s-dl-belindamutiara-2540119596-no2

May 6, 2023

1 Belinda Mutiara

1.0.1 NIM: 2540119596 | Kelas: LA09

LINK VIDEO

https://drive.google.com/drive/folders/1iIxpxIgVLqB1G-cWYxDCVfliwRd05f-k?usp=sharing

2 Mengimport Library

[ ]:
```
import os
import pandas as pd
import numpy as np
import tensorflow as tf
tf.random.set_seed(42)
import cv2
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from random import randint

from tensorflow.keras import layers
from tensorflow.keras import Model

from google.colab import drive
import keras

import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
%matplotlib inline
```

Kode diatas dilakukan untuk mengimport library yang dibutuhkan untuk keperluan analisis dan modelling.

[ ]:
```
import tensorflow as tf
tf.test.gpu_device_name()
```

[ ]:
```
'/device:GPU:0'
```

Kita akan menggunakan GPU

[ ]:
```
#import data dari google colab

drive.mount('/content/gdrive/')
```

```
Mounted at /content/gdrive/
```

Menghubungkan Goole Colab dengan Google Drive

3 Nomor 2a

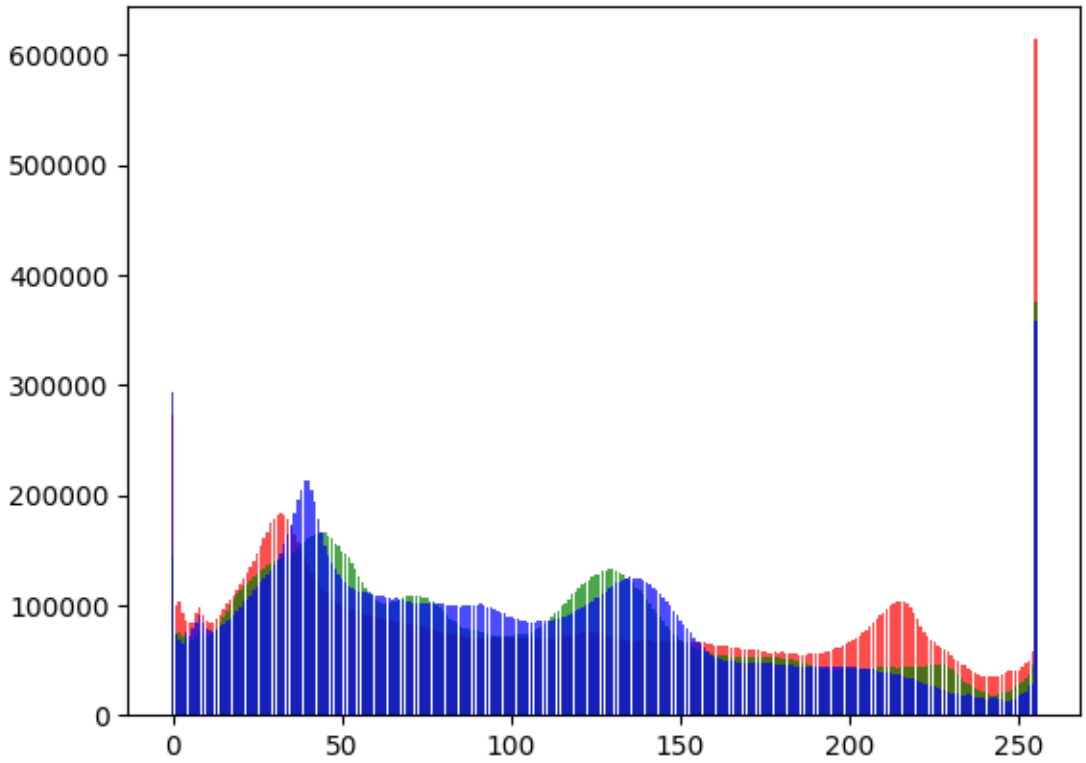
[LO 3, 5 poin] Lakukan eksplorasi terhadap data tersebut dengan melihat histogram warnanya dan lakukan proses augmentasi data jika diperlukan. Dan kemudian lakukan resize resolusi gambar menjadi 64 x 64.

3.1 Eksplorasi warna

3.1.1 Color Histogram per Kelas

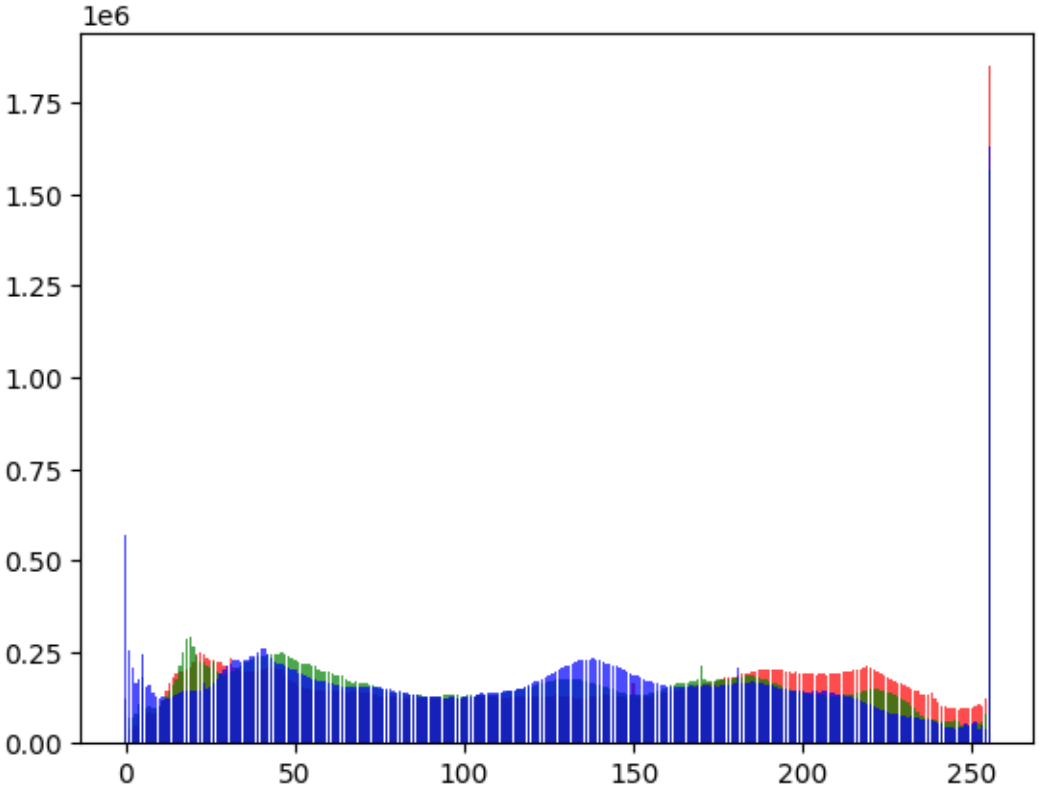
[ ]:
```
from skimage import io
import glob

nb_bins = 256
batik_ciamis = "/content/gdrive/MyDrive/Dataset2B/batik-ciamis"
root = batik_ciamis
image_files = glob.glob(os.path.join(root, "*.jpg"))

red = np.zeros(nb_bins)
green = np.zeros(nb_bins)
blue = np.zeros(nb_bins)

for img in image_files:
    x = io.imread(img)
    hist_r, _ = np.histogram(x[:,:,0], bins=nb_bins, range=[0, 255])
    hist_g, _ = np.histogram(x[:,:,1], bins=nb_bins, range=[0, 255])
    hist_b, _ = np.histogram(x[:,:,2], bins=nb_bins, range=[0, 255])
    red += hist_r
    green += hist_g
    blue += hist_b

bins = np.arange(nb_bins+1)
fig, ax = plt.subplots()
ax.bar(bins[:-1], red, color='r', alpha=0.7)
ax.bar(bins[:-1], green, color='g', alpha=0.7)
ax.bar(bins[:-1], blue, color='b', alpha=0.7)
```

```
plt.show()
```



[ ]:
```
from skimage import io
import glob

nb_bins = 256
batik_garutan = "/content/gdrive/MyDrive/Dataset2B/batik-garutan"
root = batik_garutan
image_files = glob.glob(os.path.join(root, "*.jpg"))

red = np.zeros(nb_bins)
green = np.zeros(nb_bins)
blue = np.zeros(nb_bins)

for img in image_files:
    x = io.imread(img)
    hist_r, _ = np.histogram(x[:,:,0], bins=nb_bins, range=[0, 255])
    hist_g, _ = np.histogram(x[:,:,1], bins=nb_bins, range=[0, 255])
    hist_b, _ = np.histogram(x[:,:,2], bins=nb_bins, range=[0, 255])
    red += hist_r
    green += hist_g
    blue += hist_b
```

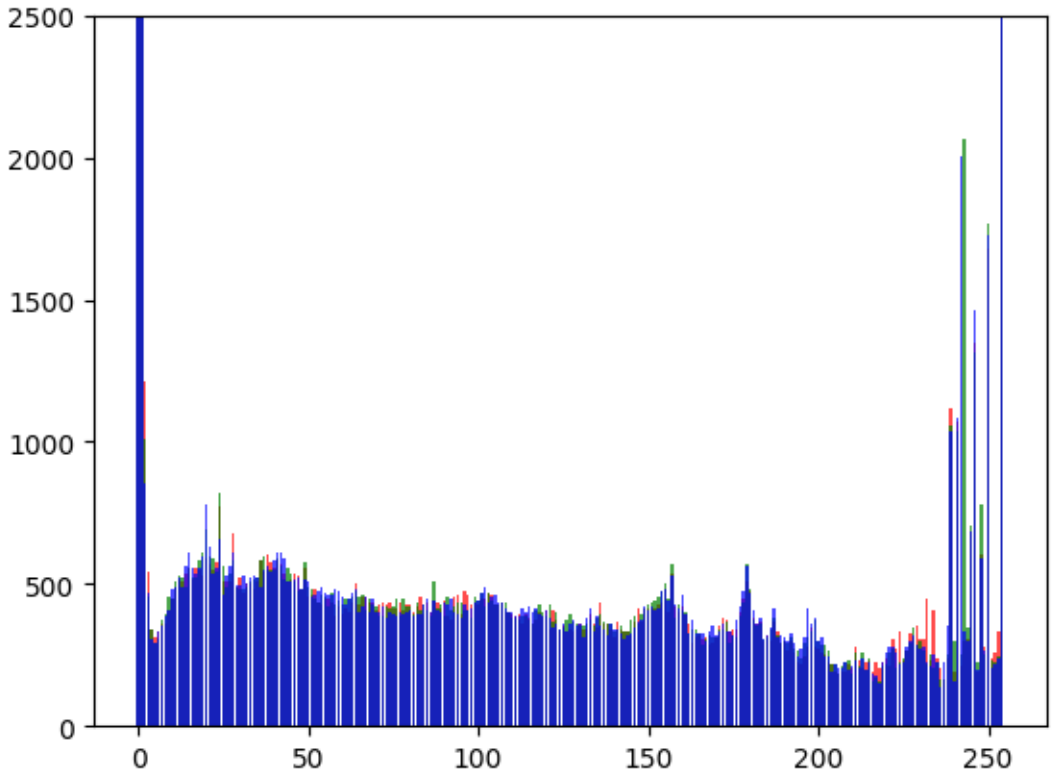
```
bins = np.arange(nb_bins+1)
fig, ax = plt.subplots()
ax.bar(bins[:-1], red, color='r', alpha=0.7)
ax.bar(bins[:-1], green, color='g', alpha=0.7)
ax.bar(bins[:-1], blue, color='b', alpha=0.7)
plt.show()
```



[ ]:
```
from skimage import io
count_r=0
count_g=0
count_b=0

batik_gentongan = "/content/gdrive/MyDrive/Dataset2B/batik-gentongan"
root = batik_gentongan
for image in os.listdir(root):
    if image.endswith('.jpg'):
        x = io.imread(root+'/'+image)
        hist_r = np.histogram(x[0], bins=255, range=[0, 255])
        hist_g = np.histogram(x[1], bins=255, range=[0, 255])
        hist_b = np.histogram(x[2], bins=255, range=[0, 255])
```

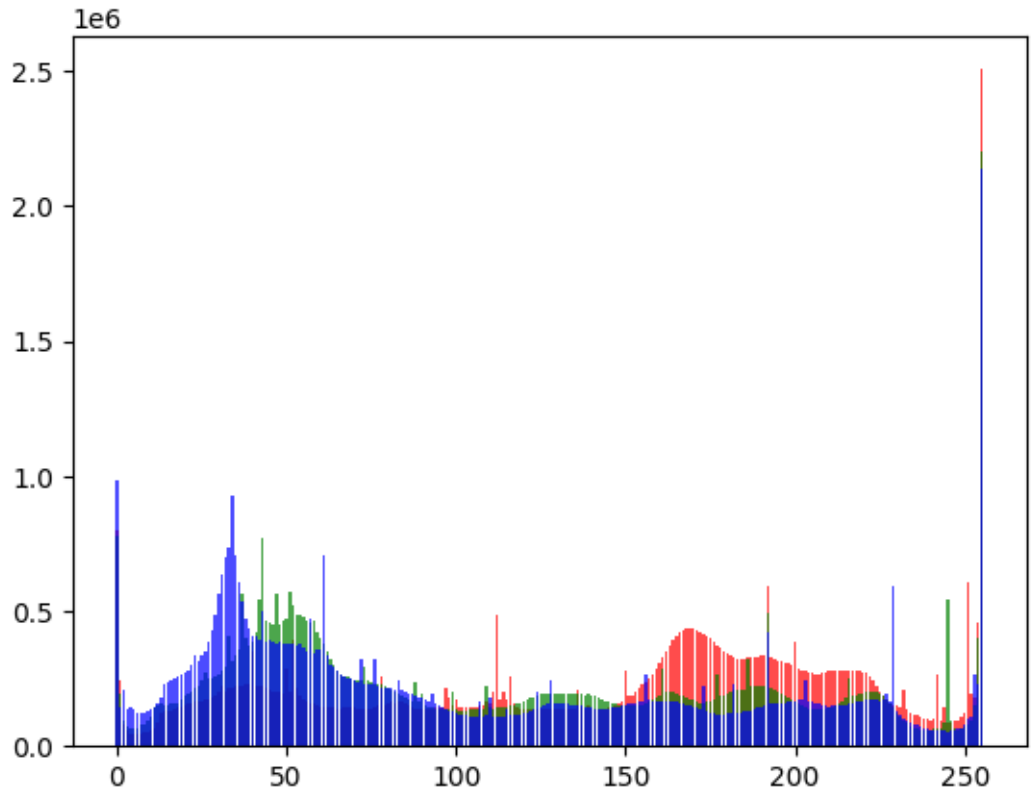
```
        count_r += hist_r[0]
        count_g += hist_g[0]
        count_b += hist_b[0]

bins = hist_r[1]
fig = plt.figure()
plt.ylim([0,2500])
plt.bar(bins[:-1], count_r, color='r', alpha=0.7)
plt.bar(bins[:-1], count_g, color='g', alpha=0.7)
plt.bar(bins[:-1], count_b, color='b', alpha=0.7)
```

[ ]: <BarContainer object of 255 artists>



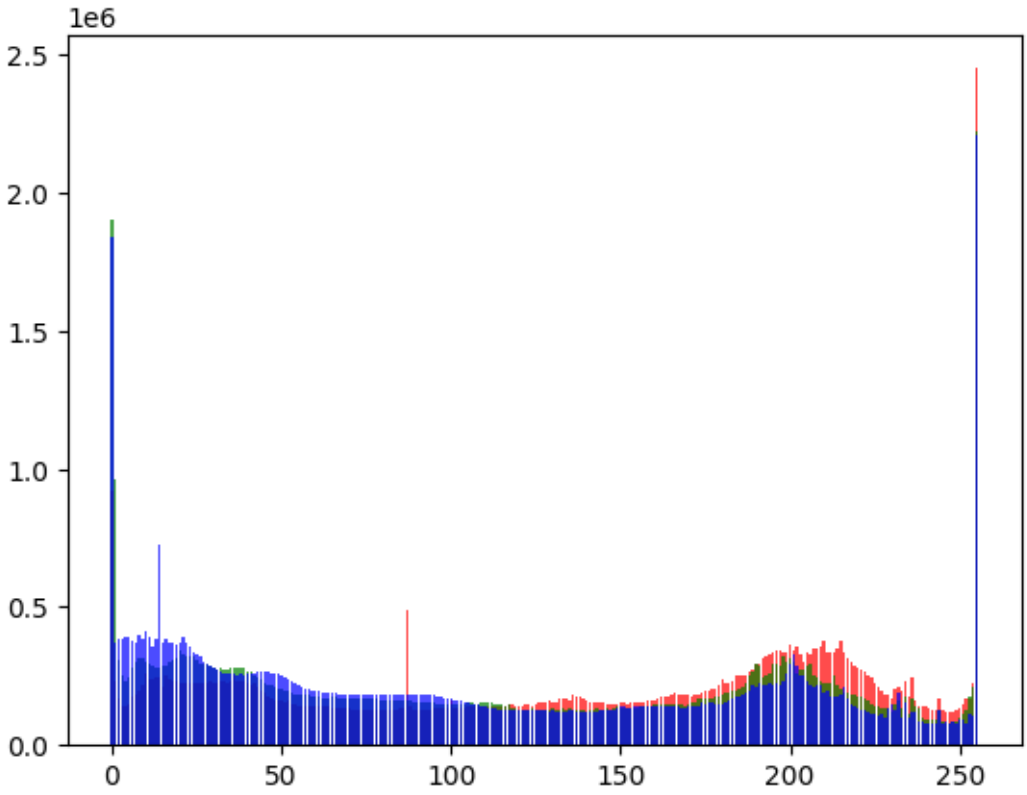
[ ]:
```
from skimage import io
import glob

nb_bins = 256
batik_kawung = "/content/gdrive/MyDrive/Dataset2B/batik-kawung"
root = batik_kawung
image_files = glob.glob(os.path.join(root, "*.jpg"))

red = np.zeros(nb_bins)
```

```
green = np.zeros(nb_bins)
blue = np.zeros(nb_bins)

for img in image_files:
    x = io.imread(img)
    hist_r, _ = np.histogram(x[:,:,0], bins=nb_bins, range=[0, 255])
    hist_g, _ = np.histogram(x[:,:,1], bins=nb_bins, range=[0, 255])
    hist_b, _ = np.histogram(x[:,:,2], bins=nb_bins, range=[0, 255])
    red += hist_r
    green += hist_g
    blue += hist_b

bins = np.arange(nb_bins+1)
fig, ax = plt.subplots()
ax.bar(bins[:-1], red, color='r', alpha=0.7)
ax.bar(bins[:-1], green, color='g', alpha=0.7)
ax.bar(bins[:-1], blue, color='b', alpha=0.7)
plt.show()
```



[ ]:
```
from skimage import io
import glob
```

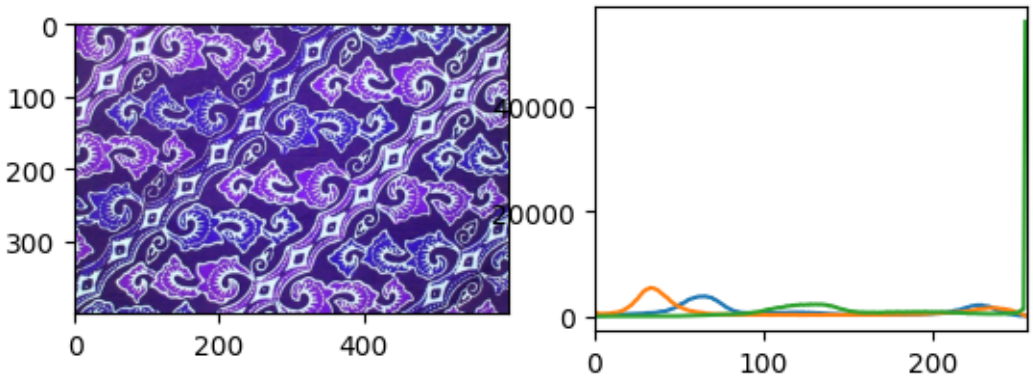
```
nb_bins = 256
batik_keraton = "/content/gdrive/MyDrive/Dataset2B/batik-keraton"
root = batik_keraton
image_files = glob.glob(os.path.join(root, "*.jpg"))

red = np.zeros(nb_bins)
green = np.zeros(nb_bins)
blue = np.zeros(nb_bins)

for img in image_files:
    x = io.imread(img)
    hist_r, _ = np.histogram(x[:,:,0], bins=nb_bins, range=[0, 255])
    hist_g, _ = np.histogram(x[:,:,1], bins=nb_bins, range=[0, 255])
    hist_b, _ = np.histogram(x[:,:,2], bins=nb_bins, range=[0, 255])
    red += hist_r
    green += hist_g
    blue += hist_b

bins = np.arange(nb_bins+1)
fig, ax = plt.subplots()
ax.bar(bins[:-1], red, color='r', alpha=0.7)
ax.bar(bins[:-1], green, color='g', alpha=0.7)
ax.bar(bins[:-1], blue, color='b', alpha=0.7)
plt.show()
```

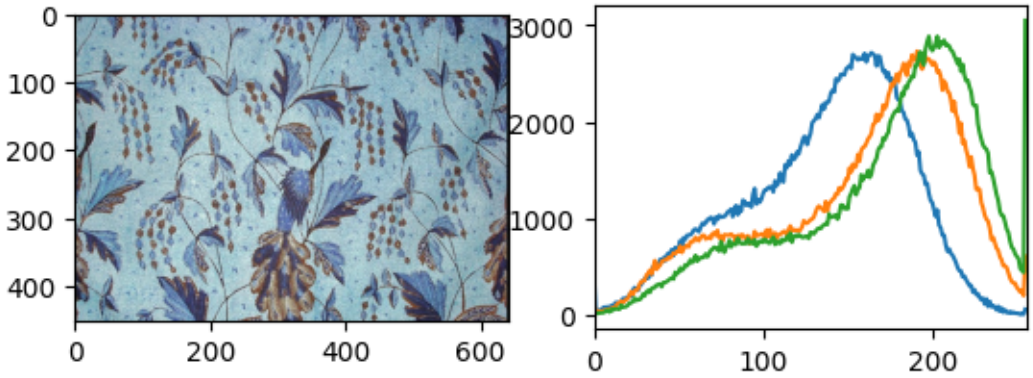


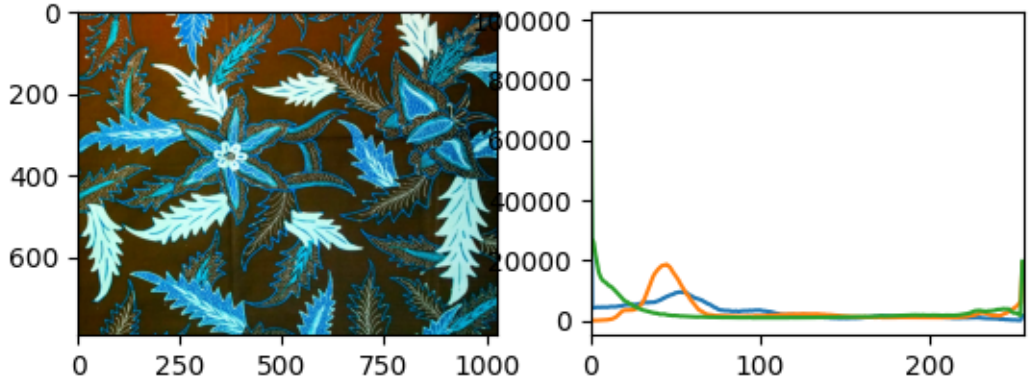
Output diatas merupaka color histogram masing-masing kelas batik, setiap kelas memiliki distribusinya masing-masing dan nampaknya tidak ada kelas yang memiliki distribusi R G B yang normal, distribusi setiap warna di masing-masing kelas yang bervasiasi.

