Sperm morphology types

By Mohamed Jamyl

http://linkedin.com/in/mohamed-jamyl

https://www.kaggle.com/mohamedjamyl

https://github.com/Mohamed-Jamyl

Project Overview

1. Normal

- **Shape:** Oval head with smooth contours.
- **Significance:** Indicates healthy sperm morphology, capable of fertilizing an egg effectively.
- **Key Features:** Balanced symmetry, no deformation.

2. Tapered

- **Shape:** Head is elongated and narrow, tapering toward the tip.
- **Significance:** Abnormal morphology, usually linked with reduced motility and fertilization potential.
- **Key Features:** Cone-like or stretched head.

3. Pyriform

- **Shape:** Pear-shaped head (broader at one end, narrowing at the other).
- **Significance:** Considered abnormal; associated with chromatin or structural defects.
- **Key Features:** Wider base, narrowed tip (like a teardrop).

4. Amorphous

- **Shape:** Irregular, distorted, or asymmetrical head without a defined shape.
- **Significance:** Severely abnormal morphology, usually infertile sperm.
- **Key Features:** Undefined structure, rough edges, lacks symmetry.

Import Libraries

In [1]:

```
import os
import shutil
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import cv2
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

import tensorflow as tf
import keras

from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix

from tqdm import tqdm
%matplotlib inline

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

```
2025-09-11 11:04:44.275527: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cu
FFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1757588684.306682    1297 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register
factory for plugin cuDNN when one has already been registered
E0000 00:00:1757588684.315268    1297 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to regist
er factory for plugin cuBLAS when one has already been registered
```

In [2]:

```
output_dir = "/kaggle/working/"

for f in os.listdir(output_dir):
    file_path = os.path.join(output_dir, f)
    try:
        if os.path.isfile(file_path) or os.path.islink(file_path):
            os.remove(file_path)  # remove file or link
        elif os.path.isdir(file_path):
            shutil.rmtree(file_path)  # remove folder
    except Exception as e:
        print(f"Error deleting {file_path}: {e}")
```

In [3]:

```
data_dir = '/kaggle/input/hushem-dataset/HuSHem'
```

In [4]:

```
for fold in os.listdir(data_dir):
    print(fold)
```

```
01_Normal
02_Tapered
04_Amorphous
03_Pyriform
```

In [5]:

```
folds = [fold for fold in os.listdir(data_dir)]
folds
```

Out[5]:

```
['01_Normal', '02_Tapered', '04_Amorphous', '03_Pyriform']
```

In [6]:

```
x = 0
data_dirr = []
for f in folds:
    x += 1
    data_dirr.append(data_dir +'/'+f)
    if x == 4:
        break
```

In [7]:

```
data_dirr
```

Out[7]:

```
['/kaggle/input/hushem-dataset/HuSHem/01_Normal',
 '/kaggle/input/hushem-dataset/HuSHem/02_Tapered',
 '/kaggle/input/hushem-dataset/HuSHem/04_Amorphous',
 '/kaggle/input/hushem-dataset/HuSHem/03_Pyriform']
```

In [8]:

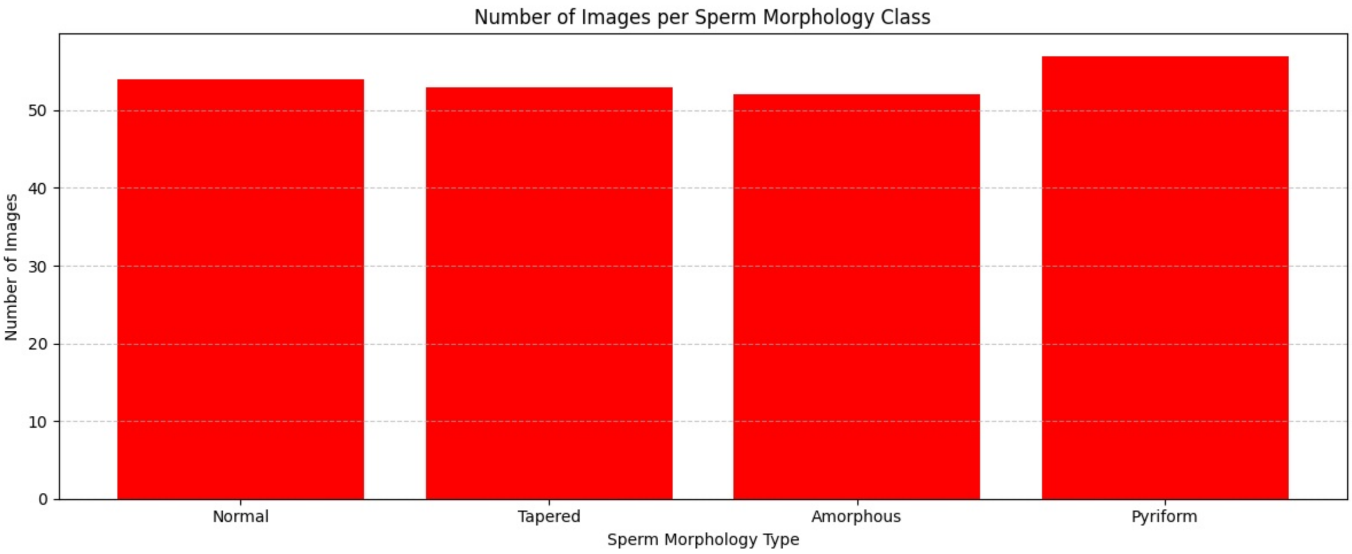
```
for fold in data_dirr:
    print(fold.split('/')[5],' : ', len(os.listdir(fold)))
```

```
01_Normal  :  54
02_Tapered  :  53
04_Amorphous  :  52
03_Pyriform  :  57
```

In [9]:

```
labels = ['Normal', 'Tapered', 'Amorphous', 'Pyriform']
values = [54, 53, 52, 57]

plt.figure(figsize=(12, 5))
plt.bar(labels, values, color='red')
plt.title('Number of Images per Sperm Morphology Class')
plt.xlabel('Sperm Morphology Type')
plt.ylabel('Number of Images')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Creating images by using Data augmentation

In [10]:

```
os.mkdir("/kaggle/working/1-Normal")
os.mkdir("/kaggle/working/2-Tapered")
os.mkdir("/kaggle/working/3-Amorphous")
os.mkdir("/kaggle/working/4-Pyriform")
```

In [11]:

```
data_dirr
```

Out[11]:

```
['/kaggle/input/hushem-dataset/HuSHem/01_Normal',
 '/kaggle/input/hushem-dataset/HuSHem/02_Tapered',
 '/kaggle/input/hushem-dataset/HuSHem/04_Amorphous',
 '/kaggle/input/hushem-dataset/HuSHem/03_Pyriform']
```

In [12]:

```
df = '/kaggle/working'
folds2 = [fold for fold in os.listdir(df)]

x = 0
df2 = []
for f in folds2:
    x += 1
    df2.append(df +'/'+f)
    if x == 4:
        break
```

In [13]:

```
df2 = sorted(df2)
```

In [14]:

```
df2
```

Out[14]:

```
['/kaggle/working/1-Normal',
 '/kaggle/working/2-Tapered',
 '/kaggle/working/3-Amorphous',
 '/kaggle/working/4-Pyriform']
```

In [15]:

```
#src_folder1 = "/kaggle/input/hushem-dataset/HuSHem/01_Normal"
#dst_folder1 = "/kaggle/working/Normal"

#for file_name in os.listdir(src_folder1):
#    src_path = os.path.join(src_folder1, file_name)
#    dst_path = os.path.join(dst_folder1, file_name)

#    if os.path.isfile(src_path):
#        shutil.copy(src_path, dst_path)
```

In [16]:

```
x = 0
for i in range(0, len(data_dirr)):
    x+=1
    for file_name in os.listdir(data_dirr[i]):
        src_path = os.path.join(data_dirr[i], file_name)
        dst_path = os.path.join(df2[i], file_name)

        if os.path.isfile(src_path):
            shutil.copy(src_path, dst_path)
    if x == 4:
        break
```

