CNN model for Binary Image Classification

Labels (Melt Pool = 1 , No Melt Pool = 0)

In [1]:
```
#Import necessary libraries for model

#for array
import numpy as np
#for reading each filein directory and writing in excel
import os
#to plot graphs
import matplotlib.pyplot as plt
#for layers
import tensorflow as tf
#for image processing
import cv2
#for training time calculation
import datetime
#for train, test spliot and machine learning models
import sklearn
#for confusion matrix
import seaborn as sns
#for layers
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import TensorBoard
from sklearn.metrics import confusion_matrix
```

In [2]:
```
# n is number of images from melt pool and no meltpool to train the model

v = int(input("Number of Images from melt pool and no melt pool : "))
total_images_as_input = print("Number of input images to the model : " + str(v*2))
e=int(input('Number of epochs :'))
f=int(input('filter size(f*f) :'))
z=int(input('number of filters:'))
```

```
Number of Images from melt pool and no melt pool : 20
Number of input images to the model : 40
Number of epochs :5
filter size(f*f) :3
number of filters:1
```

In [3]:
```
# Assigning Directory path for test images

melt_pool_folder = "C:/#Datasets/Class/train/train_melt1"
no_melt_pool_folder = "C:/#Datasets/Class/train/train_melt0"
```

In [4]:
```
#resizing images if needed for anlaysis

melt_pool_images = []
no_melt_pool_images = []

height,width=128,120

# read n images from melt pool folder
for i, file in enumerate(os.listdir(melt_pool_folder)):
    if i >= v:
        break
```

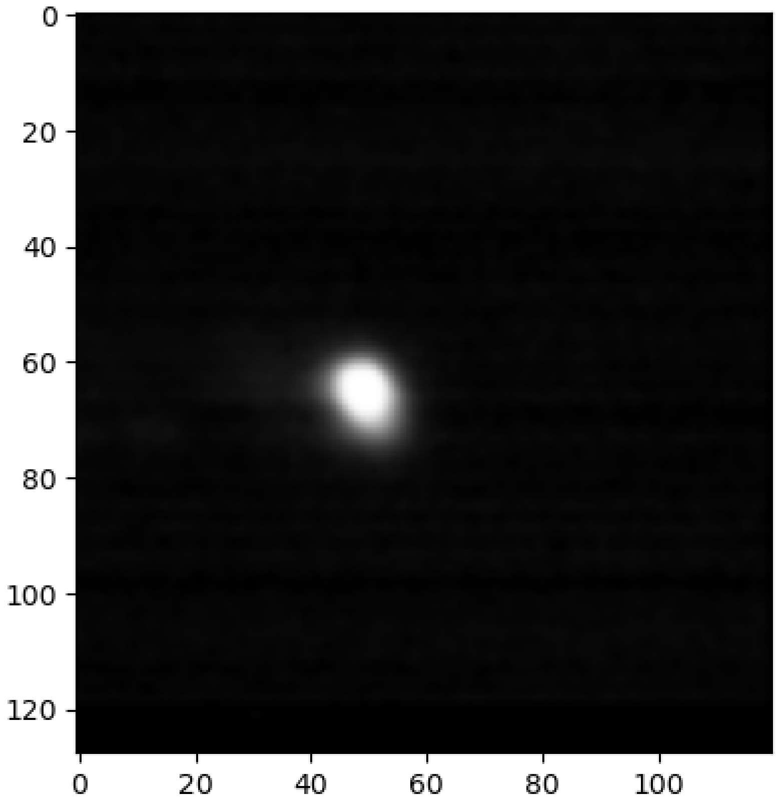
```python
    image = cv2.imread(os.path.join(melt_pool_folder, file))
    if image is not None:
        # resize image to desired dimensions
        image = cv2.resize(image, (width,height))
        melt_pool_images.append(image)

# read n images from no melt pool folder
for i, file in enumerate(os.listdir(no_melt_pool_folder)):
    if i >= v:
        break
    image = cv2.imread(os.path.join(no_melt_pool_folder, file))
    if image is not None:
        # resize image to desired dimensions
        image = cv2.resize(image, (width,height))
        no_melt_pool_images.append(image)

# convert the lists to numpy arrays and concatenate them
melt_pool_images = np.array(melt_pool_images)
no_melt_pool_images = np.array(no_melt_pool_images)
images = np.concatenate([melt_pool_images, no_melt_pool_images], axis=0)

# generate labels for the data
melt_pool_labels = np.ones(len(melt_pool_images), dtype=int)
no_melt_pool_labels = np.zeros(len(no_melt_pool_images), dtype=int)
labels = np.concatenate([melt_pool_labels, no_melt_pool_labels], axis=0)

# shuffle the data and labels
shuffled_indices = np.random.permutation(len(images))
images = images[shuffled_indices]
labels = labels[shuffled_indices]

# check the shape and labels of the data
print("Images shape:", images.shape)
print("Labels shape:", labels.shape)
print("Labels:", labels[0:10])
```

```
Images shape: (40, 128, 120, 3)
Labels shape: (40,)
Labels: [0 0 1 0 1 1 1 1 1 1]
```

In [5]:

```python
# Enter the index of the image you want to check
index = int(input('Enter Image number you want to verify:' ))

# Display the image
plt.imshow(images[index])
plt.show()

# Display the label of the image
if labels[index] == 0:
    print("No Melt Pool")
else:
    print("Melt Pool")
```

```
Enter Image number you want to verify:23
```



Melt Pool

In [6]:
```
#Array of Images
images [:1]
```

Out[6]:
```
array([[[[6, 6, 6],
          [4, 4, 4],
          [2, 2, 2],
          ...,
          [3, 3, 3],
          [3, 3, 3],
          [1, 1, 1]],

         [[8, 8, 8],
          [6, 6, 6],
          [5, 5, 5],
          ...,
          [5, 5, 5],
          [6, 6, 6],
          [3, 3, 3]],

         [[7, 7, 7],
          [6, 6, 6],
          [4, 4, 4],
          ...,
          [3, 3, 3],
          [4, 4, 4],
          [1, 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, 0, 0],
          [0, 0, 0],
          [0, 0, 0],
          ...,
          [0, 0, 0],
          [0, 0, 0],
          [0, 0, 0]]]], dtype=uint8)
```

In [7]:
```
#Array of labels

labels[:10]
```

Out[7]:
```
array([0, 0, 1, 0, 1, 1, 1, 1, 1, 1])
```

In [8]:
```
#Total number of images and labels

len(images),len(labels)
```

Out[8]:
```
(40, 40)
```

In [9]:
```
#Shape of an Image Array and label

images.shape,labels.shape
```
Out[9]:
```
((40, 128, 120, 3), (40,))
```

In [10]:
```
# Split the data into train and validation dataset

x_train, x_val, y_train, y_val = train_test_split(images, labels, test_size=0.2, ra
```

In [11]:
```
#Images in each set

len(x_train),len(x_val)
```
Out[11]:
```
(32, 8)
```

In [12]:
```
#labels in each set

len(y_train),len(y_val)
```
Out[12]:
```
(32, 8)
```

In [13]:
```
#shape of X_train and y_train

x_train.shape,y_train.shape
```
Out[13]:
```
((32, 128, 120, 3), (32,))
```