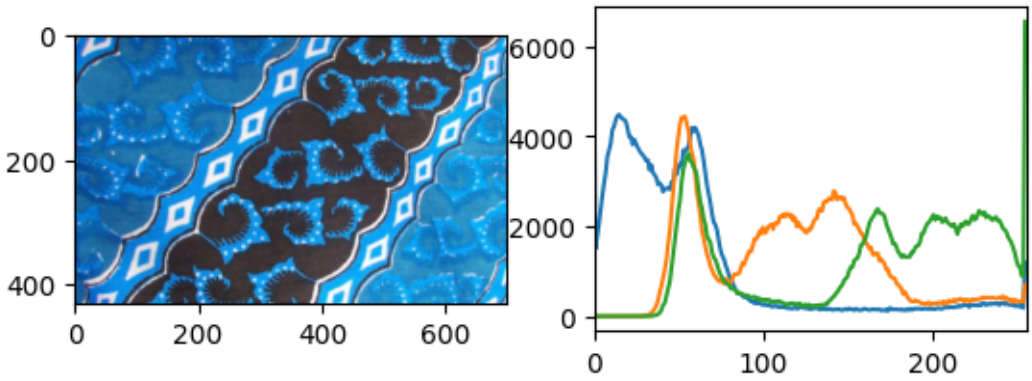
3.1.2 Color Histogram Batik ciamis

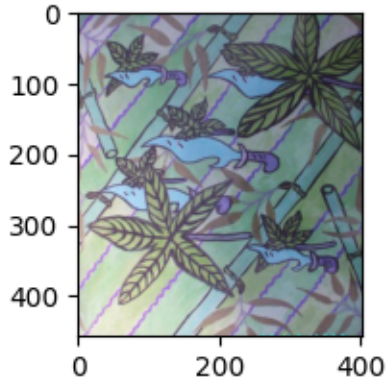
[ ]:
```
import cv2
from google.colab.patches import cv2_imshow

for i in os.listdir('/content/gdrive/MyDrive/Dataset2/Dataset2B/batik-ciamis/'):
  img = cv2.imread('/content/gdrive/MyDrive/Dataset2/Dataset2B/batik-ciamis/'+i)
  hist1 = cv2.calcHist([img],[0],None,[256],[0,256])
  hist2 = cv2.calcHist([img],[1],None,[256],[0,256])
  hist3 = cv2.calcHist([img],[2],None,[256],[0,256])

  plt.subplot(221), plt.imshow(img)
  plt.subplot(222), plt.plot(hist1), plt.plot(hist2),plt.plot(hist3)
  plt.xlim([0,256])

  plt.show()
```