In [17]:

```
for fold in df2:
    print(fold.split('/')[3],' : ', len(os.listdir(fold)))
```

```
1-Normal  :  54
2-Tapered  :  53
3-Amorphous  :  52
4-Pyriform  :  57
```

In [18]:

```
def datagen1(folder):
    datagen1 = ImageDataGenerator(
        rotation_range = -90,
        shear_range = 0.3,
        zoom_range = 0.3)
    z = 0
    for img in os.listdir(folder):
        z += 1
        x = cv2.imread(os.path.join(folder, img))
        x = tf.keras.utils.img_to_array(x)
        x = x.reshape((1, ) + x.shape)
        i = 0
        for batch in datagen1.flow(x, batch_size = 1,save_to_dir = folder,
                                   save_prefix ='image', save_format ='jpeg'):
            i += 1
            if i == 1:
                break

        if z == 1:
            break

def datagen2(folder):
    datagen2 = ImageDataGenerator(
        rotation_range = -40,
        shear_range = 0.3,
        zoom_range = 0.3)
    z = 0
    for img in os.listdir(folder):
        z += 1
        x = cv2.imread(os.path.join(folder, img))
        x = tf.keras.utils.img_to_array(x)
        x = x.reshape((1, ) + x.shape)
        i = 0
        for batch in datagen2.flow(x, batch_size = 1,save_to_dir = folder,
                                   save_prefix ='image', save_format ='jpeg'):
            i += 1
            if i == 1:
                break

        if z == 1:
            break

def datagen3(folder):
    datagen3 = ImageDataGenerator(
        rotation_range = -30,
        shear_range = 0.3,
        zoom_range = 0.3)
    z = 0
    for img in os.listdir(folder):
        z += 1
        x = cv2.imread(os.path.join(folder, img))
        x = tf.keras.utils.img_to_array(x)
        x = x.reshape((1, ) + x.shape)
        i = 0
        for batch in datagen3.flow(x, batch_size = 1,save_to_dir = folder,
                                   save_prefix ='image', save_format ='jpeg'):
            i += 1
            if i == 1:
                break

        if z == 1:
            break


def datagen4(folder):
    datagen4 = ImageDataGenerator(
        rotation_range = -30,
        shear_range = 0.3,
        zoom_range = 0.4)
    z = 0
    for img in os.listdir(folder):
        z += 1
        x = cv2.imread(os.path.join(folder, img))
        x = tf.keras.utils.img_to_array(x)
        x = x.reshape((1, ) + x.shape)
        i = 0
        for batch in datagen4.flow(x, batch_size = 1,save_to_dir = folder,
                            save_prefix ='image', save_format ='jpeg'):
            i += 1
            if i == 1:
                break

        if z == 1:
            break


def datagen5(folder):
    datagen5 = ImageDataGenerator(
            rotation_range = -30,
            shear_range = 0.3,
            zoom_range = 0.5)
    z = 0
    for img in os.listdir(folder):
        z += 1
        x = cv2.imread(os.path.join(folder, img))
        x = tf.keras.utils.img_to_array(x)
        x = x.reshape((1, ) + x.shape)
        i = 0
        for batch in datagen5.flow(x, batch_size = 1,save_to_dir = folder,
                            save_prefix ='image', save_format ='jpeg'):
            i += 1
            if i == 1:
                break

        if z == 1:
            break
```

In [19]:

```
# Normal
datagen1(df2[0])
datagen2(df2[0])
datagen3(df2[0])
```

In [20]:

```
# Tapered
datagen1(df2[1])
datagen2(df2[1])
datagen3(df2[1])
datagen4(df2[1])
```

In [21]:

```
# Amorphous
datagen1(df2[2])
datagen2(df2[2])
datagen3(df2[2])
datagen4(df2[2])
datagen5(df2[2])
```

In [22]:

```
for fold in df2:
    print(fold.split('/')[3],' : ', len(os.listdir(fold)))
```

```
1-Normal  :  57
2-Tapered  :  57
3-Amorphous  :  57
4-Pyriform  :  57
```

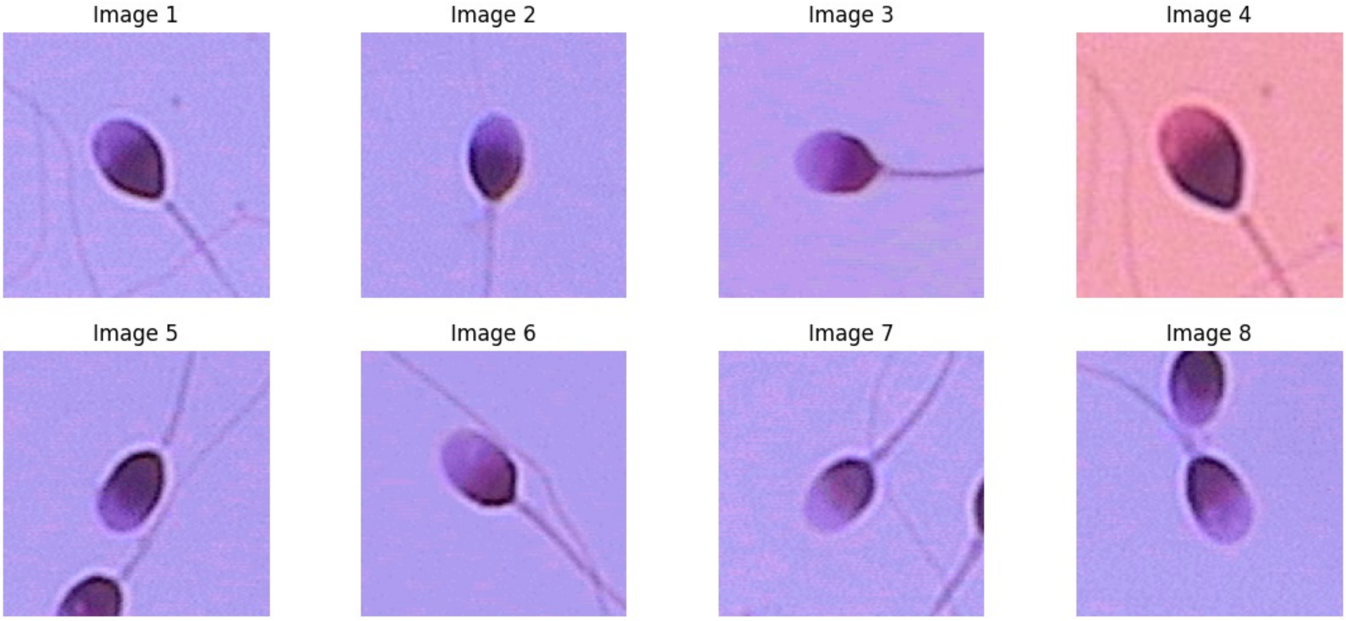
Show images

In [23]:

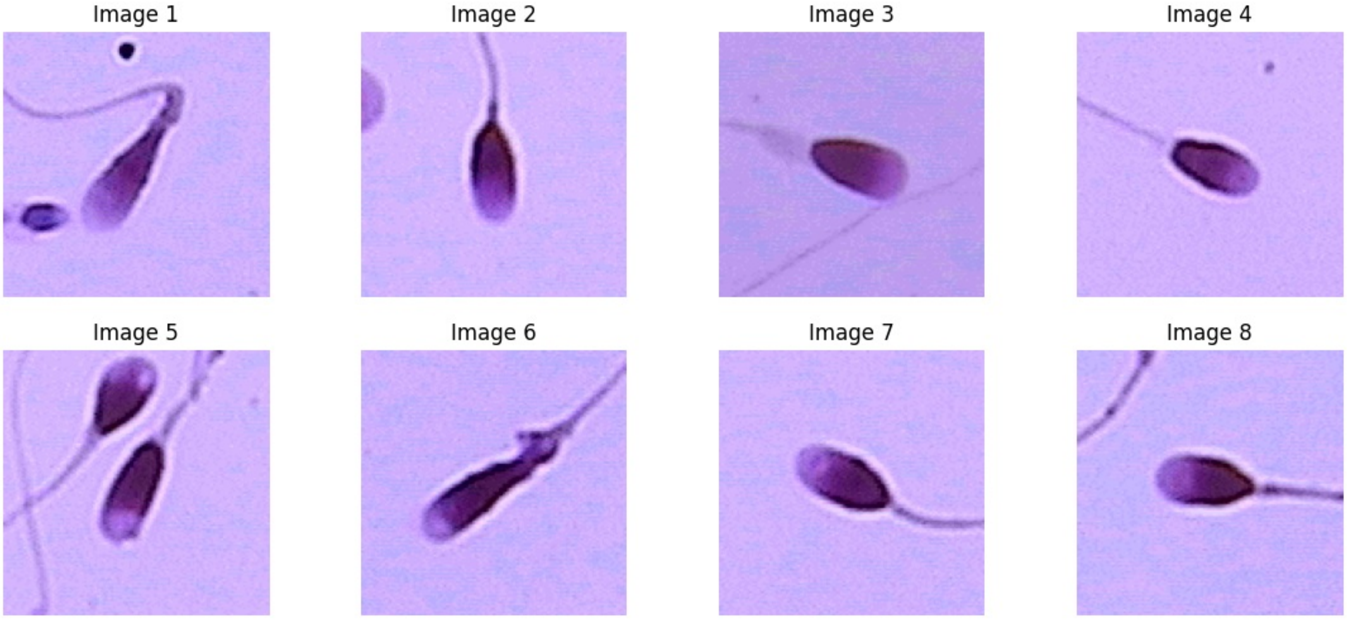
```
def show_img(folder):
    images = [cv2.imread(os.path.join(folder, img)) for img in os.listdir(folder)]
    fig = plt.figure(figsize=(14, 6))
    x = 0
    for i in range(len(images)):
        x+=1
        plt.subplot(2,4,i+1)
        plt.imshow(images[i])
        plt.axis('off')
        plt.title(f'Image {i+1}')
        if x == 8:
            break
```

In [24]:

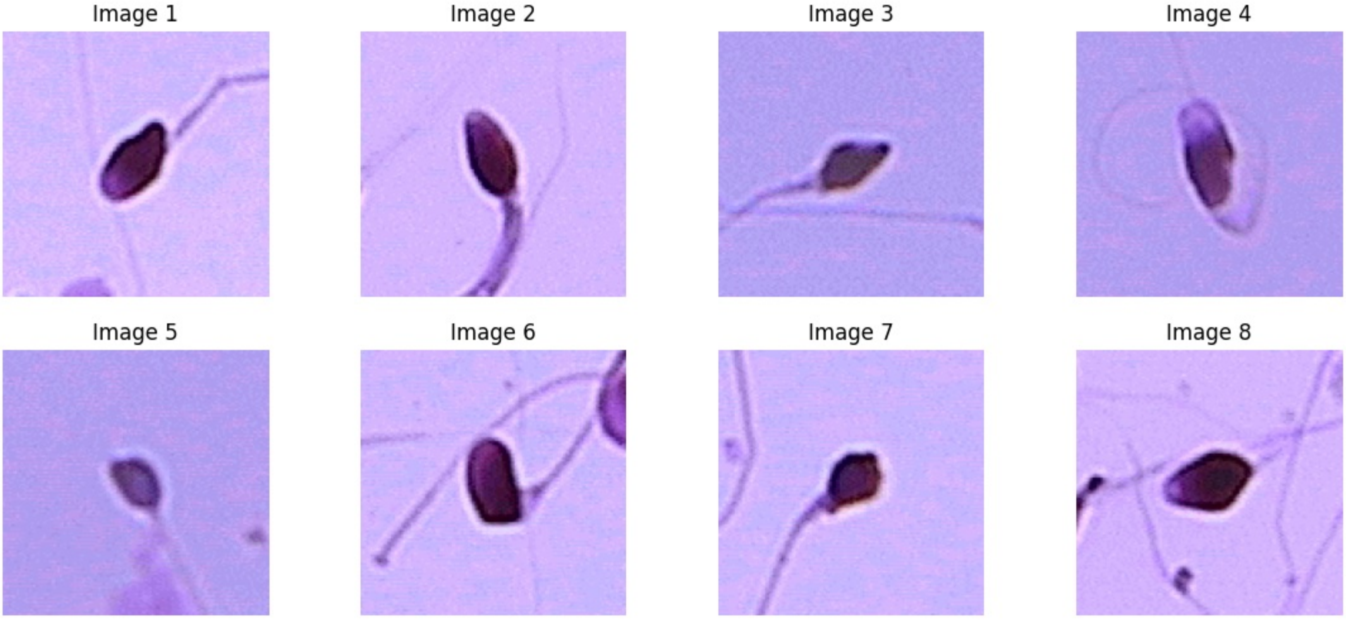
```
# Normal
show_img(df2[0])
```



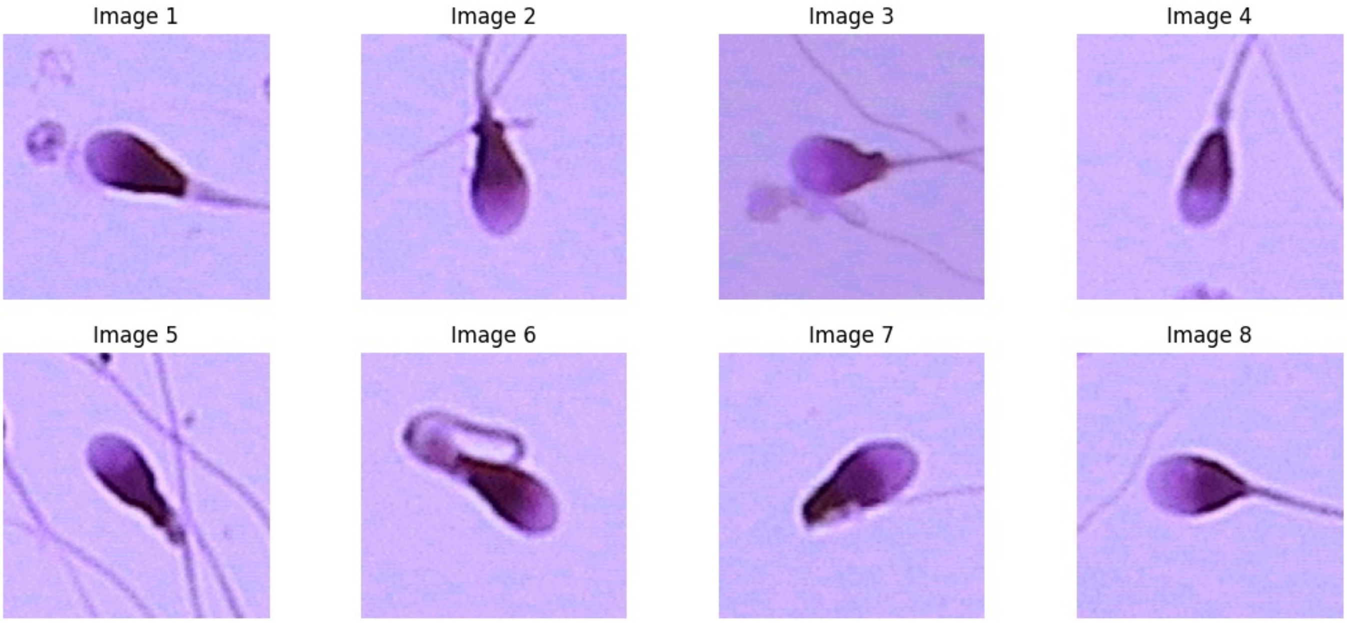
In [25]:
```
# Tapered
show_img(df2[1])
```