In [14]:
```
#%load_ext tensorboard
```

Model Architecture

In [15]:
```
# Define the CNN model architecture using the functional API

inputs = tf.keras.Input(shape=(height, width, 3))
x = tf.keras.layers.Conv2D(filters=z, kernel_size=(f, f), activation='relu')(input
x = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(x)

# x = tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(x)
# x = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(x)

# x = tf.keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu')(x
# x = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(x)

x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(units=512, activation='relu')(x)
# x = tf.keras.layers.Dropout(0.1)(x)

outputs = tf.keras.layers.Dense(units=1, activation='sigmoid')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

In [16]:
```
# Compile the model

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

In [17]:
```
#summary of model with parameters in each layer
```

```
model.summary()
```

```
Model: "model"

_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 128, 120, 3)]     0         
                                                                 
 conv2d (Conv2D)             (None, 126, 118, 1)       28        
                                                                 
 max_pooling2d (MaxPooling2D  (None, 63, 59, 1)        0         
 )                                                               
                                                                 
 flatten (Flatten)           (None, 3717)              0         
                                                                 
 dense (Dense)               (None, 512)               1903616   
                                                                 
 dense_1 (Dense)             (None, 1)                 513       
                                                                 
=================================================================
Total params: 1,904,157
Trainable params: 1,904,157
Non-trainable params: 0
_________________________________________________________________
```

In [18]:

```
# Representation of CNN model in diagram

import netron
netron.start("C:\#Datasets\Class\my_model.h5")
```

```
Serving 'C:\#Datasets\Class\my_model.h5' at http://localhost:8080
```

Out[18]:

```
('localhost', 8080)
```

In [19]:

```
#log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
#tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_j
```

In [20]:

```
#code to calcaulte time in training and each epoch

import time
start_time = time.time()
```

In [21]:

```
# Train the model on the training data and validation

history = model.fit(x_train, y_train, epochs=e
                    , batch_size=12, validation_data=(x_val, y_val), )

#callbacks=[tensorboard_callback]
```

```
Epoch 1/5
3/3 [==============================] - 1s 245ms/step - loss: 112.0637 - accuracy: 0.5312 - val_loss: 0.3992 - val_accuracy: 0.8750
Epoch 2/5
3/3 [==============================] - 0s 49ms/step - loss: 42.2155 - accuracy: 0.6562 - val_loss: 25.9841 - val_accuracy: 0.3750
Epoch 3/5
3/3 [==============================] - 0s 48ms/step - loss: 7.3077 - accuracy: 0.8125 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/5
3/3 [==============================] - 0s 76ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/5
3/3 [==============================] - 0s 70ms/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

In [22]:
```
end_time = time.time()

training_time = end_time - start_time

# Convert training time to hours, minutes, and seconds
hours, rem = divmod(training_time, 3600)
minutes, seconds = divmod(rem, 60)

training_time = "{:0>2}:{:0>2}:{:05.2f}".format(int(hours), int(minutes), seconds)

print("Training time HH:MM:SS:", training_time)
```

```
Training time HH:MM:SS: 00:00:02.01
```

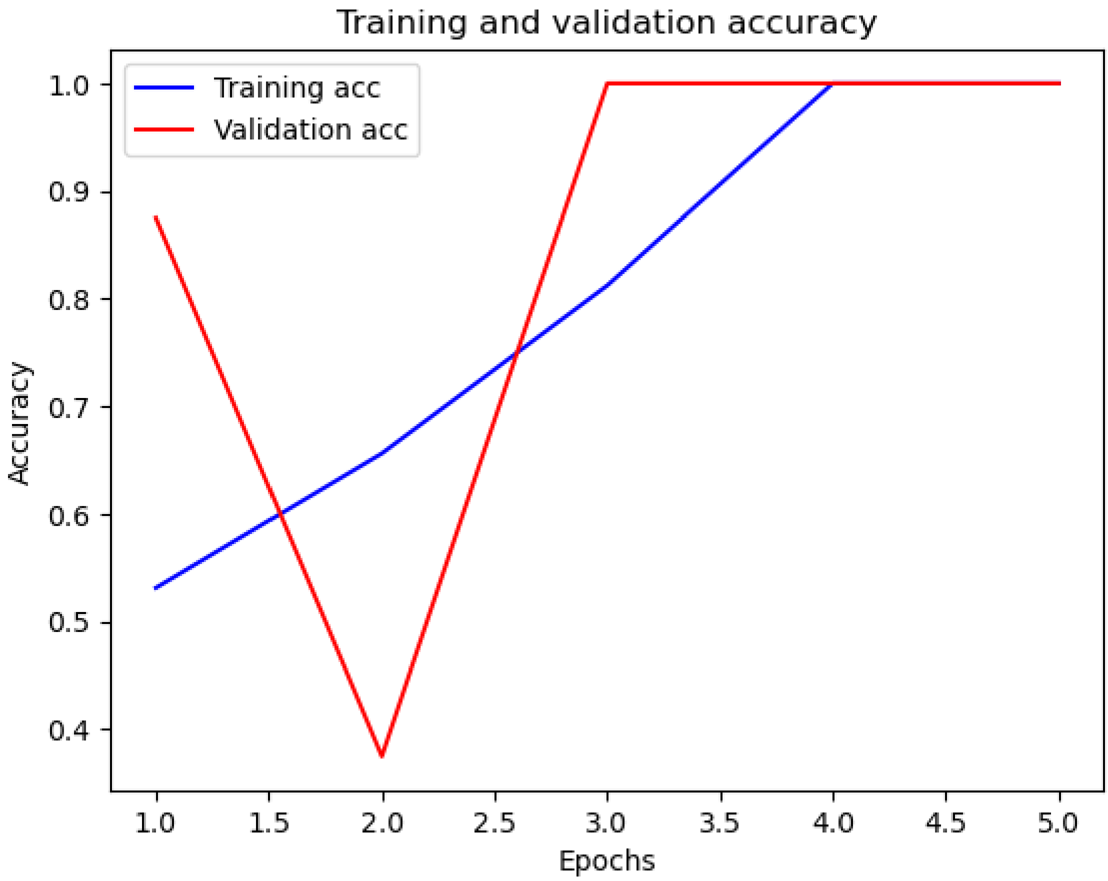
In [23]:
```
#%cd "C:\#Datasets\Class"
```

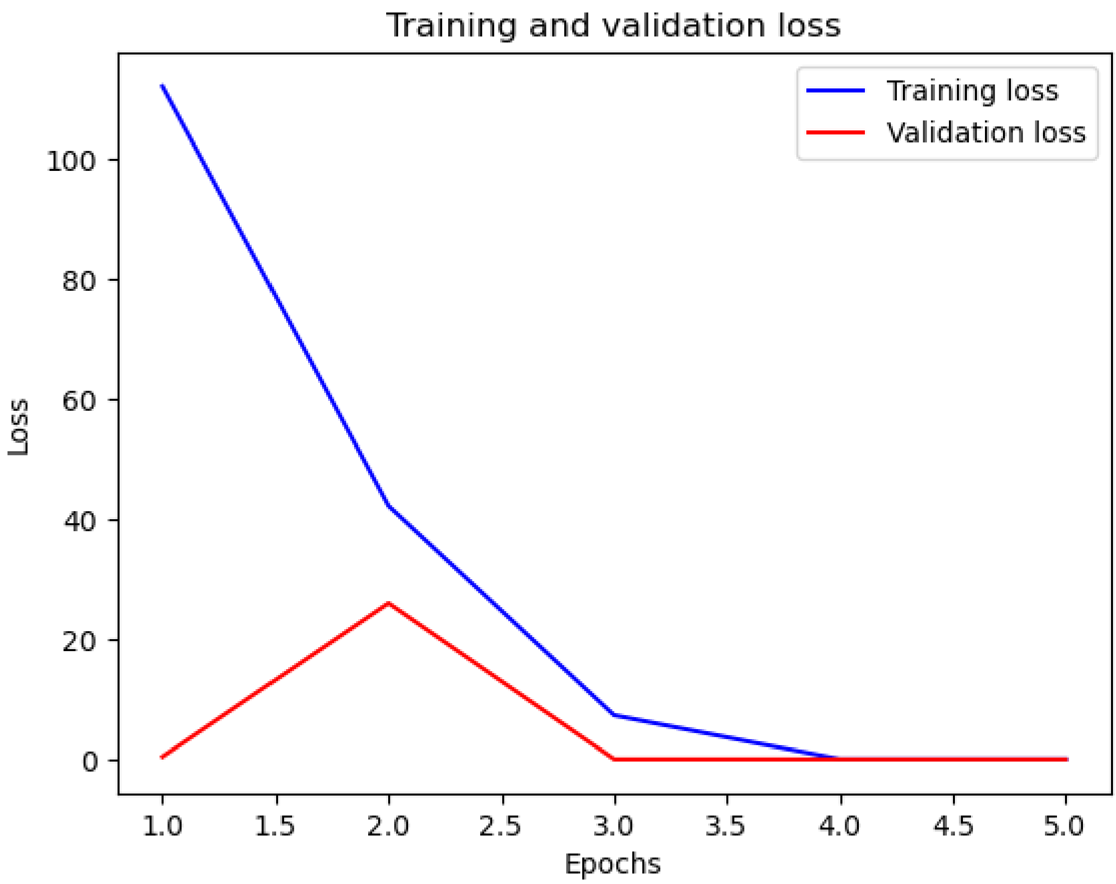
In [24]:
```
#tensorboard dev upload --logdir ./ --name "My Experiment"
```

In [25]:
```
# %tensorboard --logdir logs/fit
```

Training and Validation Accuracy

In [26]:
```
# Plot the training and validation accuracy over epochs