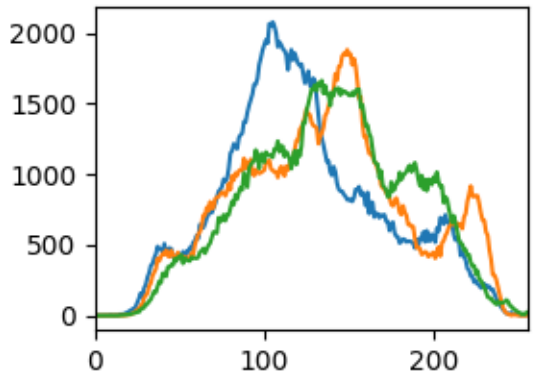

0
100
200
300
0
200
400
40000
20000
0
0
100
200

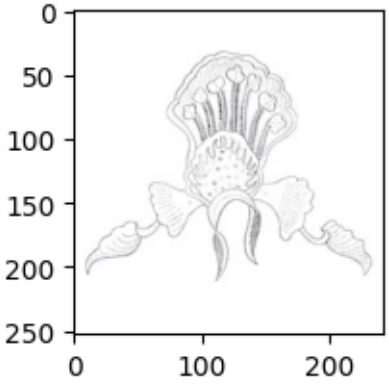

0
100
200
300
400
0
200
400
600
3000
2000
1000
0
0
100
200

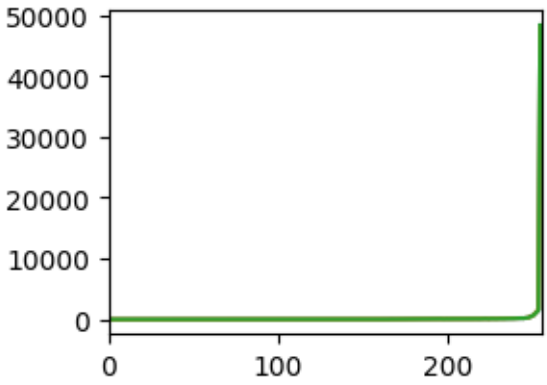

0
200
400
600
0
250
500
750
1000
100000
80000
60000
40000
20000
0
0
100
200

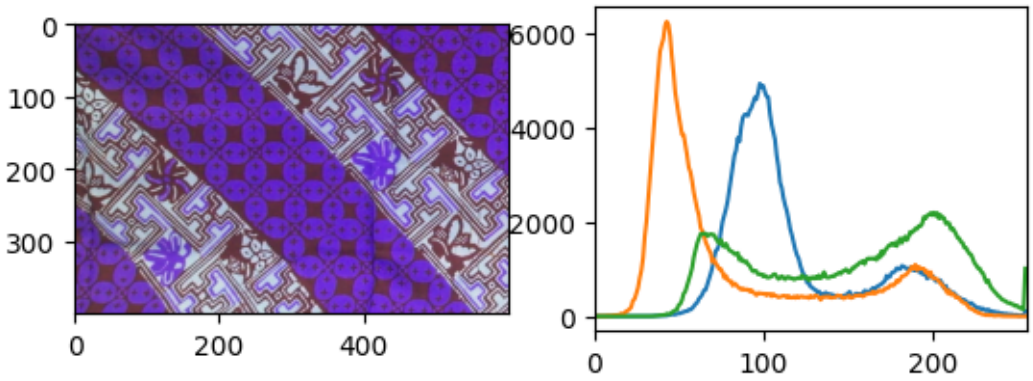

0
200
400
0
200
400
600
6000
4000
2000
0
0
100
200

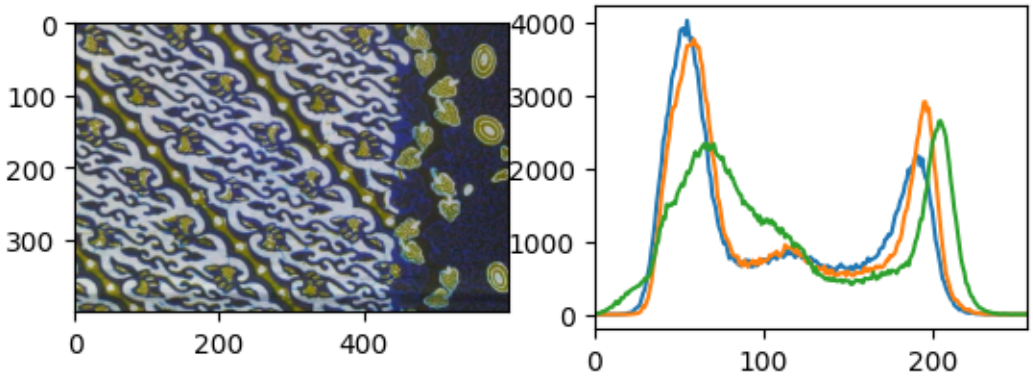

0
100
200
300
400
0
200
400

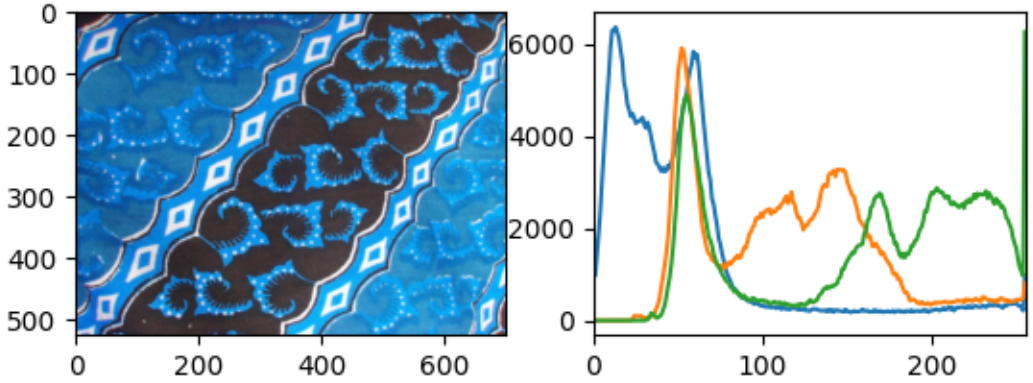

2000
1500
1000
500
0
0
100
200

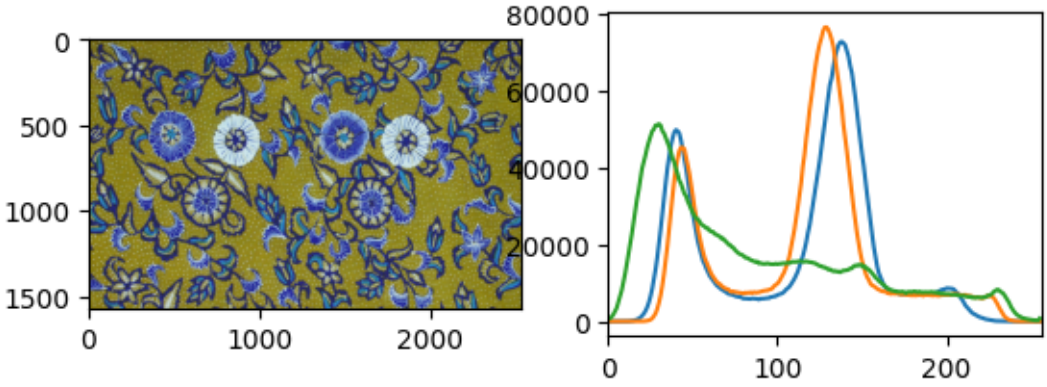

0
50
100
150
200
250
0
100
200

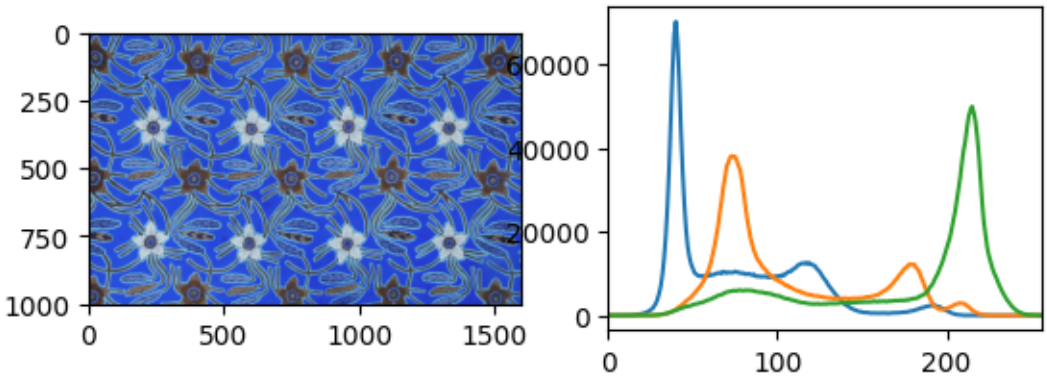

50000
40000
30000
20000
10000
0
0
100
200

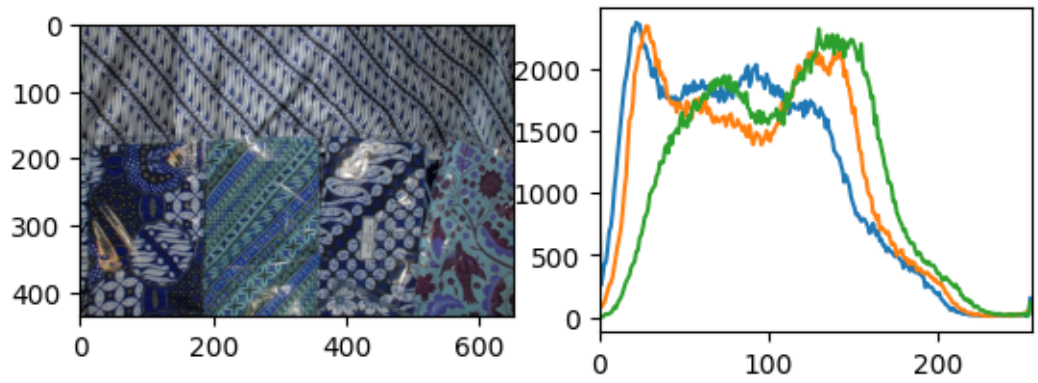

0
100
200
300
0
200
400
6000
4000
2000
0
0
100
200

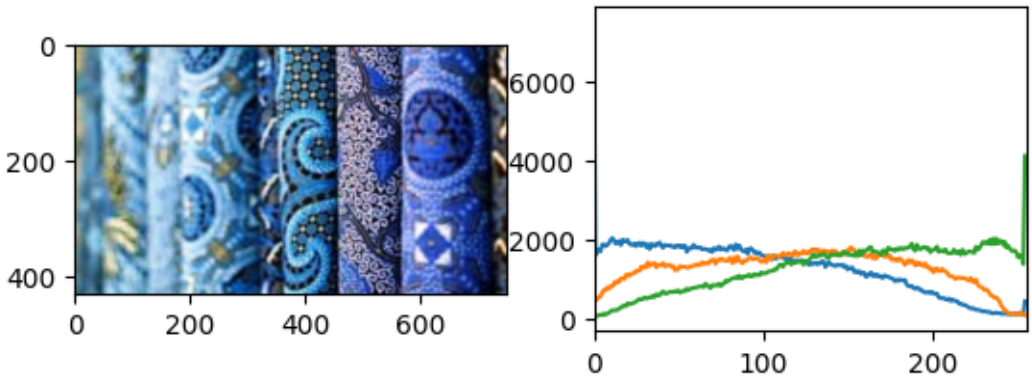

0
100
200
300
0
200
400
4000
3000
2000
1000
0
0
100
200

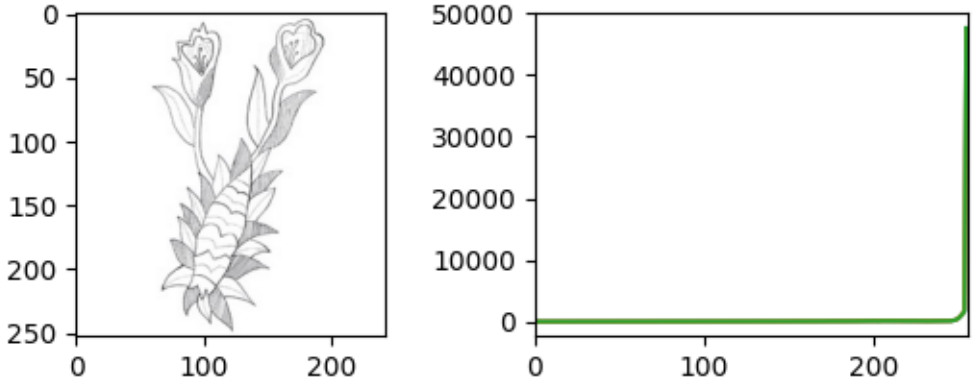

0
100
200
300
400
500
0
200
400
600
6000
4000
2000
0
0
100
200

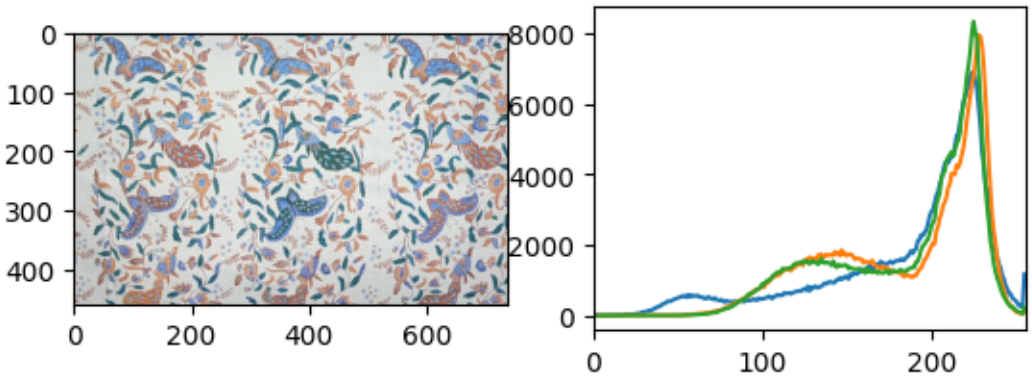

0
500
1000
1500
0
1000
2000
80000
60000
40000
20000
0
0
100
200

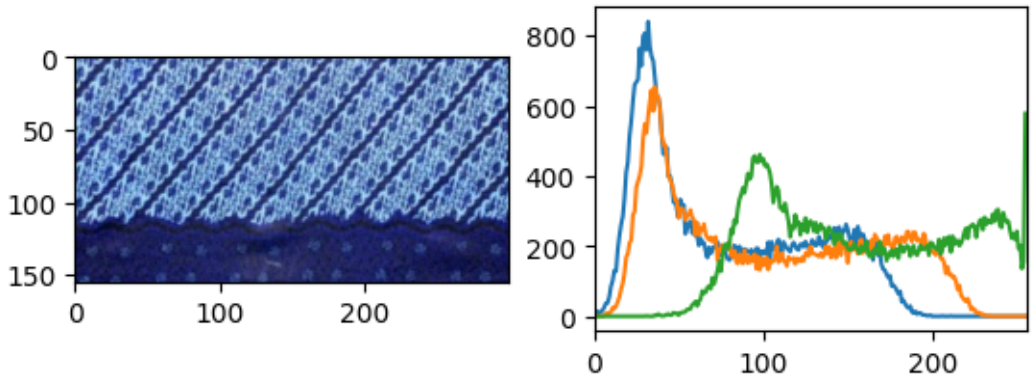

0
250
500
750
1000
0
500
1000
1500
60000
40000
20000
0
0
100
200

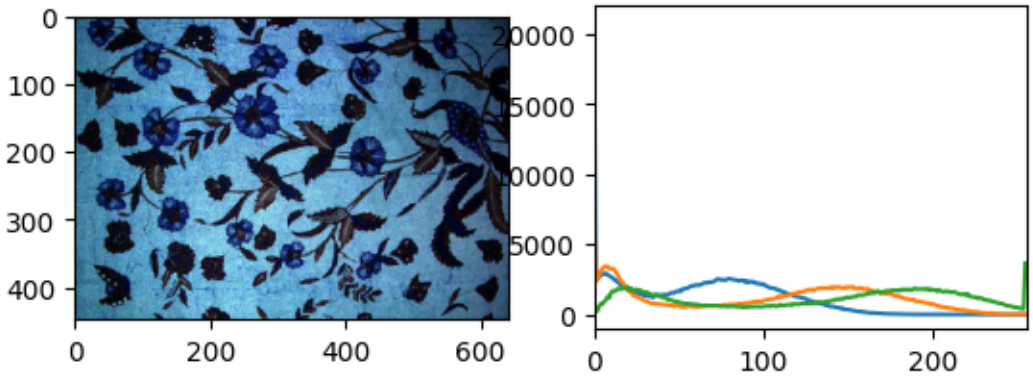

0
100
200
300
400
0
200
400
600
2000
1500
1000
500
0
0
100
200

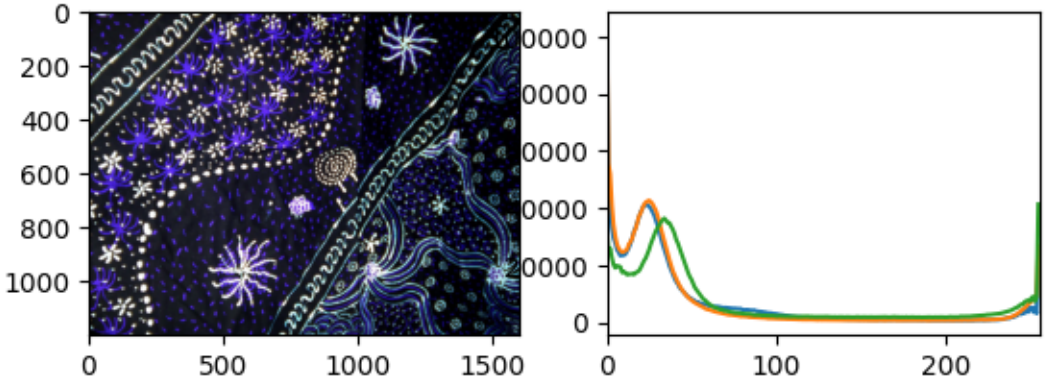

0
200
400
0
200
400
600
6000
4000
2000
0
0
100
200

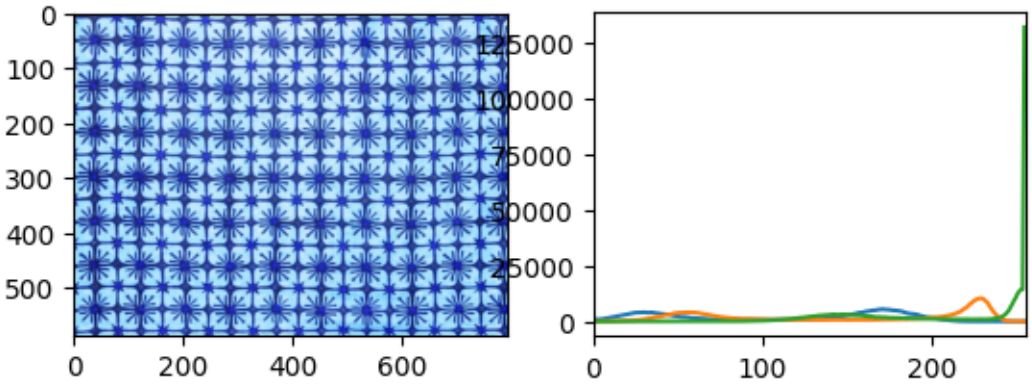

0
50
100
150
200
250
0
100
200
50000
40000
30000
20000
10000
0
0
100
200

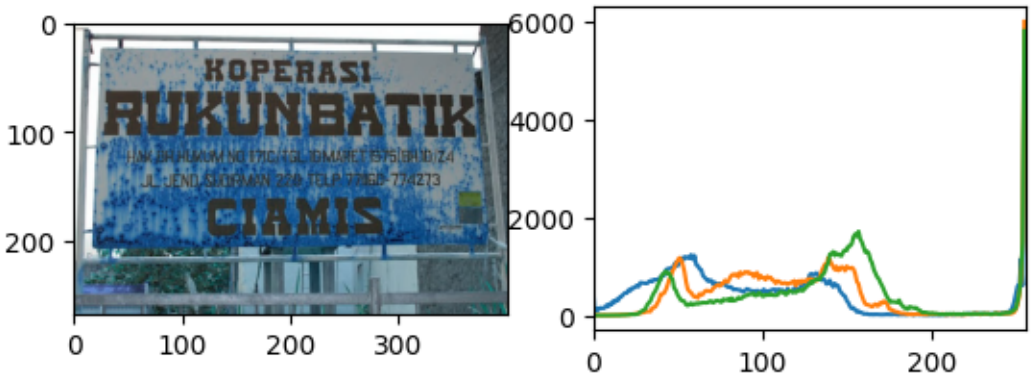

0
100
200
300
400
0
200
400
600
8000
6000
4000
2000
0
0
100
200

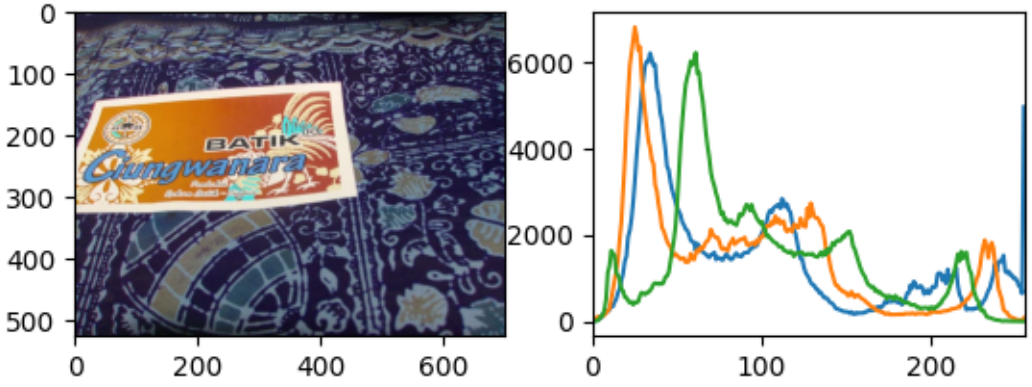

0
50
100
150
0
100
200
800
600
400
200
0
0
100
200

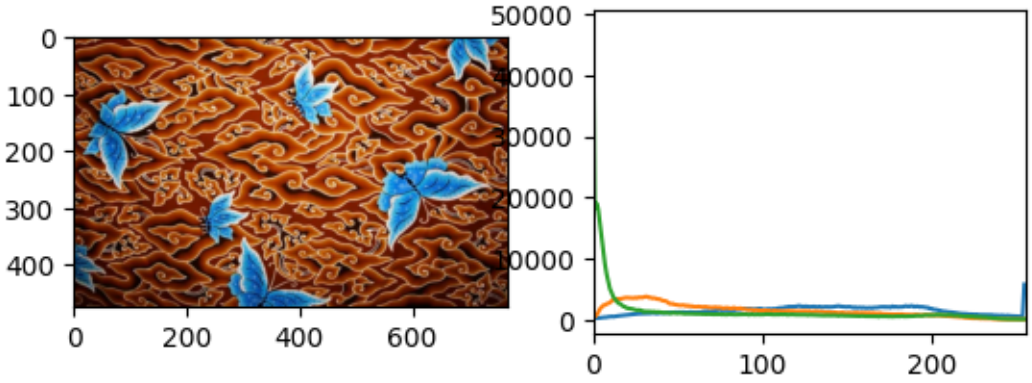

0
100
200
300
400
0
200
400
600
20000
15000
10000
5000
0
0
100
200

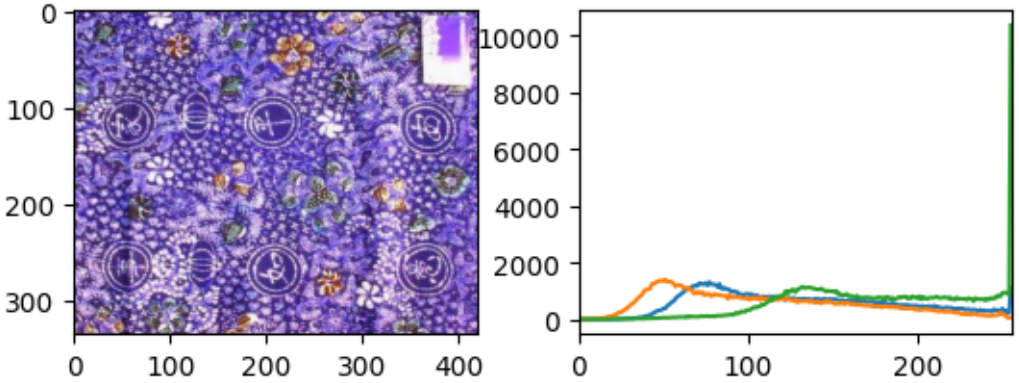

0
200
400
600
800
1000
0
500
1000
1500
0000
0000
0000
0000
0000
0
0
100
200

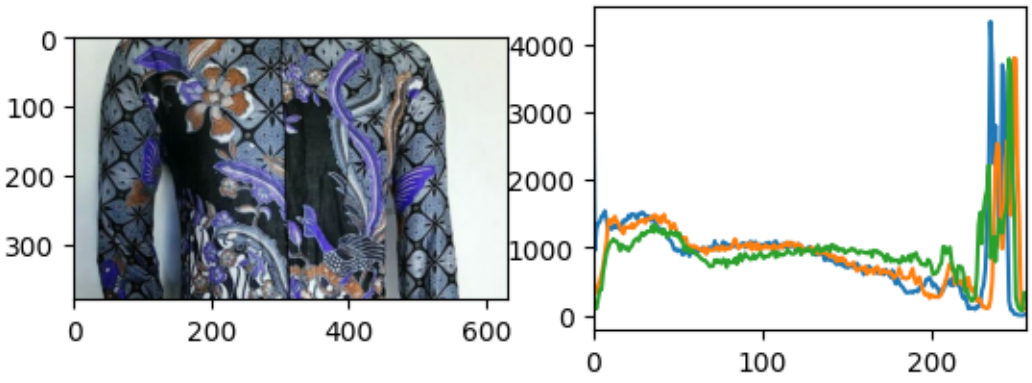

0
100
200
300
400
500
0
200
400
600
125000
100000
75000
50000
25000
0
0
100
200

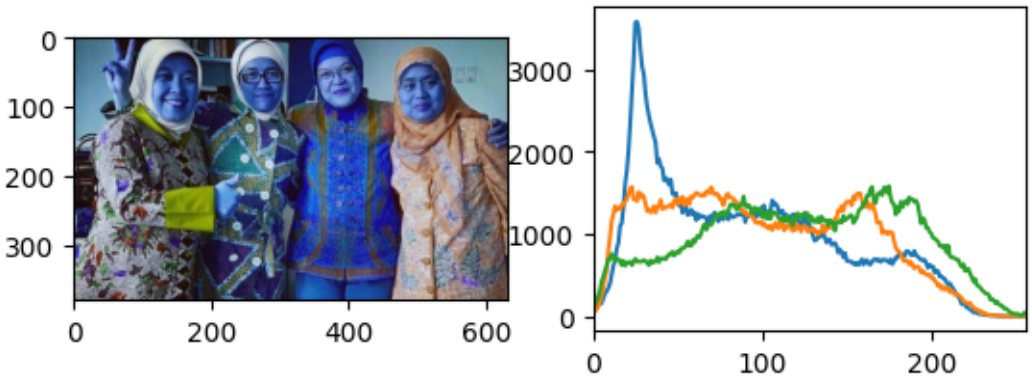

KOPERASI
RUKUNBATIK
CIAMIS
0
100
200
0
100
200
300
6000
4000
2000
0
0
100
200

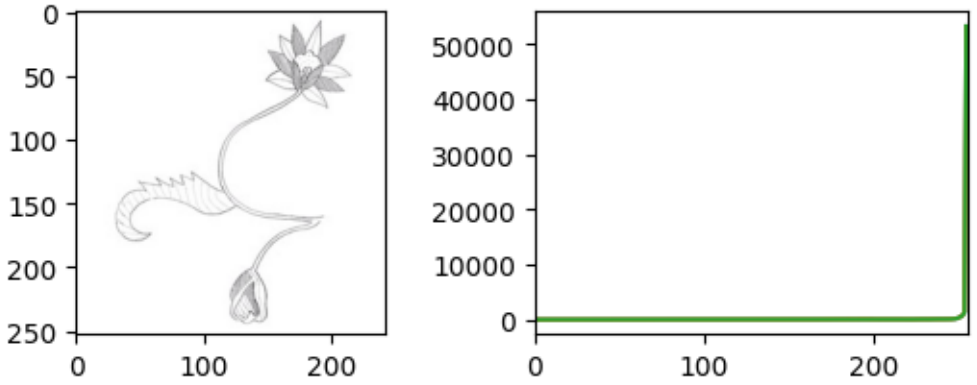

BATIK
Cungwanara
0
100
200
300
400
500
0
200
400
600
6000
4000
2000
0
0
100
200

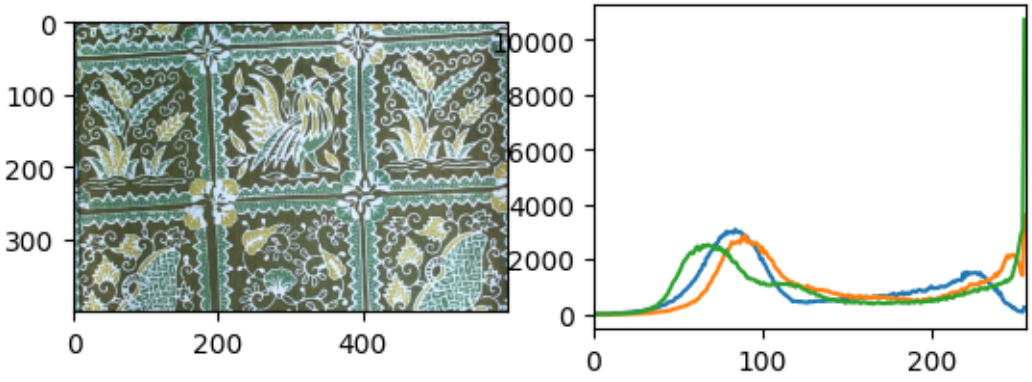

0
100
200
300
400
0
200
400
600
50000
40000
30000
20000
10000
0
0
100
200

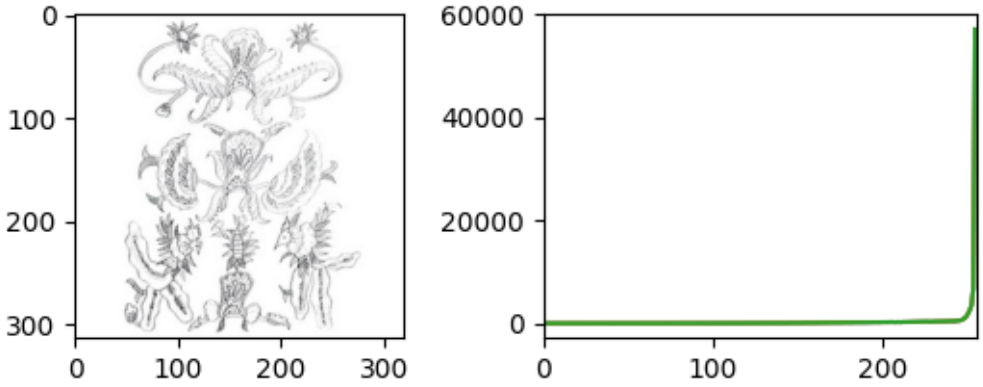

0
100
200
300
0
100
200
300
400
10000
8000
6000
4000
2000
0
0
100
200

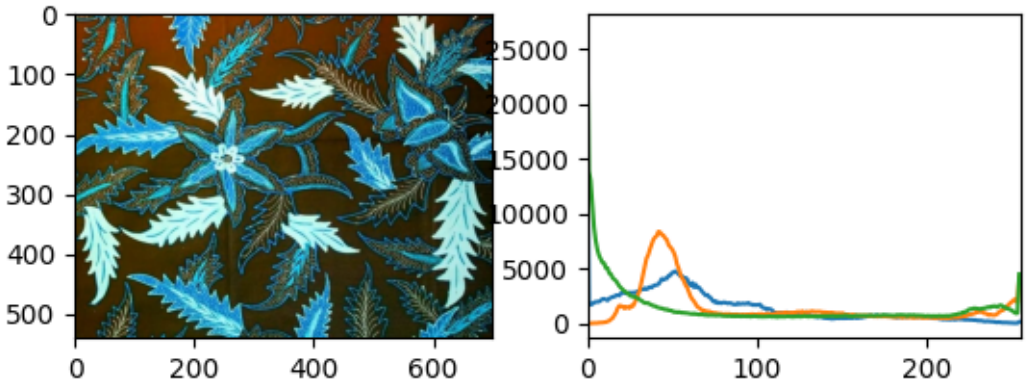

0
100
200
300
0
200
400
600
4000
3000
2000
1000
0
0
100
200

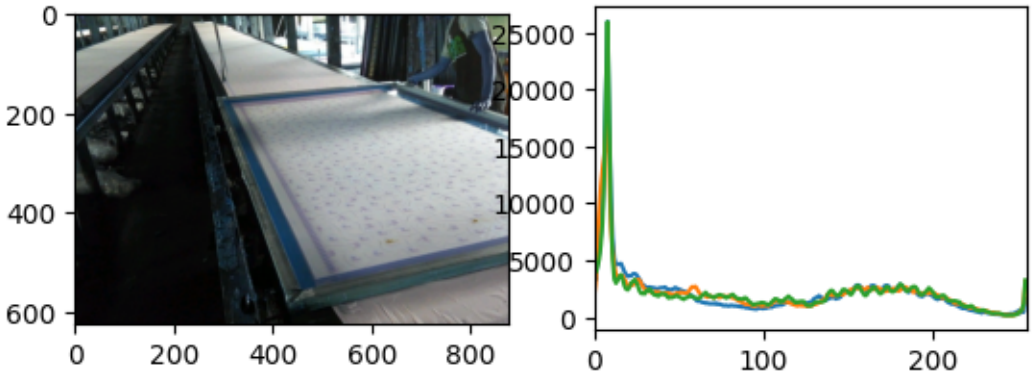

0
100
200
300
0
200
400
600
3000
2000
1000
0
0
100
200

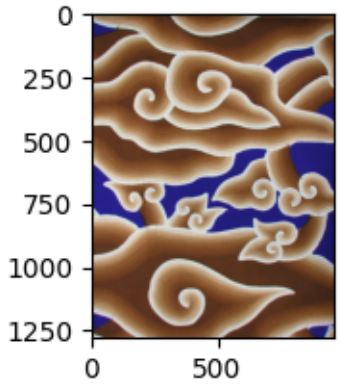

0
50
100
150
200
250
0
100
200
50000
40000
30000
20000
10000
0
0
100
200

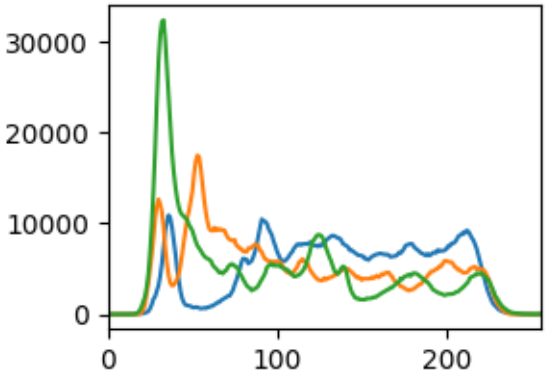

