Load Data

```
import urllib.request

zip_url= 'https://prod-dcd-datasets-cache-zipfiles.s3.eu-west-
1.amazonaws.com/ghvhwcpkbg-1.zip'

# Download the zip file
urllib.request.urlretrieve(zip_url, "file.zip")
```

```
('file.zip', <http.client.HTTPMessage at 0x7fd2b072df90>)
```

```
import zipfile


with zipfile.ZipFile("file.zip", 'r') as zip_ref:
    zip_ref.extractall("extracted_contents")

!pip install aspose-zip
```

```
Collecting aspose-zip
  Downloading aspose_zip-24.4.0-py3-none-
manylinux1_x86_64.whl.metadata (6.3 kB)
Downloading aspose_zip-24.4.0-py3-none-manylinux1_x86_64.whl (41.6 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41.6/41.6 MB 7.3 MB/s eta
0:00:00
```

```
import aspose.zip as az

# Load RAR archive
with az.rar.RarArchive("/kaggle/working/extracted_contents/Offline
Handwriting Signature/1-100.rar") as archive:

    # Extract RAR file
    archive.extract_to_directory("/kaggle/working/Data")

import aspose.zip as az

# Load RAR archive
with az.rar.RarArchive("/kaggle/working/extracted_contents/Offline
Handwriting Signature/201-300.rar") as archive:

    # Extract RAR file
    archive.extract_to_directory("/kaggle/working/Data")

import aspose.zip as az

# Load RAR archive
with az.rar.RarArchive("/kaggle/working/extracted_contents/Offline
```

```
Handwriting Signature/101-200.rar") as archive:

    # Extract RAR file
    archive.extract_to_directory("/kaggle/working/Data")

import aspose.zip as az

# Load RAR archive
with az.rar.RarArchive("/kaggle/working/extracted_contents/Offline
Handwriting Signature/401-420.rar") as archive:

    # Extract RAR file
    archive.extract_to_directory("/kaggle/working/Data")

import aspose.zip as az

# Load RAR archive
with az.rar.RarArchive("/kaggle/working/extracted_contents/Offline
Handwriting Signature/301-400.rar") as archive:

    # Extract RAR file
    archive.extract_to_directory("/kaggle/working/Data")
```

Important Libraries For Preprocessing

```
import cv2
import numpy as np
import os
import random
from scipy.signal import wiener
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Segmentation

```
def segment_images(dataset_path, output_path, min_area_threshold,
min_width_threshold, min_height_threshold):
    if not os.path.exists(output_path):
        os.makedirs(output_path)

    for cls in os.listdir(dataset_path):
        class_path = os.path.join(dataset_path, cls)
        if os.path.isdir(class_path):
            output_class_path = os.path.join(output_path, cls)
            if not os.path.exists(output_class_path):
                os.makedirs(output_class_path)
```

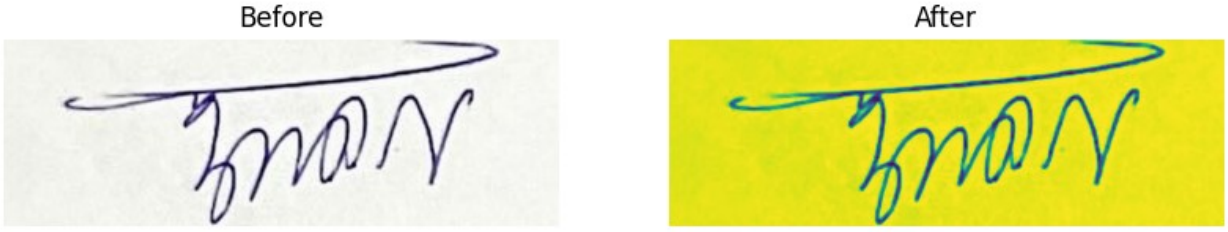
```
            for img_name in os.listdir(class_path):
                img_path = os.path.join(class_path, img_name)
                if img_name.lower().endswith(('.png', '.jpg',
'.jpeg')):
                    image = cv2.imread(img_path)
                    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                    ret, thresh = cv2.threshold(gray, 127, 255,
cv2.THRESH_BINARY_INV)
                    kernel = np.ones((5,100), np.uint8)
                    img_dilation = cv2.dilate(thresh, kernel,
iterations=1)
                    ctrs, _ = cv2.findContours(img_dilation.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
                    sorted_ctrs = sorted(ctrs, key=lambda ctr:
cv2.boundingRect(ctr)[0])

                    # Initialize max area, max width, and max height
to 0
                    max_area = 0
                    max_width = 0
                    max_height = 0
                    max_roi = None

                    for i, ctr in enumerate(sorted_ctrs):
                        x, y, w, h = cv2.boundingRect(ctr)
                        area = cv2.contourArea(ctr)
                        if area > min_area_threshold and w >
min_width_threshold and h > min_height_threshold:
                            roi = image[y:y+h, x:x+w]
                            if roi.shape[0] * roi.shape[1] > max_area
or roi.shape[0] > max_height or roi.shape[1] > max_width:
                                max_area = roi.shape[0] * roi.shape[1]
                                max_width = roi.shape[1]
                                max_height = roi.shape[0]
                                max_roi = roi

                    if max_roi is not None:
                        segment_path = os.path.join(output_class_path,
f'segment_no_0_{img_name}')
                        cv2.imwrite(segment_path, max_roi)

dataset_path = '/kaggle/working/Data'
output_path = '/kaggle/working/Dataset'
min_area_threshold = 1000
min_width_threshold = 150
min_height_threshold = 10

segment_images(dataset_path, output_path, min_area_threshold,
min_width_threshold, min_height_threshold)
```

```
# Load the images
img1 = mpimg.imread('/kaggle/working/Data/a_(47)/30.png')
img2 =
mpimg.imread('/kaggle/working/Dataset/a_(47)/segment_no_0_30.png')

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Display the images
axs[0].imshow(img1)
axs[0].set_title('Before')
axs[1].imshow(img2)
axs[1].set_title('After')

for ax in axs:
    ax.axis('off')

plt.show()
```

Remove Noise By Median Filter

```
input_dir = '/kaggle/working/Dataset'
output_dir = '/kaggle/working/MedianFilter'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for root, dirs, files in os.walk(input_dir):
    for file in files:
        if file.endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
```

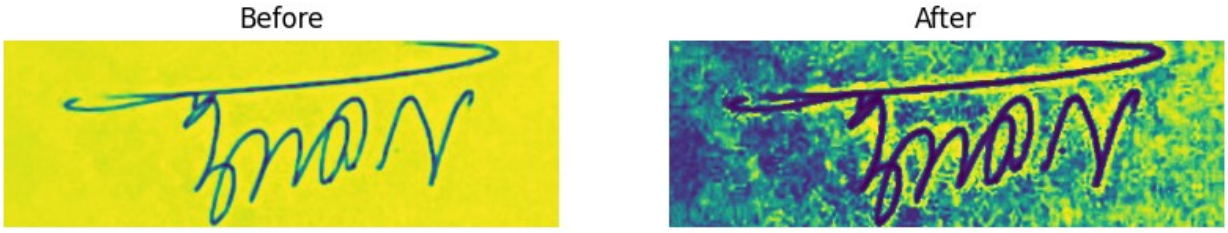
```
            median_img = cv2.medianBlur(img, 3)  # 5 is the kernel
size. You can adjust it according to your needs.
            rel_path = os.path.relpath(root, input_dir)
            median_root = os.path.join(output_dir, rel_path)
            if not os.path.exists(median_root):
                os.makedirs(median_root)
            output_path = os.path.join(median_root, file)
            cv2.imwrite(output_path, median_img)

# Load the images
img1 =
mpimg.imread('/kaggle/working/Dataset/a_(401)/segment_no_0_1.png')
img2 =
mpimg.imread('/kaggle/working/MedianFilter/a_(401)/segment_no_0_1.png'
)

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Display the images
axs[0].imshow(img1)
axs[0].set_title('Before')
axs[1].imshow(img2)
axs[1].set_title('After')

for ax in axs:
    ax.axis('off')

plt.show()
```



Histogram Equalization

```
input_dir = '/kaggle/working/MedianFilter'
output_dir = '/kaggle/working/Hist'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for root, dirs, files in os.walk(input_dir):
    for file in files:
```

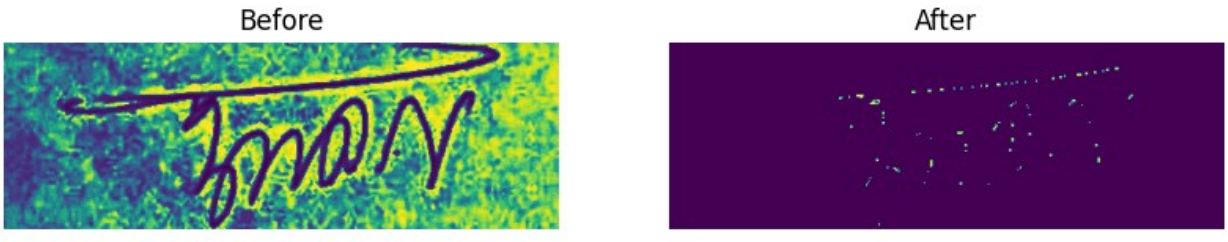
```
        if file.endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            equalized_img = cv2.equalizeHist(img)
            rel_path = os.path.relpath(root, input_dir)
            equalized_root = os.path.join(output_dir, rel_path)
            if not os.path.exists(equalized_root):
                os.makedirs(equalized_root)
            output_path = os.path.join(equalized_root, file)
            cv2.imwrite(output_path, equalized_img)

# Load the images
img1 =
mpimg.imread('/kaggle/working/MedianFilter/a_(401)/segment_no_0_1.png'
)
img2 = mpimg.imread('/kaggle/working/Hist/a_(401)/segment_no_0_1.png')

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Display the images
axs[0].imshow(img1)
axs[0].set_title('Before')
axs[1].imshow(img2)
axs[1].set_title('After')

for ax in axs:
    ax.axis('off')

plt.show()
```



Contrast Streching After Hist

```
input_dir = '/kaggle/working/Hist'
output_dir = '/kaggle/working/ContrastAfterHist'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for root, dirs, files in os.walk(input_dir):
```

```
    for file in files:
        if file.endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

            min_val = np.min(img)
            max_val = np.max(img)
            stretched_img = 255 * (img - min_val) / (max_val -
min_val)
            stretched_img = np.uint8(stretched_img)

            rel_path = os.path.relpath(root, input_dir)
            equalized_root = os.path.join(output_dir, rel_path)
            if not os.path.exists(equalized_root):
                os.makedirs(equalized_root)

            output_path = os.path.join(equalized_root, file)
            cv2.imwrite(output_path, stretched_img)

# Load the images
img1 = mpimg.imread('/kaggle/working/Hist/a_(401)/segment_no_0_1.png')
img2 =
mpimg.imread('/kaggle/working/ContrastAfterHist/a_(401)/segment_no_0_1
.png')

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Display the images
axs[0].imshow(img1)
axs[0].set_title('Before')
axs[1].imshow(img2)
axs[1].set_title('After')

for ax in axs:
    ax.axis('off')

plt.show()
```



Contrast Streching After Medien Filter

```
input_dir = '/kaggle/working/MedianFilter'
output_dir = '/kaggle/working/ContrastAfterMedienFilter'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for root, dirs, files in os.walk(input_dir):
    for file in files:
        if file.endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

            min_val = np.min(img)
            max_val = np.max(img)
            stretched_img = 255 * (img - min_val) / (max_val -
min_val)
            stretched_img = np.uint8(stretched_img)

            rel_path = os.path.relpath(root, input_dir)
            equalized_root = os.path.join(output_dir, rel_path)
            if not os.path.exists(equalized_root):
                os.makedirs(equalized_root)

            output_path = os.path.join(equalized_root, file)
            cv2.imwrite(output_path, stretched_img)

# Load the images
img1 =
mpimg.imread('/kaggle/working/MedianFilter/a_(401)/segment_no_0_1.png'
)
img2 =
mpimg.imread('/kaggle/working/ContrastAfterMedienFilter/a_(401)/segmen
t_no_0_1.png')

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Display the images
axs[0].imshow(img1)
axs[0].set_title('Before')
axs[1].imshow(img2)
axs[1].set_title('After')

for ax in axs:
    ax.axis('off')

plt.show()
```

Contrast Streching On Dataset

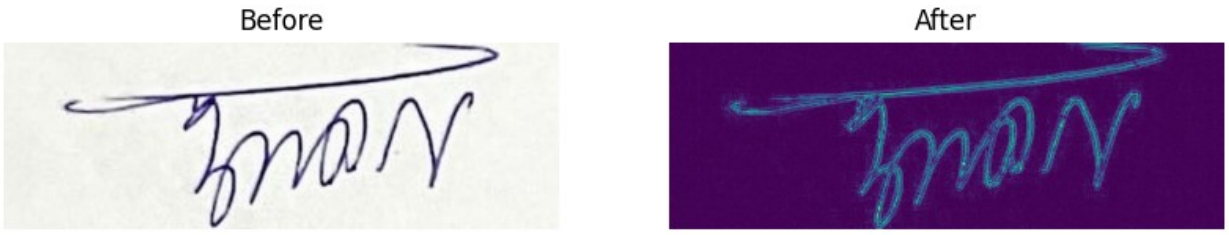
```
input_dir = '/kaggle/working/Dataset'
output_dir = '/kaggle/working/Contrast'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for root, dirs, files in os.walk(input_dir):
    for file in files:
        if file.endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

            min_val = np.min(img)
            max_val = np.max(img)
            stretched_img = 255 * (img - min_val) / (max_val -
min_val)
            stretched_img = np.uint8(stretched_img)

            rel_path = os.path.relpath(root, input_dir)
            equalized_root = os.path.join(output_dir, rel_path)
            if not os.path.exists(equalized_root):
                os.makedirs(equalized_root)

            output_path = os.path.join(equalized_root, file)
            cv2.imwrite(output_path, stretched_img)