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot the training and validation loss over epochs

loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





In [27]:

```
# Evaluate the model on the validation data
val_loss, val_acc = model.evaluate(x_val, y_val)

# Print the validation accuracy
print("Validation accuracy:", val_acc)
print("Validation loss:", val_loss)
```

```
1/1 [==============================] - 0s 48ms/step - loss: 0.0000e+00 - accuracy:
1.0000
Validation accuracy: 1.0
Validation loss: 0.0
```

In [28]:

```
#Assigning directory for testing the images after resizing.

melt_pool_test_folder = "C:/#Datasets/Class/test/test_melt1"
no_melt_pool_test_folder = "C:/#Datasets/Class/test/test_melt0"

#resizing images if needed for anlaysis
melt_pool_test_images = []
no_melt_pool_test_images = []

# read n images from melt pool folder & resize to according to trained images

for i, file in enumerate(os.listdir(melt_pool_test_folder)):
    if i >= int(v*0.2):
        break
    image = cv2.imread(os.path.join(melt_pool_test_folder, file))
    if image is not None:
        # resize image to desired dimensions
        image = cv2.resize(image, (width,height))
        melt_pool_test_images.append(image)

# read n images from no melt pool folder & resize according to trained images

for i, file in enumerate(os.listdir(no_melt_pool_test_folder)):
    if i >= int(v*0.2):
        break
    image = cv2.imread(os.path.join(no_melt_pool_test_folder, file))
    if image is not None:
        # resize image to desired dimensions
        image = cv2.resize(image,(width,height))
        no_melt_pool_test_images.append(image)

# convert the lists to numpy arrays and concatenate them
melt_pool_test_images = np.array(melt_pool_test_images)
no_melt_pool_test_images = np.array(no_melt_pool_test_images)
x_test = np.concatenate([melt_pool_test_images, no_melt_pool_test_images], axis=0)

# generate labels for the data
melt_pool_test_labels = np.ones(len(melt_pool_test_images), dtype=int)
no_melt_pool_test_labels = np.zeros(len(no_melt_pool_test_images), dtype=int)
y_test = np.concatenate([melt_pool_test_labels, no_melt_pool_test_labels], axis=0)

# shuffle the data and labels
shuffled_indices = np.random.permutation(len(x_test))
x_test = x_test[shuffled_indices]
y_test = y_test[shuffled_indices]

# check the shape and labels of the data
print("Images shape:", x_test.shape)
print("Labels shape:", y_test.shape)
print("Labels:", y_test[0:10])
```

```
Images shape: (8, 128, 120, 3)
Labels shape: (8,)
Labels: [1 1 1 0 1 0 0 0]
```

In [29]:

```
# Predict labels for each image in test data

y_pred = model.predict(x_test)
```

```python
# Iterate through each image and classify as having a melt pool or no melt pool fo
for i in range(len(x_test))[:10]:

    # Get the predicted label for the current image
    pred_label = y_pred.astype(int)[i][0]

    # Check if predicted label is less than 0.5
    if pred_label < 0.5:
        print(f"Image {i+1}: No melt pool, Pred label: {pred_label}")
    else:
        print(f"Image {i+1}: Melt pool, Pred label: {pred_label}")
```

```
Image 1: Melt pool, Pred label: 1
Image 2: Melt pool, Pred label: 1
Image 3: Melt pool, Pred label: 1
Image 4: No melt pool, Pred label: 0
Image 5: Melt pool, Pred label: 1
Image 6: No melt pool, Pred label: 0
Image 7: No melt pool, Pred label: 0
Image 8: No melt pool, Pred label: 0
```

In [30]:
```python
# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(x_test, y_test)

# Print the test loss and test accuracy
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

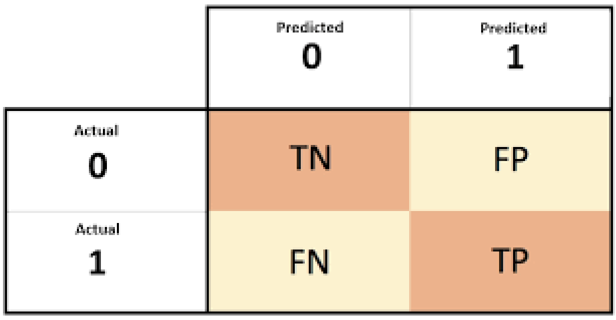
```
1/1 [==============================] - 0s 49ms/step - loss: 0.0000e+00 - accuracy:
1.0000
Test loss: 0.0
Test accuracy: 1.0
```

In [31]:
```python
#Print array of Predicted labels