0
100
200
300
0
200
400
10000
8000
6000
4000
2000
0
0
100
200

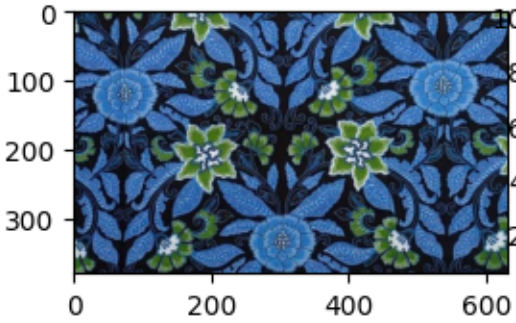

0
100
200
300
0
100
200
300
60000
40000
20000
0
0
100
200

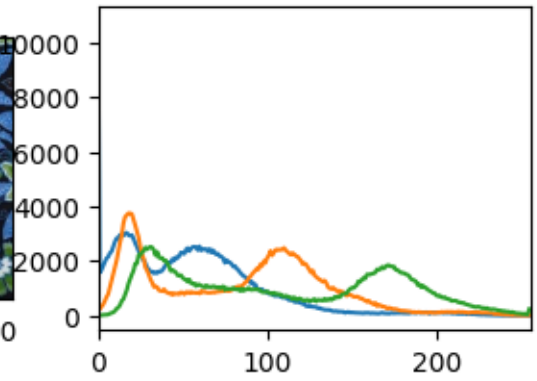

0
100
200
300
400
500
0
200
400
600
25000
20000
15000
10000
5000
0
0
100
200


0
200
400
600
0
200
400
600
800
25000
20000
15000
10000
5000
0
0
100
200

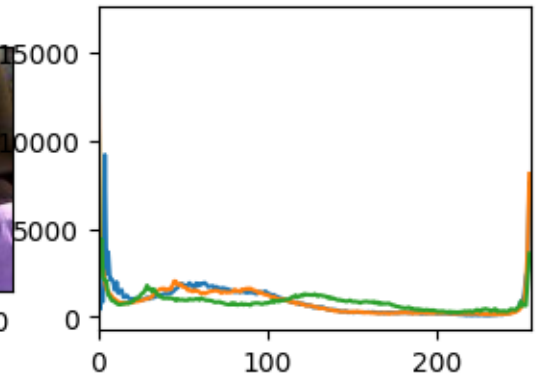

0
250
500
750
1000
1250
0
500

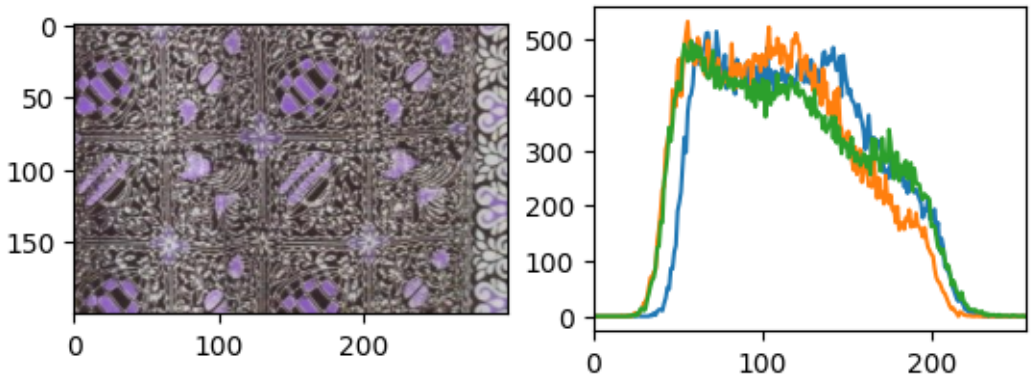

30000
20000
10000
0
0
100
200

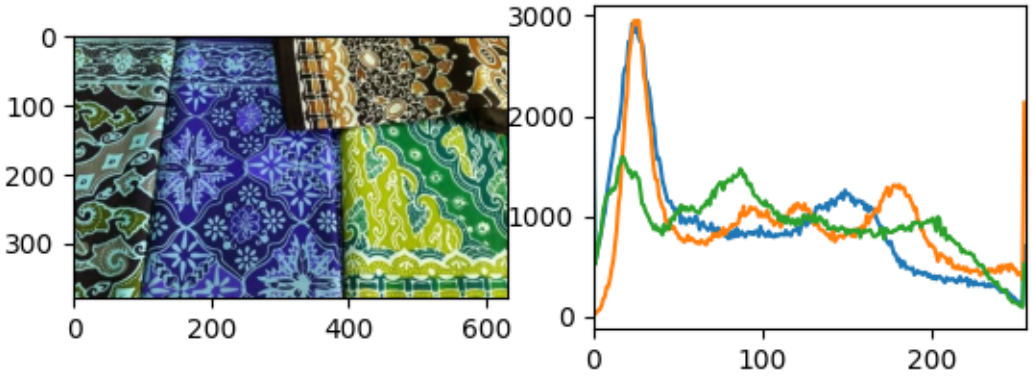

0
100
200
300
0
200
400
600

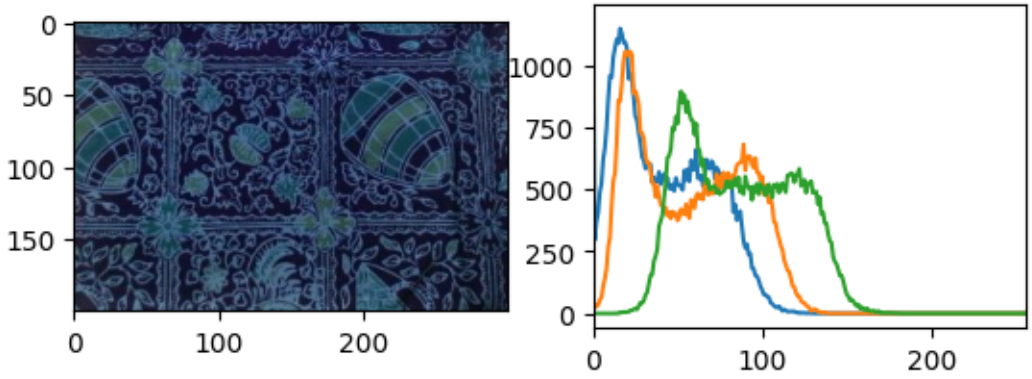

10000
8000
6000
4000
2000
0
0
100
200

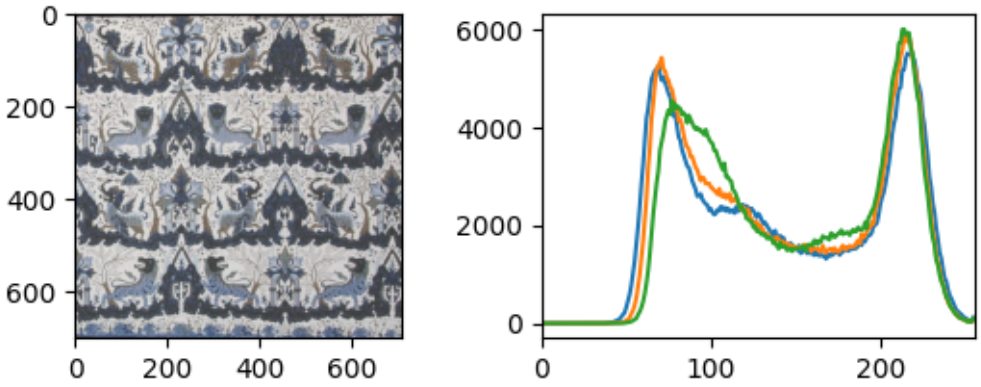

0
100
200
300
0
200
400
600

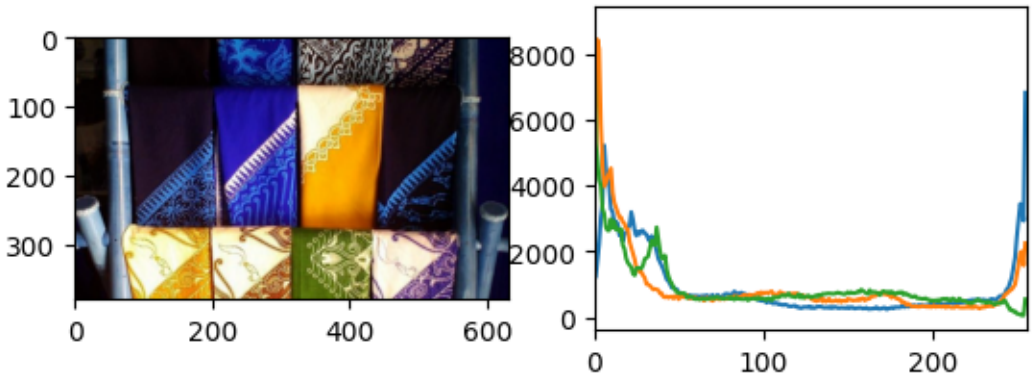

15000
10000
5000
0
0
100
200

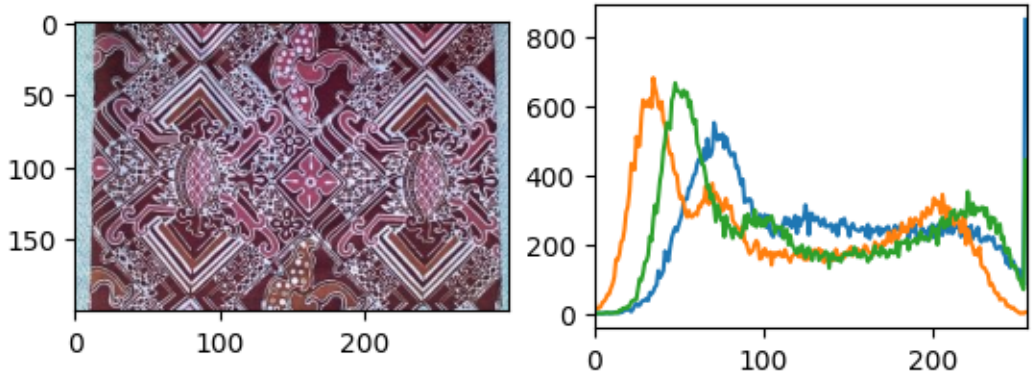

0
50
100
150
0
100
200
500
400
300
200
100
0
0
100
200

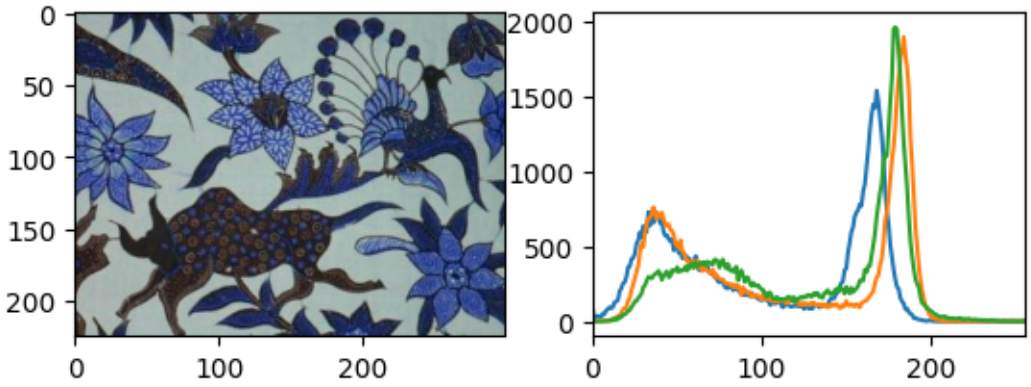

0
100
200
300
0
200
400
600
3000
2000
1000
0
0
100
200

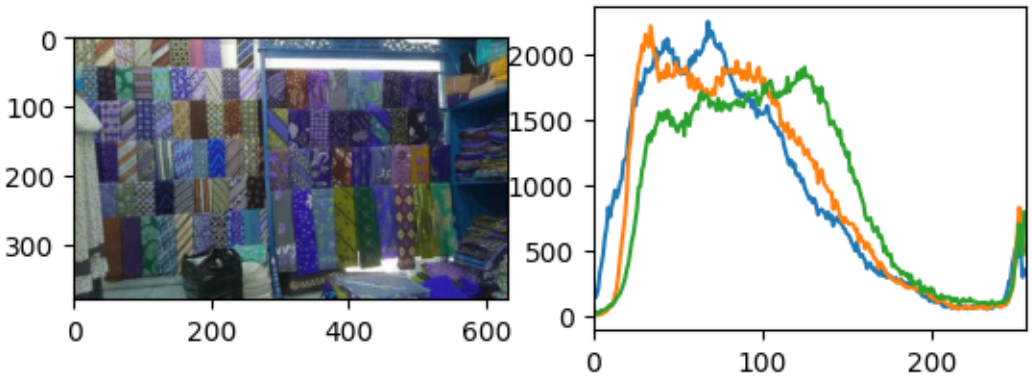

0
50
100
150
0
100
200
1000
750
500
250
0
0
100
200

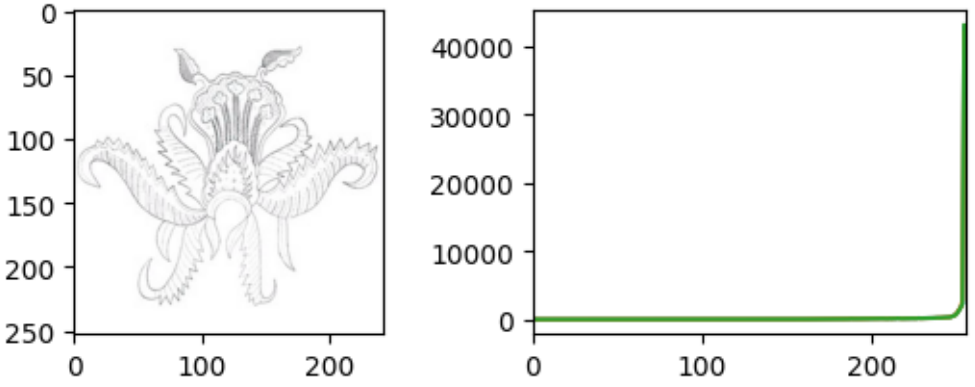

0
200
400
600
0
200
400
600
6000
4000
2000
0
0
100
200

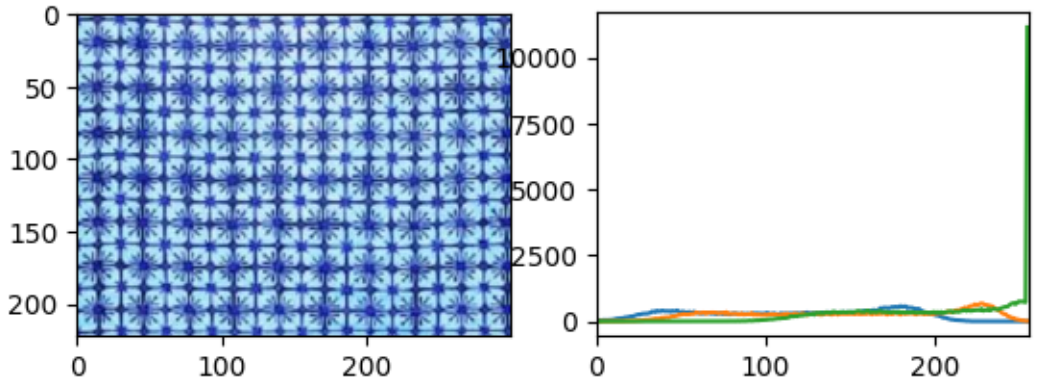

0
100
200
300
0
200
400
600
8000
6000
4000
2000
0
0
100
200

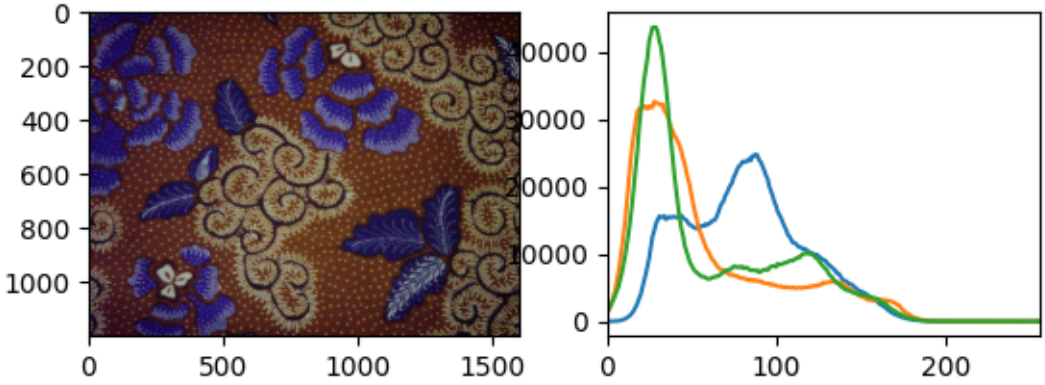

0
50
100
150
0
100
200
800
600
400
200
0
0
100
200

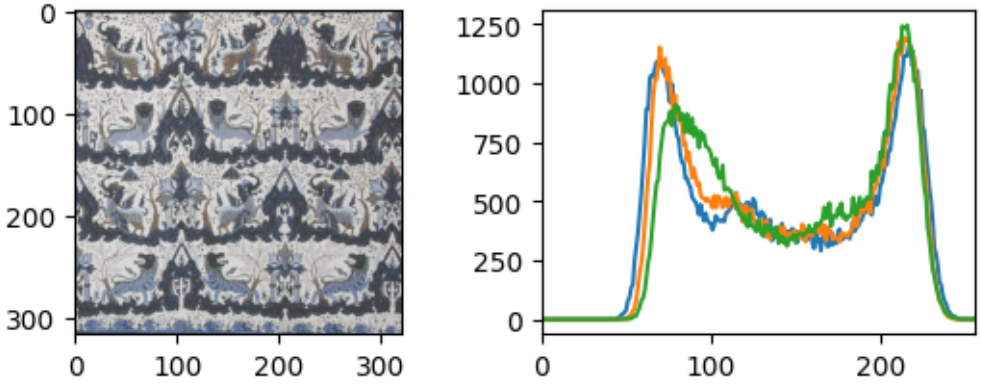

0
50
100
150
200
0
100
200
2000
1500
1000
500
0
0
100
200

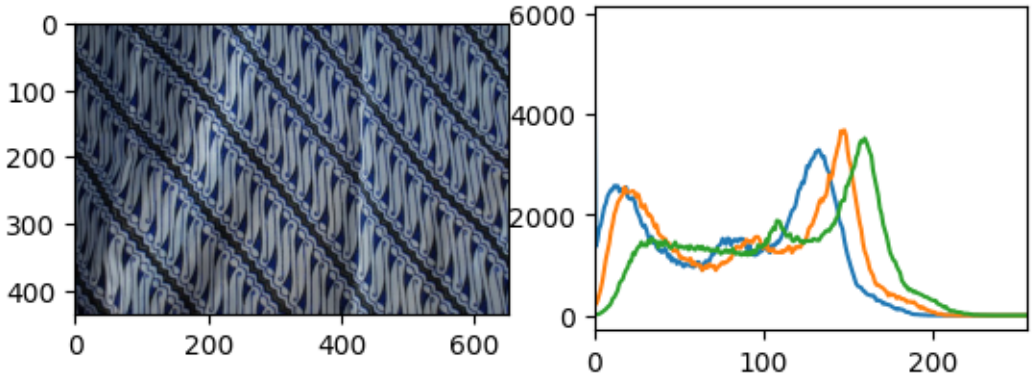

0
100
200
300
0
200
400
600
2000
1500
1000
500
0
0
100
200

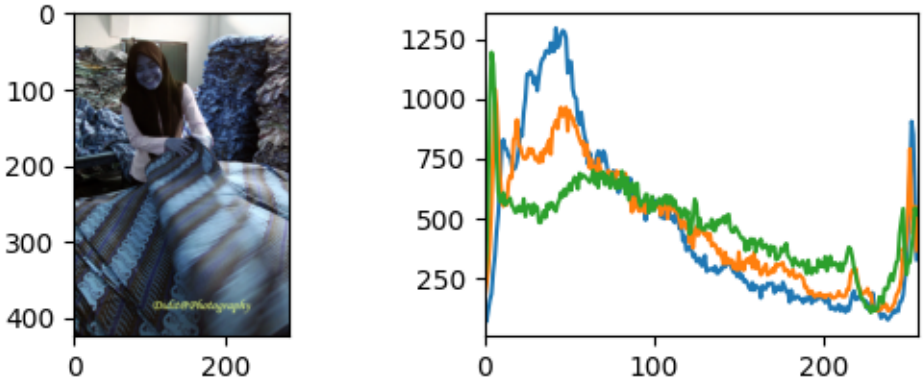

0
50
100
150
200
250
0
100
200
40000
30000
20000
10000
0
0
100
200

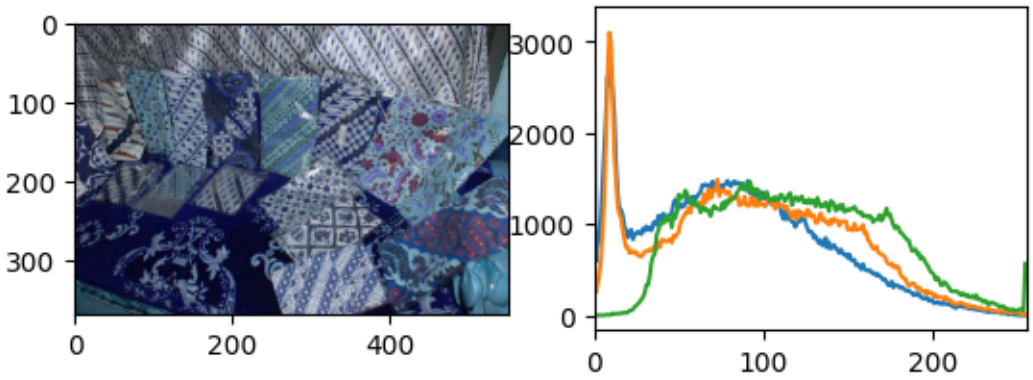

0
50
100
150
200
0
100
200
10000
7500
5000
2500
0
0
100
200

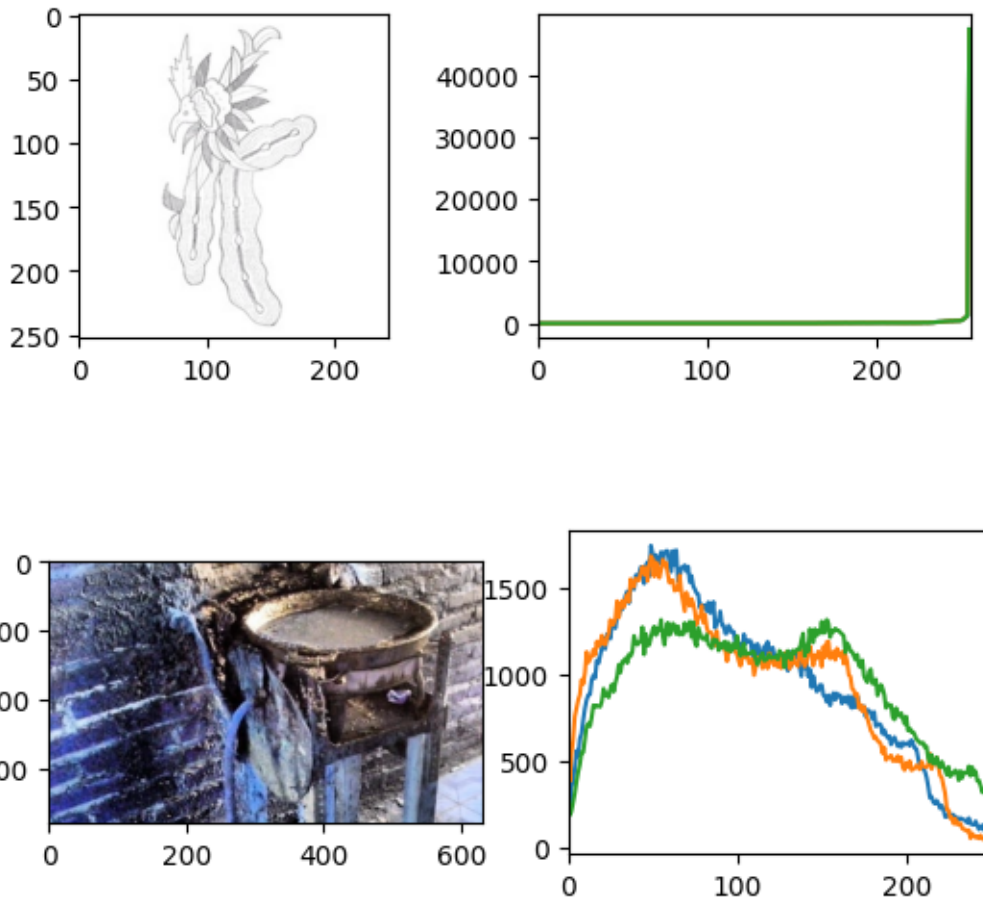

0
200
400
600
800
1000
0
500
1000
1500
0
100
200


0
100
200
300
0
100
200
300
1250
1000
750
500
250
0
0
100
200


0
100
200
300
400
0
200
400
600
6000
4000
2000
0
0
100
200

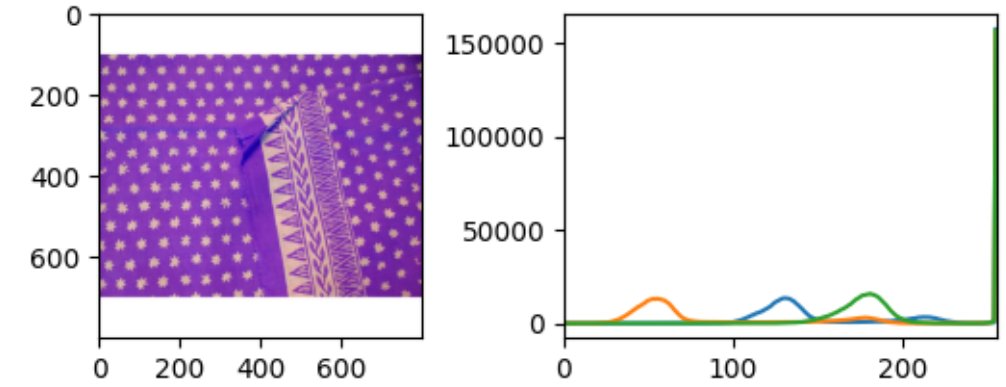

0
100
200
300
400
0
200
1250
1000
750
500
250
0
100
200

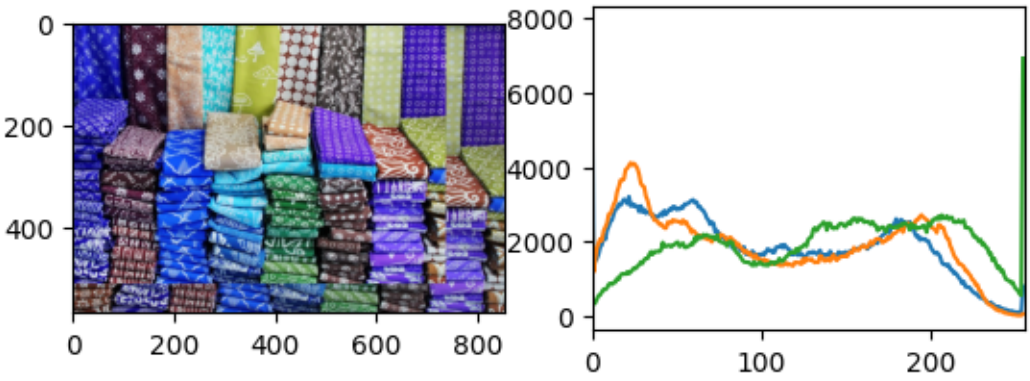

0
100
200
300
0
200
400
3000
2000
1000
0
0
100
200

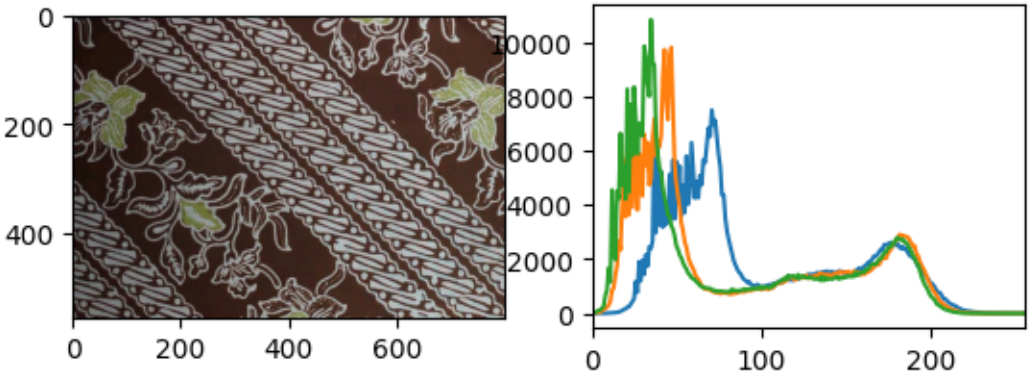


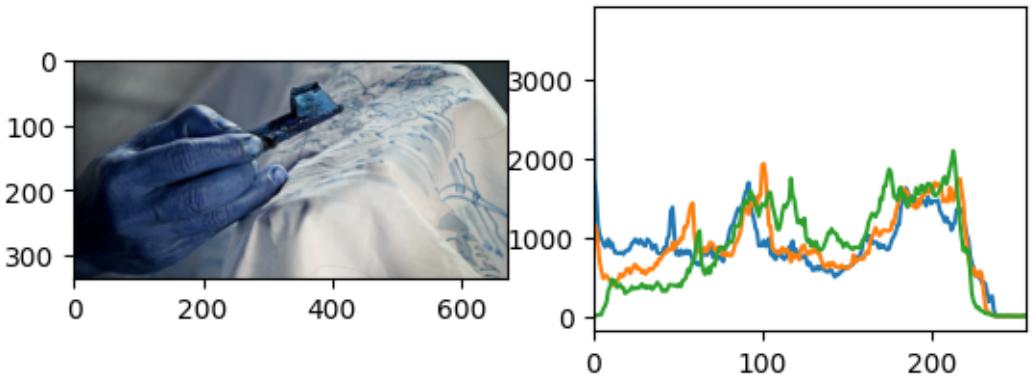
3.1.3 Color Histogram Batik Garutan

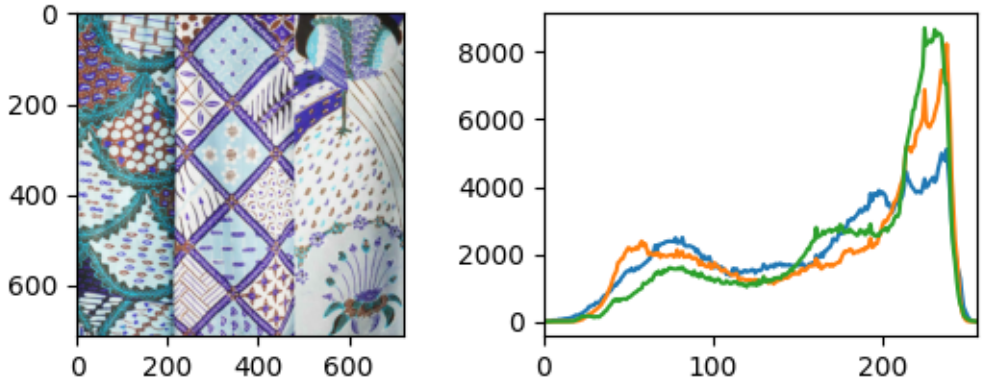
```
[ ]: for i in os.listdir('/content/gdrive/MyDrive/Dataset2/Dataset2B/batik-garutan/
     ↪'):
      img = cv2.imread('/content/gdrive/MyDrive/Dataset2/Dataset2B/batik-garutan/
     ↪'+i)
      hist1 = cv2.calcHist([img],[0],None,[256],[0,256])
      hist2 = cv2.calcHist([img],[1],None,[256],[0,256])
      hist3 = cv2.calcHist([img],[2],None,[256],[0,256])

      plt.subplot(221), plt.imshow(img)
      plt.subplot(222), plt.plot(hist1), plt.plot(hist2),plt.plot(hist3)
      plt.xlim([0,256])

      plt.show()
```