# Load the images
img1 =
mpimg.imread('/kaggle/working/Dataset/a_(401)/segment_no_0_1.png')
img2 =
mpimg.imread('/kaggle/working/Contrast/a_(401)/segment_no_0_1.png')

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Display the images
axs[0].imshow(img1)
axs[0].set_title('Before')
axs[1].imshow(img2)
```

```
axs[1].set_title('After')

for ax in axs:
    ax.axis('off')

plt.show()
```

Apply Laplacian Filter

```
input_dir = '/kaggle/working/Dataset'
output_dir = '/kaggle/working/LaplacianFilter'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for root, dirs, files in os.walk(input_dir):
    for file in files:
        if file.endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            laplacian_img = cv2.Laplacian(img, cv2.CV_64F)  # The
second argument specifies the data type of the output image.
            abs_laplacian_img = cv2.convertScaleAbs(laplacian_img)  #
This converts the image to 8-bit unsigned integers and scales the
values to the range 0-255.
            rel_path = os.path.relpath(root, input_dir)
            laplacian_root = os.path.join(output_dir, rel_path)
            if not os.path.exists(laplacian_root):
                os.makedirs(laplacian_root)
            output_path = os.path.join(laplacian_root, file)
            cv2.imwrite(output_path, abs_laplacian_img)

# Load the images
img1 =
mpimg.imread('/kaggle/working/Dataset/a_(401)/segment_no_0_1.png')
img2 =
mpimg.imread('/kaggle/working/LaplacianFilter/a_(401)/segment_no_0_1.p
ng')

# Create a figure with two subplots
```

```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Display the images
axs[0].imshow(img1)
axs[0].set_title('Before')
axs[1].imshow(img2)
axs[1].set_title('After')

for ax in axs:
    ax.axis('off')

plt.show()
```



Apply Mean Filter

```
input_dir = '/kaggle/working/Dataset'
output_dir = '/kaggle/working/MeanFilter'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for root, dirs, files in os.walk(input_dir):
    for file in files:
        if file.endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            kernel_size = (5, 5)  # You can adjust the kernel size
according to your needs.
            mean_img = cv2.blur(img, kernel_size)
            rel_path = os.path.relpath(root, input_dir)
            mean_root = os.path.join(output_dir, rel_path)
            if not os.path.exists(mean_root):
                os.makedirs(mean_root)
            output_path = os.path.join(mean_root, file)
            cv2.imwrite(output_path, mean_img)

# Load the images
img1 =
mpimg.imread('/kaggle/working/Dataset/a_(401)/segment_no_0_1.png')
img2 =
```

```
mpimg.imread('/kaggle/working/MeanFilter/a_(401)/segment_no_0_1.png')

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Display the images
axs[0].imshow(img1)
axs[0].set_title('Before')
axs[1].imshow(img2)
axs[1].set_title('After')

for ax in axs:
    ax.axis('off')

plt.show()
```

Apply a high-pass filter by subtracting a Gaussian-blurred version from the original

```
input_dir = '/kaggle/working/Dataset'
output_dir = '/kaggle/working/HighPassFilter'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for root, dirs, files in os.walk(input_dir):
    for file in files:
        if file.endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            kernel_size = (5, 5)  # You can adjust the kernel size
according to your needs.
            lowpass_img = cv2.GaussianBlur(img, kernel_size, 0)
            highpass_img = cv2.addWeighted(img, 1.0, lowpass_img, -
1.0, 0.0)
            rel_path = os.path.relpath(root, input_dir)
            highpass_root = os.path.join(output_dir, rel_path)
            if not os.path.exists(highpass_root):
                os.makedirs(highpass_root)
```

```
            output_path = os.path.join(highpass_root, file)
            cv2.imwrite(output_path, highpass_img)

# Load the images
img1 =
mpimg.imread('/kaggle/working/Dataset/a_(401)/segment_no_0_1.png')
img2 =
mpimg.imread('/kaggle/working/HighPassFilter/a_(401)/segment_no_0_1.pn
g')

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Display the images
axs[0].imshow(img1)
axs[0].set_title('Before')
axs[1].imshow(img2)
axs[1].set_title('After')

for ax in axs:
    ax.axis('off')

plt.show()
```

Load Data for Model

```
nested_extracted_path = '/kaggle/working/HighPassFilter'
folders = os.listdir(nested_extracted_path)
folder_mapping = {}

for i, folder_name in enumerate(folders):
    new_folder_name = str(i)
    os.rename(os.path.join(nested_extracted_path, folder_name),
os.path.join(nested_extracted_path, new_folder_name))
    folder_mapping[folder_name] = new_folder_name

print("Folder Mapping:")
print(folder_mapping)
```

```
Folder Mapping:
{'a_(139)': '0', 'a_(229)': '1', 'a_(230)': '2', 'a_(157)': '3',
'a_(89)': '4', 'a_(111)': '5', 'a_(7)': '6', 'a_(279)': '7',
'a_(372)': '8', 'a_(180)': '9', 'a_(148)': '10', 'a_(231)': '11',
'a_(404)': '12', 'a_(168)': '13', 'a_(49)': '14', 'a_(185)': '15',
'a_(283)': '16', 'a_(112)': '17', 'a_(156)': '18', 'a_(286)': '19',
'a_(364)': '20', 'a_(31)': '21', 'a_(312)': '22', 'a_(383)': '23',
'a_(53)': '24', 'a_(328)': '25', 'a_(50)': '26', 'a_(225)': '27',
'a_(68)': '28', 'a_(200)': '29', 'a_(184)': '30', 'a_(360)': '31',
'a_(299)': '32', 'a_(336)': '33', 'a_(145)': '34', 'a_(34)': '35',
'a_(38)': '36', 'a_(375)': '37', 'a_(278)': '38', 'a_(4)': '39',
'a_(73)': '40', 'a_(203)': '41', 'a_(71)': '42', 'a_(227)': '43',
'a_(136)': '44', 'a_(170)': '45', 'a_(137)': '46', 'a_(331)': '47',
'a_(260)': '48', 'a_(334)': '49', 'a_(354)': '50', 'a_(209)': '51',
'a_(166)': '52', 'a_(344)': '53', 'a_(52)': '54', 'a_(256)': '55',
'a_(119)': '56', 'a_(116)': '57', 'a_(207)': '58', 'a_(217)': '59',
'a_(224)': '60', 'a_(403)': '61', 'a_(169)': '62', 'a_(9)': '63',
'a_(48)': '64', 'a_(300)': '65', 'a_(17)': '66', 'a_(127)': '67',
'a_(150)': '68', 'a_(211)': '69', 'a_(251)': '70', 'a_(138)': '71',
'a_(10)': '72', 'a_(388)': '73', 'a_(121)': '74', 'a_(393)': '75',
'a_(129)': '76', 'a_(413)': '77', 'a_(323)': '78', 'a_(410)': '79',
'a_(395)': '80', 'a_(418)': '81', 'a_(61)': '82', 'a_(293)': '83',
'a_(273)': '84', 'a_(371)': '85', 'a_(65)': '86', 'a_(87)': '87',
'a_(335)': '88', 'a_(131)': '89', 'a_(268)': '90', 'a_(412)': '91',
'a_(265)': '92', 'a_(215)': '93', 'a_(22)': '94', 'a_(401)': '95',
'a_(81)': '96', 'a_(326)': '97', 'a_(58)': '98', 'a_(226)': '99',
'a_(159)': '100', 'a_(90)': '101', 'a_(408)': '102', 'a_(378)': '103',
'a_(274)': '104', 'a_(78)': '105', 'a_(199)': '106', 'a_(149)': '107',
'a_(307)': '108', 'a_(219)': '109', 'a_(195)': '110', 'a_(20)': '111',
'a_(174)': '112', 'a_(377)': '113', 'a_(228)': '114', 'a_(70)': '115',
'a_(39)': '116', 'a_(118)': '117', 'a_(384)': '118', 'a_(249)': '119',
'a_(399)': '120', 'a_(382)': '121', 'a_(387)': '122', 'a_(103)':
'123', 'a_(196)': '124', 'a_(214)': '125', 'a_(23)': '126', 'a_(133)':
'127', 'a_(30)': '128', 'a_(117)': '129', 'a_(234)': '130', 'a_(192)':
'131', 'a_(2)': '132', 'a_(191)': '133', 'a_(88)': '134', 'a_(182)':
'135', 'a_(353)': '136', 'a_(222)': '137', 'a_(381)': '138',
'a_(146)': '139', 'a_(337)': '140', 'a_(329)': '141', 'a_(3)': '142',
'a_(208)': '143', 'a_(289)': '144', 'a_(205)': '145', 'a_(257)':
'146', 'a_(13)': '147', 'a_(59)': '148', 'a_(247)': '149', 'a_(99)':
'150', 'a_(290)': '151', 'a_(126)': '152', 'a_(115)': '153',
'a_(398)': '154', 'a_(147)': '155', 'a_(246)': '156', 'a_(394)':
'157', 'a_(343)': '158', 'a_(255)': '159', 'a_(181)': '160',
'a_(341)': '161', 'a_(264)': '162', 'a_(122)': '163', 'a_(296)':
'164', 'a_(259)': '165', 'a_(95)': '166', 'a_(362)': '167', 'a_(175)':
'168', 'a_(311)': '169', 'a_(242)': '170', 'a_(54)': '171', 'a_(206)':
'172', 'a_(392)': '173', 'a_(212)': '174', 'a_(280)': '175',
'a_(236)': '176', 'a_(396)': '177', 'a_(361)': '178', 'a_(64)': '179',
'a_(120)': '180', 'a_(176)': '181', 'a_(40)': '182', 'a_(141)': '183',
'a_(218)': '184', 'a_(294)': '185', 'a_(11)': '186', 'a_(45)': '187',
'a_(276)': '188', 'a_(102)': '189', 'a_(295)': '190', 'a_(314)':
```

'191', 'a_(171)': '192', 'a_(187)': '193', 'a_(62)': '194', 'a_(97)':
'195', 'a_(198)': '196', 'a_(321)': '197', 'a_(162)': '198',
'a_(298)': '199', 'a_(315)': '200', 'a_(14)': '201', 'a_(319)': '202',
'a_(367)': '203', 'a_(261)': '204', 'a_(275)': '205', 'a_(18)': '206',
'a_(85)': '207', 'a_(406)': '208', 'a_(270)': '209', 'a_(79)': '210',
'a_(365)': '211', 'a_(43)': '212', 'a_(263)': '213', 'a_(310)': '214',
'a_(110)': '215', 'a_(350)': '216', 'a_(72)': '217', 'a_(324)': '218',
'a_(86)': '219', 'a_(82)': '220', 'a_(373)': '221', 'a_(16)': '222',
'a_(183)': '223', 'a_(66)': '224', 'a_(5)': '225', 'a_(237)': '226',
'a_(93)': '227', 'a_(74)': '228', 'a_(402)': '229', 'a_(348)': '230',
'a_(24)': '231', 'a_(36)': '232', 'a_(284)': '233', 'a_(173)': '234',
'a_(193)': '235', 'a_(51)': '236', 'a_(197)': '237', 'a_(317)': '238',
'a_(100)': '239', 'a_(96)': '240', 'a_(15)': '241', 'a_(245)': '242',
'a_(305)': '243', 'a_(269)': '244', 'a_(113)': '245', 'a_(160)':
'246', 'a_(69)': '247', 'a_(123)': '248', 'a_(340)': '249', 'a_(223)':
'250', 'a_(325)': '251', 'a_(414)': '252', 'a_(304)': '253', 'a_(57)':
'254', 'a_(327)': '255', 'a_(254)': '256', 'a_(281)': '257',
'a_(302)': '258', 'a_(172)': '259', 'a_(80)': '260', 'a_(308)': '261',
'a_(417)': '262', 'a_(135)': '263', 'a_(306)': '264', 'a_(114)':
'265', 'a_(12)': '266', 'a_(132)': '267', 'a_(28)': '268', 'a_(358)':
'269', 'a_(190)': '270', 'a_(178)': '271', 'a_(235)': '272',
'a_(379)': '273', 'a_(92)': '274', 'a_(316)': '275', 'a_(204)': '276',
'a_(318)': '277', 'a_(104)': '278', 'a_(374)': '279', 'a_(368)':
'280', 'a_(19)': '281', 'a_(380)': '282', 'a_(6)': '283', 'a_(153)':
'284', 'a_(75)': '285', 'a_(124)': '286', 'a_(42)': '287', 'a_(154)':
'288', 'a_(220)': '289', 'a_(416)': '290', 'a_(271)': '291',
'a_(347)': '292', 'a_(415)': '293', 'a_(107)': '294', 'a_(389)':
'295', 'a_(407)': '296', 'a_(91)': '297', 'a_(60)': '298', 'a_(303)':
'299', 'a_(320)': '300', 'a_(106)': '301', 'a_(140)': '302',
'a_(301)': '303', 'a_(411)': '304', 'a_(349)': '305', 'a_(297)':
'306', 'a_(151)': '307', 'a_(233)': '308', 'a_(330)': '309',
'a_(338)': '310', 'a_(291)': '311', 'a_(420)': '312', 'a_(41)': '313',
'a_(26)': '314', 'a_(177)': '315', 'a_(125)': '316', 'a_(202)': '317',
'a_(292)': '318', 'a_(370)': '319', 'a_(250)': '320', 'a_(27)': '321',
'a_(390)': '322', 'a_(83)': '323', 'a_(101)': '324', 'a_(32)': '325',
'a_(352)': '326', 'a_(385)': '327', 'a_(130)': '328', 'a_(391)':
'329', 'a_(351)': '330', 'a_(409)': '331', 'a_(142)': '332',
'a_(363)': '333', 'a_(355)': '334', 'a_(46)': '335', 'a_(285)': '336',
'a_(76)': '337', 'a_(376)': '338', 'a_(144)': '339', 'a_(189)': '340',
'a_(164)': '341', 'a_(253)': '342', 'a_(345)': '343', 'a_(33)': '344',
'a_(216)': '345', 'a_(309)': '346', 'a_(201)': '347', 'a_(239)':
'348', 'a_(313)': '349', 'a_(342)': '350', 'a_(386)': '351',
'a_(232)': '352', 'a_(35)': '353', 'a_(188)': '354', 'a_(267)': '355',
'a_(282)': '356', 'a_(143)': '357', 'a_(98)': '358', 'a_(47)': '359',
'a_(179)': '360', 'a_(29)': '361', 'a_(165)': '362', 'a_(44)': '363',
'a_(67)': '364', 'a_(322)': '365', 'a_(108)': '366', 'a_(366)': '367',
'a_(243)': '368', 'a_(105)': '369', 'a_(240)': '370', 'a_(161)':
'371', 'a_(357)': '372', 'a_(287)': '373', 'a_(238)': '374',
'a_(400)': '375', 'a_(210)': '376', 'a_(266)': '377', 'a_(339)':
'378', 'a_(397)': '379', 'a_(221)': '380', 'a_(167)': '381',