y_pred=y_pred.astype(int)
y_pred[:10]
```

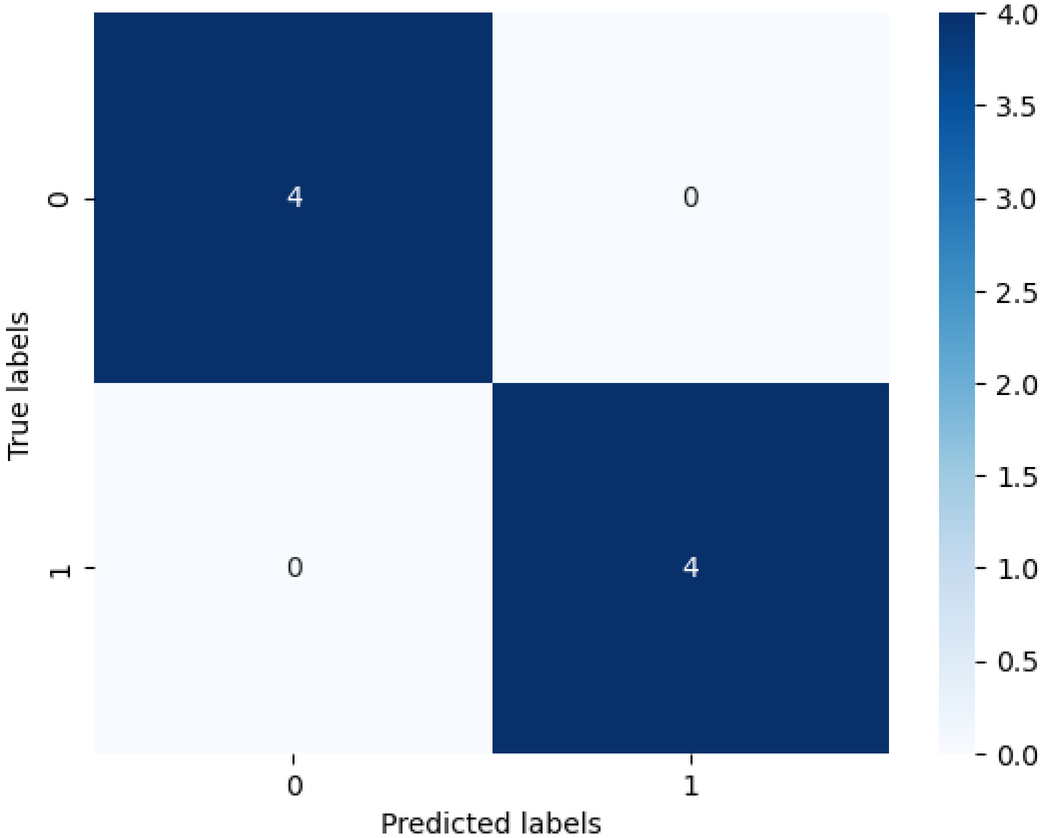
Out[31]:
```
array([[1],
       [1],
       [1],
       [0],
       [1],
       [0],
       [0],
       [0]])
```

In [32]:
```python
# # Select the index of the image you want to plot

# image_index = 1

# # Plot the selected image
# plt.imshow(x_test[image_index])
# plt.show()
# print(y_pred[image_index])
```

Confusion Matrix



In [33]:
```
# create confusion matrix
cm = confusion_matrix(y_test, y_pred)

# plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

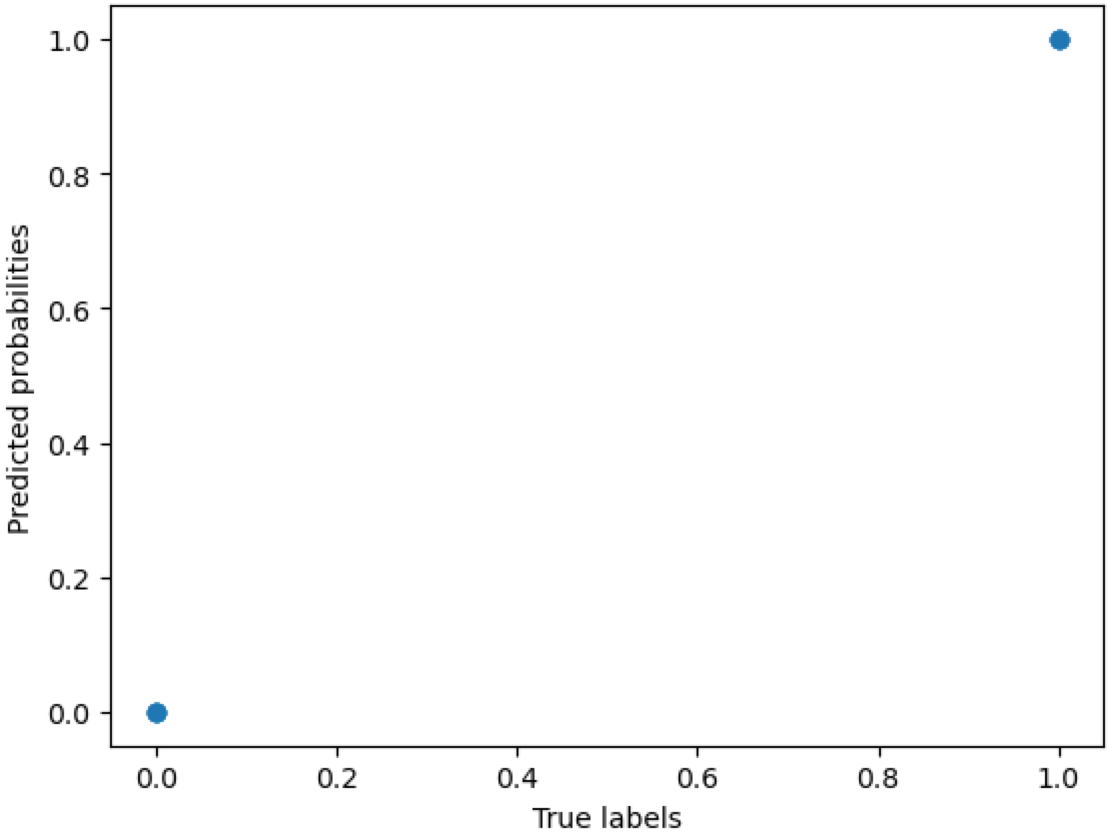


In [34]:
```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
print('True Negative =',tn)
print('False Positive =',fp)
print('False Negative =',fn)
print('True Positive =',tp)
```

```
True Negative = 4
False Positive = 0
False Negative = 0
True Positive = 4
```

