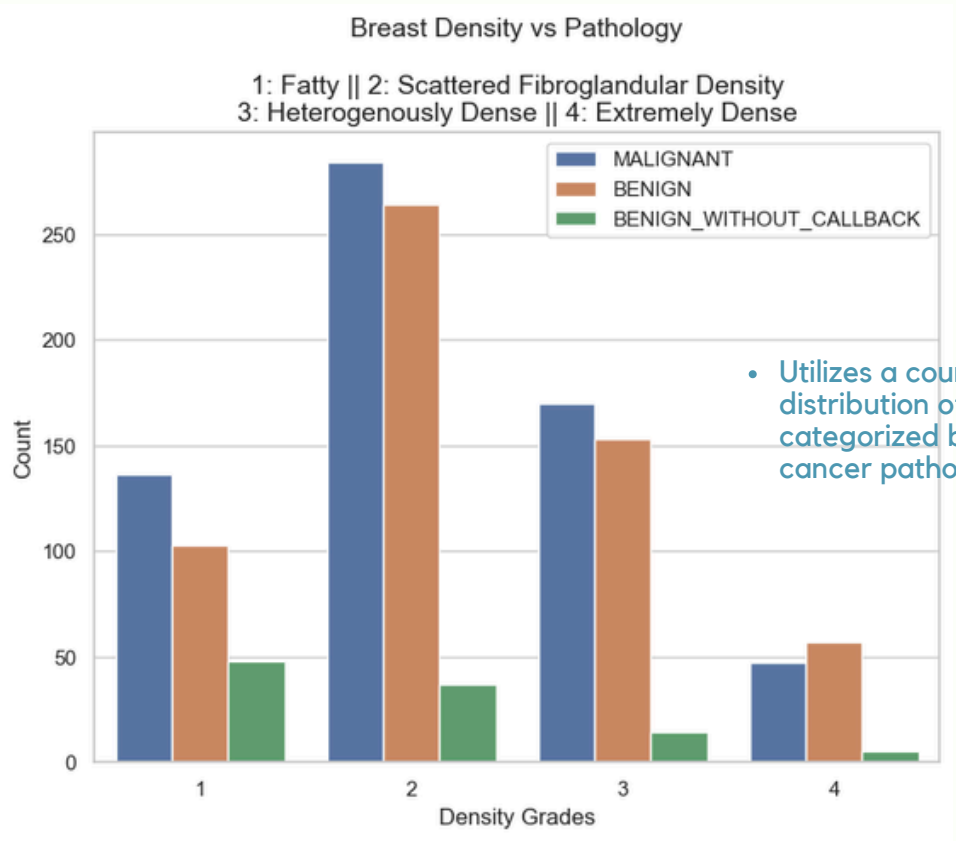
```
# examine cancer subtlety
plt.figure(figsize=(8,6))
sns.countplot(data=mass_train, x='subtlety', palette='viridis')
plt.title('Breast Cancer Mass Subtlety', fontsize=12)
plt.xlabel('Subtlety Grade')
plt.ylabel('Count')
plt.show()
```




- Displays a count plot showing the distribution of subtlety grades related to breast cancer masses.

# Count Plot: Breast Density against Pathology

```
# breast density against pathology
sns.set(style="whitegrid")

plt.figure(figsize=(8,6))

sns.countplot(data=mass_train, x='breast_density', hue='pathology')
plt.title('Breast Density vs Pathology\n\n1: Fatty || 2: Scattered Fibroglandular Density\n3: Heterogenously Dense || 4: Extreme
          fontsize=14)
plt.xlabel('Density Grades')
plt.ylabel('Count')
plt.legend()
plt.show()
```




- Utilizes a count plot to visualize the distribution of breast density grades categorized by different types of breast cancer pathology.

# 2. Image Classification:

## CNN vs SVM vs DC vs Fine-Tuned Model

1 Data Preparation:

- Image paths and labels are organized.
- Label encoding and data splitting into training and testing sets are performed.

2 Models:

- CNN
- SVM
- Dummy Classifier
- Fine-tuning a pre-trained Convolutional Neural Network (CNN)

3 Model Training and Evaluation:

- Models are trained and evaluated on the prepared data.
- Evaluation metrics such as accuracy, classification report, and confusion matrix are employed.

4 Model Comparison:

- Bar plot compares accuracy of models.
- Precision, recall, and F1-score values are compared.

5 Predictions

- Predictions are made for 4 models
- actual and predicted values are compared

# Data Preparation

```
# Prepare the data for CNN
image_paths = cropped_images.to_list() + full_mammogram_images.to_list() + ROI_mask_images.to_list()
labels =
['cropped images'] * len(cropped_images) +
['full mammogram images'] * len(full_mammogram_images) +
['ROI mask images'] * len(ROI_mask_images)
```

The code organizes data for a CNN by combining image paths from three sources into a single list (image_paths). Labels are assigned according to image categories, resulting in a list of labels (labels). This structured data is designed for CNN training, pairing each image path with its corresponding category label for model learning.

```
# Convert labels to numeric values
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
print(label_encoder)
print(labels_encoded)
```

```
LabelEncoder()
[1 1 1 ... 0 0 0]
```

The code uses a LabelEncoder to convert categorical image labels into numeric values, facilitating preparation for machine learning models. It prints information about the LabelEncoder object and displays the resulting numeric representations of the labels.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(image_paths, labels_encoded, test_size=0.2, random_state=42)
```

Splits data into training (X_train, y_train) and testing (X_test, y_test) sets, allocates 20% of the data for testing (test_size=0.2) and random_state=42 ensures reproducibility.

# Load and Preprocess Images for CNN

```
# Load and preprocess images for CNN
def load_and_preprocess_image(path):
    img = cv2.imread(path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (224, 224))
    img = img.astype('float32') / 255.0
    return img

X_train_cnn = np.array([load_and_preprocess_image(path) for path in X_train])
X_test_cnn = np.array([load_and_preprocess_image(path) for path in X_test])
print(X_train_cnn)
print(X_test_cnn)
```

Defines a function load_and_preprocess_image(path) to read, convert to grayscale, resize to (224, 224), and normalize images, X_train_cnn and X_test_cnn are created by applying the function to each image path in the training and testing sets, the resulting arrays are printed for verification.

# Grayscale Image Preprocessing output for CNN

The output presents processed grayscale images intended for a Convolutional Neural Network (CNN). Images have been standardized by converting them to a size of (224, 224) and normalizing pixel values between 0 and 1.

```
[[[0.36862746 0.38039216 0.41960785 ... 0.45490196 0.42745098 0.4509804 ]
  [0.4        0.38039216 0.41568628 ... 0.48235294 0.46666667 0.47058824]
  [0.40784314 0.39607844 0.3882353  ... 0.5176471  0.49803922 0.49019608]
  ...
  [0.44313726 0.40784314 0.40784314 ... 0.29411766 0.32156864 0.31764707]
  [0.4117647  0.40392157 0.38431373 ... 0.29411766 0.27450982 0.2627451 ]
  [0.4392157  0.40392157 0.3764706  ... 0.29411766 0.28627452 0.27058825]]

 [[0.8666667  0.84705883 0.7882353  ... 1.         1.         1.        ]
  [0.6509804  0.59607846 0.04705882 ... 0.9098039  1.         1.        ]
  [0.53333336 0.14509805 0.         ... 0.75686276 0.99607843 1.        ]
  ...
  [0.88235295 0.8627451  0.81960785 ... 0.36862746 0.32941177 1.        ]
  [0.89411765 0.8784314  0.85882354 ... 0.79607844 1.         1.        ]
  [0.9019608  0.8901961  0.88235295 ... 0.4509804  1.         1.        ]]

 [[0.         0.         0.         ... 0.         0.         0.        ]
  [0.         0.         0.         ... 0.         0.         0.        ]
  [0.         0.         0.         ... 0.         0.         0.        ]
  ...
  [0.         0.         0.         ... 0.         0.         0.        ]
  [0.         0.         0.         ... 0.         0.         0.        ]
  [0.         0.         0.         ... 0.         0.         0.        ]]

 ...

 [[0.8392157  0.8352941  0.8117647  ... 1.         1.         1.        ]
  [0.7921569  0.79607844 0.78431374 ... 1.         1.         1.        ]
  [0.7137255  0.69803923 0.6901961  ... 0.9137255  0.9882353  1.        ]
  ...
  [0.7058824  0.69411767 0.68235296 ... 0.8980392  0.8980392  0.9137255 ]
  [0.8117647  0.8039216  0.80784315 ... 0.90588236 0.91764706 0.90588236]
  [0.8509804  0.8509804  0.84705883 ... 0.91764706 0.8        1.        ]]

 [[0.9019608  0.88235295 0.88235295 ... 0.01176471 0.00392157 0.00784314]
  [0.8784314  0.8784314  0.88235295 ... 0.03529412 0.01568628 0.01960784]
  [0.88235295 0.8745098  0.88235295 ... 0.00784314 0.00392157 0.00392157]
  ...
  [0.7137255  0.69411767 0.6901961  ... 0.         0.         0.        ]
  [0.69803923 0.6862745  0.69411767 ... 0.         0.         0.        ]
  [0.69411767 0.7058824  0.6901961  ... 0.         0.         0.        ]]
```



```
 [0.4        0.38431373 0.36078432 ... 0.8117647  0.83137256 0.8117647 ]
 [0.4        0.38431373 0.36862746 ... 0.8156863  0.8156863  0.8039216 ]
 [0.3647059  0.36862746 0.38039216 ... 0.8039216  0.8117647  0.8156863 ]]

[[0.41568628 0.47058824 0.44313726 ... 0.49803922 0.4745098  0.47058824]
 [0.38431373 0.4509804  0.41568628 ... 0.48235294 0.47058824 0.45882353]
 [0.36862746 0.42745098 0.4        ... 0.47058824 0.4862745  0.46666667]
 ...
 [0.25490198 0.2901961  0.30588236 ... 0.3764706  0.40784314 0.4117647 ]
 [0.21568628 0.2509804  0.31764707 ... 0.38039216 0.39607844 0.39215687]
 [0.23137255 0.24705882 0.25882354 ... 0.3764706  0.39215687 0.39607844]]]
```

# One-Hot Encoding Labels:

```
# Convert labels to one-hot encoded vectors
y_train_cnn = to_categorical(y_train)
y_test_cnn = to_categorical(y_test)
print(y_train_cnn)
print(y_test_cnn)
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 ...
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
[[0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]
 ...
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]]
```

The code employs the to_categorical function to convert categorical labels, represented by y_train and y_test, into one-hot encoded vectors. This transformation results in the creation of y_train_cnn and y_test_cnn, which now represent the labels in a binary matrix form. In this representation, each label is converted into a vector where all elements are zero, except for the position corresponding to the label, which is set to one. The output arrays are then printed to confirm the successful one-hot encoding.

# CNN Model Construction

```
# Build the CNN model
model_cnn = Sequential()
model_cnn.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(224, 224, 1)))
model_cnn.add(MaxPooling2D(pool_size=(2, 2)))
model_cnn.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model_cnn.add(MaxPooling2D(pool_size=(2, 2)))
model_cnn.add(Flatten())
model_cnn.add(Dense(128, activation='relu'))
model_cnn.add(Dense(3, activation='softmax'))

model_cnn.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

The code builds a Convolutional Neural Network (CNN) using the Keras Sequential model. The architecture consists of two convolutional layers with max-pooling, followed by flattening and two dense layers. The model is compiled with categorical crossentropy loss, Adam optimizer, and accuracy metric. It is designed for classifying images into three categories ('cropped images', 'full mammogram images', 'ROI mask images').

# Training the CNN Model:

```
# Train the CNN model
model_cnn.fit(X_train_cnn, y_train_cnn, batch_size=32, epochs=7, validation_data=(X_test_cnn, y_test_cnn))
```

```
Epoch 1/7
242/242 [==============================] - 334s 1s/step - loss: 0.0475 - accuracy: 0.9847 - val_loss: 0.0042 - val_accuracy: 0.
9984
Epoch 2/7
242/242 [==============================] - 322s 1s/step - loss: 0.0049 - accuracy: 0.9986 - val_loss: 0.0060 - val_accuracy: 0.
9969
Epoch 3/7
242/242 [==============================] - 323s 1s/step - loss: 0.0180 - accuracy: 0.9966 - val_loss: 0.0037 - val_accuracy: 0.
9990
Epoch 4/7
242/242 [==============================] - 313s 1s/step - loss: 0.0051 - accuracy: 0.9986 - val_loss: 0.0100 - val_accuracy: 0.
9974
Epoch 5/7
242/242 [==============================] - 309s 1s/step - loss: 2.0355e-04 - accuracy: 1.0000 - val_loss: 0.0101 - val_accurac
y: 0.9979
Epoch 6/7
242/242 [==============================] - 249s 1s/step - loss: 6.6454e-05 - accuracy: 1.0000 - val_loss: 0.0107 - val_accurac
y: 0.9979
Epoch 7/7
242/242 [==============================] - 168s 693ms/step - loss: 4.1230e-05 - accuracy: 1.0000 - val_loss: 0.0116 - val_accur
acy: 0.9974
```

The CNN model (model_cnn) is trained using the fit method with a batch size of 32 for 7 epochs. Training is performed on the training data (X_train_cnn and y_train_cnn), and the model's performance is validated using the testing data (X_test_cnn and y_test_cnn).

# CNN Model Evaluation on Testing Data

```
# Evaluate the CNN model
y_pred_cnn = model_cnn.predict(X_test_cnn)
y_pred_cnn = np.argmax(y_pred_cnn, axis=1)
accuracy_cnn = accuracy_score(y_test, y_pred_cnn)
classification_report_cnn = classification_report(y_test, y_pred_cnn)
classification_report_cnn_1 = classification_report(y_test, y_pred_cnn, output_dict=True)

print('CNN Accuracy:', accuracy_cnn)
print('CNN Classification Report:')
print(classification_report_cnn)
print('CNN Classification Report Dictionary:')
print(classification_report_cnn_1)
```

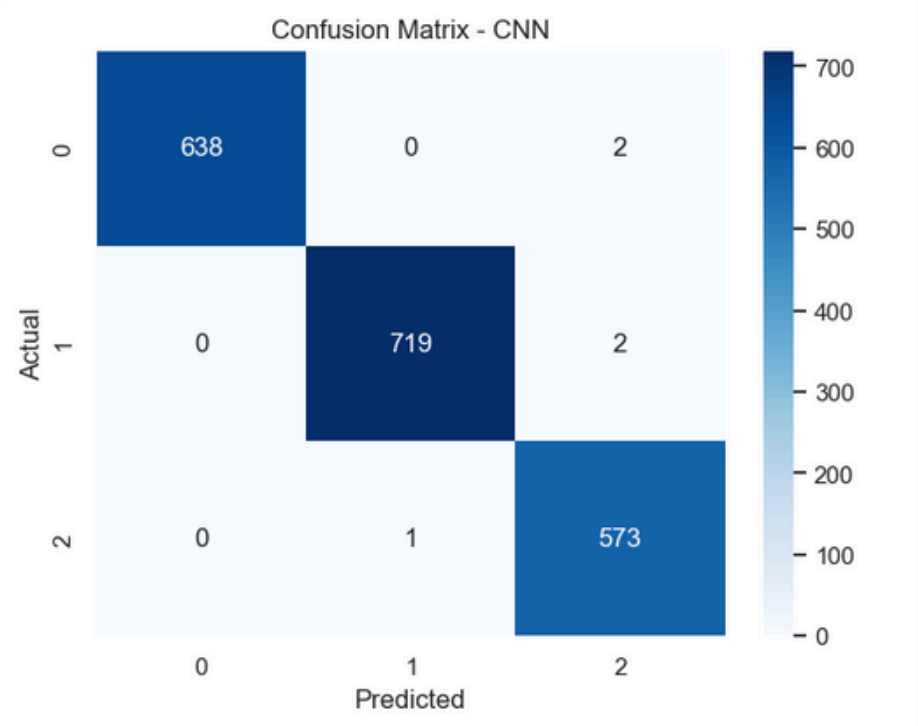
```
61/61 [==============================] - 13s 207ms/step
CNN Accuracy: 0.9974160206718347
CNN Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       640
           1       1.00      1.00      1.00       721
           2       0.99      1.00      1.00       574

    accuracy                           1.00      1935
   macro avg       1.00      1.00      1.00      1935
weighted avg       1.00      1.00      1.00      1935

CNN Classification Report Dictionary:
{'0': {'precision': 1.0, 'recall': 0.996875, 'f1-score': 0.9984350547730829, 'support': 640}, '1': {'precision': 0.998611111111
1111, 'recall': 0.9972260748959778, 'f1-score': 0.9979181124219292, 'support': 721}, '2': {'precision': 0.9930675909878682, 're
call': 0.9982578397212544, 'f1-score': 0.9956559513466551, 'support': 574}, 'accuracy': 0.9974160206718347, 'macro avg': {'prec
ision': 0.997226234032993, 'recall': 0.9974529715390773, 'f1-score': 0.9973363728472223, 'support': 1935}, 'weighted avg': {'pr
ecision': 0.9974260508207481, 'recall': 0.9974160206718347, 'f1-score': 0.9974180414387411, 'support': 1935}}
```

The code evaluates the performance of the CNN model on testing data. It computes predictions, calculates accuracy using the accuracy_score function, and generates a detailed classification report with metrics such as precision, recall, and F1-score. The results are printed and stored in both regular and dictionary formats for further analysis.

# Confusion Matrix for CNN Model:

```
# Confusion Matrix for CNN
conf_matrix_cnn = confusion_matrix(y_test, y_pred_cnn)
sns.heatmap(conf_matrix_cnn, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - CNN')
plt.show()
```

The code generates a confusion matrix for the CNN model, visually presenting key performance metrics, such as true positives, true negatives, false positives, and false negatives. The accompanying heatmap uses a blue color map, annotations, and clear axis labels for improved interpretation. The title 'Confusion Matrix - CNN' summarizes the focus of the visualization.



# Preparing Data for SVM:

```
# Prepare the data for SVM
X_train_svm = np.array([load_and_preprocess_image(path).flatten() for path in X_train])
X_test_svm = np.array([load_and_preprocess_image(path).flatten() for path in X_test])
print(X_train_svm)
print(X_test_svm)
```

```
[[0.36862746 0.38039216 0.41960785 ... 0.29411766 0.28627452 0.27058825]
 [0.8666667  0.84705883 0.7882353  ... 0.4509804  1.         1.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.8392157  0.8352941  0.8117647  ... 0.91764706 0.8        1.        ]
 [0.9019608  0.88235295 0.88235295 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
[[1.         1.         1.         ... 1.         1.         1.        ]
 [0.39607844 0.38431373 0.39215687 ... 0.08235294 0.09803922 0.08627451]
 [0.10588235 0.06666667 0.07450981 ... 0.27058825 0.25882354 0.23921569]
 ...
 [0.5137255  0.5137255  0.50980395 ... 0.25490198 0.27058825 0.27450982]
 [0.03529412 0.02352941 0.01960784 ... 0.8039216  0.8117647  0.8156863 ]
 [0.41568628 0.47058824 0.44313726 ... 0.3764706  0.39215687 0.39607844]]
```

The code prepares data for Support Vector Machine (SVM) by flattening and converting image data into numpy arrays for both training and testing sets. The load_and_preprocess_image function is utilized to preprocess the images before flattening, ensuring compatibility with the SVM model.

# Building and Training SVM Model

```
# Build and train the SVM model
model_svm = SVC(kernel='linear')
model_svm.fit(X_train_svm, y_train)
```

```
SVC(kernel='linear')
```

The code constructs an SVM model with a linear kernel and proceeds to train the model using the fit method. The training involves the preprocessed and flattened image data from the training set (X_train_svm) and their corresponding labels (y_train).