```
'a_(186)': '382', 'a_(56)': '383', 'a_(37)': '384', 'a_(272)': '385',
'a_(25)': '386', 'a_(134)': '387', 'a_(262)': '388', 'a_(419)': '389',
'a_(8)': '390', 'a_(152)': '391', 'a_(369)': '392', 'a_(163)': '393',
'a_(277)': '394', 'a_(109)': '395', 'a_(333)': '396', 'a_(241)':
'397', 'a_(155)': '398', 'a_(63)': '399', 'a_(356)': '400', 'a_(194)':
'401', 'a_(1)': '402', 'a_(128)': '403', 'a_(248)': '404', 'a_(213)':
'405', 'a_(252)': '406', 'a_(55)': '407', 'a_(332)': '408', 'a_(77)':
'409', 'a_(359)': '410', 'a_(244)': '411', 'a_(288)': '412',
'a_(346)': '413', 'a_(405)': '414', 'a_(258)': '415', 'a_(158)':
'416', 'a_(94)': '417', 'a_(84)': '418', 'a_(21)': '419'}

dataset_folder = '/kaggle/working/HighPassFilter'
total_classes = 0
total_images = 0  # Initialize a variable to store total images

for class_folder in os.listdir(dataset_folder):
  class_folder_path = os.path.join(dataset_folder, class_folder)
  if os.path.isdir(class_folder_path):
    total_classes += 1
    images = os.listdir(class_folder_path)
    num_images = len(images)
    print(f"Class: {class_folder}, Number of Images: {num_images}")
    total_images += num_images  # Add current class image count to
total

print(f"Total number of classes: {total_classes}")
print(f"Total number of images: {total_images}")
```

```
Class: 292, Number of Images: 30
Class: 260, Number of Images: 30
Class: 165, Number of Images: 30
Class: 350, Number of Images: 30
Class: 345, Number of Images: 30
Class: 144, Number of Images: 30
Class: 188, Number of Images: 30
Class: 352, Number of Images: 30
Class: 33, Number of Images: 30
Class: 206, Number of Images: 30
Class: 174, Number of Images: 30
Class: 104, Number of Images: 30
Class: 97, Number of Images: 30
Class: 98, Number of Images: 30
Class: 102, Number of Images: 30
Class: 21, Number of Images: 30
Class: 107, Number of Images: 30
Class: 419, Number of Images: 30
Class: 224, Number of Images: 30
Class: 74, Number of Images: 30
Class: 368, Number of Images: 30
Class: 295, Number of Images: 30
```

```
Class: 70, Number of Images: 30
Class: 291, Number of Images: 30
Class: 145, Number of Images: 30
Class: 401, Number of Images: 30
Class: 79, Number of Images: 30
Class: 218, Number of Images: 30
Class: 62, Number of Images: 30
Class: 225, Number of Images: 30
Class: 124, Number of Images: 30
Class: 254, Number of Images: 30
Class: 204, Number of Images: 30
Class: 212, Number of Images: 30
Class: 44, Number of Images: 30
Class: 25, Number of Images: 30
Class: 327, Number of Images: 30
Class: 67, Number of Images: 30
Class: 211, Number of Images: 30
Class: 156, Number of Images: 30
Class: 333, Number of Images: 30
Class: 37, Number of Images: 30
Class: 36, Number of Images: 30
Class: 244, Number of Images: 30
Class: 301, Number of Images: 30
Class: 184, Number of Images: 30
Class: 406, Number of Images: 30
Class: 146, Number of Images: 30
Class: 50, Number of Images: 30
Class: 166, Number of Images: 30
Class: 267, Number of Images: 30
Class: 385, Number of Images: 30
Class: 409, Number of Images: 30
Class: 363, Number of Images: 30
Class: 153, Number of Images: 30
Class: 382, Number of Images: 30
Class: 227, Number of Images: 30
Class: 349, Number of Images: 30
Class: 347, Number of Images: 30
Class: 187, Number of Images: 30
Class: 134, Number of Images: 30
Class: 119, Number of Images: 30
Class: 210, Number of Images: 30
Class: 313, Number of Images: 30
Class: 326, Number of Images: 30
Class: 11, Number of Images: 30
Class: 412, Number of Images: 30
Class: 106, Number of Images: 30
Class: 161, Number of Images: 30
Class: 200, Number of Images: 30
Class: 35, Number of Images: 30
```

```
Class: 207, Number of Images: 30
Class: 3, Number of Images: 30
Class: 398, Number of Images: 30
Class: 308, Number of Images: 30
Class: 99, Number of Images: 30
Class: 148, Number of Images: 30
Class: 111, Number of Images: 30
Class: 120, Number of Images: 30
Class: 136, Number of Images: 30
Class: 214, Number of Images: 30
Class: 261, Number of Images: 30
Class: 189, Number of Images: 30
Class: 397, Number of Images: 30
Class: 236, Number of Images: 30
Class: 309, Number of Images: 30
Class: 115, Number of Images: 30
Class: 31, Number of Images: 30
Class: 316, Number of Images: 30
Class: 334, Number of Images: 29
Class: 325, Number of Images: 30
Class: 54, Number of Images: 30
Class: 232, Number of Images: 30
Class: 87, Number of Images: 30
Class: 240, Number of Images: 30
Class: 32, Number of Images: 30
Class: 404, Number of Images: 30
Class: 399, Number of Images: 30
Class: 257, Number of Images: 30
Class: 112, Number of Images: 30
Class: 90, Number of Images: 30
Class: 410, Number of Images: 30
Class: 336, Number of Images: 30
Class: 255, Number of Images: 30
Class: 358, Number of Images: 30
Class: 274, Number of Images: 30
Class: 80, Number of Images: 30
Class: 332, Number of Images: 30
Class: 179, Number of Images: 30
Class: 88, Number of Images: 30
Class: 322, Number of Images: 30
Class: 6, Number of Images: 30
Class: 335, Number of Images: 30
Class: 151, Number of Images: 30
Class: 9, Number of Images: 30
Class: 377, Number of Images: 30
Class: 259, Number of Images: 30
Class: 408, Number of Images: 30
Class: 133, Number of Images: 30
Class: 319, Number of Images: 30
```

```
Class: 193, Number of Images: 30
Class: 95, Number of Images: 30
Class: 230, Number of Images: 30
Class: 190, Number of Images: 30
Class: 93, Number of Images: 30
Class: 417, Number of Images: 30
Class: 163, Number of Images: 30
Class: 1, Number of Images: 30
Class: 265, Number of Images: 30
Class: 154, Number of Images: 30
Class: 47, Number of Images: 30
Class: 348, Number of Images: 30
Class: 152, Number of Images: 30
Class: 52, Number of Images: 30
Class: 371, Number of Images: 30
Class: 192, Number of Images: 30
Class: 123, Number of Images: 30
Class: 343, Number of Images: 30
Class: 28, Number of Images: 30
Class: 310, Number of Images: 31
Class: 77, Number of Images: 30
Class: 183, Number of Images: 30
Class: 131, Number of Images: 30
Class: 264, Number of Images: 30
Class: 56, Number of Images: 30
Class: 323, Number of Images: 30
Class: 250, Number of Images: 30
Class: 196, Number of Images: 30
Class: 306, Number of Images: 30
Class: 364, Number of Images: 30
Class: 356, Number of Images: 30
Class: 38, Number of Images: 30
Class: 129, Number of Images: 30
Class: 217, Number of Images: 30
Class: 168, Number of Images: 30
Class: 34, Number of Images: 30
Class: 220, Number of Images: 30
Class: 351, Number of Images: 30
Class: 130, Number of Images: 30
Class: 128, Number of Images: 30
Class: 290, Number of Images: 30
Class: 253, Number of Images: 30
Class: 109, Number of Images: 30
Class: 248, Number of Images: 30
Class: 328, Number of Images: 30
Class: 101, Number of Images: 30
Class: 297, Number of Images: 30
Class: 27, Number of Images: 8
Class: 138, Number of Images: 30
```

```
Class: 94, Number of Images: 30
Class: 370, Number of Images: 30
Class: 57, Number of Images: 30
Class: 65, Number of Images: 30
Class: 294, Number of Images: 30
Class: 53, Number of Images: 30
Class: 15, Number of Images: 30
Class: 213, Number of Images: 30
Class: 407, Number of Images: 30
Class: 298, Number of Images: 30
Class: 266, Number of Images: 30
Class: 29, Number of Images: 30
Class: 110, Number of Images: 30
Class: 216, Number of Images: 30
Class: 314, Number of Images: 30
Class: 121, Number of Images: 30
Class: 91, Number of Images: 30
Class: 396, Number of Images: 30
Class: 402, Number of Images: 30
Class: 84, Number of Images: 30
Class: 357, Number of Images: 30
Class: 141, Number of Images: 30
Class: 73, Number of Images: 30
Class: 181, Number of Images: 30
Class: 43, Number of Images: 30
Class: 68, Number of Images: 30
Class: 139, Number of Images: 30
Class: 270, Number of Images: 30
Class: 231, Number of Images: 30
Class: 234, Number of Images: 30
Class: 14, Number of Images: 30
Class: 342, Number of Images: 30
Class: 205, Number of Images: 30
Class: 339, Number of Images: 30
Class: 375, Number of Images: 30
Class: 142, Number of Images: 30
Class: 413, Number of Images: 30
Class: 194, Number of Images: 30
Class: 299, Number of Images: 30
Class: 215, Number of Images: 30
Class: 16, Number of Images: 30
Class: 157, Number of Images: 30
Class: 222, Number of Images: 30
Class: 263, Number of Images: 30
Class: 394, Number of Images: 30
Class: 237, Number of Images: 30
Class: 201, Number of Images: 30
Class: 273, Number of Images: 30
Class: 209, Number of Images: 30
```

```
Class: 269, Number of Images: 30
Class: 105, Number of Images: 30
Class: 256, Number of Images: 30
Class: 387, Number of Images: 30
Class: 92, Number of Images: 30
Class: 116, Number of Images: 30
Class: 226, Number of Images: 30
Class: 20, Number of Images: 30
Class: 39, Number of Images: 30
Class: 45, Number of Images: 30
Class: 235, Number of Images: 30
Class: 178, Number of Images: 30
Class: 296, Number of Images: 30
Class: 42, Number of Images: 30
Class: 180, Number of Images: 30
Class: 81, Number of Images: 30
Class: 8, Number of Images: 30
Class: 72, Number of Images: 30
Class: 108, Number of Images: 30
Class: 395, Number of Images: 30
Class: 7, Number of Images: 30
Class: 198, Number of Images: 30
Class: 369, Number of Images: 30
Class: 405, Number of Images: 30
Class: 51, Number of Images: 30
Class: 169, Number of Images: 30
Class: 272, Number of Images: 30
Class: 223, Number of Images: 30
Class: 49, Number of Images: 30
Class: 135, Number of Images: 30
Class: 171, Number of Images: 30
Class: 75, Number of Images: 30
Class: 247, Number of Images: 30
Class: 289, Number of Images: 30
Class: 320, Number of Images: 30
Class: 219, Number of Images: 30
Class: 83, Number of Images: 30
Class: 229, Number of Images: 30
Class: 284, Number of Images: 30
Class: 17, Number of Images: 30
Class: 418, Number of Images: 30
Class: 175, Number of Images: 30
Class: 331, Number of Images: 30
Class: 307, Number of Images: 30
Class: 228, Number of Images: 30
Class: 321, Number of Images: 30
Class: 26, Number of Images: 30
Class: 113, Number of Images: 30
Class: 378, Number of Images: 30
```

```
Class: 373, Number of Images: 30
Class: 315, Number of Images: 30
Class: 353, Number of Images: 30
Class: 208, Number of Images: 30
Class: 82, Number of Images: 30
Class: 117, Number of Images: 30
Class: 337, Number of Images: 30
Class: 89, Number of Images: 30
Class: 354, Number of Images: 30
Class: 281, Number of Images: 30
Class: 122, Number of Images: 30
Class: 304, Number of Images: 30
Class: 288, Number of Images: 30
Class: 191, Number of Images: 30
Class: 390, Number of Images: 30
Class: 374, Number of Images: 30
Class: 147, Number of Images: 30
Class: 283, Number of Images: 30
Class: 96, Number of Images: 30
Class: 416, Number of Images: 30
Class: 302, Number of Images: 30
Class: 55, Number of Images: 30
Class: 13, Number of Images: 30
Class: 251, Number of Images: 30
Class: 252, Number of Images: 30
Class: 317, Number of Images: 30
Class: 18, Number of Images: 30
Class: 293, Number of Images: 30
Class: 24, Number of Images: 30
Class: 285, Number of Images: 30
Class: 403, Number of Images: 30
Class: 384, Number of Images: 30
Class: 278, Number of Images: 30
Class: 126, Number of Images: 30
Class: 241, Number of Images: 30
Class: 199, Number of Images: 30
Class: 203, Number of Images: 30
Class: 0, Number of Images: 30
Class: 355, Number of Images: 30
Class: 12, Number of Images: 30
Class: 30, Number of Images: 30
Class: 287, Number of Images: 30
Class: 268, Number of Images: 30
Class: 10, Number of Images: 30
Class: 46, Number of Images: 30
Class: 245, Number of Images: 30
Class: 242, Number of Images: 30
Class: 150, Number of Images: 30
Class: 155, Number of Images: 30
```