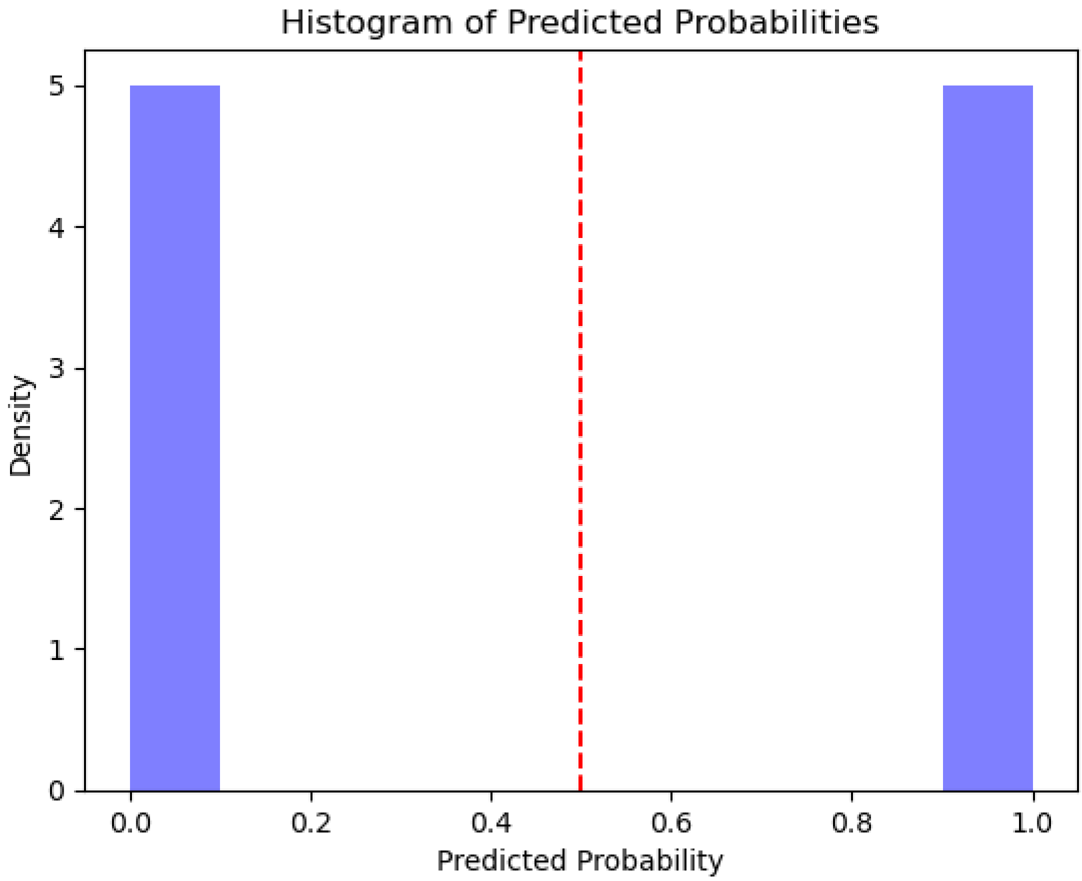
Scatter Plot

In [35]:
```python
import matplotlib.pyplot as plt


# plot scatter plot
plt.scatter(y_test, y_pred)
plt.xlabel('True labels')
plt.ylabel('Predicted probabilities')
plt.show()
```



Histogram Of Predicted Probabilites

In [36]:
```python
# Assuming you have trained a binary classification model and generated predictions
# Store the predicted probabilities in a numpy array called "y_pred"

# Create a histogram of predicted probabilities
n, bins, patches = plt.hist(y_pred, bins=10, range=(0, 1), density=True, alpha=0.5

# Add a vertical line at the 0.5 threshold
plt.axvline(x=0.5, linestyle='--', color='red')

# Add labels and title
plt.xlabel('Predicted Probability')
plt.ylabel('Density')
plt.title('Histogram of Predicted Probabilities')

plt.show()
```



New Image Prediction

In [37]:

```
# from PIL import Image

# # Load the image and convert to RGB
# image = Image.open("C:/#Datasets/Class/test/test_melt0/HW1577.bmp").convert("RGB

# # Resize the image to 128x120 pixels
# image = image.resize((120, 128))

# # Convert the image to a numpy array with shape (1, 128, 120, 3)
# image_array = np.expand_dims(np.asarray(image), axis=0)

# # Save the array to a file
# np.save("image_array.npy", image_array)

# # Print the shape of the array
# print('Shape of img_array:', image_array.shape)

# y_pred = model.predict(image_array)
# int(y_pred)
```

Accuracy

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

$$\text{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{All Samples}}$$

In [38]:
```
acc = (tp + tn) / (tp + tn + fp + fn)


print("Accuracy:", acc)
```

```
Accuracy: 1.0
```

Recall

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(FN)}$$

In [39]:
```
from sklearn.metrics import recall_score

recall = recall_score(y_test, y_pred, average='binary')

# print recall score
print('Recall score:', recall)
```

```
Recall score: 1.0
```

Precision

$$Precision = \frac{True Positive}{True Positive + False Positive}$$

In [40]:
```
from sklearn.metrics import precision_score

precision = precision_score(y_test, y_pred, average='binary')

# print precision score
print('Precision score:', precision)
```

```
Precision score: 1.0
```

F1 Score

$$\text{F1 Score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

$$\text{F1 Score} = \frac{2 \text{ x } (\text{Precision x Recall})}{\text{Precision + Recall}}$$

In [41]:

```
f1_score = 2 * (precision * recall) / (precision + recall)

# print F1 score
print('F1 score:', f1_score)
```

```
F1 score: 1.0
```

Results

In [42]:

```
#n=number of images from melt pool and no melt pool
print('Total number of Images=', 2*v)

#e=number of epochs
print("Epochs=", e)

#Data split

print("Images for Training=",len(x_train))

print("Images for Validation=",len(x_val))

print("Images for Testing=",len(x_test))

print("Labels for Training=",len(y_train))

print("Labels for Validation=",len(y_val))

print("Labels for Training=",len(y_test))


#Validation Accuracy & loss
print('Validation Accuracy=',val_acc)
print('Validation Loss=',int(val_loss))

#Test Accuracy & loss
print ('Test Accuracy=',test_acc)
print('Test Loss=',test_loss)


print('Confusion Matrix')
print(cm)

True_Negative = print('True Negative =',cm[0 , 0])
False_Positive = print('False Positive =',cm[0 , 1])
False_Negative = print('False Negative =',cm[1 , 0])
True_Positive = print('True Positive =',cm[1 , 1])
```

```
print("Accuracy:", acc)
print('Precision score:', precision)
print('Recall score:', recall)
print('F1 score:', f1_score)
```

```
Total number of Images= 40
Epochs= 5
Images for Training= 32
Images for Validation= 8
Images for Testing= 8
Labels for Training= 32
Labels for Validation= 8
Labels for Training= 8
Validation Accuracy= 1.0
Validation Loss= 0
Test Accuracy= 1.0
Test Loss= 0.0
Confusion Matrix
[[4 0]
 [0 4]]
True Negative = 4
False Positive = 0
False Negative = 0
True Positive = 4
Accuracy: 1.0
Precision score: 1.0
Recall score: 1.0
F1 score: 1.0
```

Results save for Analysis and Interpretation

In [43]:

```
from openpyxl import load_workbook

# Create a dictionary with the variable names and their values
results = {'Total Input Images':2*v,'number of filters':z,'filter dimesnion':f,'Tra
           'Testing_Images':len(x_test),'epochs': e,'Accuracy': acc,'Precision': pr
           'F1 Score': f1_score,'Validation Accuracy':'{:.2f}'.format(val_acc),'Va
          'Test_Accuracy':'{:.2f}'.format(test_acc),'Test Loss=':'{:.2f}'.format(te
           "Number of layers":len(model.layers),'Training time in HH:MM:SS': train
           'True Negative':tn,'False Positive':fp,'False Negative':fn,'True Positi

# Load the existing Excel file
workbook = load_workbook(filename="C:\#Datasets\Class\Analysis.xlsx")

# Select the worksheet by name
worksheet = workbook['Analysis1']

# Get the maximum row index
max_row = worksheet.max_row

# Write the headers to the first row
for col, header in enumerate(results.keys(), start=1):
    worksheet.cell(row=1, column=col, value=header)

# Append the new data to the next row
for col, val in enumerate(results.values(), start=1):
    worksheet.cell(row=max_row+1, column=col, value=val)

# Save the changes to the Excel file
workbook.save("C:\#Datasets\Class\Analysis.xlsx")
```

Applying our dataset on Existing Machine Learning model for analysis

In [44]:

```
#Import Necessary Library

import os
import cv2
import numpy as np
import time
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, reca
```