# SVM Model Prediction and Evaluation

```
# Predict using the SVM model
y_pred_svm = model_svm.predict(X_test_svm)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
classification_report_svm = classification_report(y_test, y_pred_svm)
classification_report_svm_1 = classification_report(y_test, y_pred_svm, output_dict=True)

print('SVM Accuracy:', accuracy_svm)
print('SVM Classification Report:')
print(classification_report_svm)
print('SVM Classification Report Dictionary:')
print(classification_report_svm_1)
```

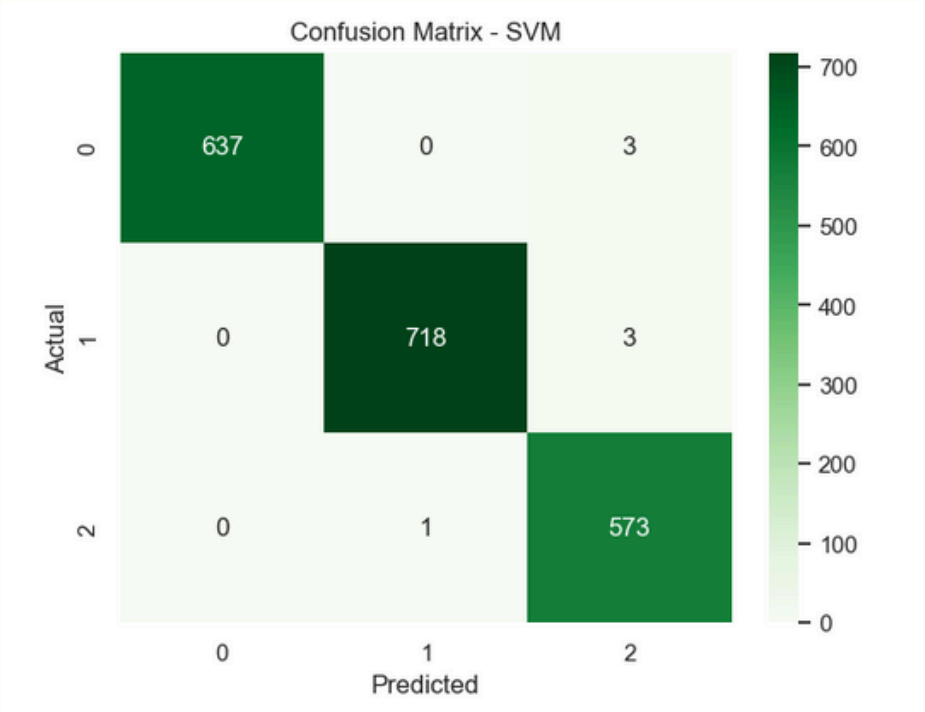
```
SVM Accuracy: 0.9963824289405685
SVM Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       640
           1       1.00      1.00      1.00       721
           2       0.99      1.00      0.99       574

    accuracy                           1.00      1935
   macro avg       1.00      1.00      1.00      1935
weighted avg       1.00      1.00      1.00      1935

SVM Classification Report Dictionary:
{'0': {'precision': 1.0, 'recall': 0.9953125, 'f1-score': 0.9976507439310885, 'support': 640}, '1': {'precision': 0.9986091794158554, 'recall': 0.9958391123439667, 'f1-score': 0.9972222222222222, 'support': 721}, '2': {'precision': 0.9896373056994818, 'recall': 0.9982578397212544, 'f1-score': 0.9939288811795316, 'support': 574}, 'accuracy': 0.9963824289405685, 'macro avg': {'precision': 0.9960821617051124, 'recall': 0.9964698173550737, 'f1-score': 0.9962672824442808, 'support': 1935}, 'weighted avg': {'precision': 0.9964077683877696, 'recall': 0.9963824289405685, 'f1-score': 0.9963870160905272, 'support': 1935}}
```

The code employs the trained Support Vector Machine (SVM) model to make predictions on the test set (X_test_svm). Subsequently, it calculates the accuracy and generates a classification report, which includes precision, recall, and F1-score metrics. The results are printed, showcasing the SVM model's performance on the test data.

# Confusion Matrix for SVM Model:

```
# Confusion Matrix for SVM
conf_matrix_svm = confusion_matrix(y_test, y_pred_svm)
sns.heatmap(conf_matrix_svm, annot=True, cmap='Greens', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - SVM')
plt.show()
```

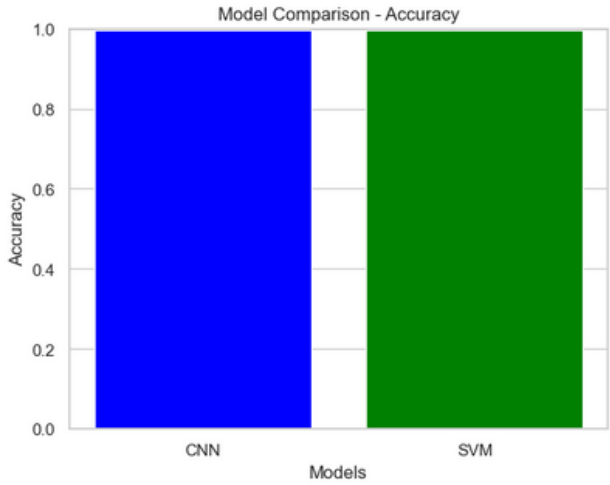
he code generates a confusion matrix for the Support Vector Machine (SVM) model's predictions on the test set (X_test_svm). It utilizes Seaborn's heatmap to visually represent the true positives, true negatives, false positives, and false negatives. The heatmap is annotated with the count values and is presented with a green color map, clear axis labels, and the title "Confusion Matrix - SVM."



# Model Comparison

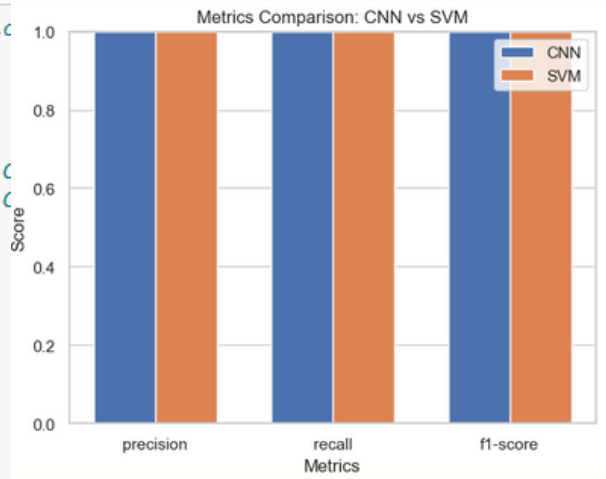
## CNN vs SVM

```
# Bar plot for model comparison
models = ['CNN', 'SVM']
accuracies = [accuracy_cnn, accuracy_svm]

plt.bar(models, accuracies, color=['blue', 'green'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Comparison - Accuracy')
plt.ylim(0, 1)  # Set y-axis limit to 0-1 for accuracy range
plt.show()
```




The code creates a bar plot to compare the accuracies of the Convolutional Neural Network (CNN) and Support Vector Machine (SVM) models. The plot uses 'blue' for CNN and 'green' for SVM,
with model names on the x-axis and accuracy values on the y-axis. The title is "Model Comparison - Accuracy," and the y-axis limit is set to 0-1 for the accuracy range

```
# Extract precision, recall, and F1-score values from classific
def extract_scores(report):
    scores = re.findall(r"\d+\.\d+", report)
    return [float(score) for score in scores]

cnn_scores = extract_scores(classification_report_cnn)[:3]  # C
svm_scores = extract_scores(classification_report_svm)[:3]  # C

metrics = ['precision', 'recall', 'f1-score']
bar_width = 0.35
index = np.arange(len(metrics))

plt.bar(index, cnn_scores, bar_width, label='CNN')
plt.bar(index + bar_width, svm_scores, bar_width, label='SVM')

plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Metrics Comparison: CNN vs SVM')
plt.xticks(index + bar_width / 2, metrics)
plt.legend()
plt.ylim([0, 1])
plt.show()
```




This code produces a bar plot comparing precision, recall, and F1-score values between the Convolutional Neural Network (CNN) and Support Vector Machine (SVM) models. The plot has a bar width of 0.35, displaying separate bars for each metric and model. Metrics are on the x-axis, and score values (ranging from 0 to 1) are on the y-axis. The title is "Metrics Comparison: CNN vs SVM."

# Dummy Classifier:

```
# Model Selection: Dummy Classifier
# Build and train the Dummy Classifier model
model_dummy = DummyClassifier(strategy='most_frequent')
model_dummy.fit(X_train_svm, y_train)
```

```
DummyClassifier(strategy='most_frequent')
```

This code segment introduces the implementation of a Dummy Classifier as part of model selection. The Dummy Classifier is built and trained with a 'most_frequent' strategy, predicting the most frequent class in the training data. This classifier is utilized as a baseline reference for comparison with more sophisticated models.