```
Class: 140, Number of Images: 30
Class: 393, Number of Images: 30
Class: 389, Number of Images: 30
Class: 414, Number of Images: 30
Class: 61, Number of Images: 30
Class: 238, Number of Images: 30
Class: 19, Number of Images: 30
Class: 366, Number of Images: 30
Class: 380, Number of Images: 30
Class: 176, Number of Images: 30
Class: 5, Number of Images: 30
Class: 271, Number of Images: 30
Class: 85, Number of Images: 30
Class: 202, Number of Images: 30
Class: 162, Number of Images: 30
Class: 262, Number of Images: 30
Class: 69, Number of Images: 30
Class: 173, Number of Images: 30
Class: 182, Number of Images: 30
Class: 164, Number of Images: 30
Class: 63, Number of Images: 30
Class: 379, Number of Images: 30
Class: 282, Number of Images: 30
Class: 137, Number of Images: 30
Class: 172, Number of Images: 30
Class: 118, Number of Images: 30
Class: 78, Number of Images: 30
Class: 362, Number of Images: 30
Class: 158, Number of Images: 30
Class: 411, Number of Images: 30
Class: 160, Number of Images: 30
Class: 177, Number of Images: 30
Class: 64, Number of Images: 30
Class: 388, Number of Images: 30
Class: 233, Number of Images: 30
Class: 185, Number of Images: 30
Class: 372, Number of Images: 30
Class: 330, Number of Images: 30
Class: 170, Number of Images: 30
Class: 344, Number of Images: 30
Class: 361, Number of Images: 30
Class: 59, Number of Images: 30
Class: 132, Number of Images: 30
Class: 41, Number of Images: 30
Class: 341, Number of Images: 30
Class: 346, Number of Images: 30
Class: 383, Number of Images: 30
Class: 221, Number of Images: 30
Class: 303, Number of Images: 30
```

```
Class: 197, Number of Images: 30
Class: 125, Number of Images: 30
Class: 243, Number of Images: 30
Class: 276, Number of Images: 30
Class: 400, Number of Images: 30
Class: 415, Number of Images: 30
Class: 100, Number of Images: 30
Class: 392, Number of Images: 30
Class: 277, Number of Images: 30
Class: 23, Number of Images: 30
Class: 186, Number of Images: 30
Class: 329, Number of Images: 30
Class: 312, Number of Images: 30
Class: 40, Number of Images: 30
Class: 311, Number of Images: 30
Class: 4, Number of Images: 30
Class: 249, Number of Images: 30
Class: 367, Number of Images: 30
Class: 48, Number of Images: 30
Class: 360, Number of Images: 30
Class: 275, Number of Images: 30
Class: 239, Number of Images: 30
Class: 340, Number of Images: 30
Class: 338, Number of Images: 30
Class: 66, Number of Images: 30
Class: 159, Number of Images: 30
Class: 386, Number of Images: 30
Class: 324, Number of Images: 30
Class: 195, Number of Images: 30
Class: 60, Number of Images: 30
Class: 114, Number of Images: 30
Class: 286, Number of Images: 30
Class: 58, Number of Images: 30
Class: 365, Number of Images: 30
Class: 391, Number of Images: 30
Class: 149, Number of Images: 30
Class: 381, Number of Images: 30
Class: 76, Number of Images: 30
Class: 279, Number of Images: 30
Class: 86, Number of Images: 30
Class: 22, Number of Images: 30
Class: 103, Number of Images: 30
Class: 143, Number of Images: 30
Class: 359, Number of Images: 30
Class: 305, Number of Images: 30
Class: 280, Number of Images: 30
Class: 258, Number of Images: 30
Class: 318, Number of Images: 30
Class: 167, Number of Images: 30
```

```
Class: 246, Number of Images: 30
Class: 71, Number of Images: 30
Class: 300, Number of Images: 30
Class: 376, Number of Images: 30
Class: 127, Number of Images: 30
Class: 2, Number of Images: 30
Total number of classes: 420
Total number of images: 12578
```

Augmentation

```
import os
import cv2
import numpy as np
import shutil
import random
import skimage.util as sk_util

def reduce_line_thickness(image_path: str, output_path: str):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    kernel = np.ones((2, 2), np.uint8)
    dilated_image = cv2.dilate(image, kernel, iterations=1)
    cv2.imwrite(output_path, dilated_image)

def random_noise(image_path: str, output_path: str):
    image = cv2.imread(image_path)
    noisy_image = sk_util.random_noise(image, mode='gaussian',
clip=True)
    noisy_image = (255 * noisy_image).astype(np.uint8)  # Convert back
to uint8 format
    cv2.imwrite(output_path, noisy_image)

def random_stretch(image_path: str, output_path: str):
    image = cv2.imread(image_path)
    stretch = (random.random() - 0.9)  # -0.5 .. +0.5
    w_stretched = max(int(image.shape[1] * (1 + stretch)), 1)  #
random width, but at least 1
    stretched_image = cv2.resize(image, (w_stretched, image.shape[0]))
# stretch horizontally by factor 0.5 .. 1.5
    cv2.imwrite(output_path, stretched_image)

dataset_dir = '/kaggle/working/HighPassFilter'
output_dir =  '/kaggle/working/AugDataset'

# Create the output directory if it does not exist
os.makedirs(output_dir, exist_ok=True)

# Iterate over each user directory
```

```
for user_dir in os.listdir(dataset_dir):
    user_dir_path = os.path.join(dataset_dir, user_dir)
    if os.path.isdir(user_dir_path):
        # Create user output directory if it does not exist
        user_output_dir = os.path.join(output_dir, user_dir)
        os.makedirs(user_output_dir, exist_ok=True)

        # Iterate over each image file in the user directory
        for image_file in os.listdir(user_dir_path):
            if image_file.endswith('.jpg') or
image_file.endswith('.png'):  # Adjust based on your image file types
                image_path = os.path.join(user_dir_path, image_file)
                original_output_path = os.path.join(user_output_dir,
image_file)  # Output path for the original image

                # Save the original image
                shutil.copy(image_path, original_output_path)

                # Augment and save the images
                reduce_line_thickness(image_path,
os.path.join(user_output_dir, f'reduced_{image_file}'))
                random_noise(image_path, os.path.join(user_output_dir,
f'noisy_{image_file}'))
                random_stretch(image_path,
os.path.join(user_output_dir, f'stretched_{image_file}'))

dataset_folder = '/kaggle/working/AugDataset'
total_classes = 0
total_images = 0  # Initialize a variable to store total images

for class_folder in os.listdir(dataset_folder):
  class_folder_path = os.path.join(dataset_folder, class_folder)
  if os.path.isdir(class_folder_path):
    total_classes += 1
    images = os.listdir(class_folder_path)
    num_images = len(images)
    print(f"Class: {class_folder}, Number of Images: {num_images}")
    total_images += num_images  # Add current class image count to
total

print(f"Total number of classes: {total_classes}")
print(f"Total number of images: {total_images}")
```

```
Class: 292, Number of Images: 120
Class: 260, Number of Images: 120
Class: 165, Number of Images: 120
Class: 350, Number of Images: 120
Class: 345, Number of Images: 120
Class: 144, Number of Images: 120
Class: 188, Number of Images: 120
```

```
Class: 352, Number of Images: 120
Class: 33, Number of Images: 120
Class: 206, Number of Images: 120
Class: 174, Number of Images: 120
Class: 104, Number of Images: 120
Class: 97, Number of Images: 120
Class: 98, Number of Images: 120
Class: 102, Number of Images: 120
Class: 21, Number of Images: 120
Class: 107, Number of Images: 120
Class: 419, Number of Images: 120
Class: 224, Number of Images: 120
Class: 74, Number of Images: 120
Class: 368, Number of Images: 120
Class: 295, Number of Images: 120
Class: 70, Number of Images: 120
Class: 291, Number of Images: 120
Class: 145, Number of Images: 120
Class: 401, Number of Images: 120
Class: 79, Number of Images: 120
Class: 218, Number of Images: 120
Class: 62, Number of Images: 120
Class: 225, Number of Images: 120
Class: 124, Number of Images: 120
Class: 254, Number of Images: 120
Class: 204, Number of Images: 120
Class: 212, Number of Images: 120
Class: 44, Number of Images: 120
Class: 25, Number of Images: 120
Class: 327, Number of Images: 120
Class: 67, Number of Images: 120
Class: 211, Number of Images: 120
Class: 156, Number of Images: 120
Class: 333, Number of Images: 120
Class: 37, Number of Images: 120
Class: 36, Number of Images: 120
Class: 244, Number of Images: 120
Class: 301, Number of Images: 120
Class: 184, Number of Images: 120
Class: 406, Number of Images: 120
Class: 146, Number of Images: 120
Class: 50, Number of Images: 120
Class: 166, Number of Images: 120
Class: 267, Number of Images: 120
Class: 385, Number of Images: 120
Class: 409, Number of Images: 120
Class: 363, Number of Images: 120
Class: 153, Number of Images: 120
Class: 382, Number of Images: 120
```

```
Class: 227, Number of Images: 120
Class: 349, Number of Images: 120
Class: 347, Number of Images: 120
Class: 187, Number of Images: 120
Class: 134, Number of Images: 120
Class: 119, Number of Images: 120
Class: 210, Number of Images: 120
Class: 313, Number of Images: 120
Class: 326, Number of Images: 120
Class: 11, Number of Images: 120
Class: 412, Number of Images: 120
Class: 106, Number of Images: 120
Class: 161, Number of Images: 120
Class: 200, Number of Images: 120
Class: 35, Number of Images: 120
Class: 207, Number of Images: 120
Class: 3, Number of Images: 120
Class: 398, Number of Images: 120
Class: 308, Number of Images: 120
Class: 99, Number of Images: 120
Class: 148, Number of Images: 120
Class: 111, Number of Images: 120
Class: 120, Number of Images: 120
Class: 136, Number of Images: 120
Class: 214, Number of Images: 120
Class: 261, Number of Images: 120
Class: 189, Number of Images: 120
Class: 397, Number of Images: 120
Class: 236, Number of Images: 120
Class: 309, Number of Images: 120
Class: 115, Number of Images: 120
Class: 31, Number of Images: 120
Class: 316, Number of Images: 120
Class: 334, Number of Images: 116
Class: 325, Number of Images: 120
Class: 54, Number of Images: 120
Class: 232, Number of Images: 120
Class: 87, Number of Images: 120
Class: 240, Number of Images: 120
Class: 32, Number of Images: 120
Class: 404, Number of Images: 120
Class: 399, Number of Images: 120
Class: 257, Number of Images: 120
Class: 112, Number of Images: 120
Class: 90, Number of Images: 120
Class: 410, Number of Images: 120
Class: 336, Number of Images: 120
Class: 255, Number of Images: 120
Class: 358, Number of Images: 120
```

```
Class: 274, Number of Images: 120
Class: 80, Number of Images: 120
Class: 332, Number of Images: 120
Class: 179, Number of Images: 120
Class: 88, Number of Images: 120
Class: 322, Number of Images: 120
Class: 6, Number of Images: 120
Class: 335, Number of Images: 120
Class: 151, Number of Images: 120
Class: 9, Number of Images: 120
Class: 377, Number of Images: 120
Class: 259, Number of Images: 120
Class: 408, Number of Images: 120
Class: 133, Number of Images: 120
Class: 319, Number of Images: 120
Class: 193, Number of Images: 120
Class: 95, Number of Images: 120
Class: 230, Number of Images: 120
Class: 190, Number of Images: 120
Class: 93, Number of Images: 120
Class: 417, Number of Images: 120
Class: 163, Number of Images: 120
Class: 1, Number of Images: 120
Class: 265, Number of Images: 120
Class: 154, Number of Images: 120
Class: 47, Number of Images: 120
Class: 348, Number of Images: 120
Class: 152, Number of Images: 120
Class: 52, Number of Images: 120
Class: 371, Number of Images: 120
Class: 192, Number of Images: 120
Class: 123, Number of Images: 120
Class: 343, Number of Images: 120
Class: 28, Number of Images: 120
Class: 310, Number of Images: 124
Class: 77, Number of Images: 120
Class: 183, Number of Images: 120
Class: 131, Number of Images: 120
Class: 264, Number of Images: 120
Class: 56, Number of Images: 120
Class: 323, Number of Images: 120
Class: 250, Number of Images: 120
Class: 196, Number of Images: 120
Class: 306, Number of Images: 120
Class: 364, Number of Images: 120
Class: 356, Number of Images: 120
Class: 38, Number of Images: 120
Class: 129, Number of Images: 120
Class: 217, Number of Images: 120
```

```
Class: 168, Number of Images: 120
Class: 34, Number of Images: 120
Class: 220, Number of Images: 120
Class: 351, Number of Images: 120
Class: 130, Number of Images: 120
Class: 128, Number of Images: 120
Class: 290, Number of Images: 120
Class: 253, Number of Images: 120
Class: 109, Number of Images: 120
Class: 248, Number of Images: 120
Class: 328, Number of Images: 120
Class: 101, Number of Images: 120
Class: 297, Number of Images: 120
Class: 27, Number of Images: 32
Class: 138, Number of Images: 120
Class: 94, Number of Images: 120
Class: 370, Number of Images: 120
Class: 57, Number of Images: 120
Class: 65, Number of Images: 120
Class: 294, Number of Images: 120
Class: 53, Number of Images: 120
Class: 15, Number of Images: 120
Class: 213, Number of Images: 120
Class: 407, Number of Images: 120
Class: 298, Number of Images: 120
Class: 266, Number of Images: 120
Class: 29, Number of Images: 120
Class: 110, Number of Images: 120
Class: 216, Number of Images: 120
Class: 314, Number of Images: 120
Class: 121, Number of Images: 120
Class: 91, Number of Images: 120
Class: 396, Number of Images: 120
Class: 402, Number of Images: 120
Class: 84, Number of Images: 120
Class: 357, Number of Images: 120
Class: 141, Number of Images: 120
Class: 73, Number of Images: 120
Class: 181, Number of Images: 120
Class: 43, Number of Images: 120
Class: 68, Number of Images: 120
Class: 139, Number of Images: 120
Class: 270, Number of Images: 120
Class: 231, Number of Images: 120
Class: 234, Number of Images: 120
Class: 14, Number of Images: 120
Class: 342, Number of Images: 120
Class: 205, Number of Images: 120
Class: 339, Number of Images: 120
```