In [45]:

```
# n is number of images from melt pool and no meltpool to train the model

v = int(input("Number of Images from melt pool and no melt pool : "))
total_images_as_input = print("Number of input images to the model : " + str(v*2))
```

```
Number of Images from melt pool and no melt pool : 30
Number of input images to the model : 60
```

Data Preparation for all Machine Learning Models

In [46]:

```
#Preparing data for machine laerning models

#assignning path for melt pool and no melt pool images
melt_pool_folder = "C:/#Datasets/Class/train/train_melt1"
no_melt_pool_folder = "C:/#Datasets/Class/train/train_melt0"

melt_pool_images = []
no_melt_pool_images = []

# read n images from melt pool folder
for i, file in enumerate(os.listdir(melt_pool_folder)):
    if i >= v:
        break
    image = cv2.imread(os.path.join(melt_pool_folder, file))
    if image is not None:
        melt_pool_images.append(image)

# read n images from no melt pool folder
for i, file in enumerate(os.listdir(no_melt_pool_folder)):
    if i >= v:
        break
    image = cv2.imread(os.path.join(no_melt_pool_folder, file))
    if image is not None:
        no_melt_pool_images.append(image)

# convert the lists to numpy arrays and concatenate them
melt_pool_images = np.array(melt_pool_images)
no_melt_pool_images = np.array(no_melt_pool_images)
images = np.concatenate([melt_pool_images, no_melt_pool_images], axis=0)

# generate labels for the data
melt_pool_labels = np.ones(len(melt_pool_images), dtype=int)
no_melt_pool_labels = np.zeros(len(no_melt_pool_images), dtype=int)
labels = np.concatenate([melt_pool_labels, no_melt_pool_labels], axis=0)
```

```
# shuffle the data and labels
shuffled_indices = np.random.permutation(len(images))
images = images[shuffled_indices]
labels = labels[shuffled_indices]

import numpy as np
from sklearn.tree import DecisionTreeClassifier

# Generate sample image data
images = np.random.rand((2*v), 128, 120, 3)

# Flatten images into 2D array
X = images.reshape((2*v), -1)

# check the shape and labels of the data
print("Images shape:", X.shape)
print("Labels shape:", labels.shape)
print("Labels:", labels[0:10])
```

```
Images shape: (60, 46080)
Labels shape: (60,)
Labels: [0 1 1 0 0 0 0 1 1 0]
```

In [47]:
```
# Split dataset into training and testing sets

x_train, x_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, rand
```

1) Decision Tree

In [48]:
```
Model= print('Decision Tree')

# Create decision tree classifier object
model = DecisionTreeClassifier()


start_time = time.time()
# Train K-Nearest Neighbors classifier on training set
model.fit(x_train, y_train)

end_time = time.time()

training_time = end_time - start_time

# Convert training time to hours, minutes, and seconds
hours, rem = divmod(training_time, 3600)
minutes, seconds = divmod(rem, 60)

training_time = "{:0>2}:{:0>2}:{:05.2f}".format(int(hours), int(minutes), seconds)

print("Training time HH:MM:SS:", training_time)

# Test K-Nearest Neighbors classifier on testing set
y_pred = model.predict(x_test)

# Evaluate performance of K-Nearest Neighbors classifier
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", '{:.2f}'.format(accuracy))
print("Precision:", '{:.2f}'.format(precision))
```

```
print("Recall:",  '{:.2f}'.format(recall))
print("F1 Score:", '{:.2f}'.format(f1))

# Calculate the confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

# Calculate the training accuracy and log loss
train_acc = model.score(x_train, y_train)
train_pred = model.predict_proba(x_train)
train_loss = log_loss(y_train, train_pred)

# Calculate the test accuracy and log loss
test_acc = model.score(x_test, y_test)
test_pred = model.predict_proba(x_test)
test_loss = log_loss(y_test, test_pred)

print("Training accuracy: ", train_acc)
print("Training log loss: ", train_loss)
print("Test accuracy: ", test_acc)
print("Test log loss: ", test_loss)

from openpyxl import load_workbook

# Create a dictionary with the variable names and their values
results = {'Model':'Decision Tree','Total Input Images':2*v,'Training_Images':len(
           'Testing_Images':len(x_test),'Accuracy':'{:.2f}'.format(accuracy),'Prec
           'Recall':'{:.2f}'.format(recall), 'F1 Score': '{:.2f}'.format(f1),"Trai
           "Training loss": '{:.2f}'.format(train_loss),'Test_Accuracy':'{:.2f}'.f
           'Training time in HH:MM:SS': training_time,
           'True Negative':tn,'False Positive':fp,'False Negative':fn,'True Positi

# Load the existing Excel file
workbook = load_workbook(filename="C:\#Datasets\Class\Analysis.xlsx")

# Select the worksheet by name
worksheet = workbook['Analysis2']

# Get the maximum row index
max_row = worksheet.max_row

# Write the headers to the first row
for col, header in enumerate(results.keys(), start=1):
    worksheet.cell(row=1, column=col, value=header)

# Append the new data to the next row
for col, val in enumerate(results.values(), start=1):
    worksheet.cell(row=max_row+1, column=col, value=val)

# Save the changes to the Excel file
workbook.save("C:\#Datasets\Class\Analysis.xlsx")
```

```
Decision Tree
Training time HH:MM:SS: 00:00:00.55
Accuracy: 0.67
Precision: 0.33
Recall: 0.33
F1 Score: 0.33
Training accuracy:  1.0
Training log loss:  9.992007221626415e-16
Test accuracy:  0.6666666666666666
Test log loss:  11.512925464970229
```

2) Naive Bayes

In [49]:

```
Model= print('Naive Bayes')

from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor

# Create Naive Bayes classifier object
model = GaussianNB()

start_time = time.time()
# Train K-Nearest Neighbors classifier on training set
model.fit(x_train, y_train)

end_time = time.time()

training_time = end_time - start_time

# Convert training time to hours, minutes, and seconds
hours, rem = divmod(training_time, 3600)
minutes, seconds = divmod(rem, 60)

training_time = "{:0>2}:{:0>2}:{:05.2f}".format(int(hours), int(minutes), seconds)

print("Training time HH:MM:SS:", training_time)

# Test K-Nearest Neighbors classifier on testing set
y_pred = model.predict(x_test)

# Evaluate performance of K-Nearest Neighbors classifier
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", '{:.2f}'.format(accuracy))
print("Precision:", '{:.2f}'.format(precision))
print("Recall:",  '{:.2f}'.format(recall))
print("F1 Score:", '{:.2f}'.format(f1))

# Calculate the confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

# Calculate the training accuracy and log loss
train_acc = model.score(x_train, y_train)
train_pred = model.predict_proba(x_train)
train_loss = log_loss(y_train, train_pred)

# Calculate the test accuracy and log loss
test_acc = model.score(x_test, y_test)
test_pred = model.predict_proba(x_test)
test_loss = log_loss(y_test, test_pred)

print("Training accuracy: ", train_acc)
print("Training log loss: ", train_loss)
print("Test accuracy: ", test_acc)
print("Test log loss: ", test_loss)

from openpyxl import load_workbook

# Create a dictionary with the variable names and their values
results = {'Model':'Naive Bayes','Total Input Images':2*v,'Training_Images':len(x_
           'Testing_Images':len(x_test),'Accuracy':'{:.2f}'.format(accuracy),'Prec
```

```
            'Recall':'{:.2f}'.format(recall), 'F1 Score': '{:.2f}'.format(f1),"Trai
            "Training loss": '{:.2f}'.format(train_loss),'Test_Accuracy':'{:.2f}'.f
            'Training time in HH:MM:SS': training_time,
            'True Negative':tn,'False Positive':fp,'False Negative':fn,'True Positi

# Load the existing Excel file
workbook = load_workbook(filename="C:\#Datasets\Class\Analysis.xlsx")

# Select the worksheet by name
worksheet = workbook['Analysis2']

# Get the maximum row index
max_row = worksheet.max_row

# Write the headers to the first row
for col, header in enumerate(results.keys(), start=1):
    worksheet.cell(row=1, column=col, value=header)

# Append the new data to the next row
for col, val in enumerate(results.values(), start=1):
    worksheet.cell(row=max_row+1, column=col, value=val)

# Save the changes to the Excel file
workbook.save("C:\#Datasets\Class\Analysis.xlsx")
```

```
Naive Bayes
Training time HH:MM:SS: 00:00:00.05
Accuracy: 0.25
Precision: 0.25
Recall: 1.00
F1 Score: 0.40
Training accuracy:  1.0
Training log loss:  9.992007221626415e-16
Test accuracy:  0.25
Test log loss:  25.90408229618301
```