In [26]:
```
# Amorphous
show_img(df2[2])
```



In [27]:
```
# Pyriform
show_img(df2[3])
```



Checking sizes of images

```
In [28]: def checking_size(folder):
             AllSizes = [(cv2.imread(os.path.join(folder, img))).shape for img in os.listdir(folder)]
             print(set(AllSizes))
```

```
In [29]: checking_size(df2[0])
```

```
{(120, 131, 3), (131, 131, 3), (127, 131, 3)}
```

```
In [30]: checking_size(df2[1])
```

```
{(131, 131, 3), (131, 118, 3)}
```

```
In [31]: checking_size(df2[2])
```

```
{(123, 131, 3), (124, 131, 3), (131, 131, 3)}
```

```
In [32]: checking_size(df2[3])
```

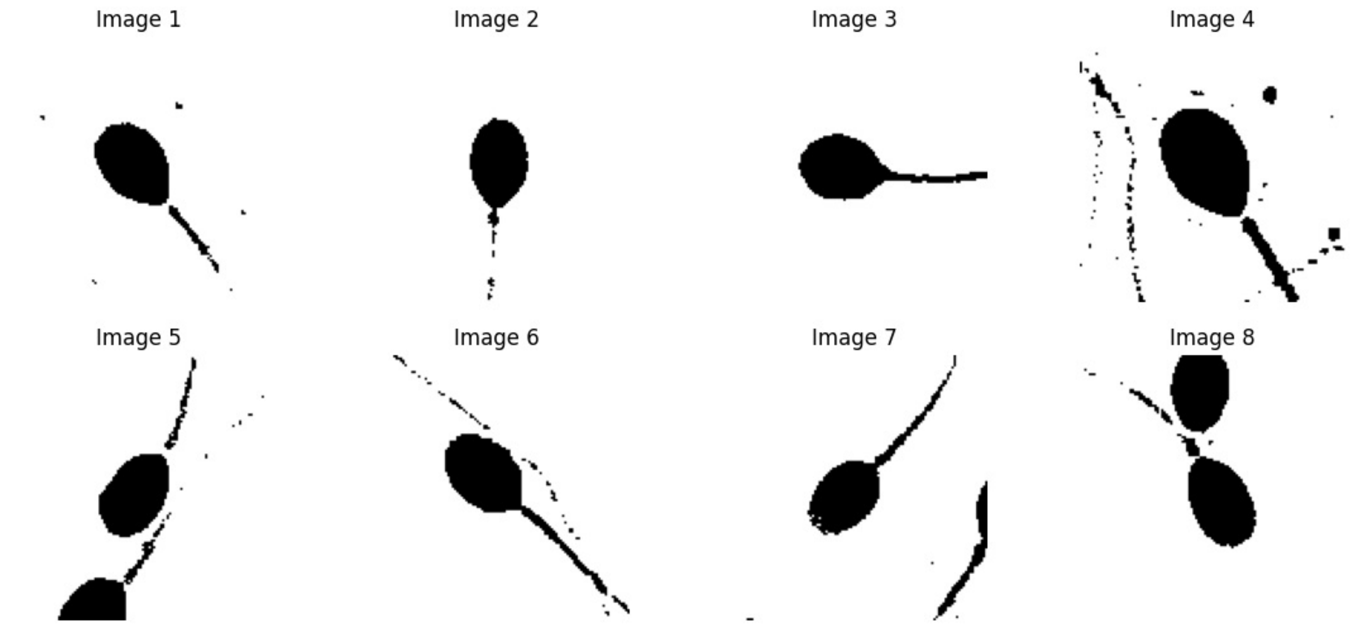
```
{(118, 131, 3), (131, 131, 3)}
```

Resizing & Masking images

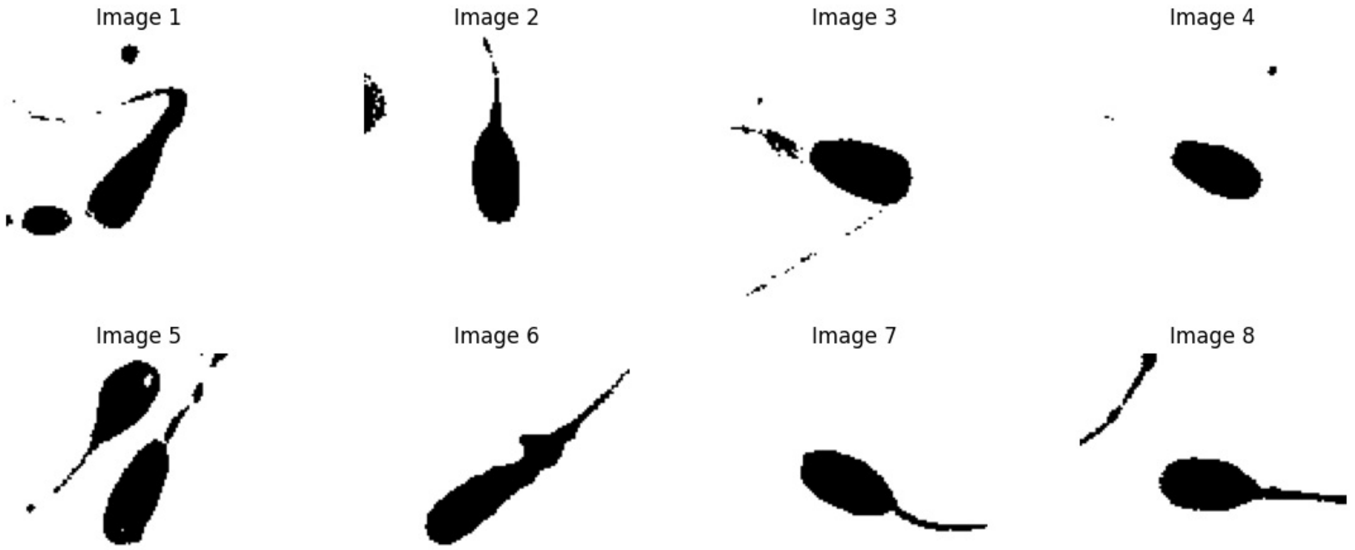
```
In [33]: img_size = 118
```

```
In [34]: def show_img_mask(folder):
             images = [cv2.imread(os.path.join(folder, img)) for img in os.listdir(folder)]
             fig = plt.figure(figsize=(14, 6))
             x = 0
             for i in range(len(images)):
                 x+=1
                 plt.subplot(2,4,i+1)
                 resize_img = cv2.resize(images[i], (img_size, img_size))
                 img_gray = cv2.cvtColor(resize_img, cv2.COLOR_BGR2GRAY)
                 _, mask = cv2.threshold(img_gray, 166, 255, cv2.THRESH_BINARY)
                 plt.imshow(mask,cmap='gray')
                 plt.axis('off')
                 plt.title(f'Image {i+1}')
                 if x == 8:
                     break
```