```
Class: 375, Number of Images: 120
Class: 142, Number of Images: 120
Class: 413, Number of Images: 120
Class: 194, Number of Images: 120
Class: 299, Number of Images: 120
Class: 215, Number of Images: 120
Class: 16, Number of Images: 120
Class: 157, Number of Images: 120
Class: 222, Number of Images: 120
Class: 263, Number of Images: 120
Class: 394, Number of Images: 120
Class: 237, Number of Images: 120
Class: 201, Number of Images: 120
Class: 273, Number of Images: 120
Class: 209, Number of Images: 120
Class: 269, Number of Images: 120
Class: 105, Number of Images: 120
Class: 256, Number of Images: 120
Class: 387, Number of Images: 120
Class: 92, Number of Images: 120
Class: 116, Number of Images: 120
Class: 226, Number of Images: 120
Class: 20, Number of Images: 120
Class: 39, Number of Images: 120
Class: 45, Number of Images: 120
Class: 235, Number of Images: 120
Class: 178, Number of Images: 120
Class: 296, Number of Images: 120
Class: 42, Number of Images: 120
Class: 180, Number of Images: 120
Class: 81, Number of Images: 120
Class: 8, Number of Images: 120
Class: 72, Number of Images: 120
Class: 108, Number of Images: 120
Class: 395, Number of Images: 120
Class: 7, Number of Images: 120
Class: 198, Number of Images: 120
Class: 369, Number of Images: 120
Class: 405, Number of Images: 120
Class: 51, Number of Images: 120
Class: 169, Number of Images: 120
Class: 272, Number of Images: 120
Class: 223, Number of Images: 120
Class: 49, Number of Images: 120
Class: 135, Number of Images: 120
Class: 171, Number of Images: 120
Class: 75, Number of Images: 120
Class: 247, Number of Images: 120
Class: 289, Number of Images: 120
```

```
Class: 320, Number of Images: 120
Class: 219, Number of Images: 120
Class: 83, Number of Images: 120
Class: 229, Number of Images: 120
Class: 284, Number of Images: 120
Class: 17, Number of Images: 120
Class: 418, Number of Images: 120
Class: 175, Number of Images: 120
Class: 331, Number of Images: 120
Class: 307, Number of Images: 120
Class: 228, Number of Images: 120
Class: 321, Number of Images: 120
Class: 26, Number of Images: 120
Class: 113, Number of Images: 120
Class: 378, Number of Images: 120
Class: 373, Number of Images: 120
Class: 315, Number of Images: 120
Class: 353, Number of Images: 120
Class: 208, Number of Images: 120
Class: 82, Number of Images: 120
Class: 117, Number of Images: 120
Class: 337, Number of Images: 120
Class: 89, Number of Images: 120
Class: 354, Number of Images: 120
Class: 281, Number of Images: 120
Class: 122, Number of Images: 120
Class: 304, Number of Images: 120
Class: 288, Number of Images: 120
Class: 191, Number of Images: 120
Class: 390, Number of Images: 120
Class: 374, Number of Images: 120
Class: 147, Number of Images: 120
Class: 283, Number of Images: 120
Class: 96, Number of Images: 120
Class: 416, Number of Images: 120
Class: 302, Number of Images: 120
Class: 55, Number of Images: 120
Class: 13, Number of Images: 120
Class: 251, Number of Images: 120
Class: 252, Number of Images: 120
Class: 317, Number of Images: 120
Class: 18, Number of Images: 120
Class: 293, Number of Images: 120
Class: 24, Number of Images: 120
Class: 285, Number of Images: 120
Class: 403, Number of Images: 120
Class: 384, Number of Images: 120
Class: 278, Number of Images: 120
Class: 126, Number of Images: 120
```

```
Class: 241, Number of Images: 120
Class: 199, Number of Images: 120
Class: 203, Number of Images: 120
Class: 0, Number of Images: 120
Class: 355, Number of Images: 120
Class: 12, Number of Images: 120
Class: 30, Number of Images: 120
Class: 287, Number of Images: 120
Class: 268, Number of Images: 120
Class: 10, Number of Images: 120
Class: 46, Number of Images: 120
Class: 245, Number of Images: 120
Class: 242, Number of Images: 120
Class: 150, Number of Images: 120
Class: 155, Number of Images: 120
Class: 140, Number of Images: 120
Class: 393, Number of Images: 120
Class: 389, Number of Images: 120
Class: 414, Number of Images: 120
Class: 61, Number of Images: 120
Class: 238, Number of Images: 120
Class: 19, Number of Images: 120
Class: 366, Number of Images: 120
Class: 380, Number of Images: 120
Class: 176, Number of Images: 120
Class: 5, Number of Images: 120
Class: 271, Number of Images: 120
Class: 85, Number of Images: 120
Class: 202, Number of Images: 120
Class: 162, Number of Images: 120
Class: 262, Number of Images: 120
Class: 69, Number of Images: 120
Class: 173, Number of Images: 120
Class: 182, Number of Images: 120
Class: 164, Number of Images: 120
Class: 63, Number of Images: 120
Class: 379, Number of Images: 120
Class: 282, Number of Images: 120
Class: 137, Number of Images: 120
Class: 172, Number of Images: 120
Class: 118, Number of Images: 120
Class: 78, Number of Images: 120
Class: 362, Number of Images: 120
Class: 158, Number of Images: 120
Class: 411, Number of Images: 120
Class: 160, Number of Images: 120
Class: 177, Number of Images: 120
Class: 64, Number of Images: 120
Class: 388, Number of Images: 120
```

```
Class: 233, Number of Images: 120
Class: 185, Number of Images: 120
Class: 372, Number of Images: 120
Class: 330, Number of Images: 120
Class: 170, Number of Images: 120
Class: 344, Number of Images: 120
Class: 361, Number of Images: 120
Class: 59, Number of Images: 120
Class: 132, Number of Images: 120
Class: 41, Number of Images: 120
Class: 341, Number of Images: 120
Class: 346, Number of Images: 120
Class: 383, Number of Images: 120
Class: 221, Number of Images: 120
Class: 303, Number of Images: 120
Class: 197, Number of Images: 120
Class: 125, Number of Images: 120
Class: 243, Number of Images: 120
Class: 276, Number of Images: 120
Class: 400, Number of Images: 120
Class: 415, Number of Images: 120
Class: 100, Number of Images: 120
Class: 392, Number of Images: 120
Class: 277, Number of Images: 120
Class: 23, Number of Images: 120
Class: 186, Number of Images: 120
Class: 329, Number of Images: 120
Class: 312, Number of Images: 120
Class: 40, Number of Images: 120
Class: 311, Number of Images: 120
Class: 4, Number of Images: 120
Class: 249, Number of Images: 120
Class: 367, Number of Images: 120
Class: 48, Number of Images: 120
Class: 360, Number of Images: 120
Class: 275, Number of Images: 120
Class: 239, Number of Images: 120
Class: 340, Number of Images: 120
Class: 338, Number of Images: 120
Class: 66, Number of Images: 120
Class: 159, Number of Images: 120
Class: 386, Number of Images: 120
Class: 324, Number of Images: 120
Class: 195, Number of Images: 120
Class: 60, Number of Images: 120
Class: 114, Number of Images: 120
Class: 286, Number of Images: 120
Class: 58, Number of Images: 120
Class: 365, Number of Images: 120
```

```
Class: 391, Number of Images: 120
Class: 149, Number of Images: 120
Class: 381, Number of Images: 120
Class: 76, Number of Images: 120
Class: 279, Number of Images: 120
Class: 86, Number of Images: 120
Class: 22, Number of Images: 120
Class: 103, Number of Images: 120
Class: 143, Number of Images: 120
Class: 359, Number of Images: 120
Class: 305, Number of Images: 120
Class: 280, Number of Images: 120
Class: 258, Number of Images: 120
Class: 318, Number of Images: 120
Class: 167, Number of Images: 120
Class: 246, Number of Images: 120
Class: 71, Number of Images: 120
Class: 300, Number of Images: 120
Class: 376, Number of Images: 120
Class: 127, Number of Images: 120
Class: 2, Number of Images: 120
Total number of classes: 420
Total number of images: 50312
```

Important Libraries for Model

```
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset, random_split
from torchvision import models,transforms, datasets
from transformers import ViTImageProcessor, ViTForImageClassification,
AutoTokenizer
import torch.optim as optim
import cv2
import os
import numpy as np
from sklearn.preprocessing import LabelEncoder
from PIL import Image
from torch.optim import lr_scheduler
```

```
2024-05-14 05:17:34.291604: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable
to register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
2024-05-14 05:17:34.291729: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
```

```
2024-05-14 05:17:34.462791: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable
to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

Constants

```
train_dir =  '/kaggle/working/AugDataset'
num_classes = total_classes
image_shape = (224, 224)
batch_size = 16
num_epochs = 10
```

preprocessing

```
class MakeDataset(Dataset):
    def __init__(self, root_folder, transform=None):
        self.root_folder = root_folder
        self.transform = transform
        self.classes = sorted(os.listdir(root_folder), key=lambda x:
int(x))
        self.class_to_idx = {cls: i for i, cls in
enumerate(self.classes)}
        self.images = self.make_dataset()

    def make_dataset(self):
        images = []
        for label in self.classes:
            label_folder = os.path.join(self.root_folder, label)
            for image_name in os.listdir(label_folder):
                image_path = os.path.join(label_folder, image_name)
                item = (image_path, self.class_to_idx[label])
                images.append(item)
        return images

    def __len__(self):
        return len(self.images)

    def __getitem__(self, index):
        image_path, label = self.images[index]
        image = Image.open(image_path).convert("RGB")

        if self.transform:
            image = self.transform(image)

        return image, label
```

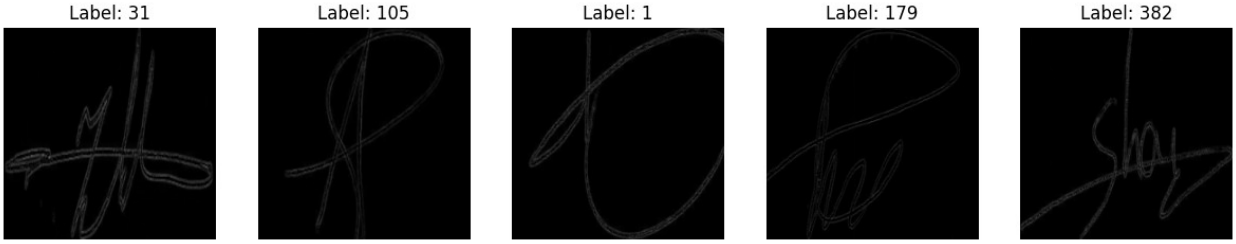
```
transform = transforms.Compose([
    transforms.Resize(image_shape),
    transforms.ToTensor()
])

dataset = MakeDataset(train_dir, transform=transform)

train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size
train_dataset, val_dataset, test_dataset = random_split(dataset,
[train_size, val_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size,
shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)
```

Visualization

```
# Define a function to visualize images from a DataLoader
def visualize(loader, num_samples=5):
    # Iterate through the DataLoader to get a batch of data
    for batch_idx, (images, labels) in enumerate(loader):
        # Plot the images
        fig, axes = plt.subplots(1, num_samples, figsize=(15, 3))
        for i in range(num_samples):
            ax = axes[i]
            ax.imshow(np.transpose(images[i], (1, 2, 0)))
            ax.set_title(f"Label: {labels[i]}")
            ax.axis('off')
        plt.show()
        break  # Stop after displaying the first batch

# Visualize samples from the training loader
print("Training Samples:")
visualize(train_loader)

# Visualize samples from the validation loader
print("Validation Samples:")
visualize(val_loader)

print("Testing Samples:")
visualize(test_loader)
```

```
Training Samples:
```


```
Validation Samples:
```




```
Testing Samples:
```

Load Model

```
model = models.efficientnet_b7(pretrained=True)
```

```
/opt/conda/lib/python3.10/site-packages/torchvision/models/
_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torchvision/models/_utils.py:2
23: UserWarning: Arguments other than a weight enum or `None` for
'weights' are deprecated since 0.13 and may be removed in the future.
The current behavior is equivalent to passing
`weights=EfficientNet_B7_Weights.IMAGENET1K_V1`. You can also use
`weights=EfficientNet_B7_Weights.DEFAULT` to get the most up-to-date
weights.
  warnings.warn(msg)
Downloading:
"https://download.pytorch.org/models/efficientnet_b7_lukemelas-
```

```
c5b4e57e.pth" to
/root/.cache/torch/hub/checkpoints/efficientnet_b7_lukemelas-
c5b4e57e.pth
100%|██████████| 255M/255M [00:01<00:00, 161MB/s]
```

```
last_layer = list(model.classifier.children())[-1]


# Replace it with the correct attribute name
if isinstance(last_layer, nn.Linear):
    in_features = last_layer.in_features
    model.classifier[-1] = nn.Linear(in_features, num_classes)
else:
    raise ValueError("The last layer of the classifier is not Linear,
please adjust the code accordingly.")

use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
print(f"Using device: {device}")
```

```
Using device: cuda
```

```
model = model.to(device)
```