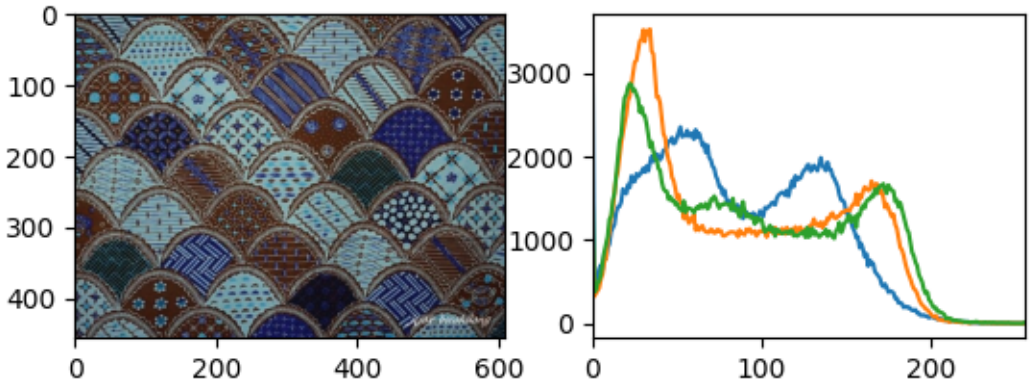

0
200
400
600
0
200
400
600
150000
100000
50000
0
0
100
200

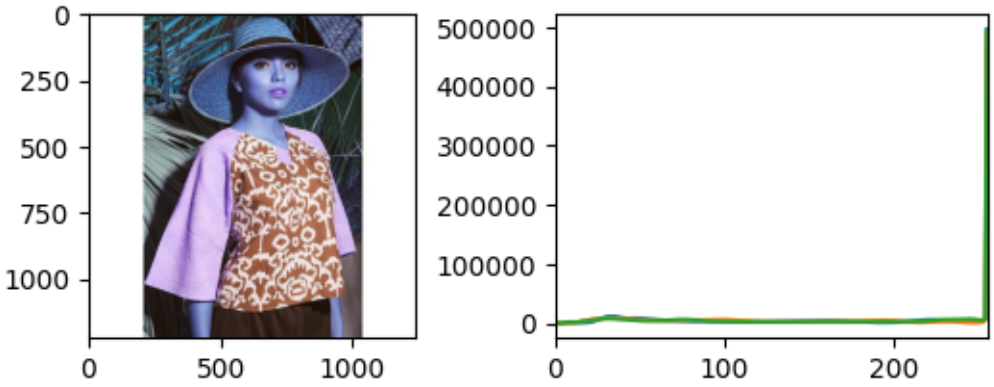

0
200
400
0
200
400
600
800
8000
6000
4000
2000
0
0
100
200

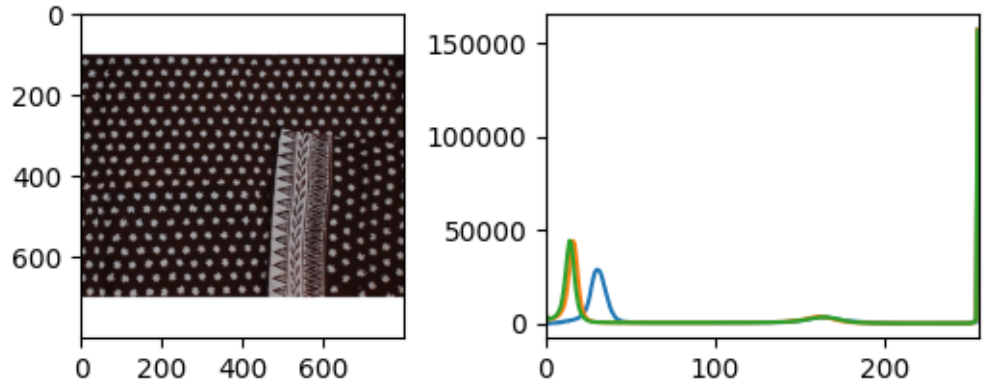

0
200
400
0
200
400
600
10000
8000
6000
4000
2000
0
0
100
200

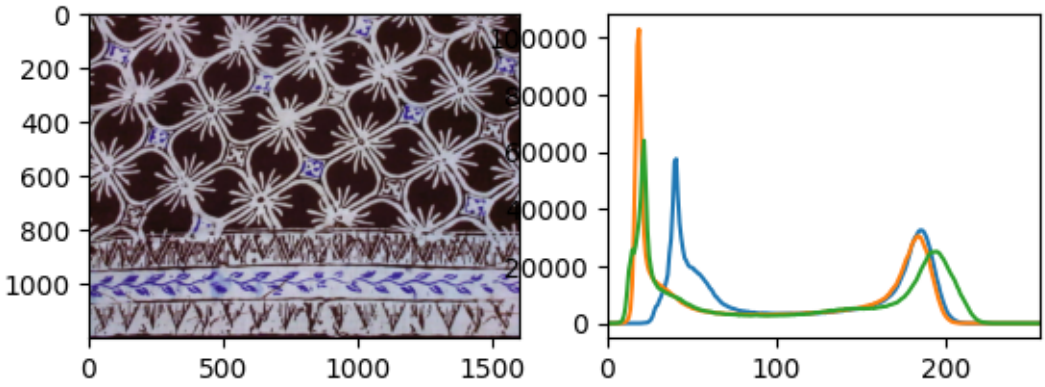

0
100
200
300
0
200
400
600
3000
2000
1000
0
0
100
200

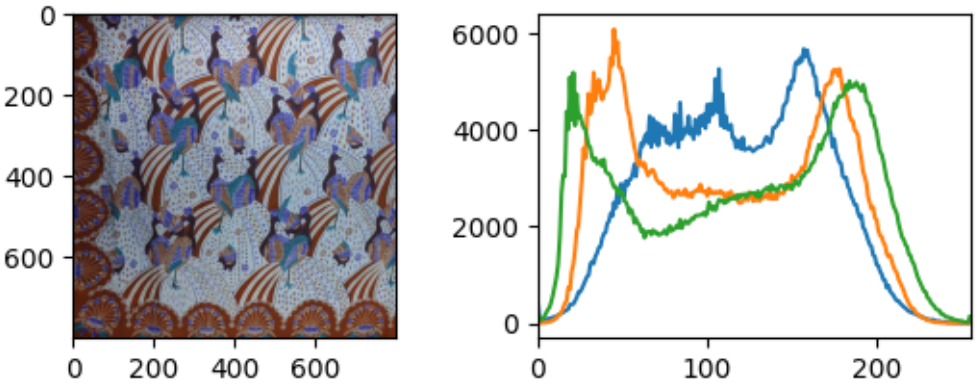

0
200
400
600
0
200
400
600
8000
6000
4000
2000
0
0
100
200

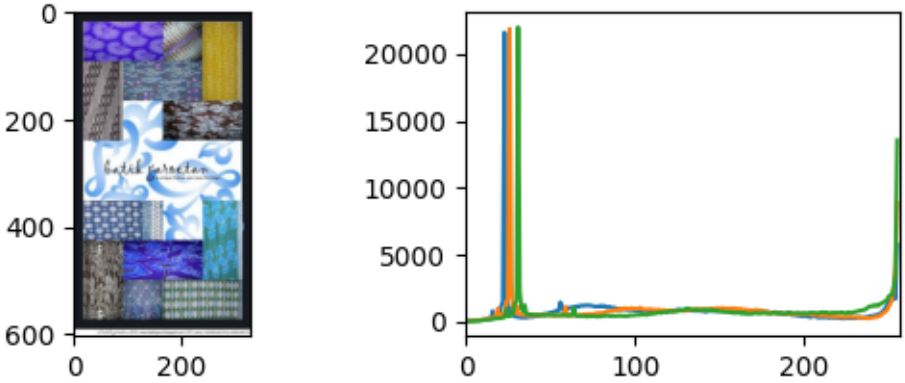

0
100
200
300
400
0
200
400
600
3000
2000
1000
0
0
100
200

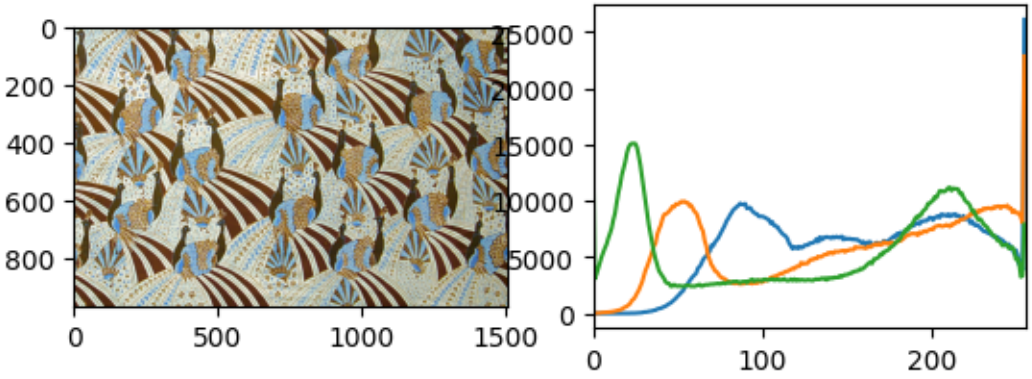

0
250
500
750
1000
0
500
1000
500000
400000
300000
200000
100000
0
0
100
200

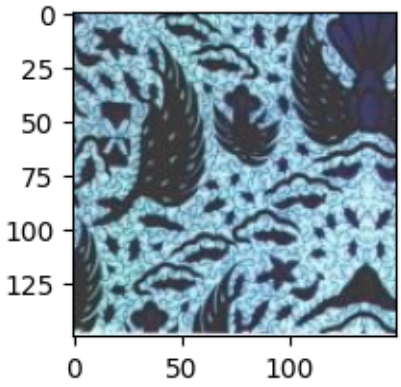

0
200
400
600
0
200
400
600
150000
100000
50000
0
0
100
200

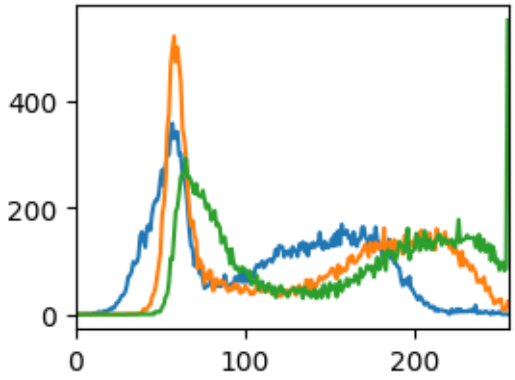

0
200
400
600
800
1000
0
500
1000
1500
0
100
200

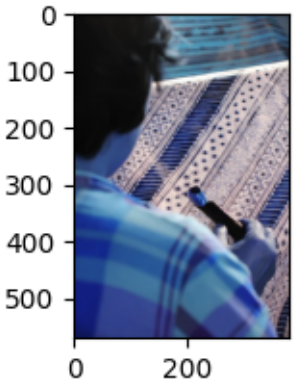

0
200
400
600
0 200 400 600
6000
4000
2000
0
0 100 200

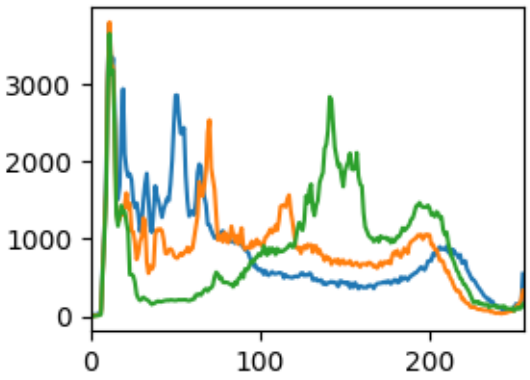

0
200
400
600
0 200
20000
15000
10000
5000
0
0 100 200

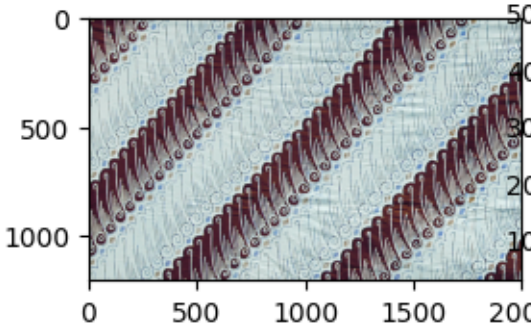

0
200
400
600
800
0 500 1000 1500
25000
20000
15000
10000
5000
0
0 100 200

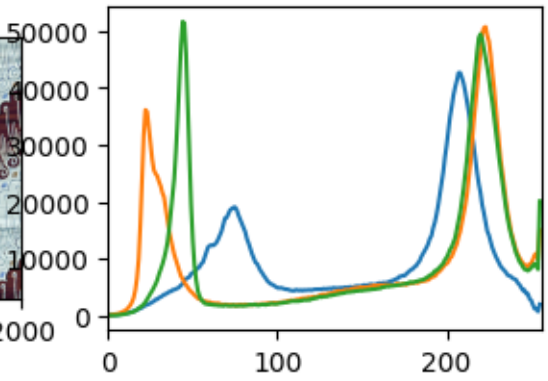

0
25
50
75
100
125
0
50
100

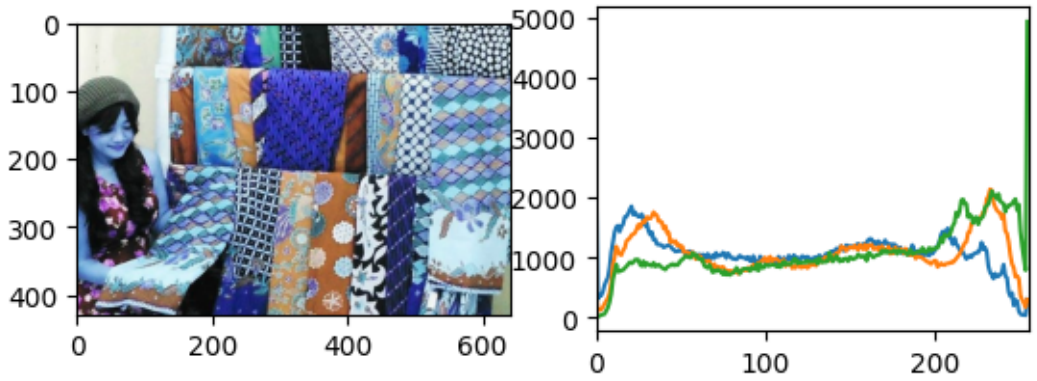

400
200
0
0
100
200

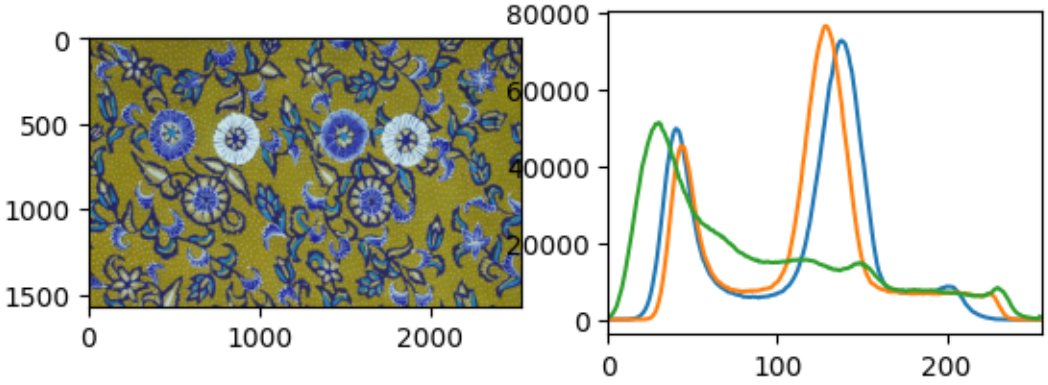

0
100
200
300
400
500
0
200

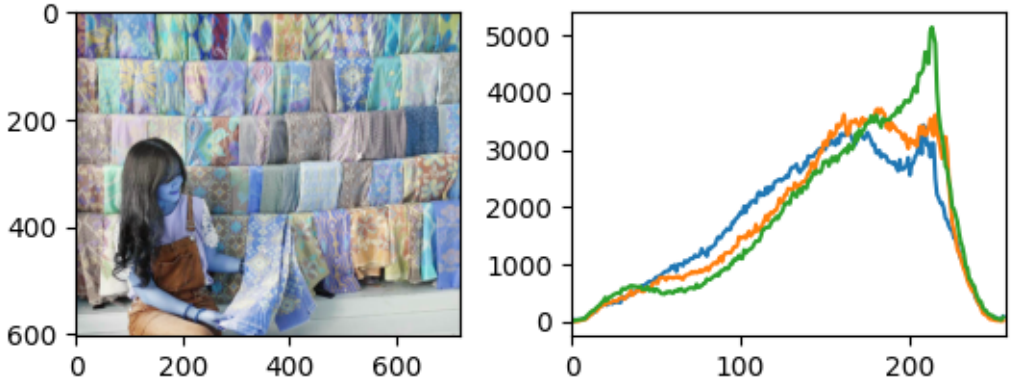

3000
2000
1000
0
0
100
200

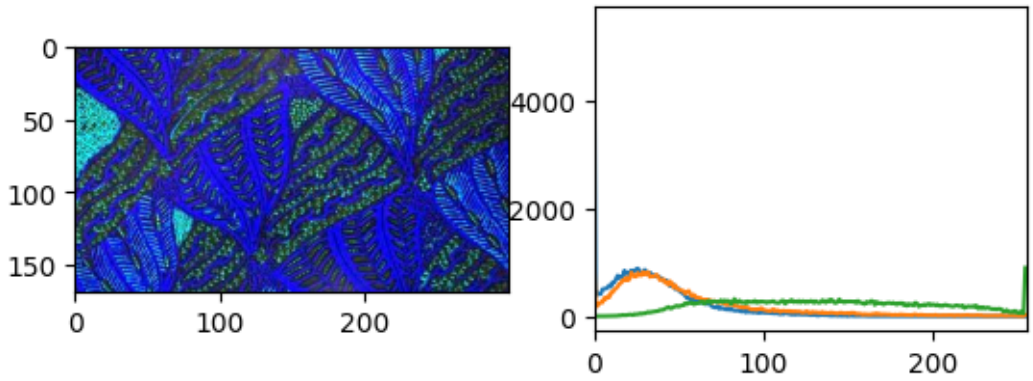

0
500
1000
0
500
1000
1500
2000

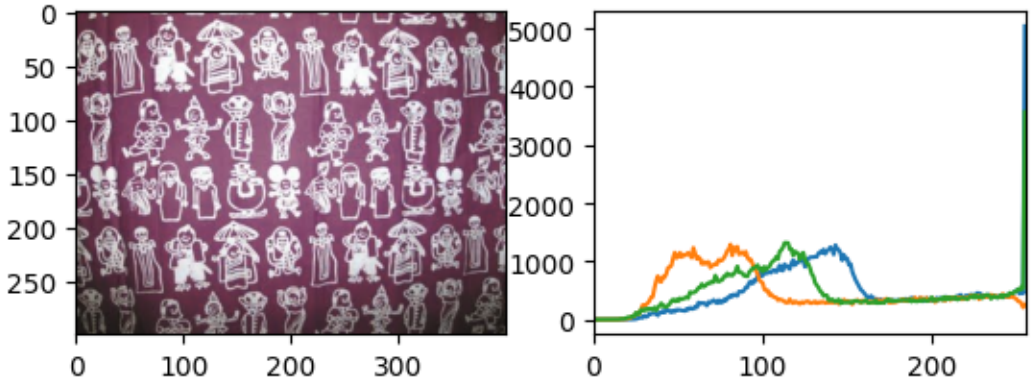

50000
40000
30000
20000
10000
0
0
100
200

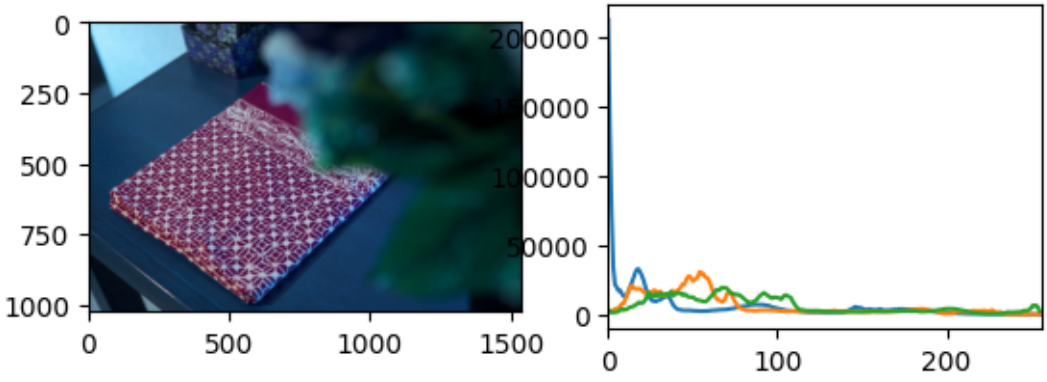

0
100
200
300
400
0
200
400
600
5000
4000
3000
2000
1000
0
0
100
200

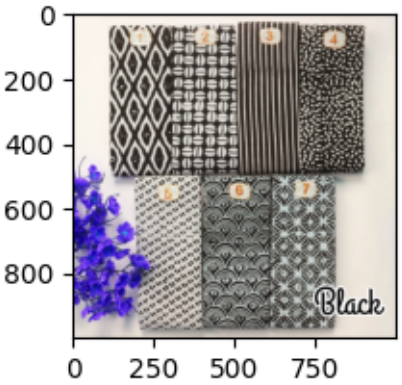

0
500
1000
1500
0
1000
2000
80000
60000
40000
20000
0
0
100
200

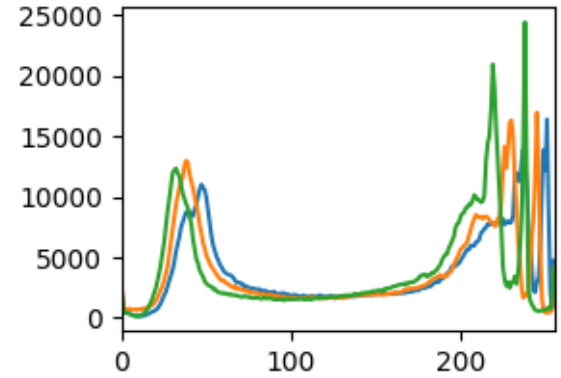

0
200
400
600
0
200
400
600
5000
4000
3000
2000
1000
0
0
100
200

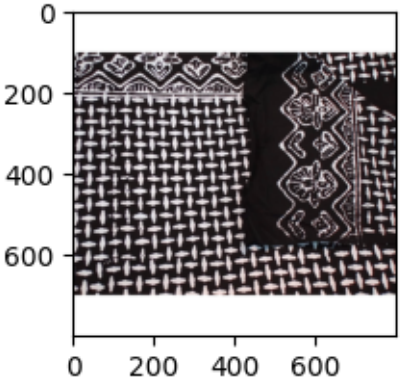

0
50
100
150
0
100
200
4000
2000
0
0
100
200

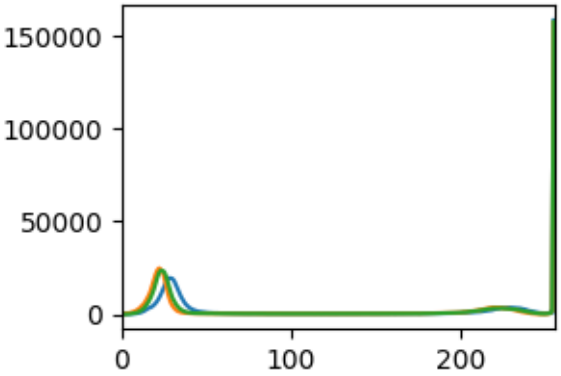

0
50
100
150
200
250
0
100
200
300
5000
4000
3000
2000
1000
0
0
100
200


0
250
500
750
1000
0
500
1000
1500
200000
150000
100000
50000
0
0
100
200

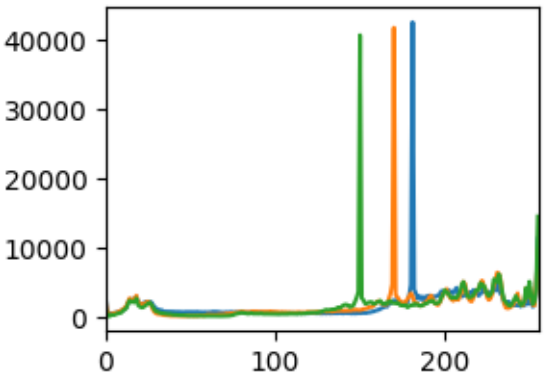

0
200
400
600
800
0
250
500
750
Black

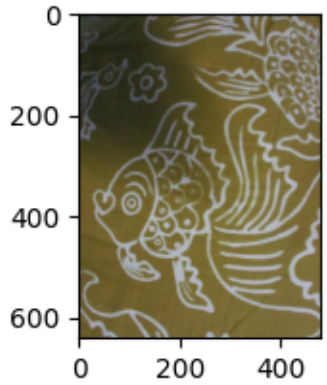

25000
20000
15000
10000
5000
0
0
100
200

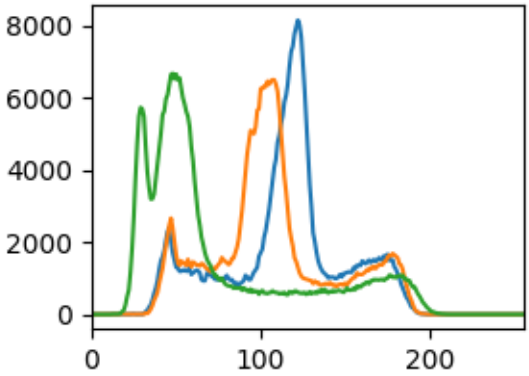

0
200
400
600
0
200
400
600

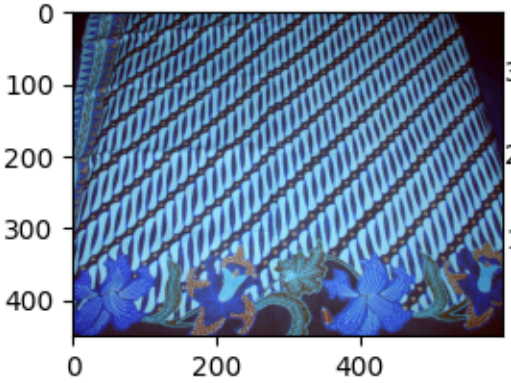

150000
100000
50000
0
0
100
200

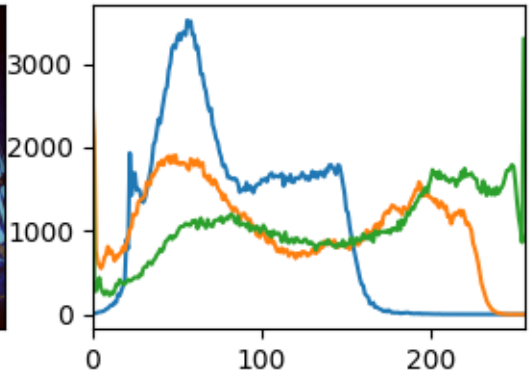

0
200
400
600
800
0
500

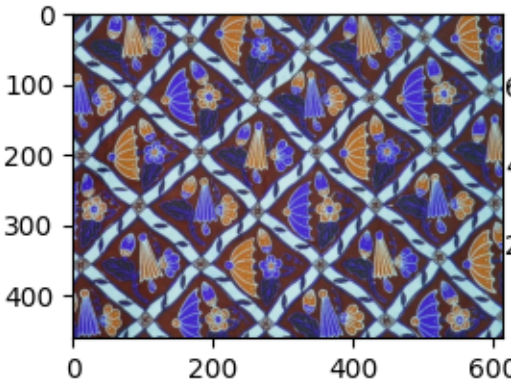

40000
30000
20000
10000
0
0
100
200

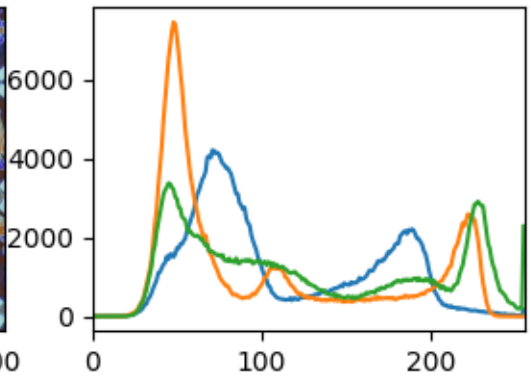

0
200
400
600
0
200
400

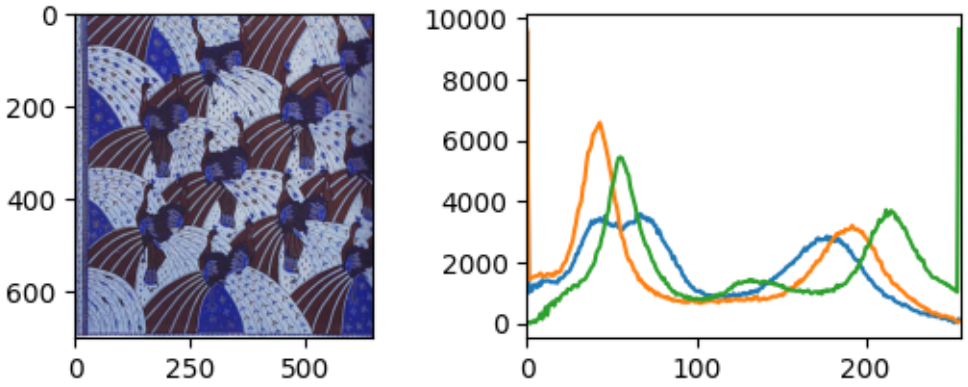

8000
6000
4000
2000
0
0
100
200

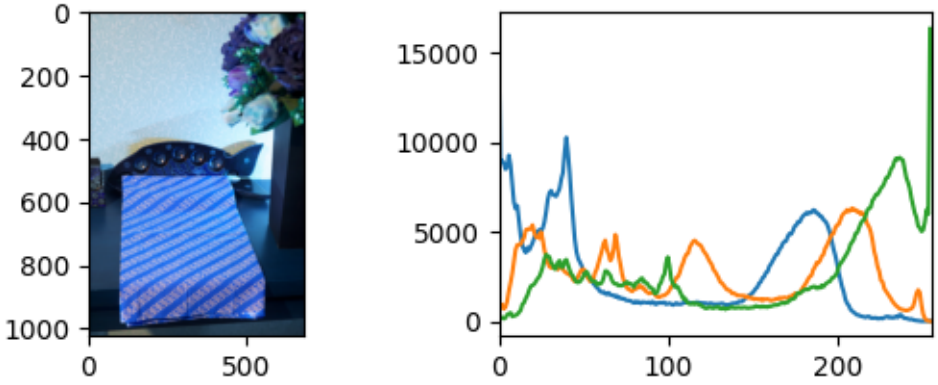

0
100
200
300
400
0
200
400

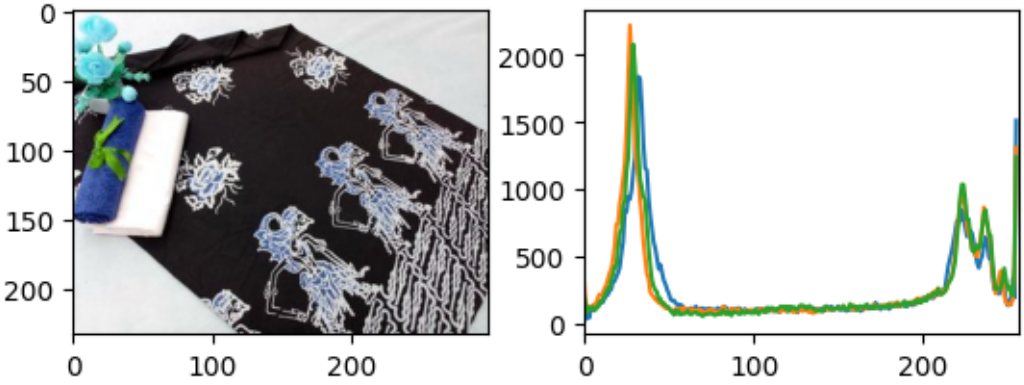

3000
2000
1000
0
0
100
200

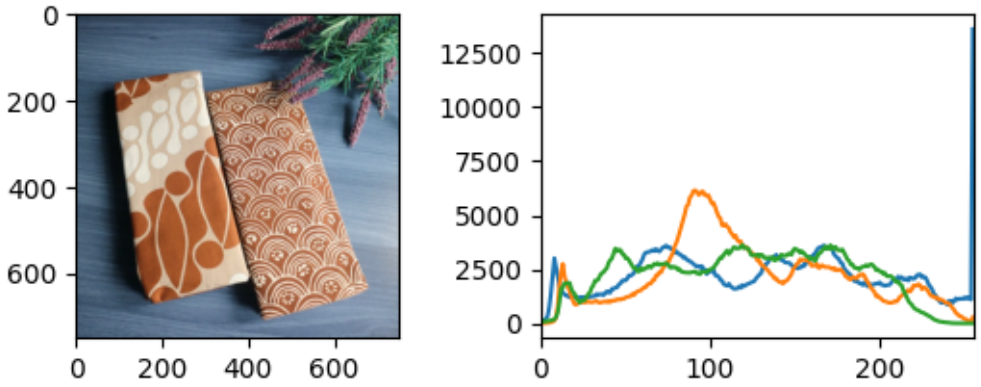