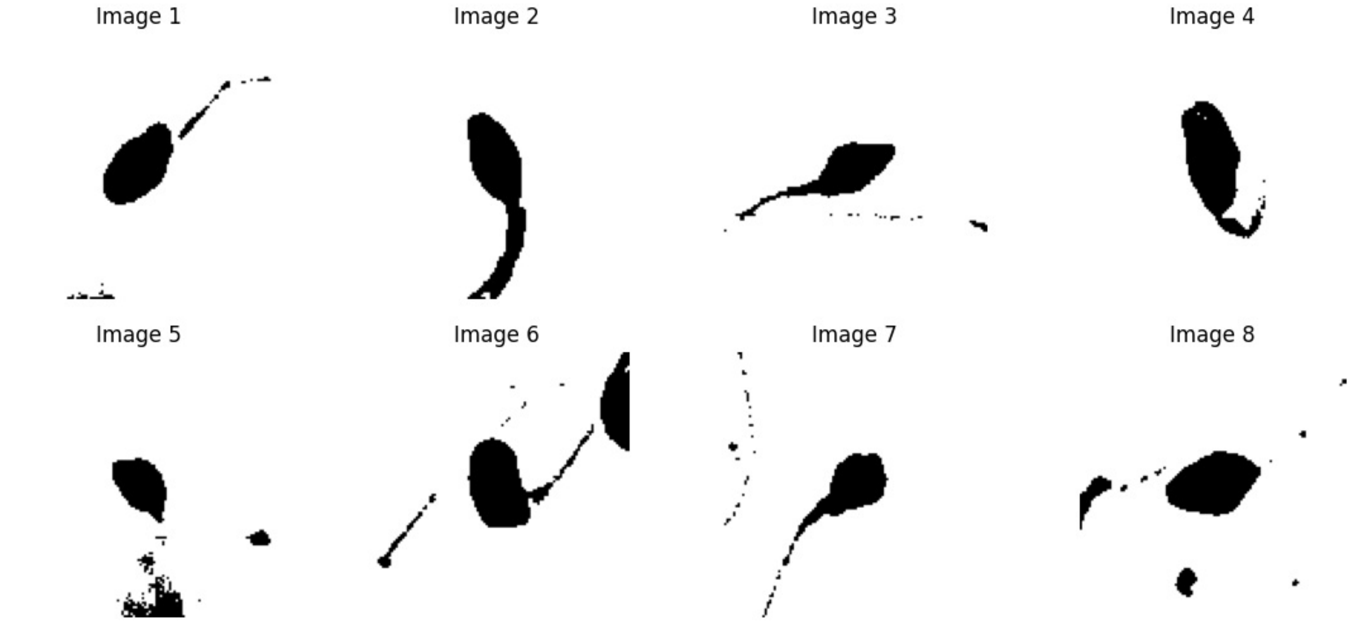
```
In [35]: show_img_mask(df2[0])
```



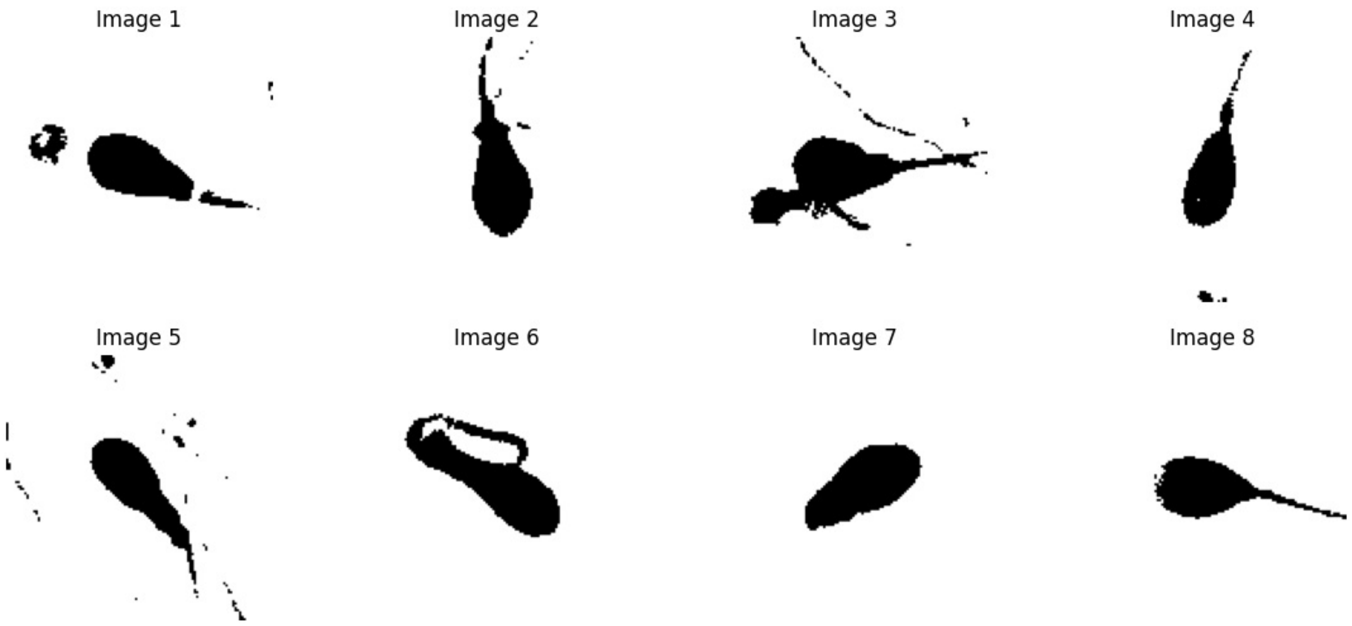
```
In [36]: show_img_mask(df2[1])
```



```
In [37]: show_img_mask(df2[2])
```



```
In [38]: show_img_mask(df2[3])
```



Show and inverting images with closing (Morphological Transformations)

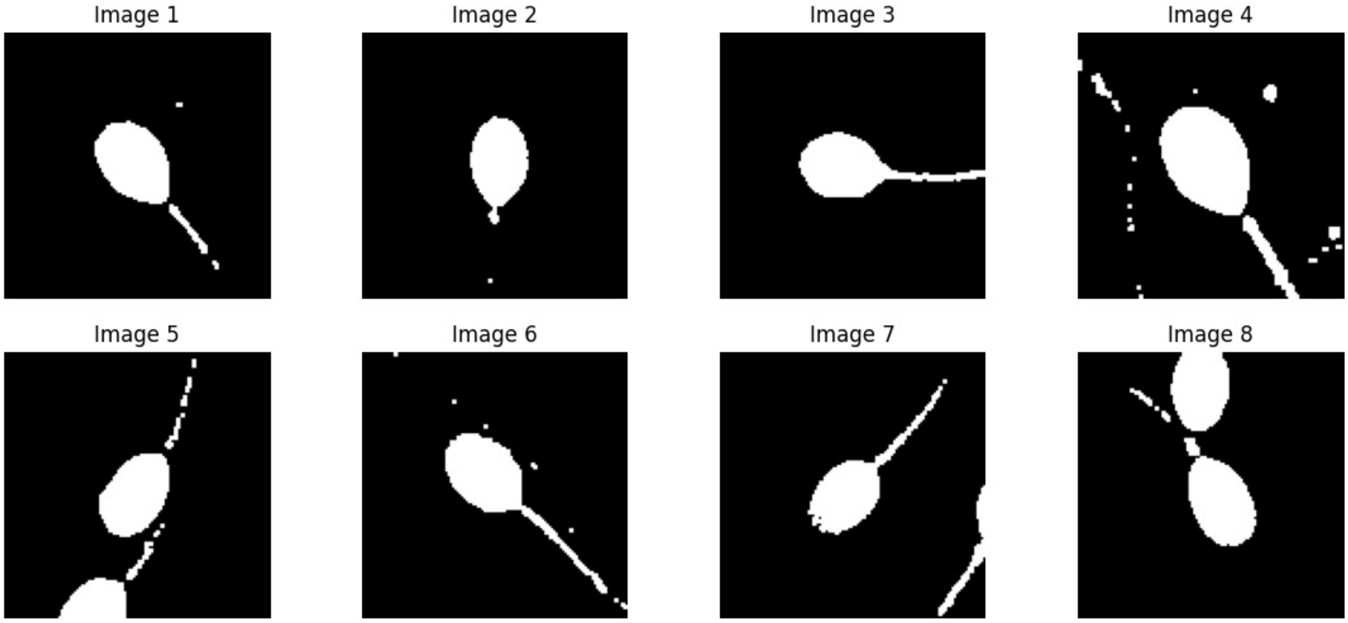
In [39]:

```python
def show_img_inverting_closing(folder):
    images = [cv2.imread(os.path.join(folder, img)) for img in os.listdir(folder)]
    fig = plt.figure(figsize=(14, 6))
    x = 0
    for i in range(len(images)):
        x+=1
        plt.subplot(2,4,i+1)
        resize_img = cv2.resize(images[i], (img_size, img_size))
        img_gray = cv2.cvtColor(resize_img, cv2.COLOR_BGR2GRAY)
        _, mask = cv2.threshold(img_gray, 166, 255, cv2.THRESH_BINARY)
        kernal = np.ones((2,2), np.uint8)
        closing = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernal)
        inverted_image = cv2.bitwise_not(closing)

        plt.imshow(inverted_image, cmap='gray')
        plt.axis('off')
        plt.title(f'Image {i+1}')
        if x == 8:
            break
```