# Dummy Classifier model Prediction:

```
# Predict using the Dummy Classifier model
y_pred_dummy = model_dummy.predict(X_test_svm)
accuracy_dummy = accuracy_score(y_test, y_pred_dummy)
classification_report_dummy = classification_report(y_test, y_pred_dummy)
classification_report_dummy_1 = classification_report(y_test, y_pred_dummy, output_dict=True)

print('Dummy Classifier Accuracy:', accuracy_dummy)
print('Dummy Classifier Classification Report:')
print(classification_report_dummy)
print('Dummy Classifier Classification Report Dictionary:')
print(classification_report_dummy_1)
```

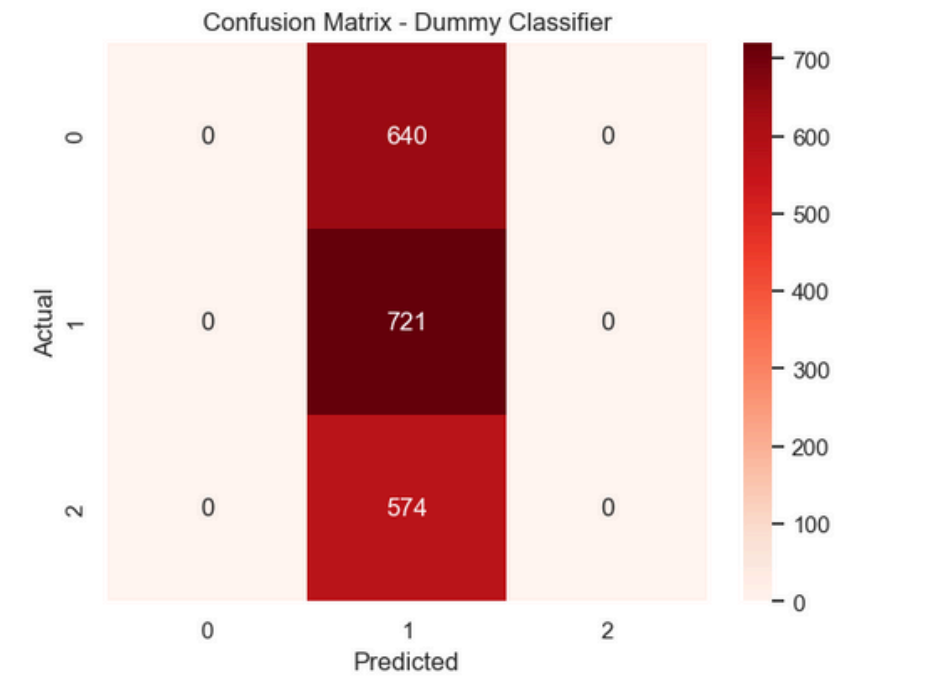
```
Dummy Classifier Accuracy: 0.37260981912144703
Dummy Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       640
           1       0.37      1.00      0.54       721
           2       0.00      0.00      0.00       574

    accuracy                           0.37      1935
   macro avg       0.12      0.33      0.18      1935
weighted avg       0.14      0.37      0.20      1935

Dummy Classifier Classification Report Dictionary:
{'0': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 640}, '1': {'precision': 0.37260981912144703, 'recall': 1.
0, 'f1-score': 0.5429216867469879, 'support': 721}, '2': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 574}, 'a
ccuracy': 0.37260981912144703, 'macro avg': {'precision': 0.12420327304048234, 'recall': 0.3333333333333333, 'f1-score': 0.1809
7389558232932, 'support': 1935}, 'weighted avg': {'precision': 0.13883807730571748, 'recall': 0.37260981912144703, 'f1-score':
0.2022979514959061, 'support': 1935}}
```

This code segment focuses on predicting using the Dummy Classifier model. It calculates the accuracy of the Dummy Classifier on the test data and generates a detailed classification report with precision, recall, and F1-score values, offering a comprehensive assessment of the Dummy Classifier's performance.

# Confusion Matrix for DC Model:

```
# Confusion Matrix for DCM
conf_matrix_DCM = confusion_matrix(y_test, y_pred_dummy)
sns.heatmap(conf_matrix_DCM, annot=True, cmap='Reds', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Dummy Classifier')
plt.show()
```

This code generates and presents a confusion matrix for the Dummy Classifier, employing a heatmap with annotated values and a 'Reds' color map. The matrix visually represents the true positives, true negatives, false positives, and false negatives, offering insights into the performance of the Dummy Classifier. The title of the visualization is "Confusion Matrix - Dummy Classifier (DCM)."

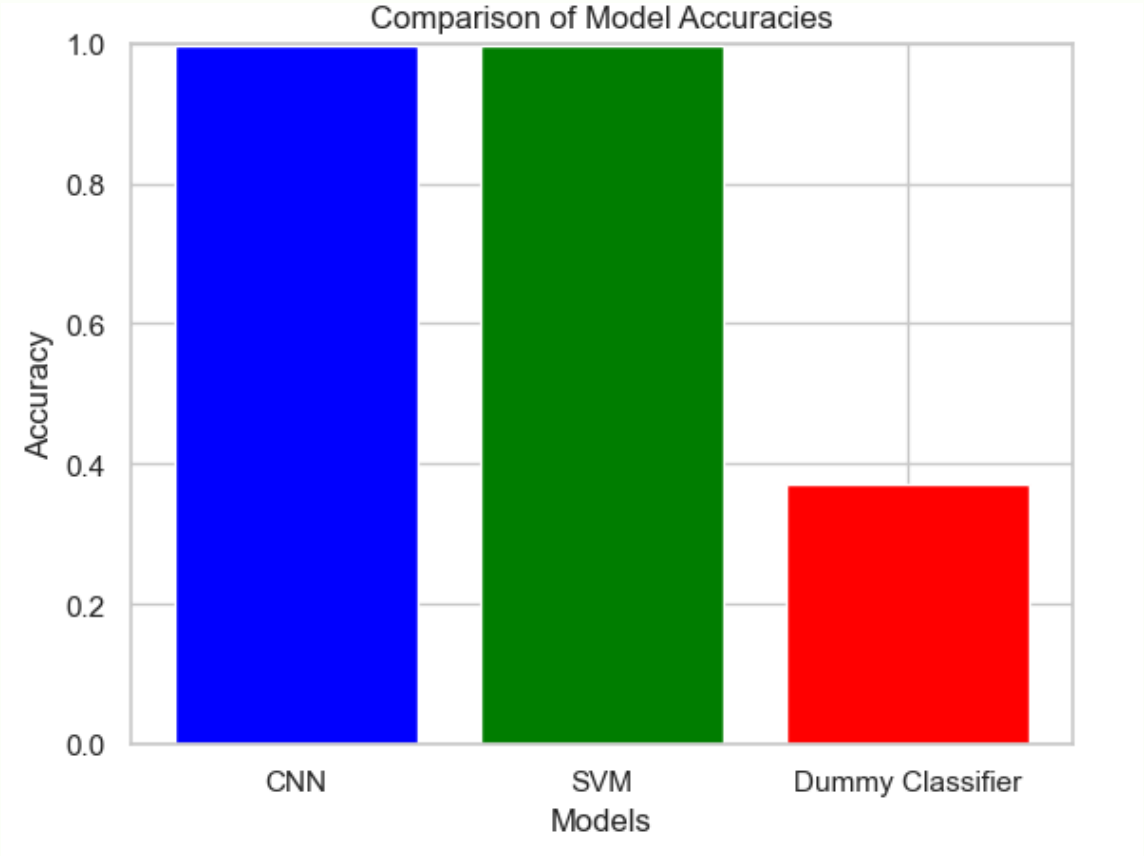


# Model Comparison

## CNN vs SVM vs DC

```
# Graph to compare accuracy of CNN, SVM, and Dummy Classifier
models = ['CNN', 'SVM', 'Dummy Classifier']
accuracies = [accuracy_cnn, accuracy_svm, accuracy_dummy]
colors = ['blue', 'green', 'red']

plt.bar(models, accuracies, color=colors)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.ylim([0, 1])
plt.show()
```

This code creates a bar plot to compare the accuracies of three models: CNN, SVM, and the Dummy Classifier. Each model is assigned a distinct color (blue for CNN, green for SVM, and red for the Dummy Classifier). The y-axis represents accuracy values, and the title of the plot is "Comparison of Model Accuracies."

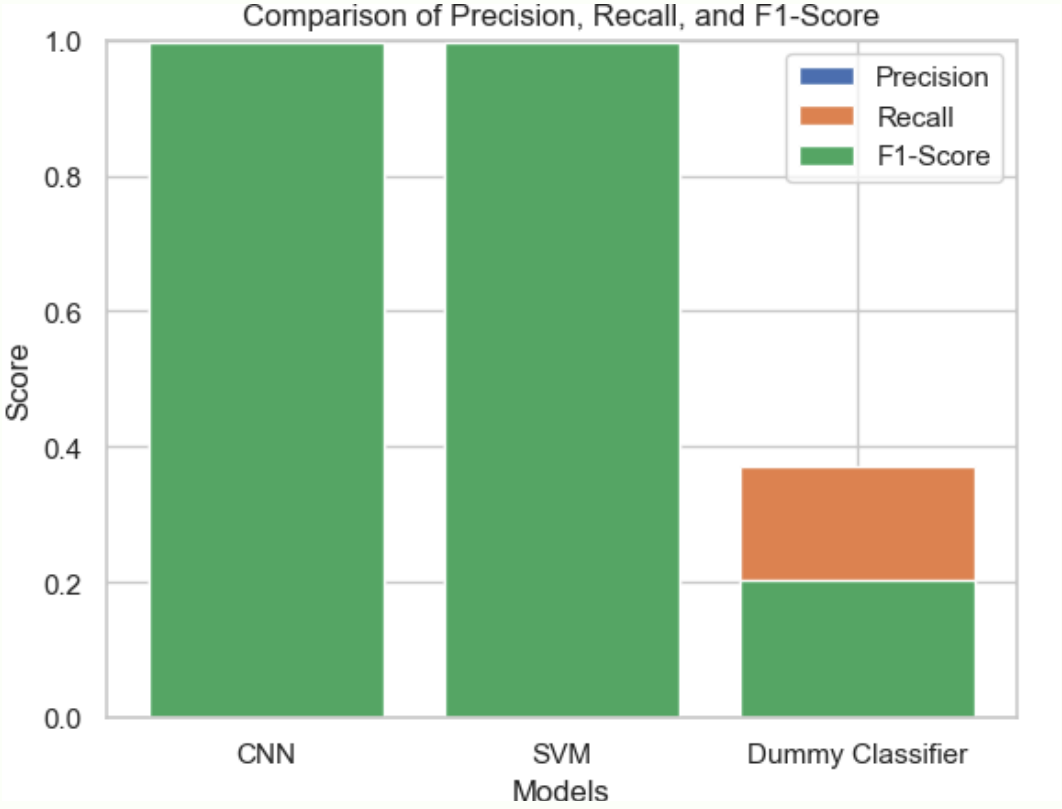


# Model Comparison cont.

## CNN vs SVM vs DC

```
# Graph to compare precision, recall, and f1-score of CNN, SVM, and Dummy Class
precision_scores = [precision_score(y_test, y_pred_cnn, average='weighted'),
                    precision_score(y_test, y_pred_svm, average='weighted'),
                    precision_score(y_test, y_pred_dummy, average='weighted')]
recall_scores = [recall_score(y_test, y_pred_cnn, average='weighted'),
                 recall_score(y_test, y_pred_svm, average='weighted'),
                 recall_score(y_test, y_pred_dummy, average='weighted')]
f1_scores = [f1_score(y_test, y_pred_cnn, average='weighted'),
             f1_score(y_test, y_pred_svm, average='weighted'),
             f1_score(y_test, y_pred_dummy, average='weighted')]

plt.bar(models, precision_scores, label='Precision')
plt.bar(models, recall_scores, label='Recall')
plt.bar(models, f1_scores, label='F1-Score')
plt.xlabel('Models')
plt.ylabel('Score')
plt.title('Comparison of Precision, Recall, and F1-Score')
plt.legend()
plt.ylim([0, 1])
plt.show()
```