0
100
200
300
400
0
200
400
600

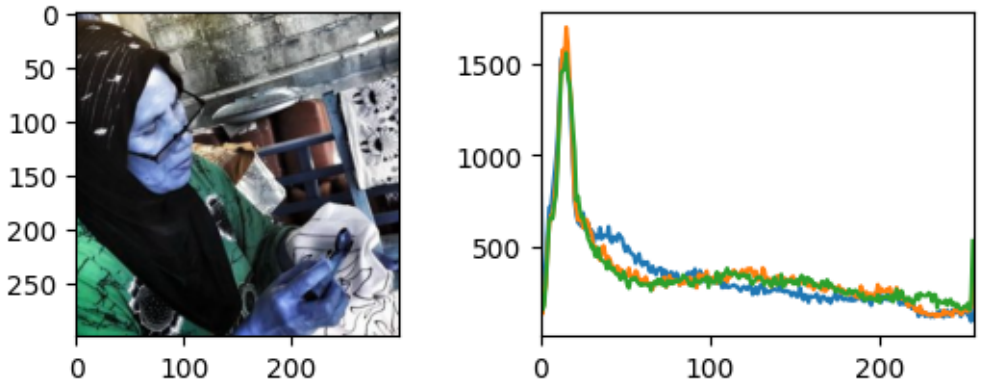

6000
4000
2000
0
0
100
200

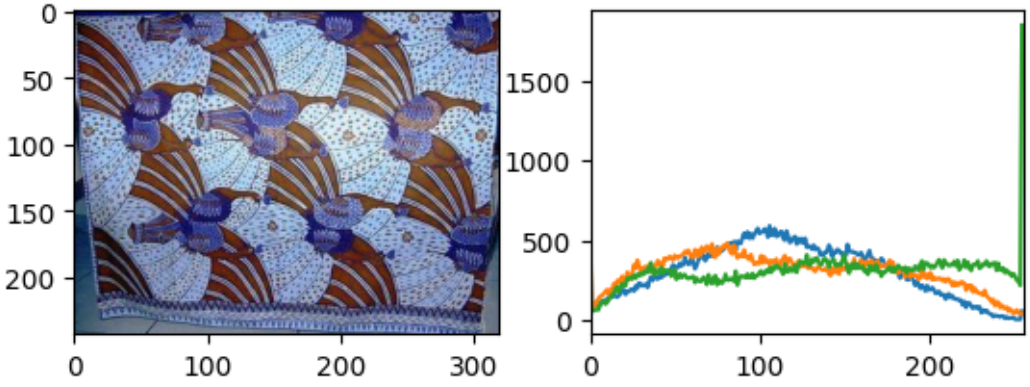

0
200
400
600
0
250
500
10000
8000
6000
4000
2000
0
0
100
200

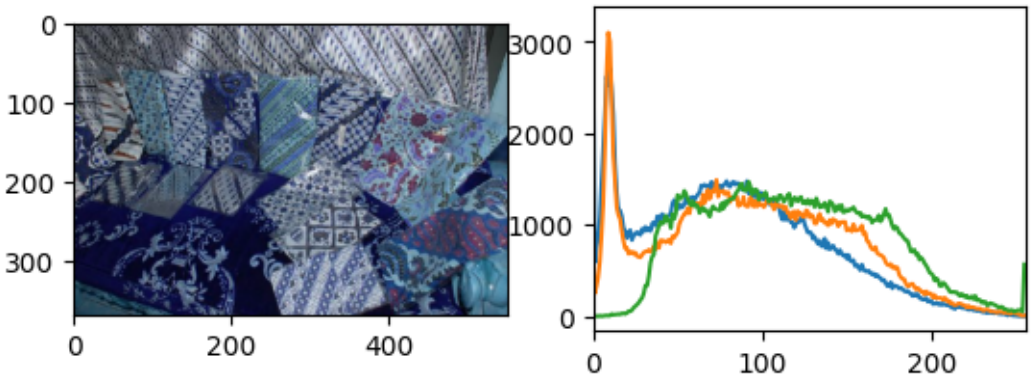

0
200
400
600
800
1000
0
500
15000
10000
5000
0
0
100
200

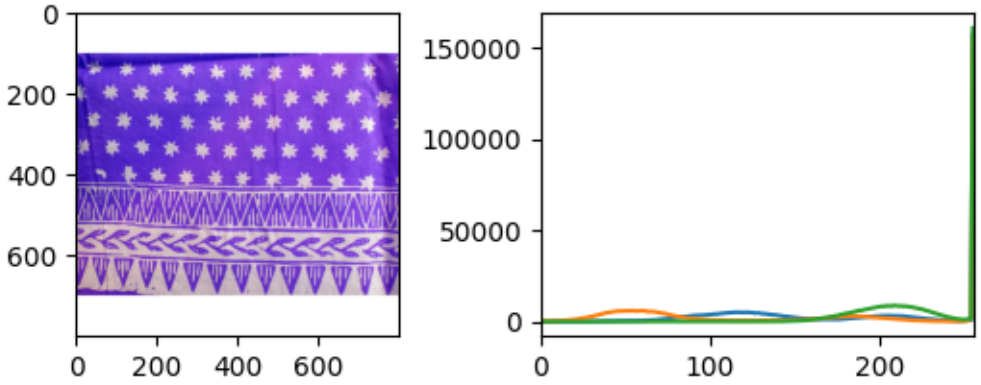

0
50
100
150
200
0
100
200
2000
1500
1000
500
0
0
100
200

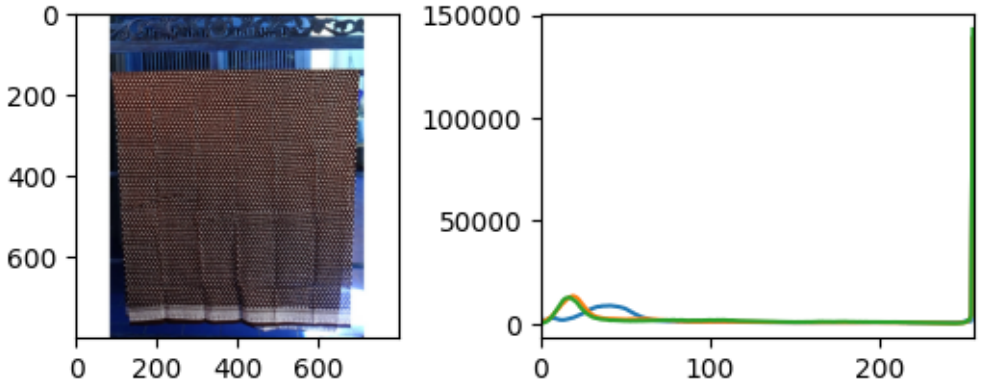

0
200
400
600
0
200
400
600
12500
10000
7500
5000
2500
0
0
100
200


0
50
100
150
200
250
0
100
200
1500
1000
500
0
100
200

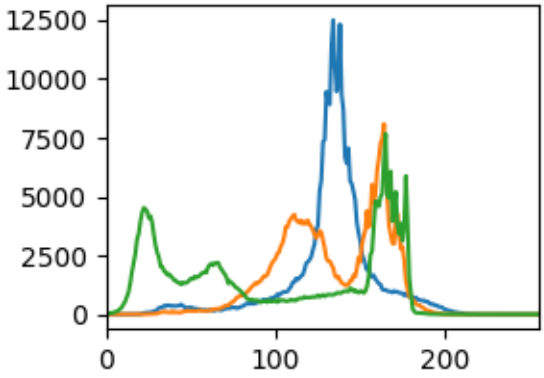

0
50
100
150
200
0
100
200
300
1500
1000
500
0
0
100
200

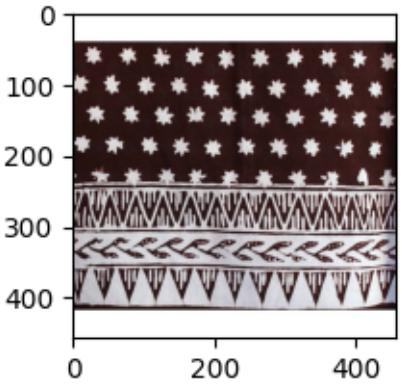

0
100
200
300
0
200
400
3000
2000
1000
0
0
100
200

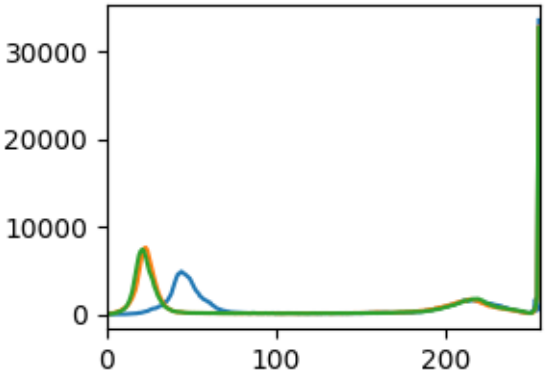

0
200
400
600
0
200
400
600
150000
100000
50000
0
0
100
200


0
200
400
600
0
200
400
600
150000
100000
50000
0
0
100
200

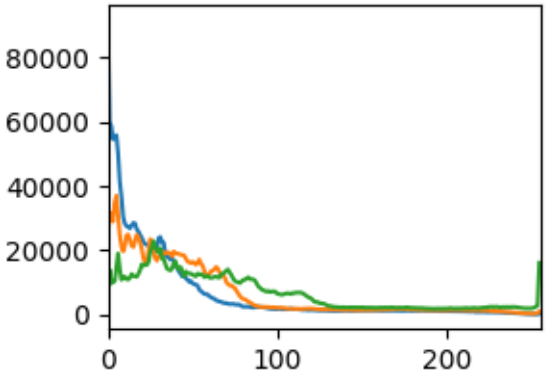

0
200
400
600
0
200
400

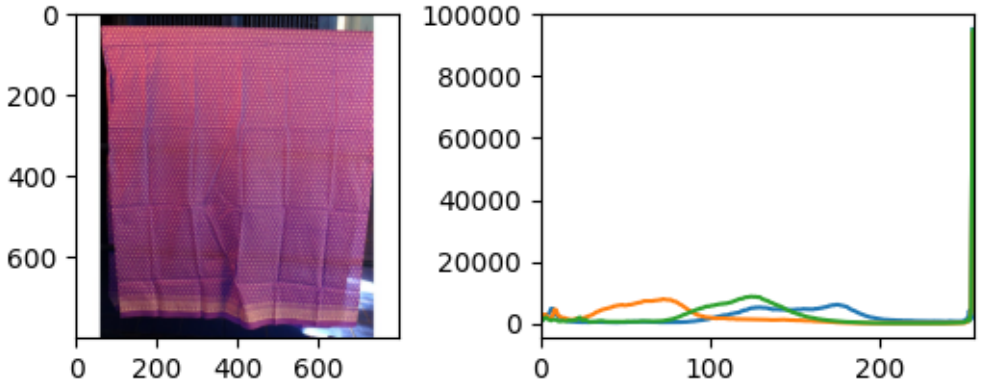

12500
10000
7500
5000
2500
0
0
100
200

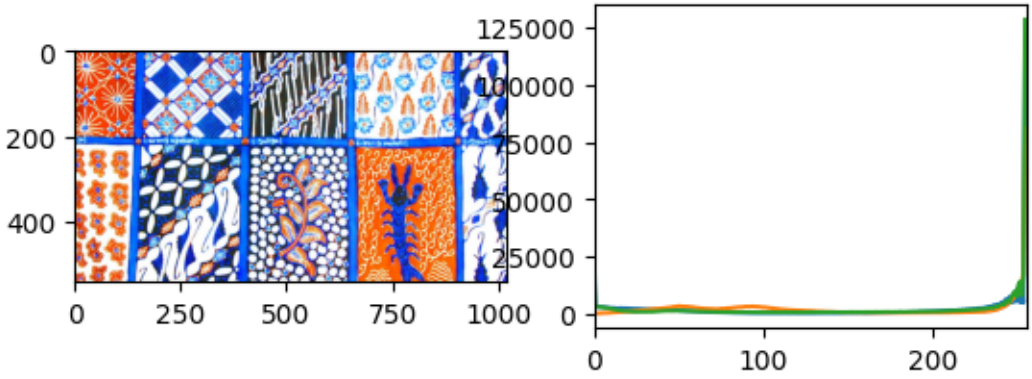

0
100
200
300
400
0
200
400

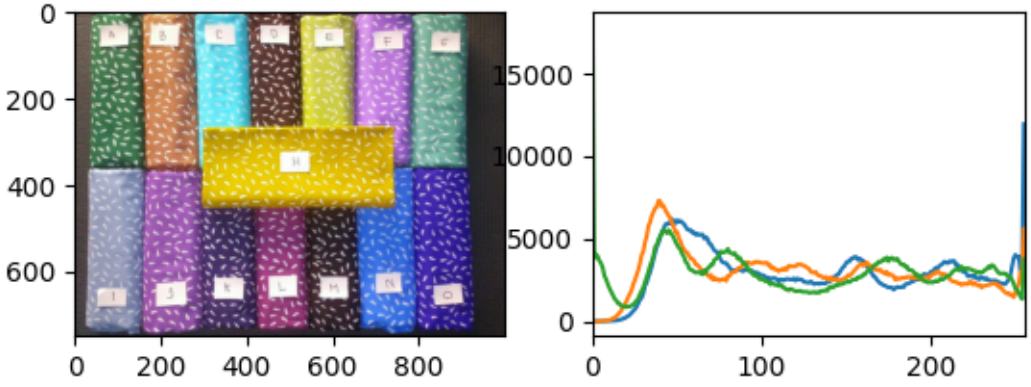

30000
20000
10000
0
0
100
200

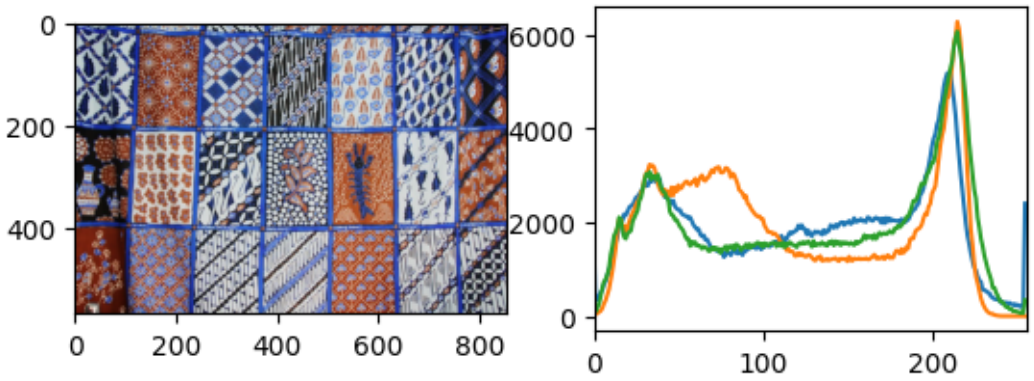

0
500
1000
1500
0
1000

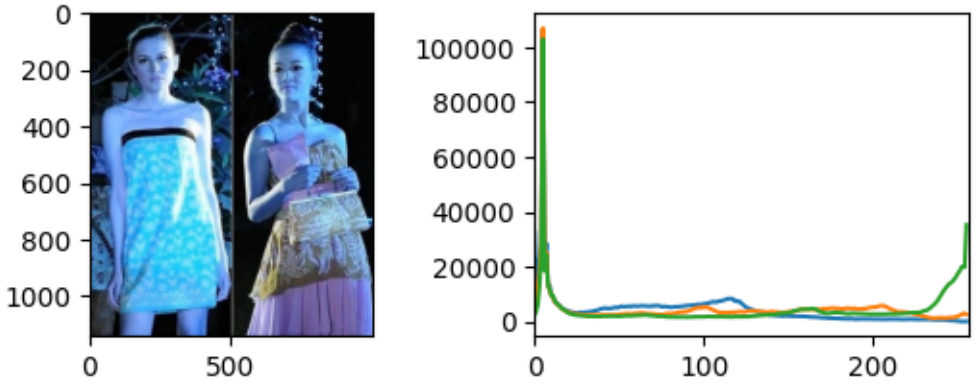

80000
60000
40000
20000
0
0
100
200

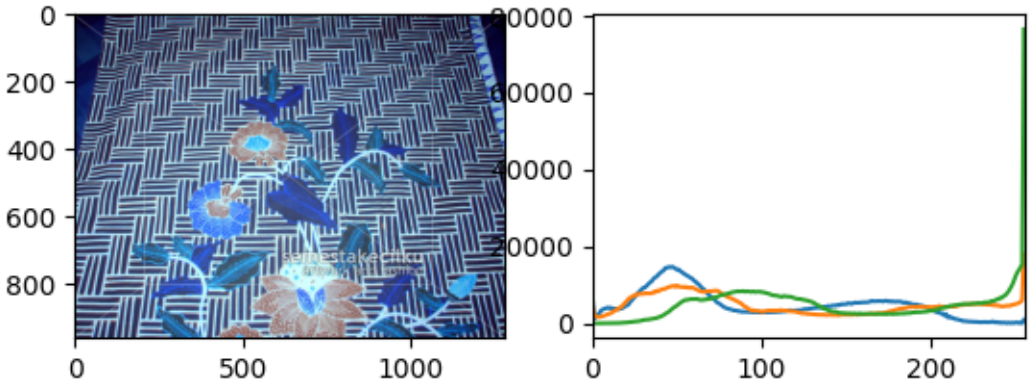

0
200
400
600
0
200
400
600
100000
80000
60000
40000
20000
0
0
100
200

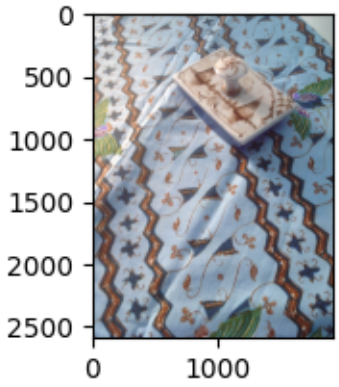

0
200
400
0
250
500
750
1000
125000
100000
75000
50000
25000
0
0
100
200

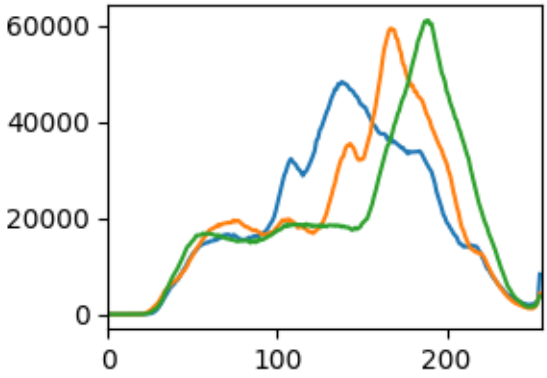

0
200
400
600
0
200
400
600
800
15000
10000
5000
0
0
100
200

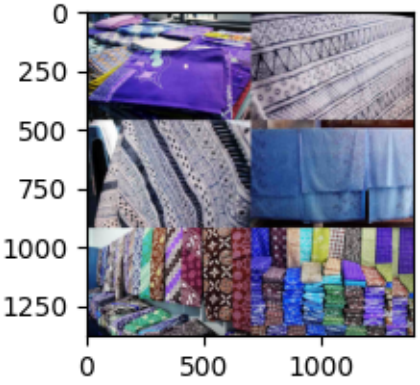

0
200
400
0
200
400
600
800
6000
4000
2000
0
0
100
200

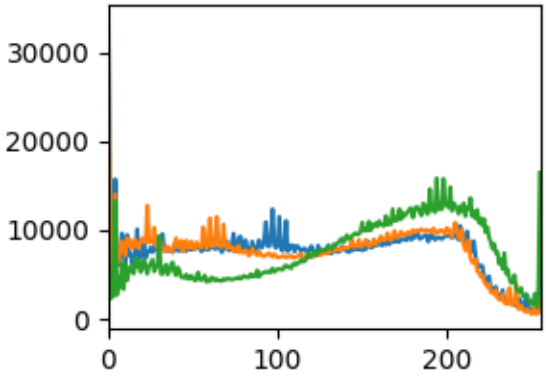

0
200
400
600
800
1000
0
500
100000
80000
60000
40000
20000
0
0
100
200

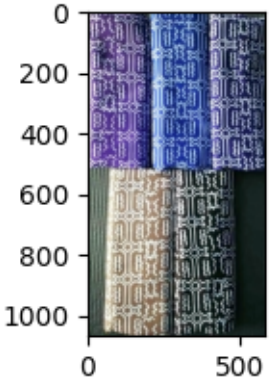

0
200
400
600
800
0
500
1000
0
100
200

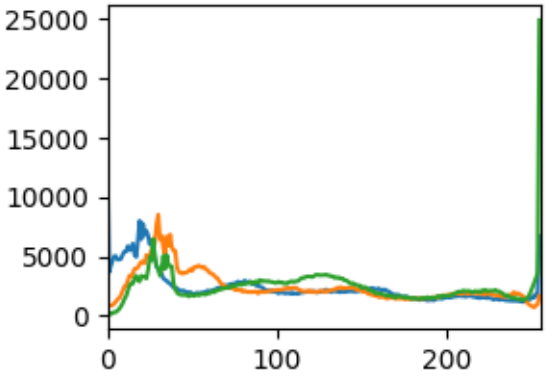

0
500
1000
1500
2000
2500
0
1000

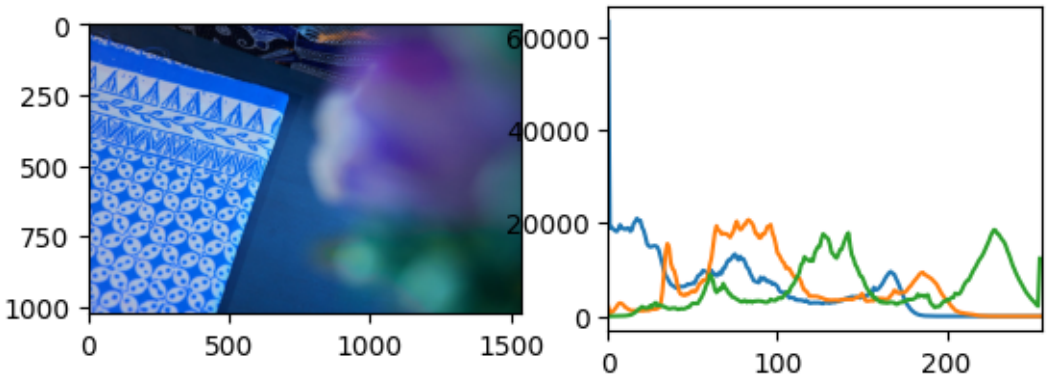

60000
40000
20000
0
0
100
200

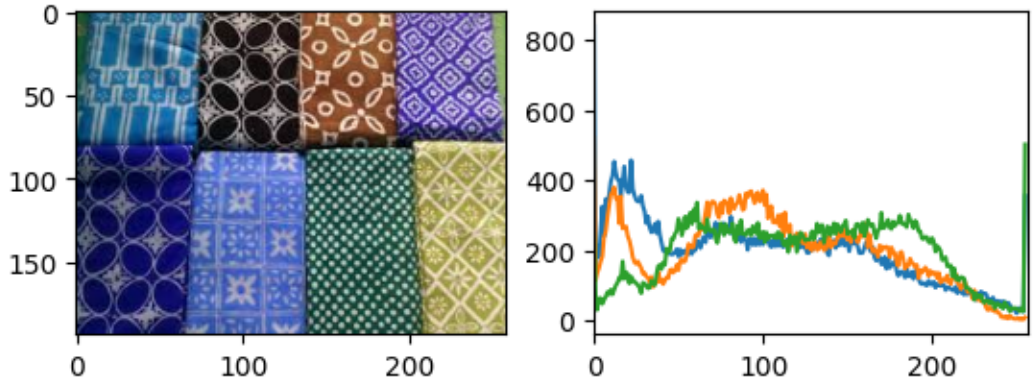

0
250
500
750
1000
1250
0
500
1000


30000
20000
10000
0
0
100
200


0
200
400
600
800
1000
0
500

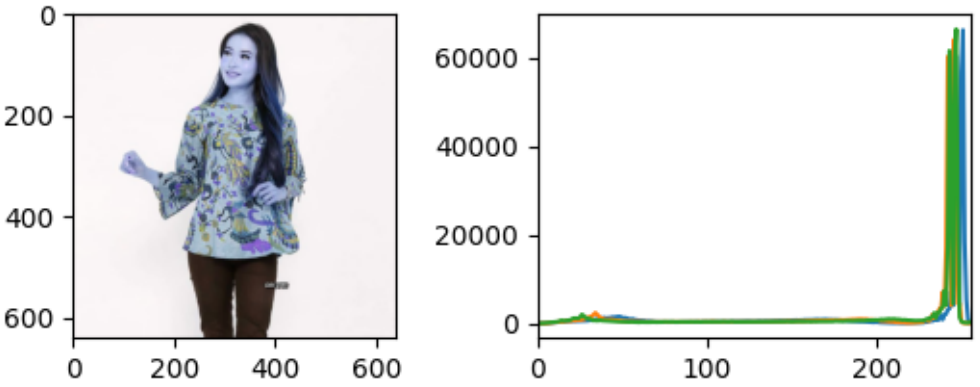

25000
20000
15000
10000
5000
0
0
100
200

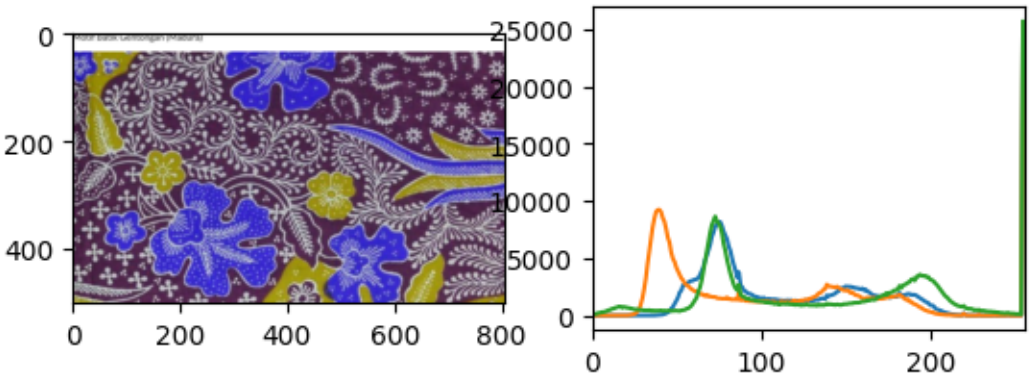


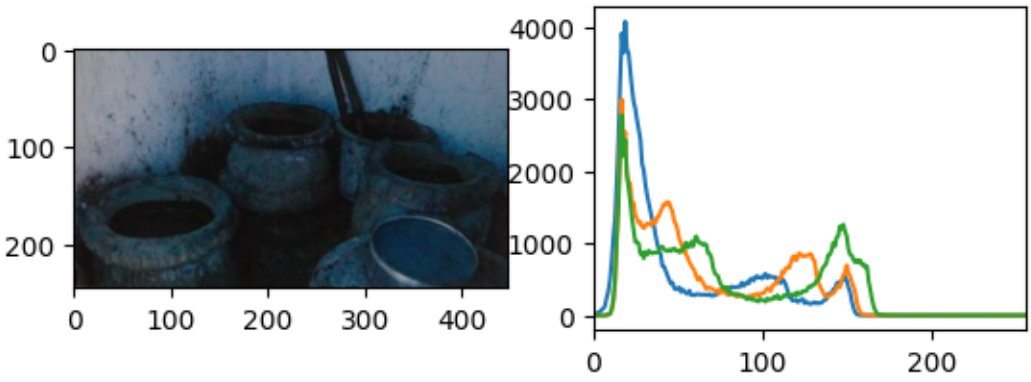


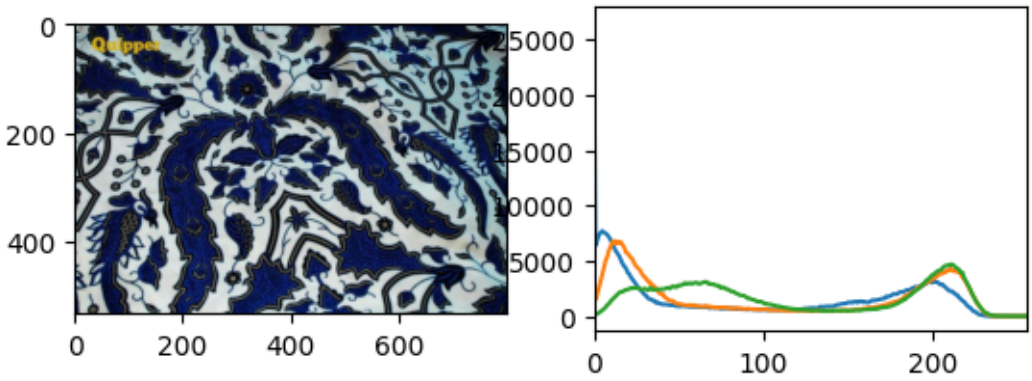
3.1.4 Color Histogram Batik Gentongan

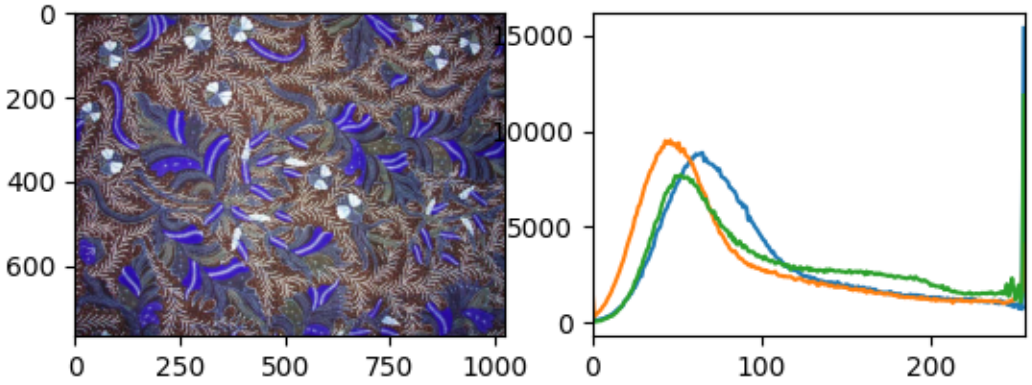
[ ]:
```
for i in os.listdir('/content/gdrive/MyDrive/Dataset2B/batik-gentongan/'):
  img = cv2.imread('/content/gdrive/MyDrive/Dataset2B/batik-gentongan/'+i)
  hist1 = cv2.calcHist([img],[0],None,[256],[0,256])
  hist2 = cv2.calcHist([img],[1],None,[256],[0,256])
  hist3 = cv2.calcHist([img],[2],None,[256],[0,256])

  plt.subplot(221), plt.imshow(img)
  plt.subplot(222), plt.plot(hist1), plt.plot(hist2),plt.plot(hist3)
  plt.xlim([0,256])

  plt.show()
```