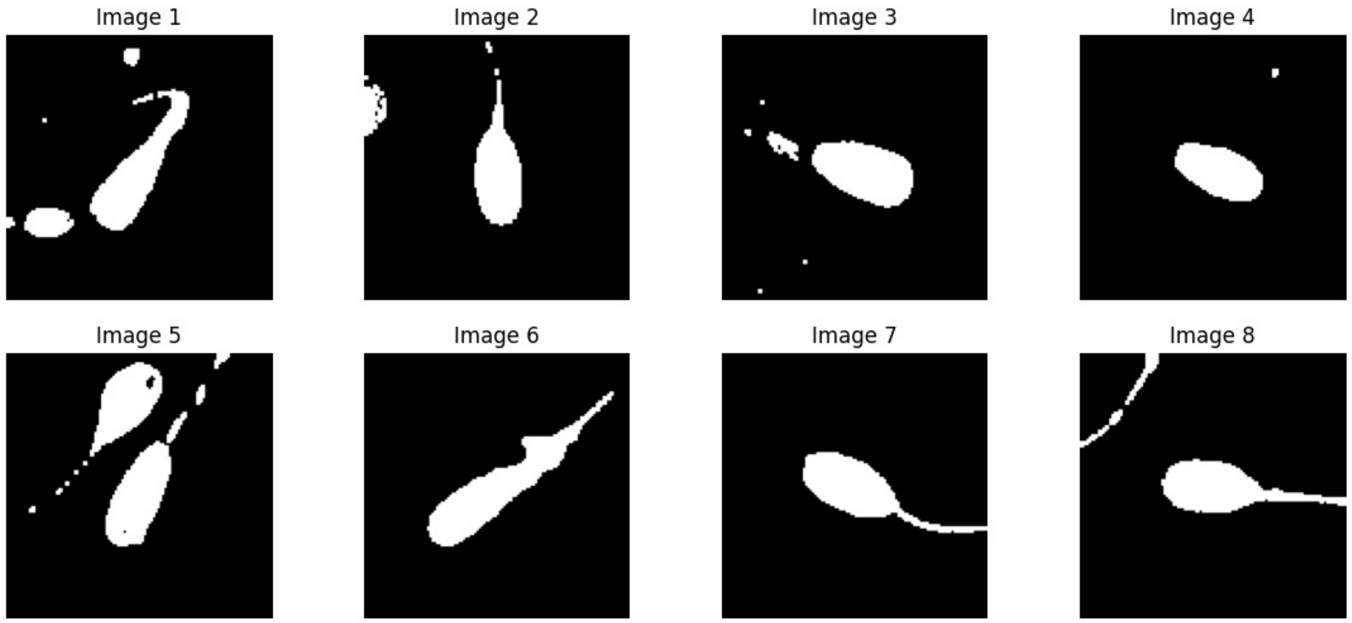
In [40]:

```python
show_img_inverting_closing(df2[0])
```

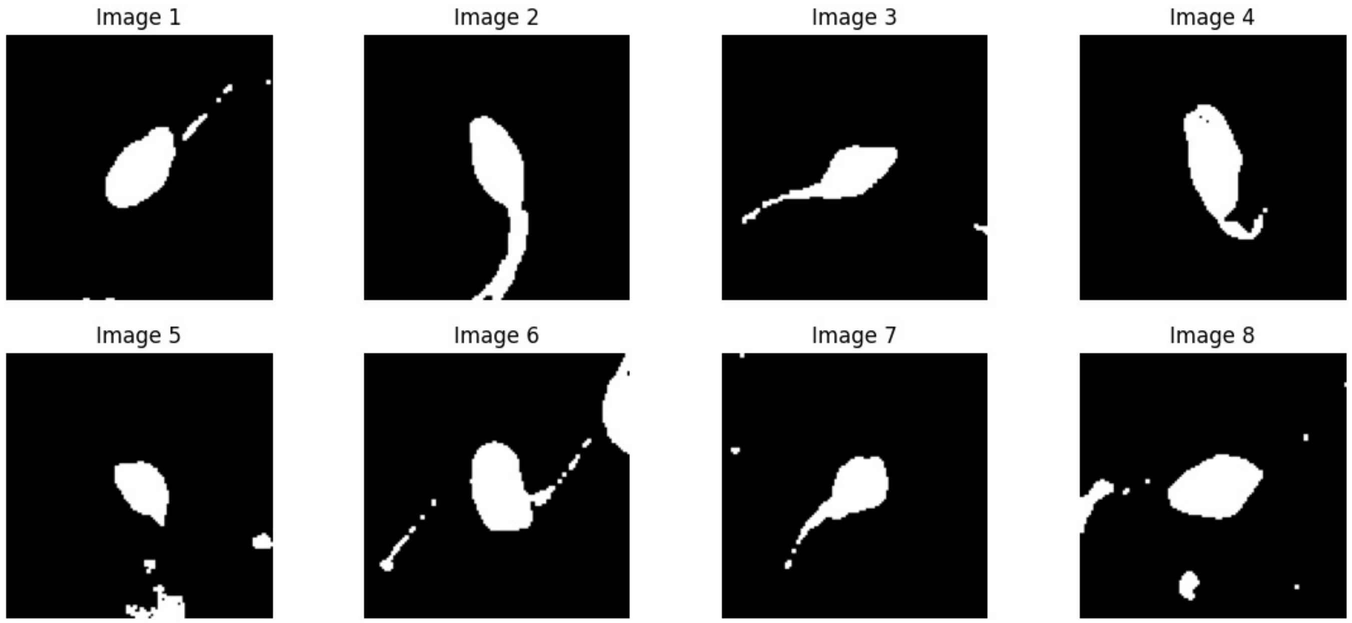


In [41]:

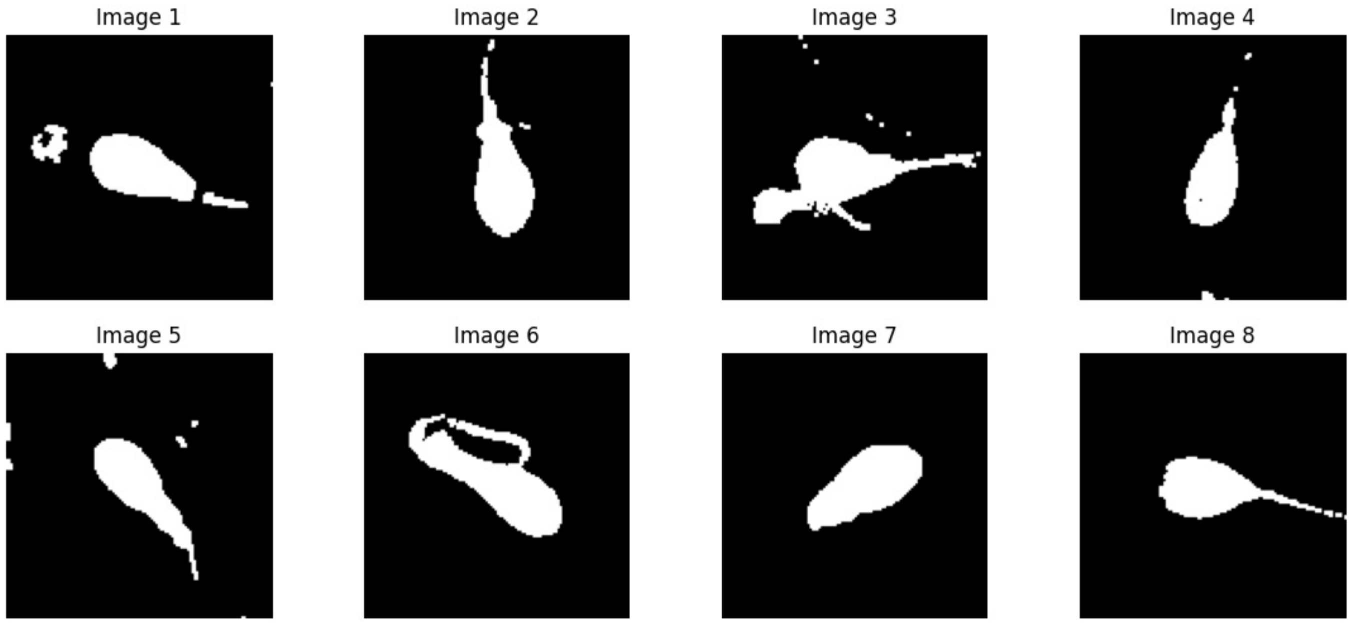
```python
show_img_inverting_closing(df2[1])
```