3) Random Forest

In [50]:
```
Model= print('Random Forest')
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor

# Create random forest classifier object
model = RandomForestClassifier()

start_time = time.time()
# Train K-Nearest Neighbors classifier on training set
model.fit(x_train, y_train)

end_time = time.time()

training_time = end_time - start_time

# Convert training time to hours, minutes, and seconds
hours, rem = divmod(training_time, 3600)
minutes, seconds = divmod(rem, 60)

training_time = "{:0>2}:{:0>2}:{:05.2f}".format(int(hours), int(minutes), seconds)

print("Training time HH:MM:SS:", training_time)
```

```
# Test K-Nearest Neighbors classifier on testing set
y_pred = model.predict(x_test)

# Evaluate performance of K-Nearest Neighbors classifier
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", '{:.2f}'.format(accuracy))
print("Precision:", '{:.2f}'.format(precision))
print("Recall:",  '{:.2f}'.format(recall))
print("F1 Score:", '{:.2f}'.format(f1))

# Calculate the confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

# Calculate the training accuracy and log loss
train_acc = model.score(x_train, y_train)
train_pred = model.predict_proba(x_train)
train_loss = log_loss(y_train, train_pred)

# Calculate the test accuracy and log loss
test_acc = model.score(x_test, y_test)
test_pred = model.predict_proba(x_test)
test_loss = log_loss(y_test, test_pred)

print("Training accuracy: ", train_acc)
print("Training log loss: ", train_loss)
print("Test accuracy: ", test_acc)
print("Test log loss: ", test_loss)

from openpyxl import load_workbook

# Create a dictionary with the variable names and their values
results = {'Model':'Random Forest','Total Input Images':2*v,'Training_Images':len(
           'Testing_Images':len(x_test),'Accuracy':'{:.2f}'.format(accuracy),'Prec
           'Recall':'{:.2f}'.format(recall), 'F1 Score': '{:.2f}'.format(f1),"Trai
           "Training loss": '{:.2f}'.format(train_loss),'Test_Accuracy':'{:.2f}'.fo
           'Training time in HH:MM:SS': training_time,
           'True Negative':tn,'False Positive':fp,'False Negative':fn,'True Positi

# Load the existing Excel file
workbook = load_workbook(filename="C:\#Datasets\Class\Analysis.xlsx")

# Select the worksheet by name
worksheet = workbook['Analysis2']

# Get the maximum row index
max_row = worksheet.max_row

# Write the headers to the first row
for col, header in enumerate(results.keys(), start=1):
    worksheet.cell(row=1, column=col, value=header)

# Append the new data to the next row
for col, val in enumerate(results.values(), start=1):
    worksheet.cell(row=max_row+1, column=col, value=val)

# Save the changes to the Excel file
workbook.save("C:\#Datasets\Class\Analysis.xlsx")
```

```
Random Forest
Training time HH:MM:SS: 00:00:00.43
Accuracy: 0.33
Precision: 0.27
Recall: 1.00
F1 Score: 0.43
Training accuracy:  1.0
Training log loss:  0.20217497941258408
Test accuracy:  0.3333333333333333
Test log loss:  0.7330078828781721
```

4) Logistic Regression

In [51]:

```
Model= print('Logistic Regression')
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor

# Create logistic regression classifier object
model = LogisticRegression()
start_time = time.time()
# Train K-Nearest Neighbors classifier on training set
model.fit(x_train, y_train)

end_time = time.time()

training_time = end_time - start_time

# Convert training time to hours, minutes, and seconds
hours, rem = divmod(training_time, 3600)
minutes, seconds = divmod(rem, 60)

training_time = "{:0>2}:{:0>2}:{:05.2f}".format(int(hours), int(minutes), seconds)

print("Training time HH:MM:SS:", training_time)

# Test K-Nearest Neighbors classifier on testing set
y_pred = model.predict(x_test)

# Evaluate performance of K-Nearest Neighbors classifier
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", '{:.2f}'.format(accuracy))
print("Precision:", '{:.2f}'.format(precision))
print("Recall:",  '{:.2f}'.format(recall))
print("F1 Score:", '{:.2f}'.format(f1))

# Calculate the confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

# Calculate the training accuracy and log loss
train_acc = model.score(x_train, y_train)
train_pred = model.predict_proba(x_train)
train_loss = log_loss(y_train, train_pred)

# Calculate the test accuracy and log loss
test_acc = model.score(x_test, y_test)
test_pred = model.predict_proba(x_test)
test_loss = log_loss(y_test, test_pred)
```

```
print("Training accuracy: ", train_acc)
print("Training log loss: ", train_loss)
print("Test accuracy: ", test_acc)
print("Test log loss: ", test_loss)

from openpyxl import load_workbook

# Create a dictionary with the variable names and their values
results = {'Model':'Logistic Regression','Total Input Images':2*v,'Training_Images
           'Testing_Images':len(x_test),'Accuracy':'{:.2f}'.format(accuracy),'Prec
           'Recall':'{:.2f}'.format(recall), 'F1 Score': '{:.2f}'.format(f1),"Trai
           "Training loss": '{:.2f}'.format(train_loss),'Test_Accuracy':'{:.2f}'.f
           'Training time in HH:MM:SS': training_time,
           'True Negative':tn,'False Positive':fp,'False Negative':fn,'True Positi

# Load the existing Excel file
workbook = load_workbook(filename="C:\#Datasets\Class\Analysis.xlsx")

# Select the worksheet by name
worksheet = workbook['Analysis2']

# Get the maximum row index
max_row = worksheet.max_row

# Write the headers to the first row
for col, header in enumerate(results.keys(), start=1):
    worksheet.cell(row=1, column=col, value=header)

# Append the new data to the next row
for col, val in enumerate(results.values(), start=1):
    worksheet.cell(row=max_row+1, column=col, value=val)

# Save the changes to the Excel file
workbook.save("C:\#Datasets\Class\Analysis.xlsx")
```

```
Logistic Regression
Training time HH:MM:SS: 00:00:00.50
Accuracy: 0.25
Precision: 0.25
Recall: 1.00
F1 Score: 0.40
Training accuracy:  1.0
Training log loss:  0.0016383819188388101
Test accuracy:  0.25
Test log loss:  1.0373960240750149
```