Training

```
# Define the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.0001)

scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)

# Lists to store training and validation metrics for plotting curves
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []
predictions = []
targets = []

import time

start_time = time.time()

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct_train = 0
```

```
    total_train = 0

    for i, (inputs, labels) in enumerate(train_loader):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

        _, predicted_train = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted_train == labels).sum().item()

        # Print training loss and accuracy every 100 batches
        if i % 100 == 99:
            print(f'Epoch {epoch + 1}/{num_epochs}, Batch {i +
1}/{len(train_loader)}, '
                  f'Training Loss: {running_loss / 100}, Training
Accuracy: {100 * correct_train / total_train}%')

    # Calculate training accuracy after the epoch
    training_accuracy = correct_train / total_train

    # Validation
    model.eval()
    correct_val = 0
    total_val = 0
    val_running_loss = 0.0

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_running_loss += loss.item()

            _, predicted_val = torch.max(outputs.data, 1)
            total_val += labels.size(0)
            correct_val += (predicted_val == labels).sum().item()

            # Append predictions and targets for confusion matrix
            predictions.append(predicted_val.cpu().numpy())
            targets.append(labels.cpu().numpy())

    # Calculate validation accuracy after the epoch
    validation_accuracy = correct_val / total_val
    average_val_loss = val_running_loss / len(val_loader)
```

```
    print(f'Epoch {epoch + 1}/{num_epochs}, '
          f'Training Loss: {running_loss / len(train_loader)}, '
          f'Training Accuracy: {100 * training_accuracy}%, '
          f'Validation Loss: {average_val_loss}, '
          f'Validation Accuracy: {100 * validation_accuracy}%')

    # Append training and validation metrics for plotting
    train_losses.append(running_loss / len(train_loader))
    train_accuracies.append(training_accuracy)
    val_losses.append(average_val_loss)
    val_accuracies.append(validation_accuracy)

    scheduler.step()
```

```
Epoch 1/10, Batch 100/2516, Training Loss: 5.996179275512695, Training
Accuracy: 1.5%
Epoch 1/10, Batch 200/2516, Training Loss: 11.797345495223999,
Training Accuracy: 3.8125%
Epoch 1/10, Batch 300/2516, Training Loss: 17.112852969169616,
Training Accuracy: 7.5625%
Epoch 1/10, Batch 400/2516, Training Loss: 21.809506084918976,
Training Accuracy: 11.6875%
Epoch 1/10, Batch 500/2516, Training Loss: 25.927587761878968,
Training Accuracy: 16.0125%
Epoch 1/10, Batch 600/2516, Training Loss: 29.446949615478516,
Training Accuracy: 20.96875%
Epoch 1/10, Batch 700/2516, Training Loss: 32.4747691988945, Training
Accuracy: 25.5%
Epoch 1/10, Batch 800/2516, Training Loss: 35.00521857619285, Training
Accuracy: 30.09375%
Epoch 1/10, Batch 900/2516, Training Loss: 37.12046372413635, Training
Accuracy: 34.46527777777778%
Epoch 1/10, Batch 1000/2516, Training Loss: 38.8263020157814, Training
Accuracy: 38.75625%
Epoch 1/10, Batch 1100/2516, Training Loss: 40.22539370238781,
Training Accuracy: 42.69318181818182%
Epoch 1/10, Batch 1200/2516, Training Loss: 41.41814249098301,
Training Accuracy: 46.192708333333336%
Epoch 1/10, Batch 1300/2516, Training Loss: 42.38459844946861,
Training Accuracy: 49.33173076923077%
Epoch 1/10, Batch 1400/2516, Training Loss: 43.19283582657576,
Training Accuracy: 52.191964285714285%
Epoch 1/10, Batch 1500/2516, Training Loss: 43.84231605023146,
Training Accuracy: 54.8125%
Epoch 1/10, Batch 1600/2516, Training Loss: 44.42286767661572,
Training Accuracy: 57.171875%
Epoch 1/10, Batch 1700/2516, Training Loss: 44.89489956498146,
Training Accuracy: 59.294117647058826%
Epoch 1/10, Batch 1800/2516, Training Loss: 45.294234987944364,
Training Accuracy: 61.267361111111114%
```

```
Epoch 1/10, Batch 1900/2516, Training Loss: 45.6503264182806, Training Accuracy: 63.046052631578945%
Epoch 1/10, Batch 2000/2516, Training Loss: 45.999908924847844, Training Accuracy: 64.621875%
Epoch 1/10, Batch 2100/2516, Training Loss: 46.29645533125848, Training Accuracy: 66.0952380952381%
Epoch 1/10, Batch 2200/2516, Training Loss: 46.56522319633514, Training Accuracy: 67.44318181818181%
Epoch 1/10, Batch 2300/2516, Training Loss: 46.813629740290345, Training Accuracy: 68.65760869565217%
Epoch 1/10, Batch 2400/2516, Training Loss: 47.040689791850745, Training Accuracy: 69.80989583333333%
Epoch 1/10, Batch 2500/2516, Training Loss: 47.27029601562768, Training Accuracy: 70.8525%
Epoch 1/10, Training Loss: 1.88004404547608, Training Accuracy: 71.01294442097941%, Validation Loss: 0.17266564604545395, Validation Accuracy: 97.63466507652554%
Epoch 2/10, Batch 100/2516, Training Loss: 0.1386658306606114, Training Accuracy: 97.4375%
Epoch 2/10, Batch 200/2516, Training Loss: 0.26485020542517307, Training Accuracy: 97.78125%
Epoch 2/10, Batch 300/2516, Training Loss: 0.38666368352249264, Training Accuracy: 97.625%
Epoch 2/10, Batch 400/2516, Training Loss: 0.5213257530704141, Training Accuracy: 97.484375%
Epoch 2/10, Batch 500/2516, Training Loss: 0.6704047287069261, Training Accuracy: 97.3%
Epoch 2/10, Batch 600/2516, Training Loss: 0.789502317328006, Training Accuracy: 97.33333333333333%
Epoch 2/10, Batch 700/2516, Training Loss: 0.899028177112341, Training Accuracy: 97.35714285714286%
Epoch 2/10, Batch 800/2516, Training Loss: 1.0203817201033234, Training Accuracy: 97.3359375%
Epoch 2/10, Batch 900/2516, Training Loss: 1.1486644044332206, Training Accuracy: 97.28472222222223%
Epoch 2/10, Batch 1000/2516, Training Loss: 1.2576496709231286, Training Accuracy: 97.3%
Epoch 2/10, Batch 1100/2516, Training Loss: 1.3592891620565206, Training Accuracy: 97.3409090909091%
Epoch 2/10, Batch 1200/2516, Training Loss: 1.4422313105408102, Training Accuracy: 97.40104166666667%
Epoch 2/10, Batch 1300/2516, Training Loss: 1.5332853290624917, Training Accuracy: 97.4375%
Epoch 2/10, Batch 1400/2516, Training Loss: 1.6253873700276018, Training Accuracy: 97.46875%
Epoch 2/10, Batch 1500/2516, Training Loss: 1.7272951519209891, Training Accuracy: 97.45%
Epoch 2/10, Batch 1600/2516, Training Loss: 1.828808587519452, Training Accuracy: 97.421875%
```

```
Epoch 2/10, Batch 1700/2516, Training Loss: 1.9178817442245781,
Training Accuracy: 97.45220588235294%
Epoch 2/10, Batch 1800/2516, Training Loss: 2.023833102202043,
Training Accuracy: 97.43055555555556%
Epoch 2/10, Batch 1900/2516, Training Loss: 2.1074492303468286,
Training Accuracy: 97.47039473684211%
Epoch 2/10, Batch 2000/2516, Training Loss: 2.186568476213142,
Training Accuracy: 97.5125%
Epoch 2/10, Batch 2100/2516, Training Loss: 2.285234524952248,
Training Accuracy: 97.50892857142857%
Epoch 2/10, Batch 2200/2516, Training Loss: 2.3787159983161836,
Training Accuracy: 97.51420454545455%
Epoch 2/10, Batch 2300/2516, Training Loss: 2.4634865264594556,
Training Accuracy: 97.5461956521739%
Epoch 2/10, Batch 2400/2516, Training Loss: 2.5385260411165653,
Training Accuracy: 97.5625%
Epoch 2/10, Batch 2500/2516, Training Loss: 2.611386020826176,
Training Accuracy: 97.5875%
Epoch 2/10, Training Loss: 0.10449472023396263, Training Accuracy:
97.58254863474869%, Validation Loss: 0.05157308728981113, Validation
Accuracy: 98.37010534684953%
Epoch 3/10, Batch 100/2516, Training Loss: 0.06201296983286739,
Training Accuracy: 98.0%
Epoch 3/10, Batch 200/2516, Training Loss: 0.116493658144027, Training
Accuracy: 98.25%
Epoch 3/10, Batch 300/2516, Training Loss: 0.17479165651835502,
Training Accuracy: 98.33333333333333%
Epoch 3/10, Batch 400/2516, Training Loss: 0.2505105167767033,
Training Accuracy: 98.1875%
Epoch 3/10, Batch 500/2516, Training Loss: 0.32207775961607693,
Training Accuracy: 98.1125%
Epoch 3/10, Batch 600/2516, Training Loss: 0.38559787678997964,
Training Accuracy: 98.15625%
Epoch 3/10, Batch 700/2516, Training Loss: 0.4415608226461336,
Training Accuracy: 98.21428571428571%
Epoch 3/10, Batch 800/2516, Training Loss: 0.49200263132108374,
Training Accuracy: 98.21875%
Epoch 3/10, Batch 900/2516, Training Loss: 0.5511886221403256,
Training Accuracy: 98.22222222222223%
Epoch 3/10, Batch 1000/2516, Training Loss: 0.6232008668896742,
Training Accuracy: 98.1875%
Epoch 3/10, Batch 1100/2516, Training Loss: 0.6969626904162578,
Training Accuracy: 98.14772727272727%
Epoch 3/10, Batch 1200/2516, Training Loss: 0.7558359588379971,
Training Accuracy: 98.15625%
Epoch 3/10, Batch 1300/2516, Training Loss: 0.8190116431354545,
Training Accuracy: 98.14903846153847%
Epoch 3/10, Batch 1400/2516, Training Loss: 0.8787309599667787,
Training Accuracy: 98.15178571428571%
```

```
Epoch 3/10, Batch 1500/2516, Training Loss: 0.9345415918878279,
Training Accuracy: 98.16666666666667%
Epoch 3/10, Batch 1600/2516, Training Loss: 0.9951731755188666,
Training Accuracy: 98.1875%
Epoch 3/10, Batch 1700/2516, Training Loss: 1.0772509133885615,
Training Accuracy: 98.1654411764706%
Epoch 3/10, Batch 1800/2516, Training Loss: 1.1509865978709422,
Training Accuracy: 98.125%
Epoch 3/10, Batch 1900/2516, Training Loss: 1.2046312233363279,
Training Accuracy: 98.1217105263158%
Epoch 3/10, Batch 2000/2516, Training Loss: 1.265583147443831,
Training Accuracy: 98.1125%
Epoch 3/10, Batch 2100/2516, Training Loss: 1.3406873255316167,
Training Accuracy: 98.07440476190476%
Epoch 3/10, Batch 2200/2516, Training Loss: 1.416288069402799,
Training Accuracy: 98.07102272727273%
Epoch 3/10, Batch 2300/2516, Training Loss: 1.4717021658853628,
Training Accuracy: 98.08152173913044%
Epoch 3/10, Batch 2400/2516, Training Loss: 1.5321786467614584,
Training Accuracy: 98.08333333333333%
Epoch 3/10, Batch 2500/2516, Training Loss: 1.5906262213876472,
Training Accuracy: 98.105%
Epoch 3/10, Training Loss: 0.06364889756233368, Training Accuracy:
98.10678526174563%, Validation Loss: 0.05841517908764737, Validation
Accuracy: 97.85330948121646%
Epoch 4/10, Batch 100/2516, Training Loss: 0.06111187021480873,
Training Accuracy: 97.75%
Epoch 4/10, Batch 200/2516, Training Loss: 0.11815807869774289,
Training Accuracy: 98.09375%
Epoch 4/10, Batch 300/2516, Training Loss: 0.16749343938543462,
Training Accuracy: 98.16666666666667%
Epoch 4/10, Batch 400/2516, Training Loss: 0.20983030941803008,
Training Accuracy: 98.25%
Epoch 4/10, Batch 500/2516, Training Loss: 0.266706886816537, Training
Accuracy: 98.15%
Epoch 4/10, Batch 600/2516, Training Loss: 0.3048357498459518,
Training Accuracy: 98.26041666666667%
Epoch 4/10, Batch 700/2516, Training Loss: 0.3577361670037499,
Training Accuracy: 98.22321428571429%
Epoch 4/10, Batch 800/2516, Training Loss: 0.42248887359455695,
Training Accuracy: 98.1953125%
Epoch 4/10, Batch 900/2516, Training Loss: 0.47358851767668964,
Training Accuracy: 98.19444444444444%
Epoch 4/10, Batch 1000/2516, Training Loss: 0.5156708131410415,
Training Accuracy: 98.25%
Epoch 4/10, Batch 1100/2516, Training Loss: 0.562173026985838,
Training Accuracy: 98.25%
Epoch 4/10, Batch 1200/2516, Training Loss: 0.6163419672386954,
Training Accuracy: 98.23958333333333%
```

```
Epoch 4/10, Batch 1300/2516, Training Loss: 0.6662178674660391,
Training Accuracy: 98.23076923076923%
Epoch 4/10, Batch 1400/2516, Training Loss: 0.7104584924405208,
Training Accuracy: 98.29017857142857%
Epoch 4/10, Batch 1500/2516, Training Loss: 0.755560848999885,
Training Accuracy: 98.2875%
Epoch 4/10, Batch 1600/2516, Training Loss: 0.8225712564348941,
Training Accuracy: 98.23828125%
Epoch 4/10, Batch 1700/2516, Training Loss: 0.877169515529531,
Training Accuracy: 98.24264705882354%
Epoch 4/10, Batch 1800/2516, Training Loss: 0.9573648815596243,
Training Accuracy: 98.19097222222223%
Epoch 4/10, Batch 1900/2516, Training Loss: 1.0127715103310766,
Training Accuracy: 98.19407894736842%
Epoch 4/10, Batch 2000/2516, Training Loss: 1.0664648812421365,
Training Accuracy: 98.203125%
Epoch 4/10, Batch 2100/2516, Training Loss: 1.1156224757194286,
Training Accuracy: 98.21428571428571%
Epoch 4/10, Batch 2200/2516, Training Loss: 1.170424773307168,
Training Accuracy: 98.19318181818181%
Epoch 4/10, Batch 2300/2516, Training Loss: 1.2099407556053483,
Training Accuracy: 98.20652173913044%
Epoch 4/10, Batch 2400/2516, Training Loss: 1.25003007849853, Training
Accuracy: 98.21354166666667%
Epoch 4/10, Batch 2500/2516, Training Loss: 1.2942429473792436,
Training Accuracy: 98.215%
Epoch 4/10, Training Loss: 0.051887705857279365, Training Accuracy:
98.21362021416681%, Validation Loss: 0.03733394294921752, Validation
Accuracy: 98.27072152653548%
Epoch 5/10, Batch 100/2516, Training Loss: 0.047154763219878076,
Training Accuracy: 98.125%
Epoch 5/10, Batch 200/2516, Training Loss: 0.08992291783797554,
Training Accuracy: 98.25%
Epoch 5/10, Batch 300/2516, Training Loss: 0.13920996188186108,
Training Accuracy: 98.20833333333333%
Epoch 5/10, Batch 400/2516, Training Loss: 0.18255711622186938,
Training Accuracy: 98.1875%
Epoch 5/10, Batch 500/2516, Training Loss: 0.24148912977078, Training
Accuracy: 98.15%
Epoch 5/10, Batch 600/2516, Training Loss: 0.2890118679602165,
Training Accuracy: 98.22916666666667%
Epoch 5/10, Batch 700/2516, Training Loss: 0.32995251692889727,
Training Accuracy: 98.23214285714286%
Epoch 5/10, Batch 800/2516, Training Loss: 0.37767582935921384,
Training Accuracy: 98.2109375%
Epoch 5/10, Batch 900/2516, Training Loss: 0.4249792020133464,
Training Accuracy: 98.21527777777777%
Epoch 5/10, Batch 1000/2516, Training Loss: 0.4658966335246805,
Training Accuracy: 98.21875%
```

```
Epoch 5/10, Batch 1100/2516, Training Loss: 0.514651203873218,
Training Accuracy: 98.1875%
Epoch 5/10, Batch 1200/2516, Training Loss: 0.5568585974647431,
Training Accuracy: 98.17708333333333%
Epoch 5/10, Batch 1300/2516, Training Loss: 0.6088103606004733,
Training Accuracy: 98.20192307692308%
Epoch 5/10, Batch 1400/2516, Training Loss: 0.6506350884307176,
Training Accuracy: 98.22321428571429%
Epoch 5/10, Batch 1500/2516, Training Loss: 0.7074213981447974,
Training Accuracy: 98.17916666666666%
Epoch 5/10, Batch 1600/2516, Training Loss: 0.743631916429731,
Training Accuracy: 98.20703125%
Epoch 5/10, Batch 1700/2516, Training Loss: 0.784357934477157,
Training Accuracy: 98.21323529411765%
Epoch 5/10, Batch 1800/2516, Training Loss: 0.8364082994265482,
Training Accuracy: 98.21180555555556%
Epoch 5/10, Batch 1900/2516, Training Loss: 0.8692859663441778,
Training Accuracy: 98.22368421052632%
Epoch 5/10, Batch 2000/2516, Training Loss: 0.9261744313896634,
Training Accuracy: 98.2125%
Epoch 5/10, Batch 2100/2516, Training Loss: 0.9724665842991089,
Training Accuracy: 98.23214285714286%
Epoch 5/10, Batch 2200/2516, Training Loss: 1.0177620790275979,
Training Accuracy: 98.2159090909091%
Epoch 5/10, Batch 2300/2516, Training Loss: 1.0717410795448814,
Training Accuracy: 98.20652173913044%
Epoch 5/10, Batch 2400/2516, Training Loss: 1.1226058700028807,
Training Accuracy: 98.20833333333333%
Epoch 5/10, Batch 2500/2516, Training Loss: 1.174645365587203,
Training Accuracy: 98.2075%
Epoch 5/10, Training Loss: 0.04695368865507637, Training Accuracy:
98.20865114661234%, Validation Loss: 0.03229725282294639, Validation
Accuracy: 98.50924269528922%
Epoch 6/10, Batch 100/2516, Training Loss: 0.03636334760725731,
Training Accuracy: 98.3125%
Epoch 6/10, Batch 200/2516, Training Loss: 0.0675714618570055,
Training Accuracy: 98.59375%
Epoch 6/10, Batch 300/2516, Training Loss: 0.11656142055377132,
Training Accuracy: 98.35416666666667%
Epoch 6/10, Batch 400/2516, Training Loss: 0.15904655672115042,
Training Accuracy: 98.390625%
Epoch 6/10, Batch 500/2516, Training Loss: 0.18953516628564102,
Training Accuracy: 98.4875%
Epoch 6/10, Batch 600/2516, Training Loss: 0.22354653961694568,
Training Accuracy: 98.55208333333333%
Epoch 6/10, Batch 700/2516, Training Loss: 0.2592967641502037,
Training Accuracy: 98.57142857142857%
Epoch 6/10, Batch 800/2516, Training Loss: 0.2982073899320676,
Training Accuracy: 98.5625%
```

```
Epoch 6/10, Batch 900/2516, Training Loss: 0.33869117445632585,
Training Accuracy: 98.52777777777777%
Epoch 6/10, Batch 1000/2516, Training Loss: 0.3854184261444607,
Training Accuracy: 98.5125%
Epoch 6/10, Batch 1100/2516, Training Loss: 0.43244263268308714,
Training Accuracy: 98.5%
Epoch 6/10, Batch 1200/2516, Training Loss: 0.48296376323502044,
Training Accuracy: 98.453125%
Epoch 6/10, Batch 1300/2516, Training Loss: 0.5467381993599701,
Training Accuracy: 98.40384615384616%
Epoch 6/10, Batch 1400/2516, Training Loss: 0.5927412230172194,
Training Accuracy: 98.37946428571429%
Epoch 6/10, Batch 1500/2516, Training Loss: 0.654224824232224,
Training Accuracy: 98.34166666666667%
Epoch 6/10, Batch 1600/2516, Training Loss: 0.6972760663207737,
Training Accuracy: 98.34765625%
Epoch 6/10, Batch 1700/2516, Training Loss: 0.7288366744012456,
Training Accuracy: 98.37132352941177%
Epoch 6/10, Batch 1800/2516, Training Loss: 0.7591246635370772,
Training Accuracy: 98.38541666666667%
Epoch 6/10, Batch 1900/2516, Training Loss: 0.8037980179753503,
Training Accuracy: 98.38157894736842%
Epoch 6/10, Batch 2000/2516, Training Loss: 0.8355664690365665,
Training Accuracy: 98.390625%
Epoch 6/10, Batch 2100/2516, Training Loss: 0.862529831819993,
Training Accuracy: 98.41369047619048%
Epoch 6/10, Batch 2200/2516, Training Loss: 0.8866779304711963,
Training Accuracy: 98.44318181818181%
Epoch 6/10, Batch 2300/2516, Training Loss: 0.9376331455190666,
Training Accuracy: 98.4211956521739%
Epoch 6/10, Batch 2400/2516, Training Loss: 0.9799143562268, Training
Accuracy: 98.40104166666667%
Epoch 6/10, Batch 2500/2516, Training Loss: 1.0431879104109247,
Training Accuracy: 98.3725%
Epoch 6/10, Training Loss: 0.04179583201949712, Training Accuracy:
98.37014584213273%, Validation Loss: 0.04840734452355675, Validation
Accuracy: 98.09183064997018%
Epoch 7/10, Batch 100/2516, Training Loss: 0.029013790719618557,
Training Accuracy: 98.75%
Epoch 7/10, Batch 200/2516, Training Loss: 0.0666044698053156,
Training Accuracy: 98.5625%
Epoch 7/10, Batch 300/2516, Training Loss: 0.09519726117388927,
Training Accuracy: 98.625%
Epoch 7/10, Batch 400/2516, Training Loss: 0.13132140546440496,
Training Accuracy: 98.53125%
Epoch 7/10, Batch 500/2516, Training Loss: 0.17780837284386508,
Training Accuracy: 98.45%
Epoch 7/10, Batch 600/2516, Training Loss: 0.22355246721213917,
Training Accuracy: 98.38541666666667%
```

```
Epoch 7/10, Batch 700/2516, Training Loss: 0.25926348302396945,
Training Accuracy: 98.41071428571429%
Epoch 7/10, Batch 800/2516, Training Loss: 0.29239296491243294,
Training Accuracy: 98.4140625%
Epoch 7/10, Batch 900/2516, Training Loss: 0.3188156681456894,
Training Accuracy: 98.46527777777777%
Epoch 7/10, Batch 1000/2516, Training Loss: 0.3511689405688958,
Training Accuracy: 98.5%
Epoch 7/10, Batch 1100/2516, Training Loss: 0.3878037970951118,
Training Accuracy: 98.46022727272727%
Epoch 7/10, Batch 1200/2516, Training Loss: 0.42198862884877597,
Training Accuracy: 98.46875%
Epoch 7/10, Batch 1300/2516, Training Loss: 0.46463517058771686,
Training Accuracy: 98.45673076923077%
Epoch 7/10, Batch 1400/2516, Training Loss: 0.5085803959188343,
Training Accuracy: 98.45535714285714%
Epoch 7/10, Batch 1500/2516, Training Loss: 0.5663164756605693,
Training Accuracy: 98.4%
Epoch 7/10, Batch 1600/2516, Training Loss: 0.6049309342059133,
Training Accuracy: 98.41015625%
Epoch 7/10, Batch 1700/2516, Training Loss: 0.6488308947453334,
Training Accuracy: 98.4154411764706%
Epoch 7/10, Batch 1800/2516, Training Loss: 0.6850759523258603,
Training Accuracy: 98.42361111111111%
Epoch 7/10, Batch 1900/2516, Training Loss: 0.7359064183705777,
Training Accuracy: 98.39802631578948%
Epoch 7/10, Batch 2000/2516, Training Loss: 0.7944743393974204,
Training Accuracy: 98.365625%
Epoch 7/10, Batch 2100/2516, Training Loss: 0.8274382109752332,
Training Accuracy: 98.37202380952381%
Epoch 7/10, Batch 2200/2516, Training Loss: 0.8554304344199772,
Training Accuracy: 98.40056818181819%
Epoch 7/10, Batch 2300/2516, Training Loss: 0.880300539163145,
Training Accuracy: 98.42934782608695%
Epoch 7/10, Batch 2400/2516, Training Loss: 0.9157923007465434,
Training Accuracy: 98.42708333333333%
Epoch 7/10, Batch 2500/2516, Training Loss: 0.9566197237980668,
Training Accuracy: 98.4075%
Epoch 7/10, Training Loss: 0.0381303364719733, Training Accuracy:
98.41238291634575%, Validation Loss: 0.030691356967595528, Validation
Accuracy: 98.48936593122639%
Epoch 8/10, Batch 100/2516, Training Loss: 0.032674312344170175,
Training Accuracy: 98.4375%
Epoch 8/10, Batch 200/2516, Training Loss: 0.058434538284054725,
Training Accuracy: 98.59375%
Epoch 8/10, Batch 300/2516, Training Loss: 0.07437996375927469,
Training Accuracy: 98.875%
Epoch 8/10, Batch 400/2516, Training Loss: 0.10230860901559936,
Training Accuracy: 98.765625%
```

```
Epoch 8/10, Batch 500/2516, Training Loss: 0.12352610065499903,
Training Accuracy: 98.775%
Epoch 8/10, Batch 600/2516, Training Loss: 0.14734750019561033,
Training Accuracy: 98.79166666666667%
Epoch 8/10, Batch 700/2516, Training Loss: 0.17027060616266682,
Training Accuracy: 98.78571428571429%
Epoch 8/10, Batch 800/2516, Training Loss: 0.19103745533167968,
Training Accuracy: 98.8046875%
Epoch 8/10, Batch 900/2516, Training Loss: 0.21734561930803464,
Training Accuracy: 98.78472222222223%
Epoch 8/10, Batch 1000/2516, Training Loss: 0.24339707757317228,
Training Accuracy: 98.79375%
Epoch 8/10, Batch 1100/2516, Training Loss: 0.27085757125270904,
Training Accuracy: 98.74431818181819%
Epoch 8/10, Batch 1200/2516, Training Loss: 0.2938187688180187,
Training Accuracy: 98.734375%
Epoch 8/10, Batch 1300/2516, Training Loss: 0.3145702907720988,
Training Accuracy: 98.7451923076923%
Epoch 8/10, Batch 1400/2516, Training Loss: 0.33345970791553553,
Training Accuracy: 98.77232142857143%
Epoch 8/10, Batch 1500/2516, Training Loss: 0.36272496676909216,
Training Accuracy: 98.74583333333334%
Epoch 8/10, Batch 1600/2516, Training Loss: 0.3855004233390355,
Training Accuracy: 98.75%
Epoch 8/10, Batch 1700/2516, Training Loss: 0.4132960731328785,
Training Accuracy: 98.74632352941177%
Epoch 8/10, Batch 1800/2516, Training Loss: 0.43393348897363465,
Training Accuracy: 98.75347222222223%
Epoch 8/10, Batch 1900/2516, Training Loss: 0.4556851610410376,
Training Accuracy: 98.7532894736842%
Epoch 8/10, Batch 2000/2516, Training Loss: 0.47964721197960897,
Training Accuracy: 98.74375%
Epoch 8/10, Batch 2100/2516, Training Loss: 0.507342219310376,
Training Accuracy: 98.72916666666667%
Epoch 8/10, Batch 2200/2516, Training Loss: 0.5286579472299491,
Training Accuracy: 98.73295454545455%
Epoch 8/10, Batch 2300/2516, Training Loss: 0.5564235220233968,
Training Accuracy: 98.72554347826087%
Epoch 8/10, Batch 2400/2516, Training Loss: 0.5780088294715097,
Training Accuracy: 98.73697916666667%
Epoch 8/10, Batch 2500/2516, Training Loss: 0.5961120705171197,
Training Accuracy: 98.75%
Epoch 8/10, Training Loss: 0.02391452968792485, Training Accuracy:
98.7477949762727%, Validation Loss: 0.02323993330011858, Validation
Accuracy: 98.50924269528922%
Epoch 9/10, Batch 100/2516, Training Loss: 0.017964392185167526,
Training Accuracy: 99.125%
Epoch 9/10, Batch 200/2516, Training Loss: 0.03650088421702094,
Training Accuracy: 99.125%
```

```
Epoch 9/10, Batch 300/2516, Training Loss: 0.05656146264169365,
Training Accuracy: 99.02083333333333%
Epoch 9/10, Batch 400/2516, Training Loss: 0.07706600634221104,
Training Accuracy: 98.953125%
Epoch 9/10, Batch 500/2516, Training Loss: 0.10473568054614589,
Training Accuracy: 98.8375%
Epoch 9/10, Batch 600/2516, Training Loss: 0.13729002713058436,
Training Accuracy: 98.73958333333333%
Epoch 9/10, Batch 700/2516, Training Loss: 0.16203633166092912,
Training Accuracy: 98.74107142857143%
Epoch 9/10, Batch 800/2516, Training Loss: 0.18447892940399468,
Training Accuracy: 98.7265625%
Epoch 9/10, Batch 900/2516, Training Loss: 0.20110679394390898,
Training Accuracy: 98.76388888888889%
Epoch 9/10, Batch 1000/2516, Training Loss: 0.221773546373297,
Training Accuracy: 98.76875%
Epoch 9/10, Batch 1100/2516, Training Loss: 0.2521783314427012,
Training Accuracy: 98.7215909090909%
Epoch 9/10, Batch 1200/2516, Training Loss: 0.2711052672000369,
Training Accuracy: 98.73958333333333%
Epoch 9/10, Batch 1300/2516, Training Loss: 0.2973692724884313,
Training Accuracy: 98.70673076923077%
Epoch 9/10, Batch 1400/2516, Training Loss: 0.3254913341326028,
Training Accuracy: 98.6875%
Epoch 9/10, Batch 1500/2516, Training Loss: 0.3482122656404681,
Training Accuracy: 98.69583333333334%
Epoch 9/10, Batch 1600/2516, Training Loss: 0.3742874039470189,
Training Accuracy: 98.69140625%
Epoch 9/10, Batch 1700/2516, Training Loss: 0.38981074022893153,
Training Accuracy: 98.71691176470588%
Epoch 9/10, Batch 1800/2516, Training Loss: 0.4139894893194287,
Training Accuracy: 98.70486111111111%
Epoch 9/10, Batch 1900/2516, Training Loss: 0.4369372720067986,
Training Accuracy: 98.69407894736842%
Epoch 9/10, Batch 2000/2516, Training Loss: 0.4541004669128961,
Training Accuracy: 98.71875%
Epoch 9/10, Batch 2100/2516, Training Loss: 0.48011593935454583,
Training Accuracy: 98.70238095238095%
Epoch 9/10, Batch 2200/2516, Training Loss: 0.49722414342049887,
Training Accuracy: 98.72727272727273%
Epoch 9/10, Batch 2300/2516, Training Loss: 0.5148861160943125,
Training Accuracy: 98.73641304347827%
Epoch 9/10, Batch 2400/2516, Training Loss: 0.5340861266455977,
Training Accuracy: 98.75260416666667%
Epoch 9/10, Batch 2500/2516, Training Loss: 0.5556405458097652,
Training Accuracy: 98.7525%
Epoch 9/10, Training Loss: 0.02212972485293266, Training Accuracy:
98.76021764515889%, Validation Loss: 0.023274461654508496, Validation
Accuracy: 98.44961240310077%
```

```
Epoch 10/10, Batch 100/2516, Training Loss: 0.02380431375117041,
Training Accuracy: 98.625%
Epoch 10/10, Batch 200/2516, Training Loss: 0.04378400606779906,
Training Accuracy: 98.8125%
Epoch 10/10, Batch 300/2516, Training Loss: 0.06978223512822296,
Training Accuracy: 98.70833333333333%
Epoch 10/10, Batch 400/2516, Training Loss: 0.09017917046792717,
Training Accuracy: 98.734375%
Epoch 10/10, Batch 500/2516, Training Loss: 0.1104544474676004,
Training Accuracy: 98.7625%
Epoch 10/10, Batch 600/2516, Training Loss: 0.13158709921557601,
Training Accuracy: 98.80208333333333%
Epoch 10/10, Batch 700/2516, Training Loss: 0.15496166238863224,
Training Accuracy: 98.78571428571429%
Epoch 10/10, Batch 800/2516, Training Loss: 0.17964275891059514,
Training Accuracy: 98.75%
Epoch 10/10, Batch 900/2516, Training Loss: 0.20210671087759693,
Training Accuracy: 98.77083333333333%
Epoch 10/10, Batch 1000/2516, Training Loss: 0.21389626716816565,
Training Accuracy: 98.84375%
Epoch 10/10, Batch 1100/2516, Training Loss: 0.2312716622323205,
Training Accuracy: 98.86363636363636%
Epoch 10/10, Batch 1200/2516, Training Loss: 0.2504331963868026,
Training Accuracy: 98.86458333333333%
Epoch 10/10, Batch 1300/2516, Training Loss: 0.2735243685824753,
Training Accuracy: 98.85576923076923%
Epoch 10/10, Batch 1400/2516, Training Loss: 0.2996611150564786,
Training Accuracy: 98.83482142857143%
Epoch 10/10, Batch 1500/2516, Training Loss: 0.32236979220684586,
Training Accuracy: 98.825%
Epoch 10/10, Batch 1600/2516, Training Loss: 0.3427519785382174,
Training Accuracy: 98.83203125%
Epoch 10/10, Batch 1700/2516, Training Loss: 0.3669331928174506,
Training Accuracy: 98.83088235294117%
Epoch 10/10, Batch 1800/2516, Training Loss: 0.3847696058469592,
Training Accuracy: 98.84027777777777%
Epoch 10/10, Batch 1900/2516, Training Loss: 0.4038002115565905,
Training Accuracy: 98.84210526315789%
Epoch 10/10, Batch 2000/2516, Training Loss: 0.4250163232724299,
Training Accuracy: 98.834375%
Epoch 10/10, Batch 2100/2516, Training Loss: 0.44416316614358947,
Training Accuracy: 98.82738095238095%
Epoch 10/10, Batch 2200/2516, Training Loss: 0.4667313715176351,
Training Accuracy: 98.82670454545455%
Epoch 10/10, Batch 2300/2516, Training Loss: 0.48281628241798896,
Training Accuracy: 98.85054347826087%
Epoch 10/10, Batch 2400/2516, Training Loss: 0.4988508267433281,
Training Accuracy: 98.86197916666667%
Epoch 10/10, Batch 2500/2516, Training Loss: 0.5179003997812106,
Training Accuracy: 98.8675%
```