In [42]:
```
show_img_inverting_closing(df2[2])
```



In [43]:
```
show_img_inverting_closing(df2[3])
```



Creating x and y

In [44]:

```
x = []
y = []

for label, folder in enumerate(df2):
    for img in tqdm(os.listdir(folder)):
        img_path = os.path.join(folder, img)
        img_gray = cv2.imread(img_path, cv2.COLOR_BGR2GRAY)
        img_gray = cv2.cvtColor(img_gray, cv2.COLOR_BGR2GRAY)
        _, mask = cv2.threshold(img_gray, 175, 255, cv2.THRESH_BINARY)
        kernal = np.ones((2,2), np.uint8)
        closing = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernal)
        inverted_image = cv2.bitwise_not(closing)
        resize_img = cv2.resize(inverted_image, (img_size, img_size))

        x.append(resize_img)
        y.append(label)

x = np.array(x).reshape(-1, img_size, img_size, 1)
y = np.array(y)

print(f"x shape: {x.shape}")
print(f"y shape: {y.shape}")
```

```
100%|██████████| 57/57 [00:00<00:00, 5518.06it/s]
100%|██████████| 57/57 [00:00<00:00, 5635.51it/s]
100%|██████████| 57/57 [00:00<00:00, 5835.66it/s]
100%|██████████| 57/57 [00:00<00:00, 5544.55it/s]
x shape: (228, 118, 118, 1)
y shape: (228,)
```

In [45]:

```
x[0:1]
```

Out[45]:

```
array([[[[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 [46]:

```
y
```

Out[46]:

```
array([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, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3])
```

Spliting Data

In [47]:

```
train_images, test_images, train_labels, test_labels = train_test_split(x,y, train_size=0.8,random_state=1234)
print(train_images.shape)
print(test_images.shape)
print(train_labels.shape)
print(test_labels.shape)
```

```
(182, 118, 118, 1)
(46, 118, 118, 1)
(182,)
(46,)
```

In [48]:

```
train_images, train_labels = shuffle(train_images, train_labels, random_state=25)
```

Training data by using VGG16

In [49]:

```
if train_images.shape[-1] == 1:
    train_images = np.repeat(train_images, 3, axis=-1)
    test_images = np.repeat(test_images, 3, axis=-1)
```

In [50]:

```
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
```

In [51]:

```
batch_size = 16
img_size = (118, 118)
channels = 3
img_shape = (img_size[0], img_size[1], channels)

base_model = VGG16(weights='imagenet', include_top=False, input_shape=img_shape)
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
predictions = Dense(4, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers[-4:]:
    layer.trainable = True
```

```
2025-09-11 11:04:59.539014: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to cuI
nit: INTERNAL: CUDA error: Failed call to cuInit: UNKNOWN ERROR (303)
```

In [52]:

```
model.compile(optimizer=Adam(learning_rate=1e-5),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

In [53]:

```
model.summary()
```

```
Model: "functional"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)        │ (None, 118, 118, 3)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block1_conv1 (Conv2D)           │ (None, 118, 118, 64)   │         1,792 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block1_conv2 (Conv2D)           │ (None, 118, 118, 64)   │        36,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block1_pool (MaxPooling2D)      │ (None, 59, 59, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block2_conv1 (Conv2D)           │ (None, 59, 59, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block2_conv2 (Conv2D)           │ (None, 59, 59, 128)    │       147,584 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block2_pool (MaxPooling2D)      │ (None, 29, 29, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_conv1 (Conv2D)           │ (None, 29, 29, 256)    │       295,168 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_conv2 (Conv2D)           │ (None, 29, 29, 256)    │       590,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_conv3 (Conv2D)           │ (None, 29, 29, 256)    │       590,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_pool (MaxPooling2D)      │ (None, 14, 14, 256)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block4_conv1 (Conv2D)           │ (None, 14, 14, 512)    │     1,180,160 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block4_conv2 (Conv2D)           │ (None, 14, 14, 512)    │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block4_conv3 (Conv2D)           │ (None, 14, 14, 512)    │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block4_pool (MaxPooling2D)      │ (None, 7, 7, 512)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block5_conv1 (Conv2D)           │ (None, 7, 7, 512)      │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block5_conv2 (Conv2D)           │ (None, 7, 7, 512)      │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block5_conv3 (Conv2D)           │ (None, 7, 7, 512)      │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block5_pool (MaxPooling2D)      │ (None, 3, 3, 512)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ global_average_pooling2d        │ (None, 512)            │             0 │
│ (GlobalAveragePooling2D)        │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 256)            │       131,328 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 256)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 128)            │        32,896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 4)              │           516 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 14,879,428 (56.76 MB)
 Trainable params: 14,879,428 (56.76 MB)
 Non-trainable params: 0 (0.00 B)
```

In [54]:

```
early_stopping = keras.callbacks.EarlyStopping(
                              patience=3,
                              restore_best_weights=True)
```

In [55]:

```
epochs = 24
history = model.fit(train_images,
                   train_labels,
                   epochs=epochs,
                   validation_data=(test_images, test_labels),
                   callbacks=[early_stopping]
                   )
```