This code creates a bar plot to compare precision, recall, and F1-score values for three models: CNN, SVM, and DC, The plot depicts precision, recall, and F1-score values on the y-axis, and the title is "Comparison of Precision, Recall, and F1-Score."

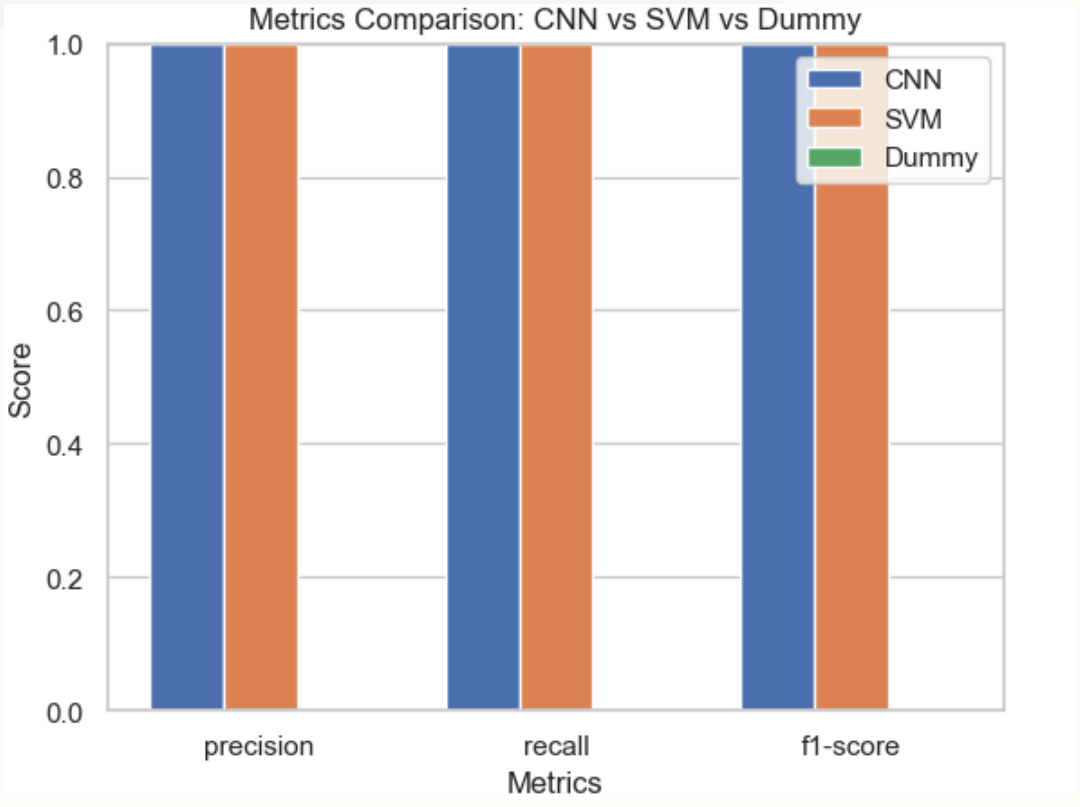


# Model Comparison cont.

## CNN vs SVM vs DC

```
# Extract precision, recall, and F1-score values from classification re
def extract_scores(report):
    scores = re.findall(r"\d+\.\d+", report)
    return [float(score) for score in scores]

cnn_scores = extract_scores(classification_report_cnn)[:3]  # Consider
svm_scores = extract_scores(classification_report_svm)[:3]  # Consider
dummy_scores = extract_scores(classification_report_dummy)[:3]  # Consi

metrics = ['precision', 'recall', 'f1-score']
bar_width = 0.25
index = np.arange(len(metrics))

plt.bar(index, cnn_scores, bar_width, label='CNN')
plt.bar(index + bar_width, svm_scores, bar_width, label='SVM')
plt.bar(index + 2 * bar_width, dummy_scores, bar_width, label='Dummy')

plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Metrics Comparison: CNN vs SVM vs Dummy')
plt.xticks(index + bar_width, metrics)
plt.legend()
plt.ylim([0, 1])
plt.show()
```

This code extracts precision, recall, and F1-score values from classification reports of CNN, SVM, and the Dummy Classifier, presenting the results in a bar plot. Each model is differentiated by color



# Fine-Tuning a Pre-Trained CNN Model

```
#fine tunig algorithm
# Assuming you have a pre-trained model loaded into pretrained_model
pretrained_model = model_cnn  # model_cnn with the highest accuracy

# Remove the dense layers from the pre-trained model
pretrained_model.layers.pop()
pretrained_model.layers.pop()

# Freeze the layers of the pre-trained model
for layer in pretrained_model.layers:
    layer.trainable = False

# Build your new model on top of the pre-trained model
fine_tuned_model = Sequential()
fine_tuned_model.add(pretrained_model)
fine_tuned_model.add(Dense(128, activation='relu'))
fine_tuned_model.add(Dense(3, activation='softmax'))

# Compile the fine-tuned model
fine_tuned_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

This code demonstrates fine-tuning of a pre-trained CNN model for breast mass classification. The pre-trained model, assumed to be model_cnn with the highest accuracy, has its dense layers removed and is frozen. A new model is then built on top of the pre-trained layers, adding a Dense layer with ReLU activation and a final Dense layer with softmax activation. The fine-tuned model is compiled using categorical crossentropy loss and the Adam optimizer for accuracy metrics.

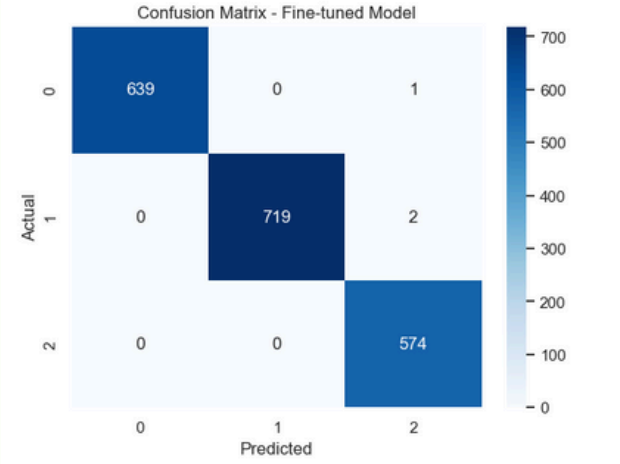
# Training and Evaluating:

```
# Train the fine-tuned model
fine_tuned_model.fit(X_train_cnn, y_train_cnn, batch_size=32, epochs=5, validation_data=(X_test_cnn, y_test_cnn))

# Evaluate the fine-tuned model
y_pred_fine_tuned = fine_tuned_model.predict(X_test_cnn)
y_pred_fine_tuned = np.argmax(y_pred_fine_tuned, axis=1)

# Calculate accuracy
accuracy_fine_tuned = accuracy_score(np.argmax(y_test_cnn, axis=1), y_pred_fine_tuned)

# Generate classification report
classification_report_fine_tuned = classification_report(np.argmax(y_test_cnn, axis=1), y_pred_fine_tuned)
classification_report_fine_tuned_dict = classification_report(np.argmax(y_test_cnn, axis=1), y_pred_fine_tuned, output_dict=True)

# Confusion Matrix for fine-tuned model
conf_matrix_fine_tuned = confusion_matrix(np.argmax(y_test_cnn, axis=1), y_pred_fine_tuned)
sns.heatmap(conf_matrix_fine_tuned, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Fine-tuned Model')
plt.show()

# Display metrics
print('Fine-tuned Model Accuracy:', accuracy_fine_tuned)
print('Fine-tuned Model Classification Report:')
print(classification_report_fine_tuned)
print('Fine-tuned Model Classification Report Dictionary:')
print(classification_report_fine_tuned_dict)
```

This code trains a fine-tuned CNN model on breast mass classification using the pre-trained model with added layers. The model is trained for 5 epochs and evaluated on the test set. Accuracy, a confusion matrix, and a classification report are generated, providing insights into the performance of the fine-tuned model. The classification report includes precision, recall, and F1-score values, while the confusion matrix visualizes true positives, true negatives, false positives, and false negatives.

# Fine-Tuned CNN Model Training and Evaluation Output