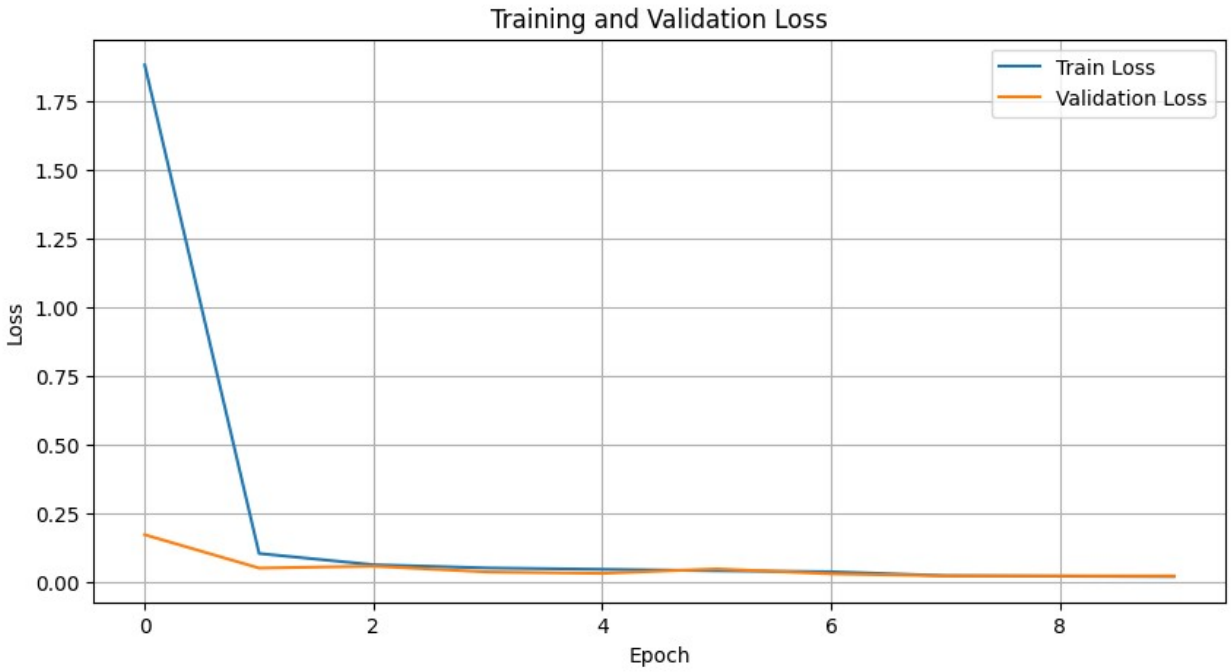
```
Epoch 10/10, Training Loss: 0.02077530661442913, Training Accuracy: 98.86456806380282%, Validation Loss: 0.023002289704466723, Validation Accuracy: 98.60862651560326%

end_time = time.time()

duration_seconds = end_time - start_time
duration_minutes = duration_seconds / 60

print("Training duration: {:.2f} minutes".format(duration_minutes))

Training duration: 277.99 minutes
```