5) K-Nearest Neighbors

In [52]:

```
Model=print('K-Nearest Neighbors')
import time
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, reca
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Create K-Nearest Neighbors classifier object
model = KNeighborsClassifier()

start_time = time.time()
# Train K-Nearest Neighbors classifier on training set
model.fit(x_train, y_train)
```

```python
end_time = time.time()

training_time = end_time - start_time

# Convert training time to hours, minutes, and seconds
hours, rem = divmod(training_time, 3600)
minutes, seconds = divmod(rem, 60)

training_time = "{:0>2}:{:0>2}:{:05.2f}".format(int(hours), int(minutes), seconds)

print("Training time HH:MM:SS:", training_time)

# Test K-Nearest Neighbors classifier on testing set
y_pred = model.predict(x_test)

# Evaluate performance of K-Nearest Neighbors classifier
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", '{:.2f}'.format(accuracy))
print("Precision:", '{:.2f}'.format(precision))
print("Recall:",  '{:.2f}'.format(recall))
print("F1 Score:", '{:.2f}'.format(f1))

# Calculate the confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

# Calculate the training accuracy and log loss
train_acc = model.score(x_train, y_train)
train_pred = model.predict_proba(x_train)
train_loss = log_loss(y_train, train_pred)

# Calculate the test accuracy and log loss
test_acc = model.score(x_test, y_test)
test_pred = model.predict_proba(x_test)
test_loss = log_loss(y_test, test_pred)

print("Training accuracy: ", train_acc)
print("Training log loss: ", train_loss)
print("Test accuracy: ", test_acc)
print("Test log loss: ", test_loss)

from openpyxl import load_workbook

# Create a dictionary with the variable names and their values
results = {'Model':'K-Nearest Neighbors','Total Input Images':2*v,'Training_Images
           'Testing_Images':len(x_test),'Accuracy':'{:.2f}'.format(accuracy),'Prec
           'Recall':'{:.2f}'.format(recall), 'F1 Score': '{:.2f}'.format(f1),"Trai
           "Training loss": '{:.2f}'.format(train_loss),'Test_Accuracy':'{:.2f}'.f
           'Training time in HH:MM:SS': training_time,
           'True Negative':tn,'False Positive':fp,'False Negative':fn,'True Positi

# Load the existing Excel file
workbook = load_workbook(filename="C:\#Datasets\Class\Analysis.xlsx")

# Select the worksheet by name
worksheet = workbook['Analysis2']

# Get the maximum row index
max_row = worksheet.max_row

# Write the headers to the first row
```

```
for col, header in enumerate(results.keys(), start=1):
    worksheet.cell(row=1, column=col, value=header)

# Append the new data to the next row
for col, val in enumerate(results.values(), start=1):
    worksheet.cell(row=max_row+1, column=col, value=val)

# Save the changes to the Excel file
workbook.save("C:\#Datasets\Class\Analysis.xlsx")
```

```
K-Nearest Neighbors
Training time HH:MM:SS: 00:00:00.00
Accuracy: 0.25
Precision: 0.12
Recall: 0.33
F1 Score: 0.18
Training accuracy:  0.6458333333333334
Training log loss:  0.6181347372785942
Test accuracy:  0.25
Test log loss:  1.1952868041361
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:22
8: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.
11.0, this behavior will change: the default value of `keepdims` will become Fals
e, the `axis` over which the statistic is taken will be eliminated, and the value
None will no longer be accepted. Set `keepdims` to True or False to avoid this war
ning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\User\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:22
8: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.
11.0, this behavior will change: the default value of `keepdims` will become Fals
e, the `axis` over which the statistic is taken will be eliminated, and the value
None will no longer be accepted. Set `keepdims` to True or False to avoid this war
ning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\User\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:22
8: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the
default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.
11.0, this behavior will change: the default value of `keepdims` will become Fals
e, the `axis` over which the statistic is taken will be eliminated, and the value
None will no longer be accepted. Set `keepdims` to True or False to avoid this war
ning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

6) Support Vector Machine

In [53]:

```
Model= print('Support Vector Machine')

from sklearn.svm import SVC
from sklearn import svm
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor

# Create support vector classifier object
# Creating an instance of SVM with probability estimates
model = svm.SVC(kernel='linear', C=1, probability=True)

start_time = time.time()
# Train K-Nearest Neighbors classifier on training set
model.fit(x_train, y_train)
```

```
end_time = time.time()

training_time = end_time - start_time

# Convert training time to hours, minutes, and seconds
hours, rem = divmod(training_time, 3600)
minutes, seconds = divmod(rem, 60)

training_time = "{:0>2}:{:0>2}:{:05.2f}".format(int(hours), int(minutes), seconds)

print("Training time HH:MM:SS:", training_time)

# Test K-Nearest Neighbors classifier on testing set
y_pred = model.predict(x_test)

# Evaluate performance of K-Nearest Neighbors classifier
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", '{:.2f}'.format(accuracy))
print("Precision:", '{:.2f}'.format(precision))
print("Recall:",  '{:.2f}'.format(recall))
print("F1 Score:", '{:.2f}'.format(f1))

# Calculate the confusion matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

# Calculate the training accuracy and log loss
train_acc = model.score(x_train, y_train)
train_pred = model.predict_proba(x_train)
train_loss = log_loss(y_train, train_pred)

# Calculate the test accuracy and log loss
test_acc = model.score(x_test, y_test)
test_pred = model.predict_proba(x_test)
test_loss = log_loss(y_test, test_pred)

print("Training accuracy: ", train_acc)
print("Training log loss: ", train_loss)
print("Test accuracy: ", test_acc)
print("Test log loss: ", test_loss)

from openpyxl import load_workbook

# Create a dictionary with the variable names and their values
results = {'Model':'Support Vector Machine','Total Input Images':2*v,'Training_Imag
           'Testing_Images':len(x_test),'Accuracy':'{:.2f}'.format(accuracy),'Prec
           'Recall':'{:.2f}'.format(recall), 'F1 Score': '{:.2f}'.format(f1),"Trai
           "Training loss": '{:.2f}'.format(train_loss),'Test_Accuracy':'{:.2f}'.f
           'Training time in HH:MM:SS': training_time,
           'True Negative':tn,'False Positive':fp,'False Negative':fn,'True Positi

# Load the existing Excel file
workbook = load_workbook(filename="C:\#Datasets\Class\Analysis.xlsx")

# Select the worksheet by name
worksheet = workbook['Analysis2']

# Get the maximum row index
max_row = worksheet.max_row
```

```python
# Write the headers to the first row
for col, header in enumerate(results.keys(), start=1):
    worksheet.cell(row=1, column=col, value=header)

# Append the new data to the next row
for col, val in enumerate(results.values(), start=1):
    worksheet.cell(row=max_row+1, column=col, value=val)

# Save the changes to the Excel file
workbook.save("C:\#Datasets\Class\Analysis.xlsx")
```

```
Support Vector Machine
Training time HH:MM:SS: 00:00:01.25
Accuracy: 0.25
Precision: 0.25
Recall: 1.00
F1 Score: 0.40
Training accuracy:  1.0
Training log loss:  2.2612508273730945
Test accuracy:  0.25
Test log loss:  0.7590746595394711
```

In [ ]:

```python

```