```
Epoch 1/5
242/242 [==============================] - 65s 249ms/step - loss: 0.3166 - accuracy: 0.9696 - val_loss: 0.0346 - val_accuracy:
0.9984
Epoch 2/5
242/242 [==============================] - 54s 224ms/step - loss: 0.0138 - accuracy: 1.0000 - val_loss: 0.0129 - val_accuracy:
0.9984
Epoch 3/5
242/242 [==============================] - 53s 218ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.0102 - val_accuracy:
0.9984
Epoch 4/5
242/242 [==============================] - 57s 237ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.0097 - val_accuracy:
0.9984
Epoch 5/5
242/242 [==============================] - 56s 231ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.0098 - val_accuracy:
0.9984
61/61 [==============================] - 13s 164ms/step
```




The fine-tuned CNN model underwent 5 epochs of training, displaying significant accuracy improvements over epochs. Starting with a loss of 0.3166 and accuracy of 96.96%, it rapidly converged to a final loss of 0.0011 and 100% accuracy. The confusion matrix graph, showcasing the model's predictions against actual values

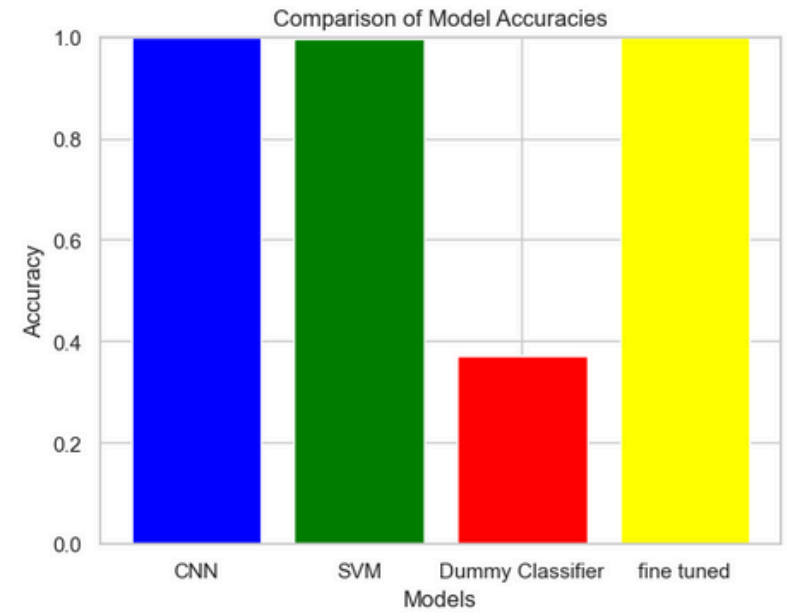
The accuracy of the fine-tuned model is 99.84%, and the classification report details precision, recall, and F1-score for each class (0, 1, 2). The weighted average F1-score is 99.84%, indicating high overall model performance across all classes. The detailed classification report dictionary provides further insights into precision, recall, and F1-score values for each class and overall performance metrics.

```
Fine-tuned Model Accuracy: 0.9984496124031008
Fine-tuned Model Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       640
           1       1.00      1.00      1.00       721
           2       0.99      1.00      1.00       574

    accuracy                           1.00      1935
   macro avg       1.00      1.00      1.00      1935
weighted avg       1.00      1.00      1.00      1935

Fine-tuned Model Classification Report Dictionary:
{'0': {'precision': 1.0, 'recall': 0.9984375, 'f1-score': 0.9992181391712275, 'support': 640}, '1': {'precision': 1.0, 'recal
l': 0.9972260748959778, 'f1-score': 0.9986111111111111, 'support': 721}, '2': {'precision': 0.9948006932409013, 'recall': 1.0,
'f1-score': 0.9973935708079931, 'support': 574}, 'accuracy': 0.9984496124031008, 'macro avg': {'precision': 0.9982668977469671,
'recall': 0.9985545249653259, 'f1-score': 0.9984076070301106, 'support': 1935}, 'weighted avg': {'precision': 0.998457673343812
5, 'recall': 0.9984496124031008, 'f1-score': 0.9984507130875889, 'support': 1935}}
```

# Model Comparison

## CNN vs SVM vs DC vs Fine-Tuned Model

```
# Graph to compare accuracy of CNN, SVM,  Dummy Classifier and fine tuned
models = ['CNN', 'SVM', 'Dummy Classifier','fine tuned']
accuracies = [accuracy_cnn, accuracy_svm, accuracy_dummy, accuracy_fine_tuned]
colors = ['blue', 'green', 'red','yellow']

plt.bar(models, accuracies, color=colors)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.ylim([0, 1])
plt.show()
```




This bar graph visually compares the accuracy of different models, including CNN, SVM, Dummy Classifier, and the Fine-Tuned model. Each model is represented by a distinct color (blue, green, red, and yellow, respectively). The y-axis reflects accuracy, while the x-axis displays the model names. This graphical representation allows for a quick assessment of the relative performance of each model in terms of accuracy, providing a comprehensive overview of their comparative strengths.

# Class Mapping for Model Output Labels:

```
class_mapping = {
    0: 'MALIGNANT',
    2: 'BENIGN',
    1: 'BENIGN_WITHOUT_CALLBACK',
}
```

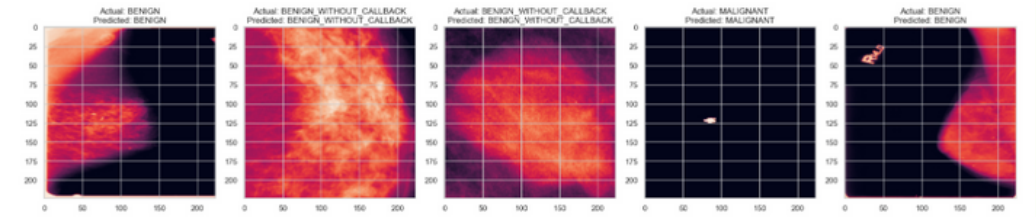
The provided class mapping assigns human-readable labels to the numerical output classes generated by a model. Specifically, it maps class 0 to 'MALIGNANT,' class 2 to 'BENIGN,' and class 1 to 'BENIGN_WITHOUT_CALLBACK.' This mapping aids in interpreting and communicating the predictions made by the model, offering a clearer understanding of the pathology classifications associated with each numerical class output.

# Predictions Visualization:

This code generates visual comparisons of predictions made by different models, including CNN, SVM, Dummy Classifier, and a fine-tuned model. For each model, it displays a specified number of images with their actual and predicted labels. The titles of the subplots provide insights into the performance of each model on individual images, aiding in understanding how well the models align with the ground truth. The images showcase a diverse set of predictions, contributing to a comprehensive evaluation of the models' capabilities.

```python
#CNN

# Number of images to display
num_images = 5

# Create a subplot with the specified number of columns and a wider layout
fig, axs = plt.subplots(1, num_images, figsize=(20, 4))

# Loop through the specified number of images
for i in range(num_images):
    # Get the image, actual label, and make a prediction
    image = X_test_cnn[i]
    actual_label = np.argmax(y_test_cnn[i])
    predicted_probabilities = model_cnn.predict(np.expand_dims(image, axis=0))[0]
    predicted_label_index = np.argmax(predicted_probabilities)

    # Map the numeric label to the actual class name
    if 'class_mapping' in locals():
        actual_class_name = class_mapping.get(actual_label, actual_label)
        predicted_class_name = class_mapping.get(predicted_label_index, predicted_label_index)
    else:
        actual_class_name = actual_label
        predicted_class_name = predicted_label_index

    # Display the image in the current subplot
    axs[i].imshow(image)
    axs[i].set_title(f"Actual: {actual_class_name}\nPredicted: {predicted_class_name}")

# Adjust layout to prevent clipping of titles
plt.tight_layout()
plt.show()
```

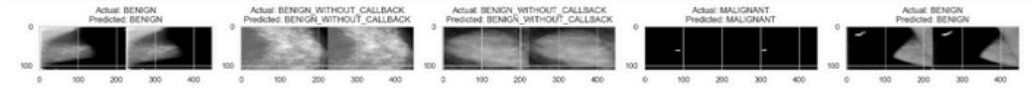
```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 106ms/step
```



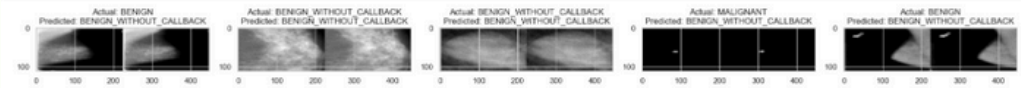
# Predictions Visualization:

cont.

```
#SVM

# Number of images to display
num_images = 5

# Create a subplot with the specified number of columns
fig, axs = plt.subplots(1, num_images, figsize=(20, 4))

# Loop through the specified number of images
for i in range(num_images):
    # Get the flattened image, actual label, and make a prediction
    flattened_image = X_test_svm[i]
    actual_label = np.argmax(y_test_cnn[i])
    predicted_label = model_svm.predict(flattened_image.reshape(1, -1))

    # Infer dimensions from the flattened image size
    side_length = int(np.sqrt(len(flattened_image)))
    inferred_height = side_length // 2
    inferred_width = side_length * 2

    # Map the numeric label to the actual class name
    if 'class_mapping' in locals():
        actual_class_name = class_mapping.get(actual_label, actual_label)
        predicted_class_name = class_mapping.get(predicted_label[0], predicted_label[0])
    else:
        actual_class_name = actual_label
        predicted_class_name = predicted_label[0]

    # Display the flattened image in the current subplot
    axs[i].imshow(flattened_image.reshape((inferred_height, inferred_width)), cmap='gray')
    # Adjust based on your original image shape
    axs[i].set_title(f"Actual: {actual_class_name}\nPredicted: {predicted_class_name}")

# Adjust layout to prevent clipping of titles
plt.tight_layout()
plt.show()
```




```
#DUMMY

# Number of images to display
num_images = 5

# Create a subplot with the specified number of columns
fig, axs = plt.subplots(1, num_images, figsize=(20, 4))

# Loop through the specified number of images
for i in range(num_images):
    # Get the flattened image, actual label, and make a prediction
    flattened_image = X_test_svm[i]
    actual_label = np.argmax(y_test_cnn[i])
    predicted_label = y_pred_dummy[i]

    # Infer dimensions from the flattened image size
    side_length = int(np.sqrt(len(flattened_image)))
    inferred_height = side_length // 2
    inferred_width = side_length * 2