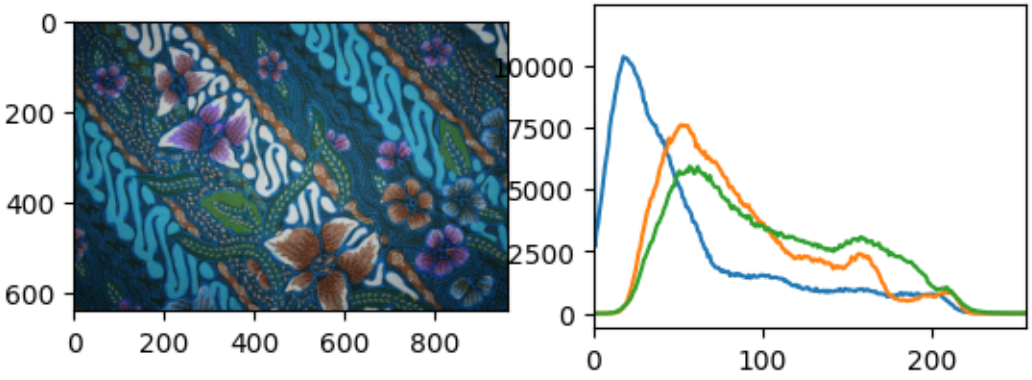

0
200
400
600
0
200
400
600
60000
40000
20000
0
0
100
200

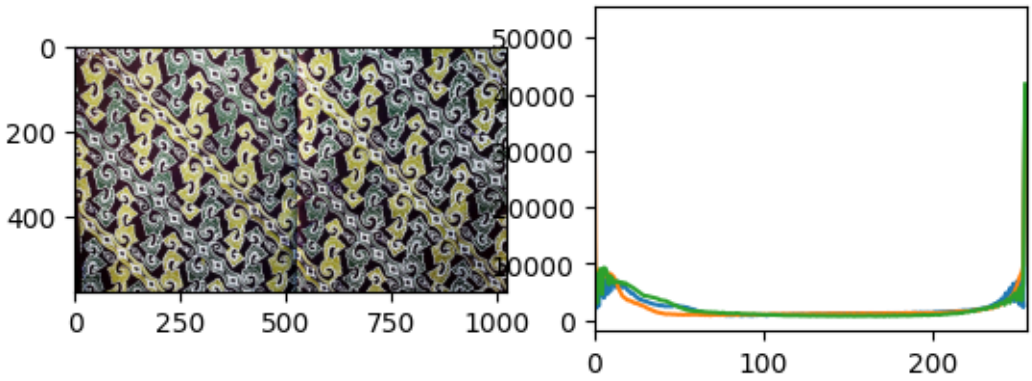

0
200
400
0
200
400
600
800
25000
20000
15000
10000
5000
0
0
100
200

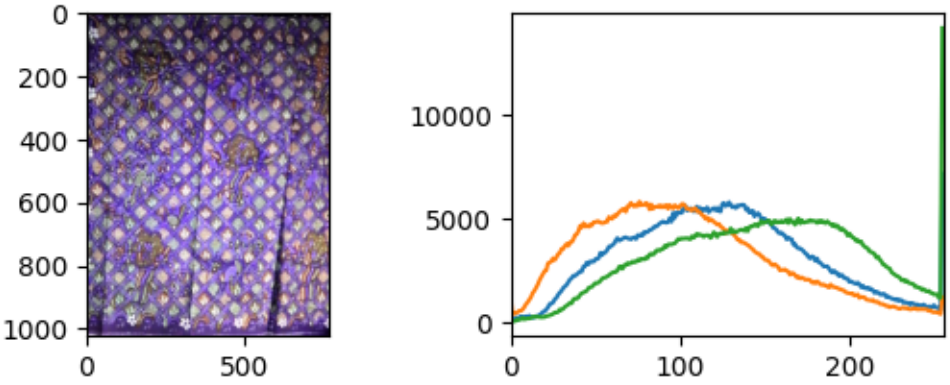

0
100
200
0
100
200
300
400
4000
3000
2000
1000
0
0
100
200

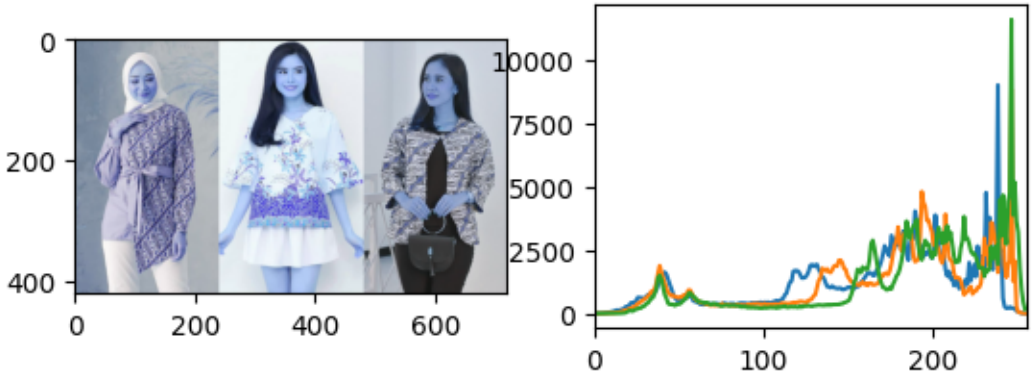

0
200
400
0
200
400
600
25000
20000
15000
10000
5000
0
0
100
200

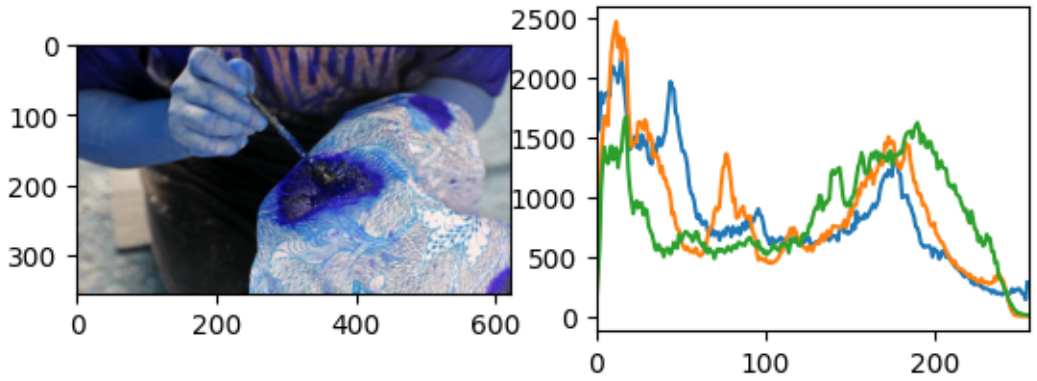

0
200
400
600
0
250
500
750
1000
15000
10000
5000
0
0
100
200

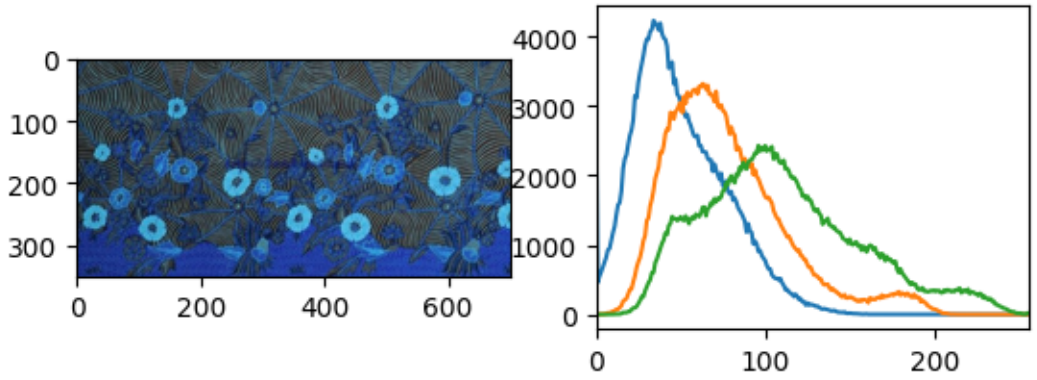

0
200
400
600
0
200
400
600
800
10000
7500
5000
2500
0
0
100
200

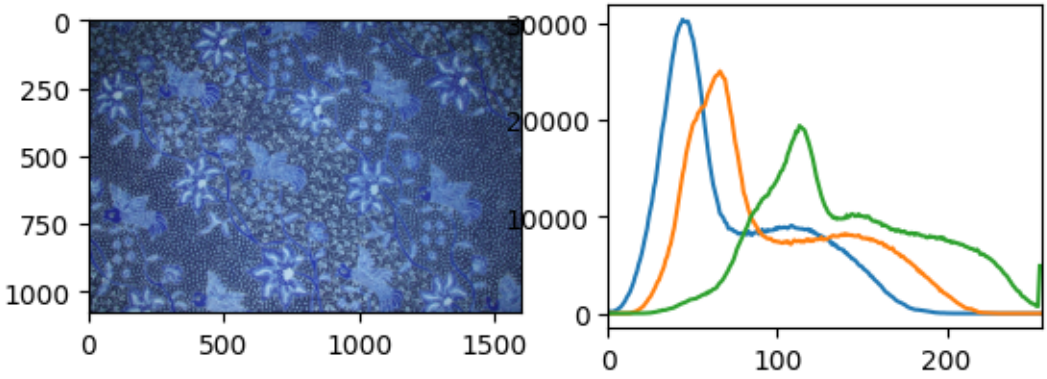

0
200
400
0
250
500
750
1000
50000
0
100
200

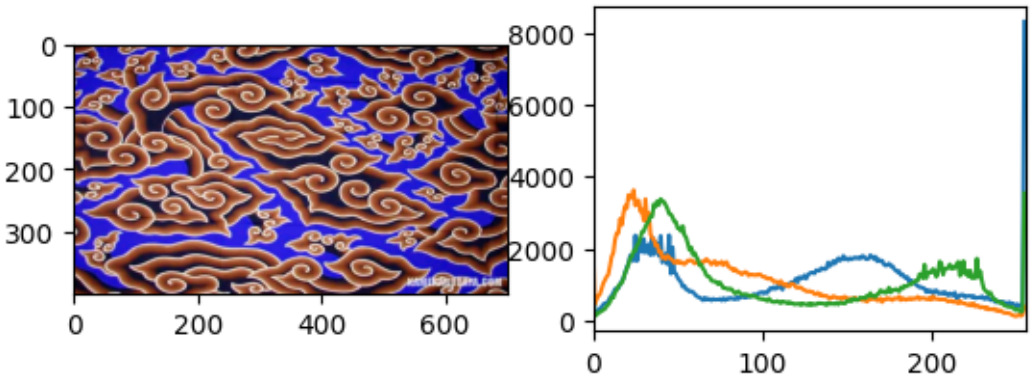

0
200
400
600
800
1000
0
500
10000
5000
0
100
200

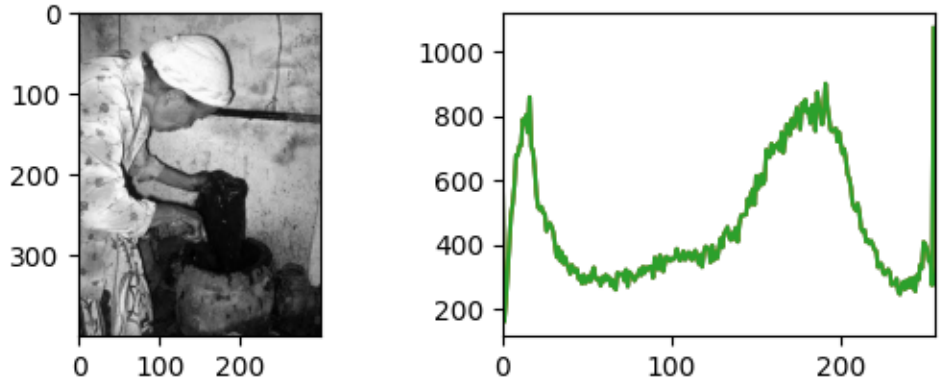

0
200
400
0
200
400
600
10000
7500
5000
2500
0
100
200

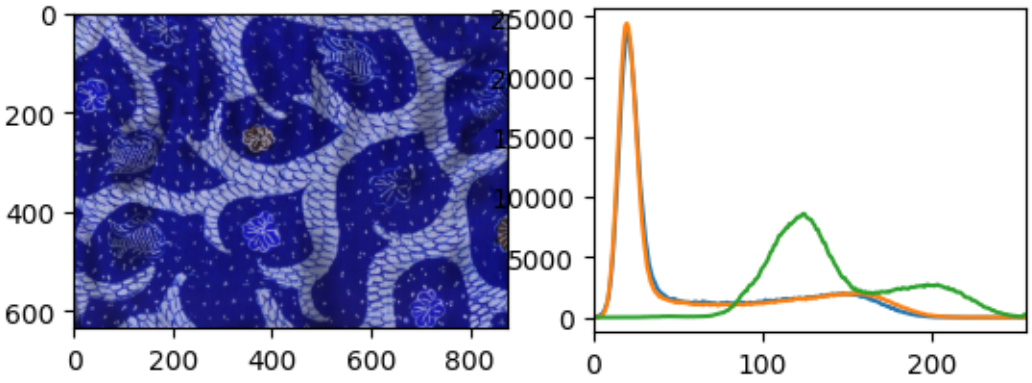

0
100
200
300
0
200
400
600
2500
2000
1500
1000
500
0
0
100
200

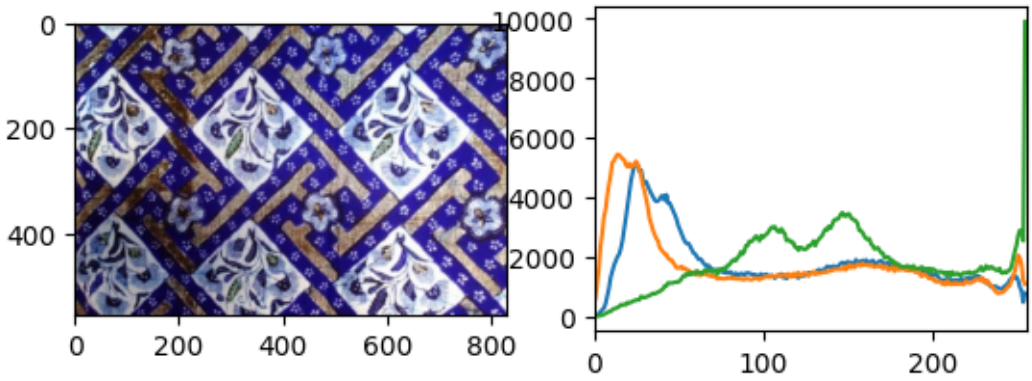

0
100
200
300
0
200
400
600
4000
3000
2000
1000
0
0
100
200

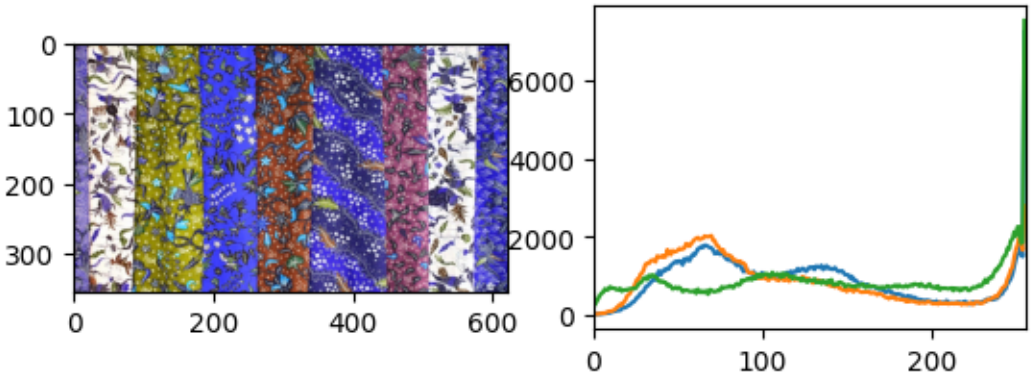

0
250
500
750
1000
0
500
1000
1500
30000
20000
10000
0
0
100
200

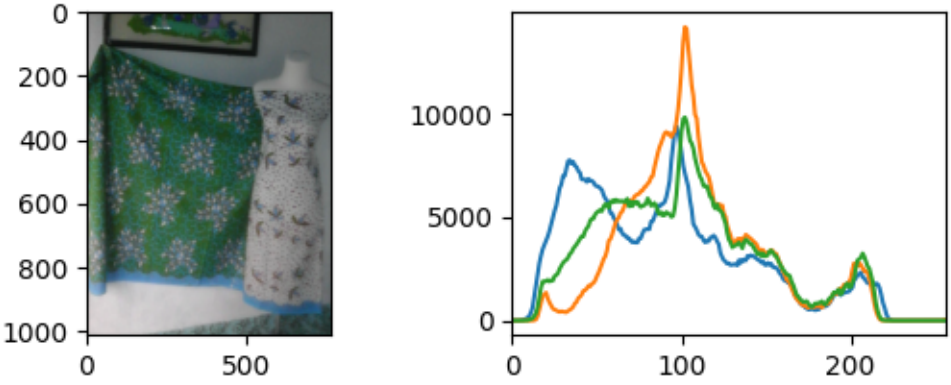

0
100
200
300
0
200
400
600
8000
6000
4000
2000
0
0
100
200

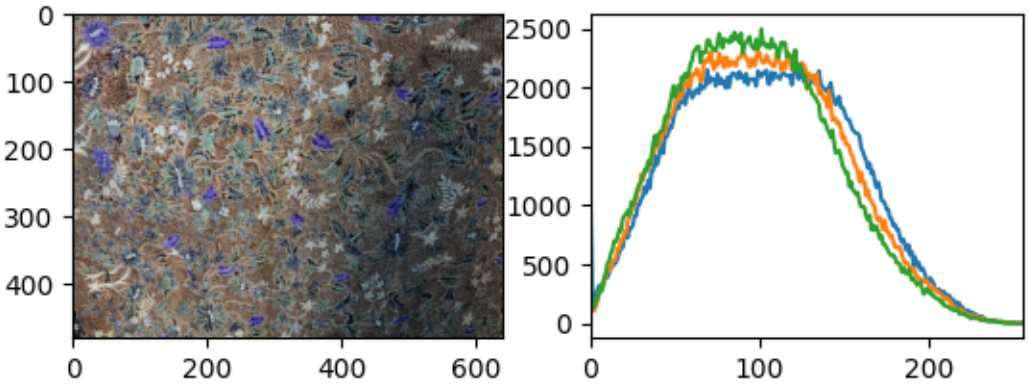

0
100
200
300
0
100
200
1000
800
600
400
200
0
100
200

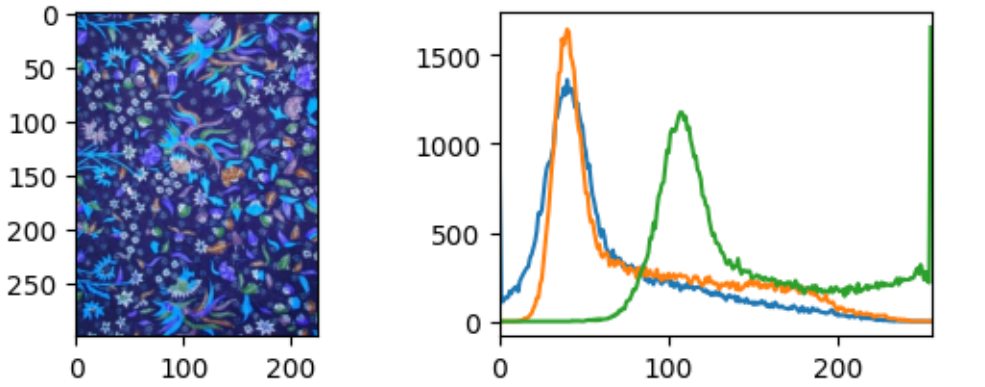

0
200
400
600
0
200
400
600
800
25000
20000
15000
10000
5000
0
0
100
200

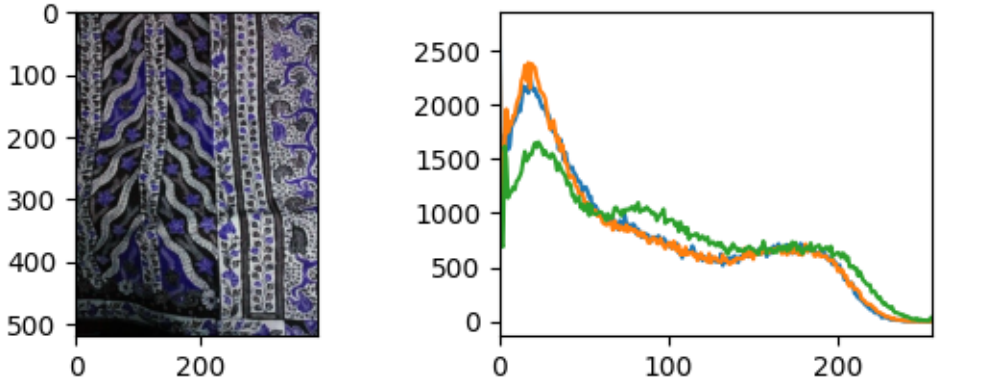

0
200
400
0
200
400
600
800
10000
8000
6000
4000
2000
0
0
100
200

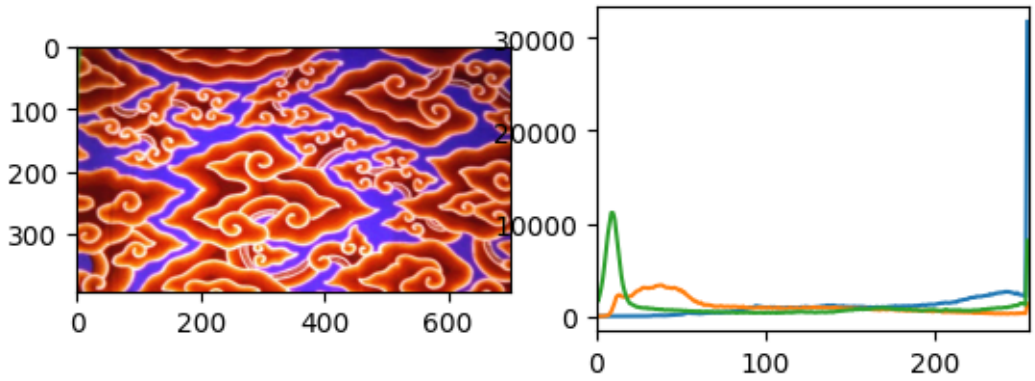

0
100
200
300
0
200
400
600
6000
4000
2000
0
0
100
200

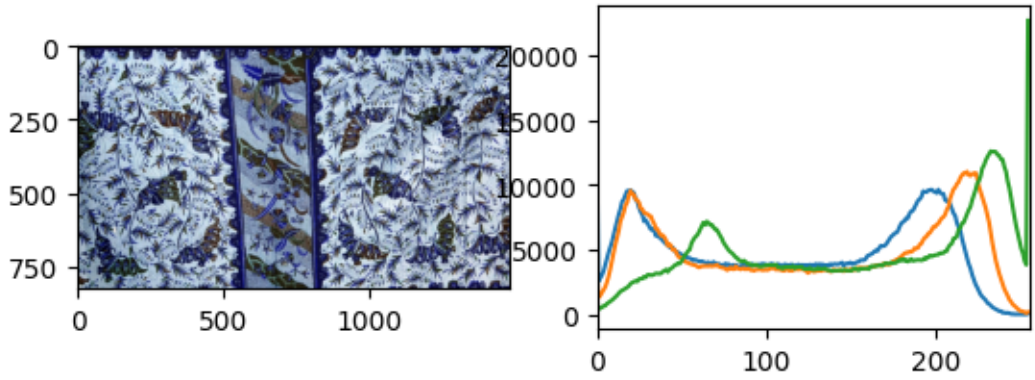

0
200
400
600
800
1000
0
500
10000
5000
0
0
100
200

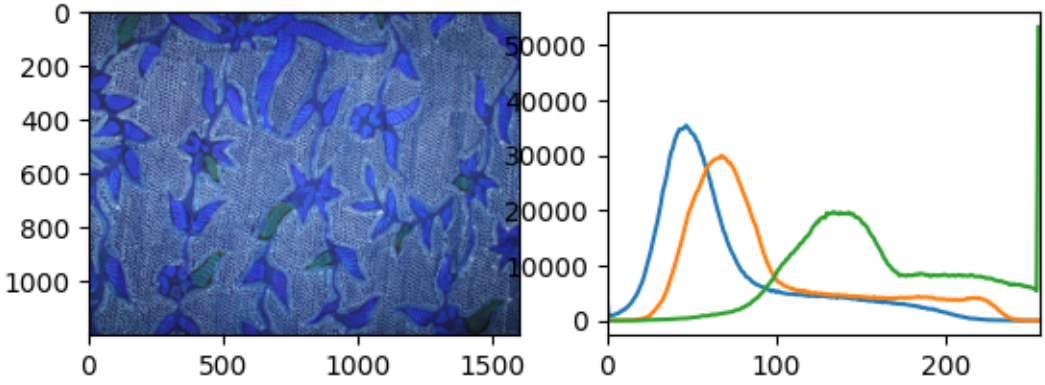

0
100
200
300
400
0
200
400
600
2500
2000
1500
1000
500
0
0
100
200

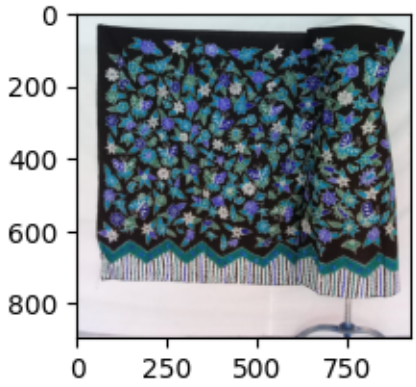

0
50
100
150
200
250
0
100
200
1500
1000
500
0
0
100
200

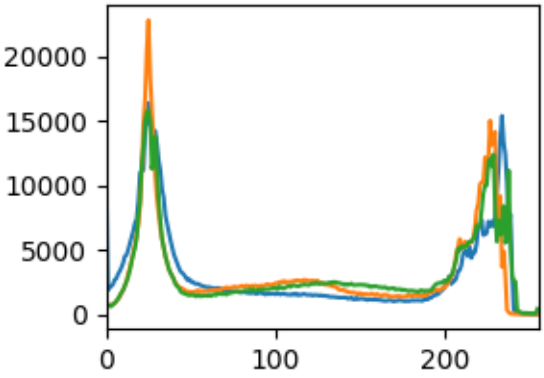

0
100
200
300
400
500
0
200
2500
2000
1500
1000
500
0
0
100
200

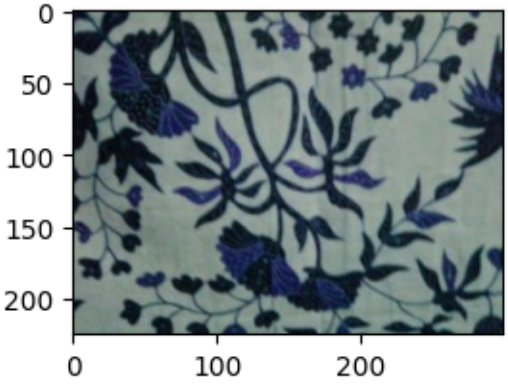

0
100
200
300
0
200
400
600
30000
20000
10000
0
0
100
200

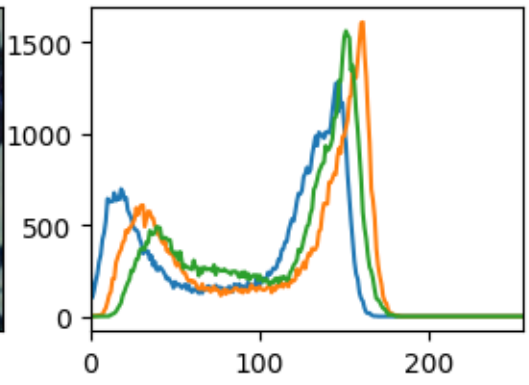

0
250
500
750
0
500
1000
20000
15000
10000
5000
0
0
100
200

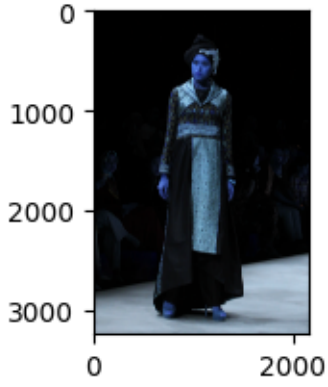

0
200
400
600
800
1000
0
500
1000
1500
50000
40000
30000
20000
10000
0
0
100
200

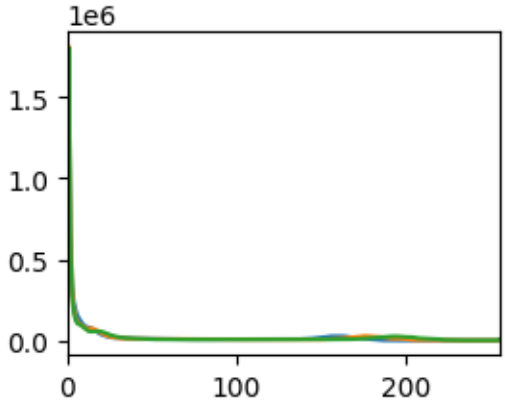