Testing

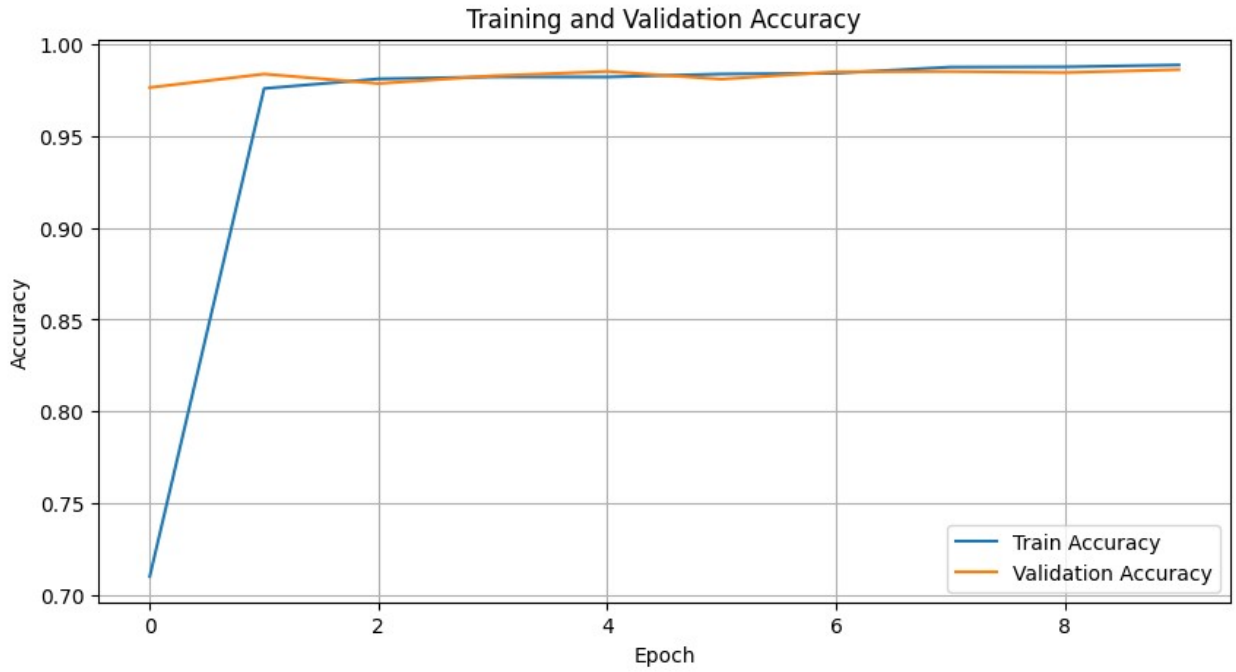
```
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

test_accuracy = correct / total
print(f'Test Accuracy: {100 * test_accuracy}%')

Test Accuracy: 98.60890302066773%
```

Plotting training and validation accuracy & loss

```
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid(True)
plt.show()
```




```
plt.figure(figsize=(10, 5))
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



XOR for Encryption and Decryption

```
def encrypt_image(image, key):
    # Flatten the image
    image_flat = image.flatten()

    # Encrypt the image
    encrypted_flat = np.bitwise_xor(image_flat, key)

    # Reshape the encrypted data
    encrypted_image = encrypted_flat.reshape(image.shape)

    return encrypted_image

def decrypt_image(encrypted_image, key):
    # Flatten the encrypted image
    encrypted_flat = encrypted_image.flatten()

    # Decrypt the image
    decrypted_flat = np.bitwise_xor(encrypted_flat, key)

    # Reshape the decrypted data
    decrypted_image = decrypted_flat.reshape(encrypted_image.shape)

    return decrypted_image

# Load an image
image =
```

```
cv2.imread('/kaggle/working/HighPassFilter/0/segment_no_0_1.png',
cv2.IMREAD_GRAYSCALE)

# Generate a random key
key = np.random.randint(0, 256, size=image.size, dtype=np.uint8)

# Encrypt the image
encrypted_image = encrypt_image(image, key)

# Decrypt the image
decrypted_image = decrypt_image(encrypted_image, key)

# Display the images
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

# Original Image
axs[0].imshow(image, cmap='gray')
axs[0].set_title('Original Image')
axs[0].axis('off')

# Encrypted Image
axs[1].imshow(encrypted_image, cmap='gray')
axs[1].set_title('Encrypted Image')
axs[1].axis('off')

# Decrypted Image
axs[2].imshow(decrypted_image, cmap='gray')
axs[2].set_title('Decrypted Image')
axs[2].axis('off')

plt.show()
```