    # Map the numeric label to the actual class name
    if 'class_mapping' in locals():
        actual_class_name = class_mapping.get(actual_label, actual_label)
        predicted_class_name = class_mapping.get(predicted_label, predicted_label)
    else:
        actual_class_name = actual_label
        predicted_class_name = predicted_label

    # Display the flattened image in the current subplot
    axs[i].imshow(flattened_image.reshape((inferred_height, inferred_width)), cmap='gray')
    # Adjust based on your original image shape
    axs[i].set_title(f"Actual: {actual_class_name}\nPredicted: {predicted_class_name}")

# Adjust layout to prevent clipping of titles
plt.tight_layout()
plt.show()
```

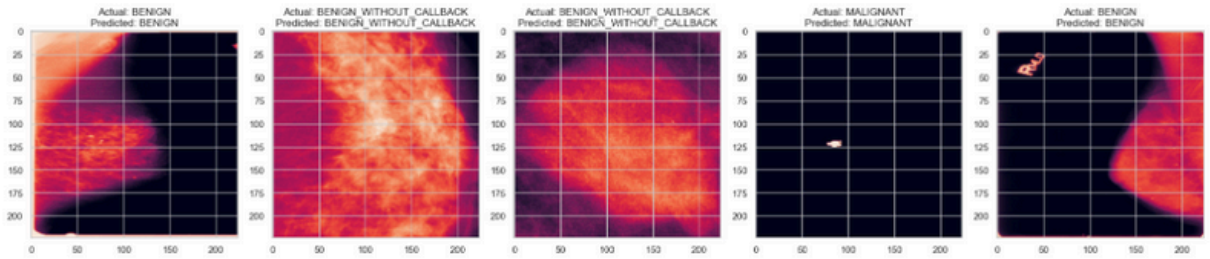


# Predictions Visualization:

cont.

```
#FINE TUNED

# Number of images to display
num_images = 5

# Create a subplot with the specified number of columns
fig, axs = plt.subplots(1, num_images, figsize=(20, 4))

# Loop through the specified number of images
for i in range(num_images):
    # Get the image, actual label, and make a prediction
    image = X_test_cnn[i]
    actual_label = np.argmax(y_test_cnn[i])
    predicted_probabilities = fine_tuned_model.predict(np.expand_dims(image, axis=0))[0]
    predicted_label_index = np.argmax(predicted_probabilities)

    # Map the numeric label to the actual class name
    if 'class_mapping' in locals():
        actual_class_name = class_mapping.get(actual_label, actual_label)
        predicted_class_name = class_mapping.get(predicted_label_index, predicted_label_index)
    else:
        actual_class_name = actual_label
        predicted_class_name = predicted_label_index

    # Display the image in the current subplot
    axs[i].imshow(image)
    axs[i].set_title(f"Actual: {actual_class_name}\nPredicted: {predicted_class_name}")

# Adjust layout to prevent clipping of titles
plt.tight_layout()
plt.show()
```

```
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 83ms/step
```



# 3. ML Exploration

## with CBIS-DDSM

1

### Feature Selection

- Filter Methods
- Wrapper Methods

2

### Dimensionality Reduction

- Principal Component Analysis (PCA)

3

### Classification in Various Approaches

- Random Forest Classification with PCA
- Support Vector Machine (SVM) Classification with PCA
- k-Nearest Neighbors (k-NN) Classification

4

### K-Means Clustering with PCA Visualization

- Principal Component Analysis (PCA) for K-Means Clustering on Reduced-Dimension Data

# Feature Selection

## *Filter Methods*

- **Chi-Squared for RandomForestClassifier**

```
#feature selection(filter)chi-squared
# Load your mass training data
mass_train = pd.read_csv(r"D:\UNI\fifth semester\data tools\csv\mass_case_description_train_set.csv")

# Select relevant features (for demonstration purposes, considering only numerical columns)
numerical_features = mass_train.select_dtypes(include=['int64', 'float64']).columns

X = mass_train[numerical_features]
y = mass_train['pathology']  # Assuming 'pathology' is your target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply chi-squared for feature selection
k_best = SelectKBest(score_func=chi2, k='all')  # Use k='all' to select all features
X_train_selected = k_best.fit_transform(X_train, y_train)
X_test_selected = k_best.transform(X_test)

# Train a model on the selected features
k_best = SelectKBest(score_func=chi2, k='all')  # Use k='all' to select all features
X_train_selected = k_best.fit_transform(X_train, y_train)
X_test_selected = k_best.transform(X_test)

# Train a model on the selected features
model = RandomForestClassifier(random_state=42)
model.fit(X_train_selected, y_train)
y_pred = model.predict(X_test_selected)

# Evaluate the model
accuracy_chi2 = accuracy_score(y_test, y_pred)
print("Accuracy (chi-squared):", accuracy_chi2)
```

In this code, chi-squared statistical test is applied for feature selection on a breast mass dataset. Numerical features are selected from the dataset, and the chi-squared test is employed to choose relevant features. A RandomForestClassifier model is then trained on the selected features, and its accuracy is evaluated on the test set. The resulting accuracy score is labeled as "Accuracy (chi-squared)."

- **evaluate the model (actual & predicted) models**

```
# Evaluate the model
print("Accuracy (chi-squared):", accuracy_chi2)
print("Actual Labels for 10 instances:")
print(y_test.values[:10])  # Show actual labels for the first 10 instances

print("Predicted Labels for 10 instances:")
print(y_pred[:10])  # Show predicted labels for the first 10 instances
```

```
Accuracy (chi-squared): 0.7386363636363636
Actual Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'MALIGNANT' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'BENIGN']
Predicted Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'MALIGNANT']
```

# Feature Selection

## *Filter Methods*

- **Mutual Information for RandomForestClassifier**

```
#feauture selection (filter)using  mutual information
# Load your mass training data
mass_train = pd.read_csv(r"D:\UNI\fifth semester\data tools\csv\mass_case_description_train_set.csv")

# Select relevant features (for demonstration purposes, considering only numerical columns)
numerical_features = mass_train.select_dtypes(include=['int64', 'float64']).columns

X = mass_train[numerical_features]
y = mass_train['pathology']  # Assuming 'pathology' is your target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply mutual information for feature selection
mutual_info_scores = mutual_info_classif(X_train, y_train)
feature_indices = sorted(range(len(mutual_info_scores)), key=lambda i: mutual_info_scores[i], reverse=True)
selected_features = X_train.columns[feature_indices[:5]]  # Selecting the top 5 features

X2_train_selected = X_train[selected_features]
X2_test_selected = X_test[selected_features]

# Train a model on the selected features
model = RandomForestClassifier(random_state=42)  # Set random_state for reproducibility
model.fit(X2_train_selected, y_train)

# Make predictions on the test set
y_pred = model.predict(X2_test_selected)

# Evaluate the model
accuracy_mi = accuracy_score(y_test, y_pred)
print("Accuracy (mutual information):", accuracy_mi)
```

```
Accuracy (mutual information): 0.7424242424242424
```

This code focuses on feature selection using mutual information on a breast mass dataset. Numerical features are chosen from the dataset, and mutual information scores are calculated to identify relevant features. The top 5 features are selected based on these scores, and a RandomForestClassifier model is trained on this subset of features. The accuracy of the model is evaluated on the test set, and the result is presented as "Accuracy (mutual information)."

- **evaluate the model (actual & predicted) models**

```
# Evaluate the model
print("Accuracy (mutual information):", accuracy_mi)
print("Actual Labels for 10 instances:")
print(y_test.values[:10])  # Show actual labels for the first 10 instances

print("Predicted Labels for 10 instances:")
print(y_pred[:10])  # Show predicted labels for the first 10 instances
```

```
Accuracy (mutual information): 0.7424242424242424
Actual Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'MALIGNANT' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'BENIGN']
Predicted Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'MALIGNANT']
```

# Feature Selection

## *Wrapper Methods* cont.

- **RFE and RandomForestClassifier**

```
# Load the mass dataset
mass_train2 = pd.read_csv(r"D:\UNI\fifth semester\data tools\csv\mass_case_description_train_set.csv")

# Separate the features (X) and the target variable (y)
x2 = mass_train[numerical_features]
y2 = mass_train2['pathology']

# Split the data into training and testing sets
X_train2, X_test2, y_train2,y_test2= train_test_split(x2, y2, test_size=0.2, random_state=42)

# Apply Recursive Feature Elimination (RFE)
model_rfe = RandomForestClassifier(random_state=42)  # Set random_state for reproducibility
rfe = RFE(model_rfe, n_features_to_select=5)  # Selecting the top 5 features
X_train_rfe_selected = rfe.fit_transform(X_train2,y_train2)
X_test_rfe_selected = rfe.transform(X_test2)

# Train a model on the selected features (RFE)
model_rfe_selected = RandomForestClassifier(random_state=42)  # Set random_state for reproducibility
model_rfe_selected.fit(X_train_rfe_selected, y_train2)
y_pred_rfe = model_rfe_selected.predict(X_test_rfe_selected)

# Evaluate the model (RFE)
accuracy_rfe = accuracy_score(y_test2,y_pred_rfe)
print("Accuracy (RFE):", accuracy_rfe)
```

```
Accuracy (RFE): 0.7386363636363636
```

The code loads a mass dataset from a CSV file, separates features (X) and the target variable (y), and then splits the data into training and testing sets. It applies Recursive Feature Elimination (RFE) using a RandomForestClassifier to select the top 5 features. A new model is trained on the selected features, and its performance is evaluated using accuracy. The resulting accuracy is printed.

- **evaluate the model (actual & predicted) models**

```
# Evaluate the model
print("Accuracy (RFE):", accuracy_rfe)
print("Actual Labels for 10 instances:")
print(y_test2.values[:10])  # Show actual labels for the first 10 instances

print("Predicted Labels for 10 instances:")
print(y_pred_rfe[:10])  # Show predicted labels for the first 10 instances
```