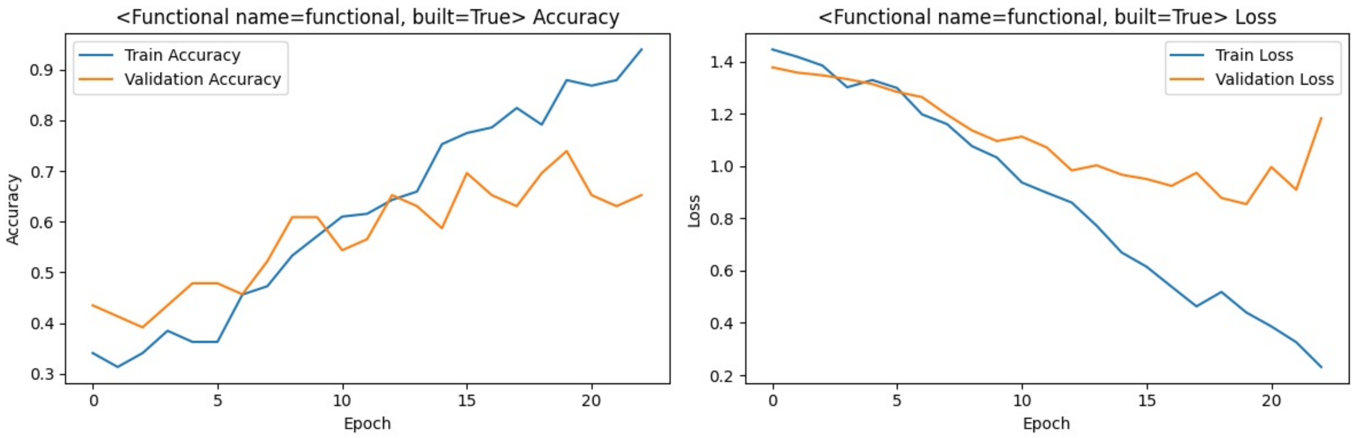
```
Epoch 1/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 62s 9s/step - accuracy: 0.3248 - loss: 1.5052 - val_accuracy: 0.4348 - val_loss: 1.3775
Epoch 2/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 54s 9s/step - accuracy: 0.2923 - loss: 1.4419 - val_accuracy: 0.4130 - val_loss: 1.3574
Epoch 3/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 54s 9s/step - accuracy: 0.3458 - loss: 1.3806 - val_accuracy: 0.3913 - val_loss: 1.3472
Epoch 4/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 52s 9s/step - accuracy: 0.3612 - loss: 1.3124 - val_accuracy: 0.4348 - val_loss: 1.3326
Epoch 5/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 54s 9s/step - accuracy: 0.3300 - loss: 1.3407 - val_accuracy: 0.4783 - val_loss: 1.3141
Epoch 6/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 54s 9s/step - accuracy: 0.3712 - loss: 1.3005 - val_accuracy: 0.4783 - val_loss: 1.2835
Epoch 7/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 54s 9s/step - accuracy: 0.4178 - loss: 1.2348 - val_accuracy: 0.4565 - val_loss: 1.2636
Epoch 8/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 56s 9s/step - accuracy: 0.4753 - loss: 1.1510 - val_accuracy: 0.5217 - val_loss: 1.1967
Epoch 9/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 53s 9s/step - accuracy: 0.5340 - loss: 1.0959 - val_accuracy: 0.6087 - val_loss: 1.1362
Epoch 10/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 53s 9s/step - accuracy: 0.5677 - loss: 1.0279 - val_accuracy: 0.6087 - val_loss: 1.0956
Epoch 11/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 54s 9s/step - accuracy: 0.5708 - loss: 0.9760 - val_accuracy: 0.5435 - val_loss: 1.1125
Epoch 12/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 54s 9s/step - accuracy: 0.6147 - loss: 0.8899 - val_accuracy: 0.5652 - val_loss: 1.0711
Epoch 13/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 79s 9s/step - accuracy: 0.6473 - loss: 0.8918 - val_accuracy: 0.6522 - val_loss: 0.9833
Epoch 14/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 52s 9s/step - accuracy: 0.6266 - loss: 0.7939 - val_accuracy: 0.6304 - val_loss: 1.0027
Epoch 15/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 54s 9s/step - accuracy: 0.7641 - loss: 0.6571 - val_accuracy: 0.5870 - val_loss: 0.9669
Epoch 16/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 52s 9s/step - accuracy: 0.7612 - loss: 0.6329 - val_accuracy: 0.6957 - val_loss: 0.9503
Epoch 17/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 52s 9s/step - accuracy: 0.7482 - loss: 0.5710 - val_accuracy: 0.6522 - val_loss: 0.9241
Epoch 18/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 53s 9s/step - accuracy: 0.8253 - loss: 0.4741 - val_accuracy: 0.6304 - val_loss: 0.9740
Epoch 19/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 52s 9s/step - accuracy: 0.7935 - loss: 0.5105 - val_accuracy: 0.6957 - val_loss: 0.8779
Epoch 20/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 52s 9s/step - accuracy: 0.8675 - loss: 0.4467 - val_accuracy: 0.7391 - val_loss: 0.8543
Epoch 21/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 52s 9s/step - accuracy: 0.8976 - loss: 0.3821 - val_accuracy: 0.6522 - val_loss: 0.9963
Epoch 22/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 54s 9s/step - accuracy: 0.8858 - loss: 0.3233 - val_accuracy: 0.6304 - val_loss: 0.9093
Epoch 23/24
6/6 ━━━━━━━━━━━━━━━━━━━━ 52s 9s/step - accuracy: 0.9617 - loss: 0.1916 - val_accuracy: 0.6522 - val_loss: 1.1826
```

In [56]:

```python
def plot_history(history, model_name):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{model_name} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{model_name} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

In [57]:

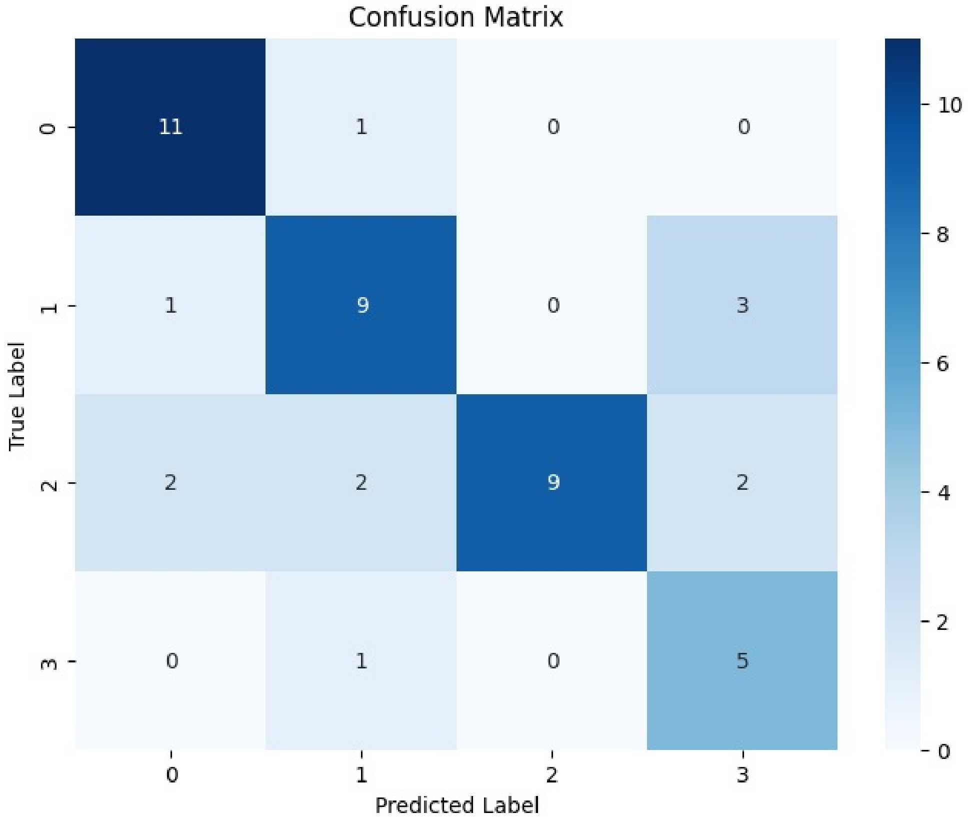
```python
plot_history(history, model)
```



In [58]:
```python
def evaluate_model(model, test_gen, test_labels):
    #test_gen.reset()
    y_pred = model.predict(test_gen)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true = test_labels
    cm = confusion_matrix(y_true, y_pred_classes)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
    print("Classification Report:")
    print(classification_report(y_true, y_pred_classes))
```

In [59]:
```python
evaluate_model(model, test_images, test_labels)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step
```



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

           0       0.79      0.92      0.85        12
           1       0.69      0.69      0.69        13
           2       1.00      0.60      0.75        15
           3       0.50      0.83      0.62         6

    accuracy                           0.74        46
   macro avg       0.74      0.76      0.73        46
weighted avg       0.79      0.74      0.74        46
```

Conclusion

The project successfully developed a deep learning model using VGG16 to classify sperm morphology into four types: Normal, Tapered, Amorphous, and Pyriform. The model achieved an overall accuracy of 74%. While it demonstrated strong performance in identifying Class 0, with a recall of 92%, and had perfect precision for Class 2, its overall effectiveness varied by class. Notably, the model showed a weakness in correctly predicting Class 3, where its precision was only 50%. These results suggest that while the model is a viable starting point for automated sperm classification, further improvements are needed to enhance its ability to accurately distinguish between all morphology types, particularly the more challenging ones.

```
In [ ]: # !jupyter nbconvert --to html "filename.ipynb"
```