```
Accuracy (RFE): 0.7386363636363636
Actual Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'MALIGNANT' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'BENIGN']
Predicted Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'MALIGNANT']
```

# Feature Selection

## *Wrapper Methods* cont.

- **SFS and RandomForestClassifier**

```
# Assuming X_train2, y_train2, X_test2, and y_test2 are defined

# Apply Sequential Feature Selection (SFS)
model_sfs = RandomForestClassifier(random_state=42)
# Set to None to select all features
sfs = SequentialFeatureSelector(model_sfs, n_features_to_select=None, direction='forward')
X_train_sfs_selected = sfs.fit_transform(X_train2, y_train2)
X_test_sfs_selected = sfs.transform(X_test2)

# Train a model on the selected features (SFS)
model_sfs_selected = RandomForestClassifier(random_state=42)
model_sfs_selected.fit(X_train_sfs_selected, y_train2)
y_pred_sfs = model_sfs_selected.predict(X_test_sfs_selected)

# Evaluate the model (SFS)
accuracy_sfs = accuracy_score(y_test2, y_pred_sfs)
print("Accuracy (SFS):", accuracy_sfs)
```

```
Accuracy (SFS): 0.6931818181818182
```

This code applies Sequential Feature Selection (SFS) using a RandomForestClassifier on the given training and testing sets. The Sequential Feature Selector is configured to select all features in a forward direction. A new model is trained on the features selected by SFS, and its accuracy is evaluated. The resulting accuracy is printed as "Accuracy (SFS)."

- **evaluate the model (actual & predicted) models**

```
# Evaluate the model
print("Accuracy (SFS):", accuracy_sfs)
print("Actual Labels for 10 instances:")
print(y_test2.values[:10])  # Show actual labels for the first 10 instances

print("Predicted Labels for 10 instances:")
print(y_pred_sfs[:10])  # Show predicted labels for the first 10 instances
```

```
Accuracy (SFS): 0.6931818181818182
Actual Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'MALIGNANT' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'BENIGN']
Predicted Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'MALIGNANT' 'MALIGNANT' 'BENIGN' 'MALIGNANT'
 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'MALIGNANT']
```

# Dimensionality Reduction

- ## Principal Component Analysis (PCA)

for RandomForestClassifier Trained on RFE-Selected Features

```
# Apply PCA for dimensionality reduction
n_components = min(X_train_rfe_selected.shape[0], X_train_rfe_selected.shape[1]) - 1
# n-components --> Adjusted to be within a valid range
pca_rfe = PCA(n_components=n_components)
X_train_rfe_pca = pca_rfe.fit_transform(X_train_rfe_selected)
X_test_rfe_pca = pca_rfe.transform(X_test_rfe_selected)

# Train a model on the reduced-dimension data
model_pca = RandomForestClassifier(random_state=42)  # Set random_state for reproducibility
model_pca.fit(X_train_rfe_pca, y_train2)

# Make predictions on the test set
y_pred_pca = model_pca.predict(X_test_rfe_pca)

# Evaluate the model (RFE + PCA)
accuracy_rfe_pca = accuracy_score(y_test2, y_pred_pca)
print("Accuracy (RFE + PCA):", accuracy_rfe_pca)
```

```
Accuracy (RFE + PCA): 0.7424242424242424
```

This code applies PCA for dimensionality reduction to the RandomForestClassifier trained on features selected by RFE. The number of components is determined based on the minimum between the number of samples and features minus one. A new model is trained on the reduced-dimension data, and its accuracy is evaluated on the test set, yielding the accuracy score labeled as "Accuracy (RFE + PCA)."

- ## evaluate the model (actual & predicted) models

```
# Evaluate the RFE + PCA model
print("Accuracy (RFE + PCA):", accuracy_rfe_pca)
print("Actual Labels for 10 instances:")
print(y_test2.values[:10])  # Show actual labels for the first 10 instances

print("Predicted Labels for 10 instances:")
print(y_pred_pca[:10])  # Show predicted labels for the first 10 instances
```

```
Accuracy (RFE + PCA): 0.7424242424242424
Actual Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'MALIGNANT' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'BENIGN']
Predicted Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'MALIGNANT']
```

# Dimensionality Reduction

- ## Principal Component Analysis (PCA)

  for RandomForestClassifier Trained on Features Selected by Mutual Information

```
# Apply PCA for dimensionality reduction
n_components = min(X2_train_selected.shape[0], X2_train_selected.shape[1])
pca = PCA(n_components=n_components)
X2_train_pca = pca.fit_transform(X2_train_selected)
X2_test_pca = pca.transform(X2_test_selected)

# Train a model on the reduced-dimension data
model = RandomForestClassifier(random_state=42)  # Set random_state for reproducibility
model.fit(X2_train_pca, y_train)

# Make predictions on the test set
y_pred = model.predict(X2_test_pca)

# Evaluate the model
accuracy_mi_pca = accuracy_score(y_test, y_pred)
print("Accuracy (mutual information + PCA):", accuracy_mi_pca)
```

```
Accuracy (mutual information + PCA): 0.7424242424242424
```

This code employs PCA for dimensionality reduction on features selected by Mutual Information. The number of components is set based on the minimum between the number of samples and selected features. A RandomForestClassifier model is then trained on the reduced-dimension data, and its accuracy is evaluated on the test set. The resulting accuracy score is labeled as "Accuracy (mutual information + PCA)."

- ## evaluate the model (actual & predicted) models

```
# Evaluate the Mutual Information + PCA model
print("Accuracy (mutual information + PCA):", accuracy_mi_pca)
print("Actual Labels for 10 instances:")
print(y_test.values[:10])  # Show actual labels for the first 10 instances

print("Predicted Labels for 10 instances:")
print(y_pred[:10])  # Show predicted labels for the first 10 instances
```

```
Accuracy (mutual information + PCA): 0.7424242424242424
Actual Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'MALIGNANT' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'BENIGN']
Predicted Labels for 10 instances:
['BENIGN' 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK'
 'MALIGNANT' 'BENIGN' 'MALIGNANT' 'BENIGN_WITHOUT_CALLBACK' 'MALIGNANT']
```

# Classification in Various Approaches

- ## Support Vector Machine (SVM) Classification with PCA

  Principal Component Analysis (PCA) for Support Vector Machine (SVM) Trained on RFE-Selected Features

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Assuming X2_train_pca and X2_test_pca are the reduced-dimension datasets after PCA
# Assuming y_train and y_test are your target variables for classification

# Train a Support Vector Machine (SVM) model
svm_model = SVC(random_state=42)
svm_model.fit(X_train_rfe_pca, y_train)

# Make predictions on the test set
y_pred_svm = svm_model.predict(X_test_rfe_pca)

# Evaluate the model
accuracy_svm = accuracy_score(y_test2, y_pred_svm)
print("Accuracy (SVM):", accuracy_svm)

# Print classification report for more detailed evaluation
print("Classification Report:")
print(classification_report(y_test, y_pred_svm))
```

```
Accuracy (SVM): 0.7462121212121212
Classification Report:
                         precision    recall  f1-score   support

                 BENIGN       0.70      0.74      0.72       112
BENIGN_WITHOUT_CALLBACK       1.00      0.52      0.69        23
              MALIGNANT       0.77      0.79      0.78       129

               accuracy                           0.75       264
              macro avg       0.82      0.68      0.73       264
           weighted avg       0.76      0.75      0.75       264
```

This code uses a Support Vector Machine (SVM) for classification after reducing the dataset's dimensionality with Principal Component Analysis (PCA). The SVM model is trained on the reduced-dimension training set, and predictions are made on the test set. The resulting accuracy is printed as "Accuracy (SVM)," and a detailed classification report provides additional insights into the model's performance across different classes. The random state is set to 42 for result reproducibility.

- ## k-Nearest Neighbors (k-NN) Classification

  k-NN Classifier Trained on PCA-Reduced Datasets

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Assuming X2_train_pca and X2_test_pca are the reduced-dimension datasets after PCA

# Train a k-NN classifier
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X2_train_pca, y_train)

# Make predictions on the test set
y_pred_knn = knn_classifier.predict(X2_test_pca)

# Evaluate the k-NN classifier
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print("Accuracy (k-NN):", accuracy_knn)
print(classification_report(y_test, y_pred_knn))
```

```
Accuracy (k-NN): 0.7007575757575758
                         precision    recall  f1-score   support

                 BENIGN       0.63      0.76      0.69       112
BENIGN_WITHOUT_CALLBACK       1.00      0.39      0.56        23
              MALIGNANT       0.76      0.71      0.73       129

               accuracy                           0.70       264
              macro avg       0.80      0.62      0.66       264
           weighted avg       0.72      0.70      0.70       264
```

This code uses the scikit-learn library to implement a k-Nearest Neighbors (k-NN) classifier. The classifier is trained on a dataset with reduced dimensionality achieved through Principal Component Analysis (PCA). After training, predictions are made on a test set, and the accuracy of the k-NN classifier is calculated and printed for evaluation, and a detailed classification report provides additional insights into the model's performance across different classes.

# K-Means Clustering with PCA Visualization

```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Assuming X2_train_pca is the reduced-dimension dataset after PCA

# Choose the number of clusters (you may need to adjust this based on your data)
n_clusters = 3

# Apply K-Means clustering
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
pathology_clusters = kmeans.fit_predict(X2_train_pca)

# Visualize the clusters (assuming 2D data after PCA)
plt.scatter(X2_train_pca[:, 0], X2_train_pca[:, 1], c=pathology_clusters, cmap='viridis', edgecolor='k')
plt.title('K-Means Clustering of Pathology Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

This code applies K-Means clustering to a dataset reduced through PCA. The number of clusters is set to 3, but this can be adjusted based on the data characteristics. The KMeans algorithm is utilized with a random state for reproducibility. The resulting clusters are visualized in a scatter plot, assuming 2D data after PCA. Each point is colored based on its assigned cluster, providing insight into the grouping patterns within the reduced-dimensional dataset. The plot is titled 'K-Means Clustering of Pathology Data,' and the axes represent the first and second principal components.