0
200
400
600
800
0
250
500
750

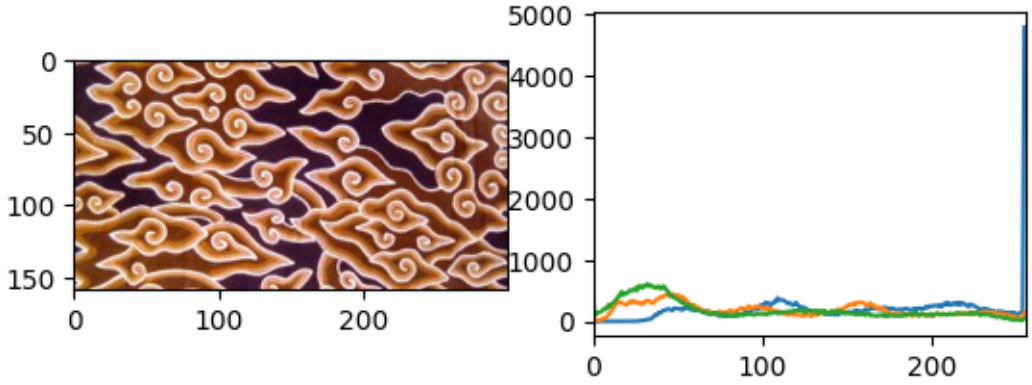

20000
15000
10000
5000
0
0
100
200

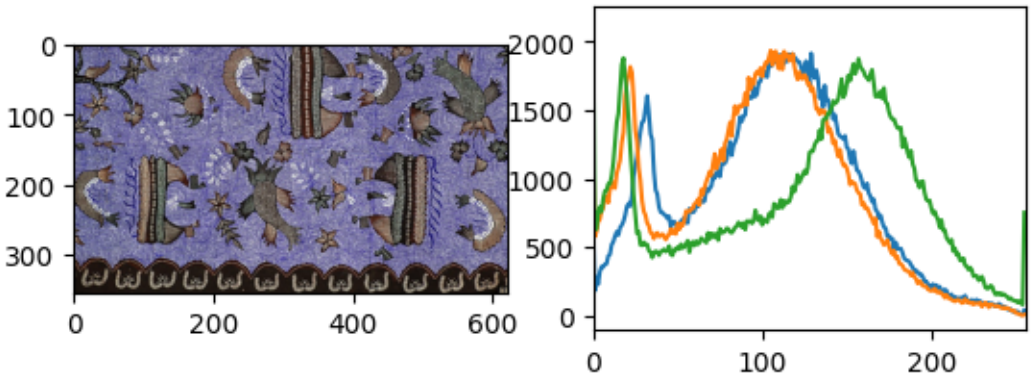

0
50
100
150
200
0
100
200

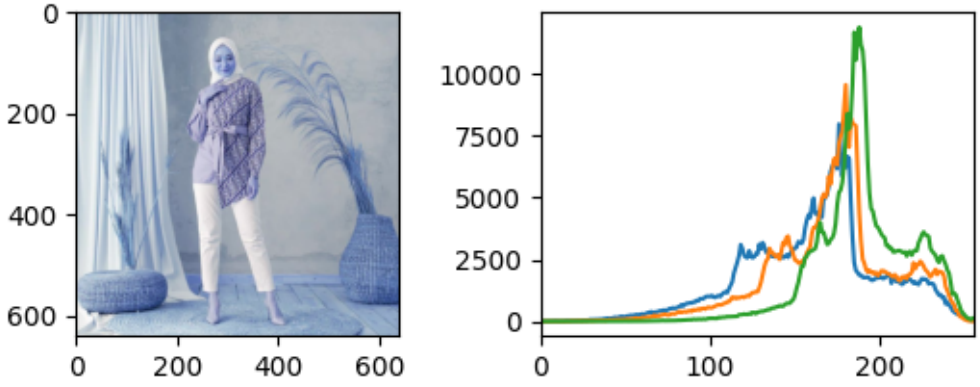

1500
1000
500
0
0
100
200

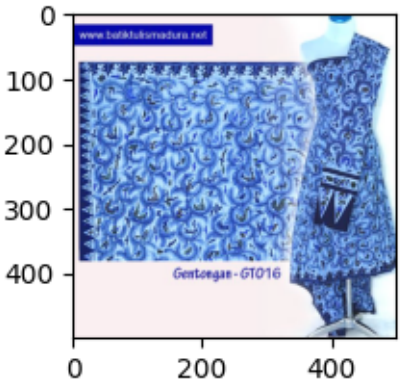

0
1000
2000
3000
0
2000

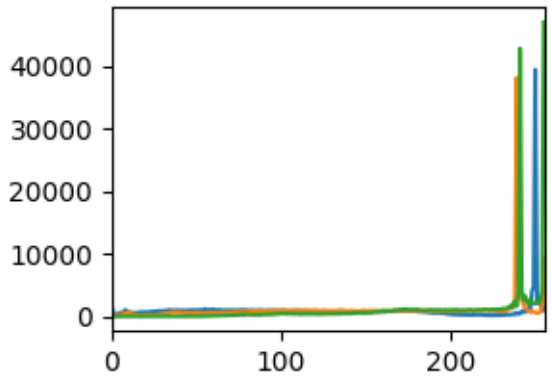

1e6
1.5
1.0
0.5
0.0
0
100
200

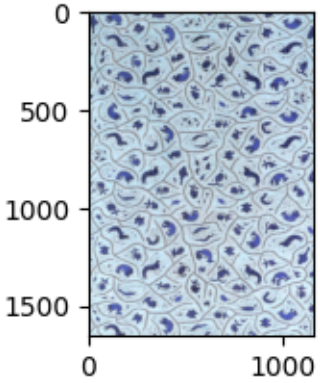

0
50
100
150
0
100
200
5000
4000
3000
2000
1000
0
0
100
200

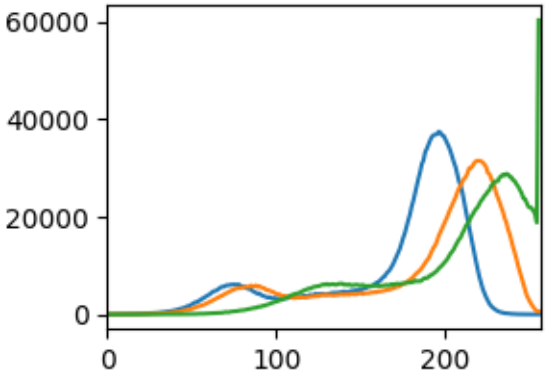

0
100
200
300
0
200
400
600
2000
1500
1000
500
0
0
100
200

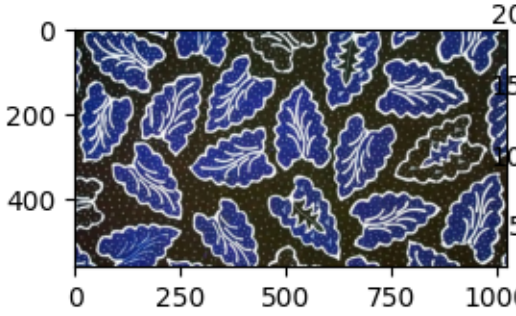

0
200
400
600
0
200
400
600
10000
7500
5000
2500
0
0
100
200

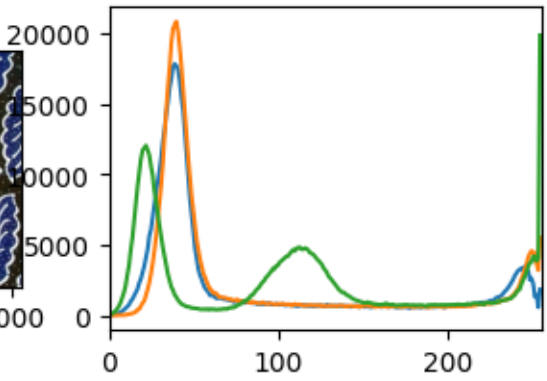

0
100
200
300
400
0
200
400
Gentorgan - GT016

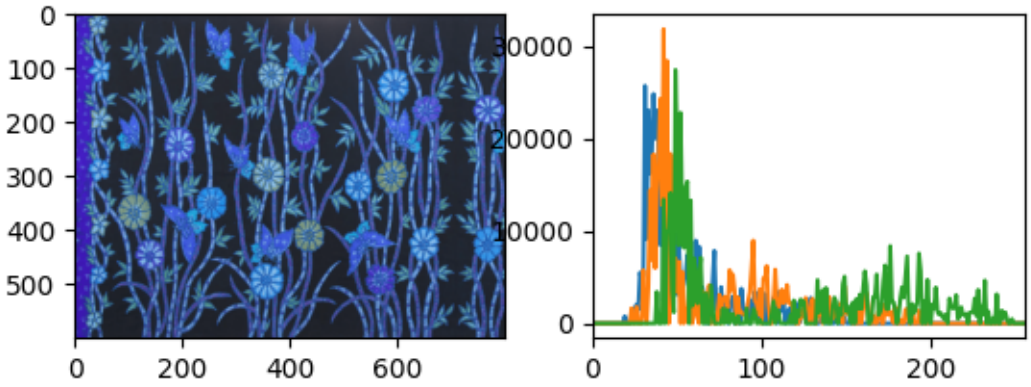

40000
30000
20000
10000
0
0
100
200

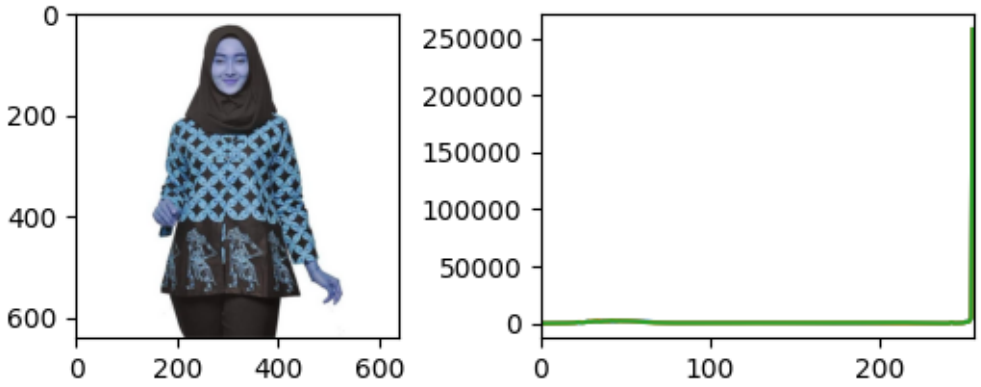

0
500
1000
1500
0
1000

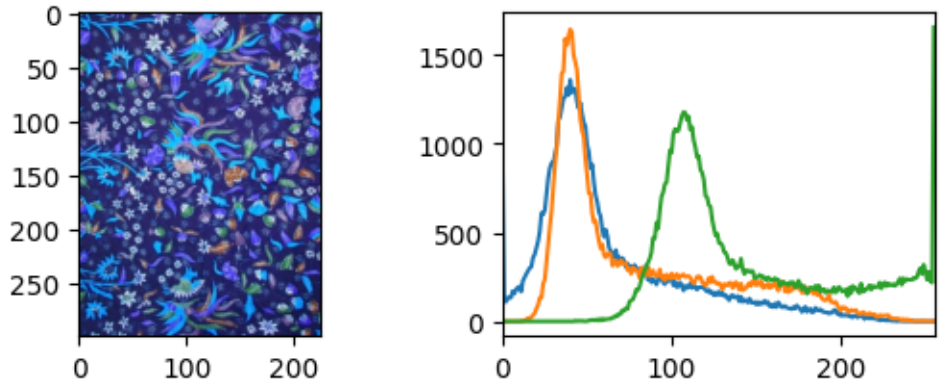

60000
40000
20000
0
0
100
200

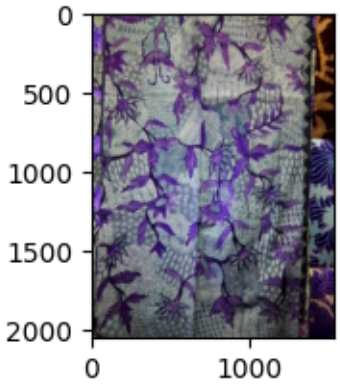

0
200
400
0
250
500
750
1000

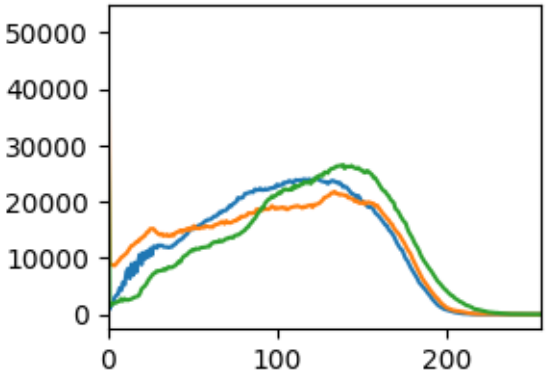

20000
15000
10000
5000
0
0
100
200

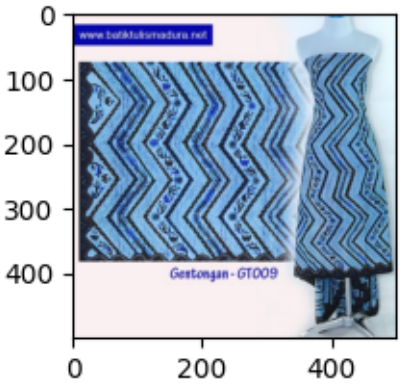

0
100
200
300
400
500
0
200
400
600
30000
20000
10000
0
0
100
200

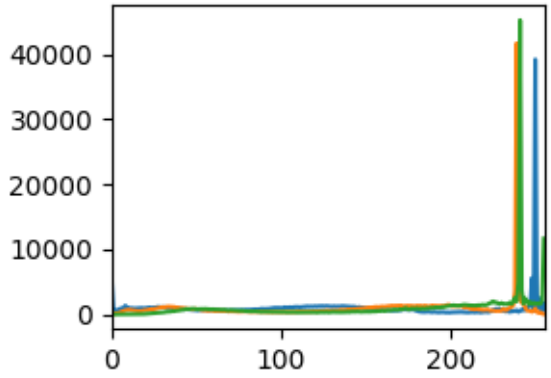

0
200
400
600
0
200
400
600
250000
200000
150000
100000
50000
0
0
100
200

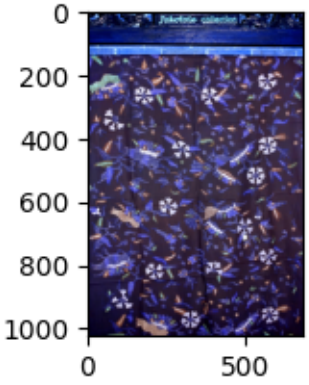

0
50
100
150
200
250
0
100
200
1500
1000
500
0
0
100
200

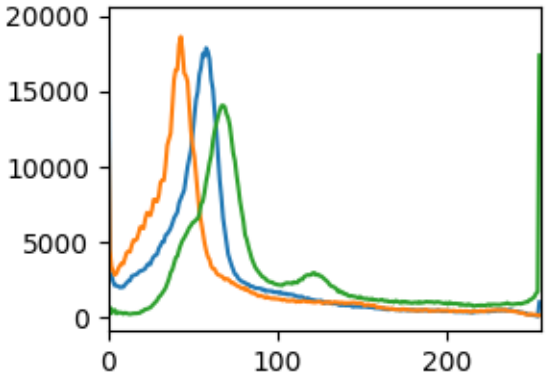

0
500
1000
1500
2000
0
1000

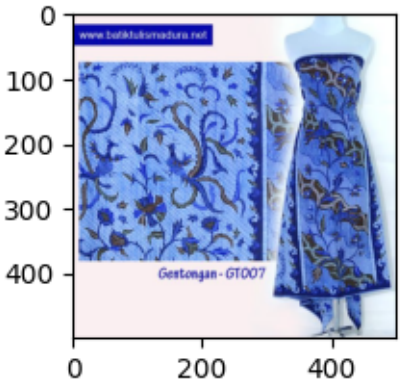

50000
40000
30000
20000
10000
0
0
100
200

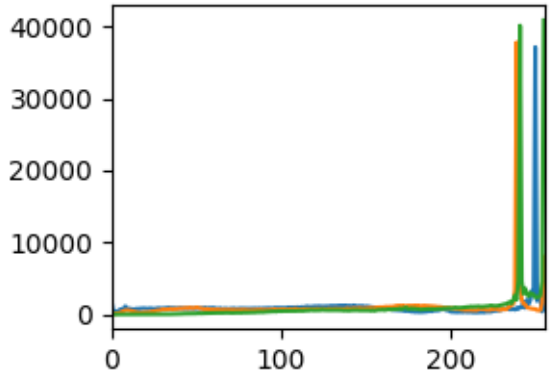

0
100
200
300
400
Gestongan - GT009
0
200
400

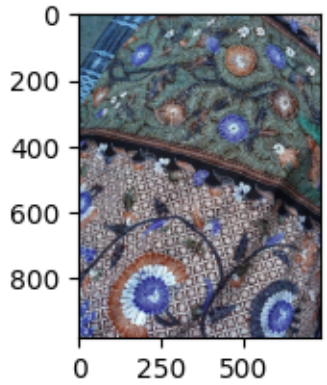

40000
30000
20000
10000
0
0
100
200

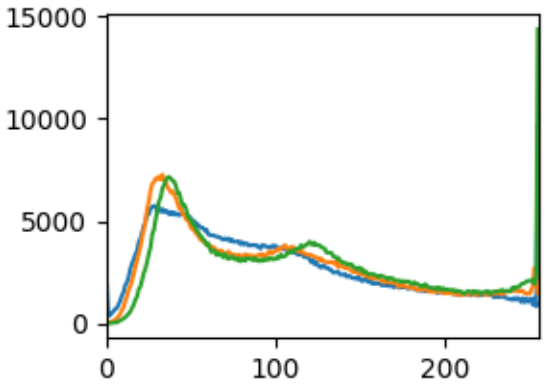

0
200
400
600
800
1000
0
500

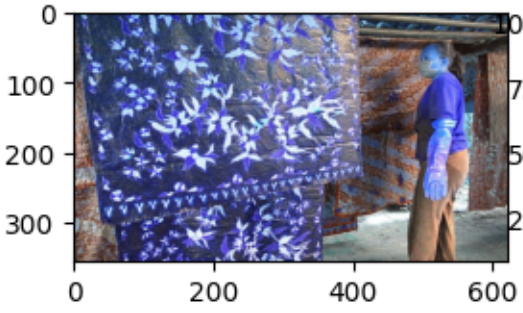

20000
15000
10000
5000
0
0
100
200

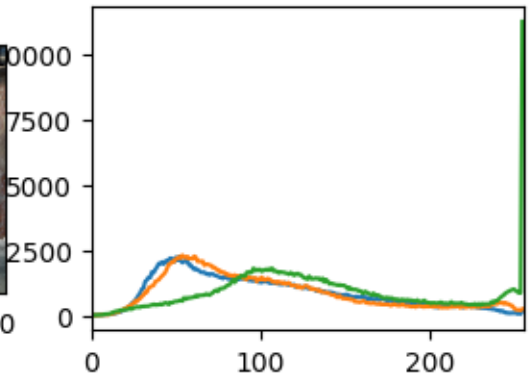

0
100
200
300
400
0
200
400
Gentongan - GT007

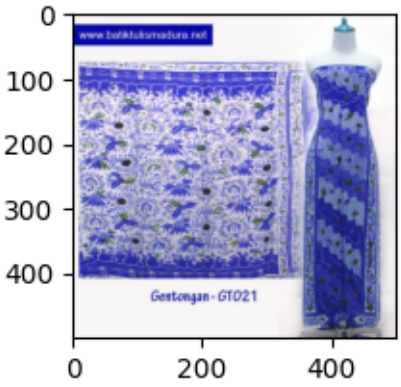

40000
30000
20000
10000
0
0
100
200

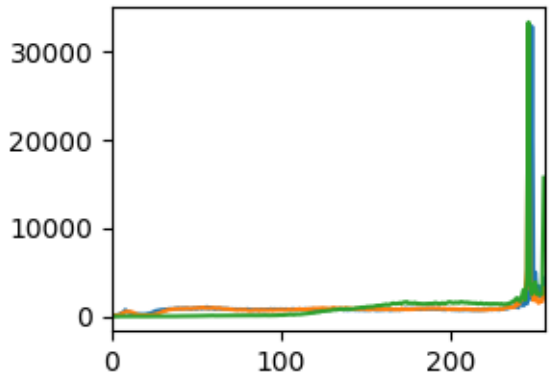

0
200
400
600
800
0
250
500

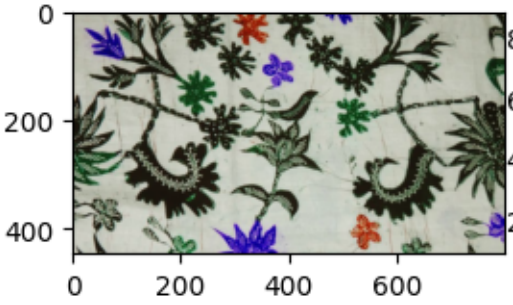

15000
10000
5000
0
0
100
200

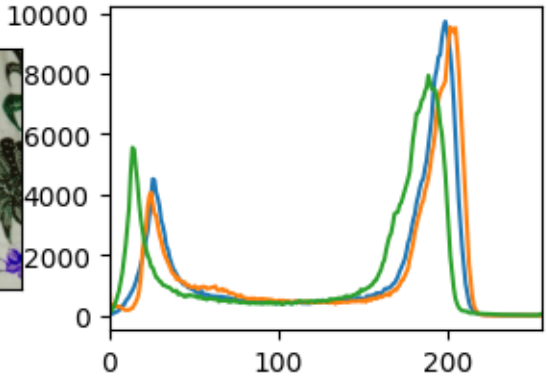

0
100
200
300
0
200
400
600

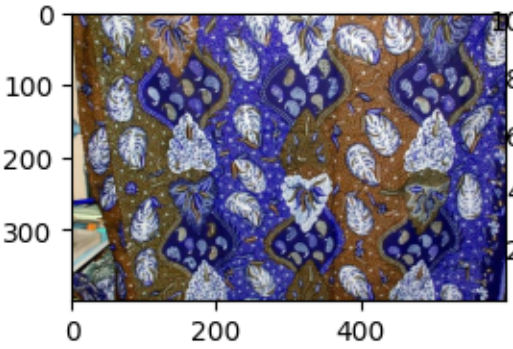

0000
7500
5000
2500
0
0
100
200

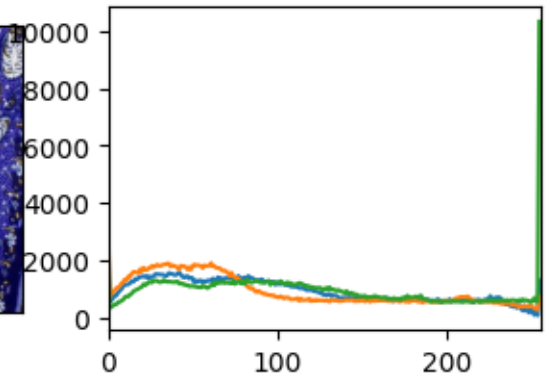

0
100
200
300
400
0
200
400
Gentongan - GTD21

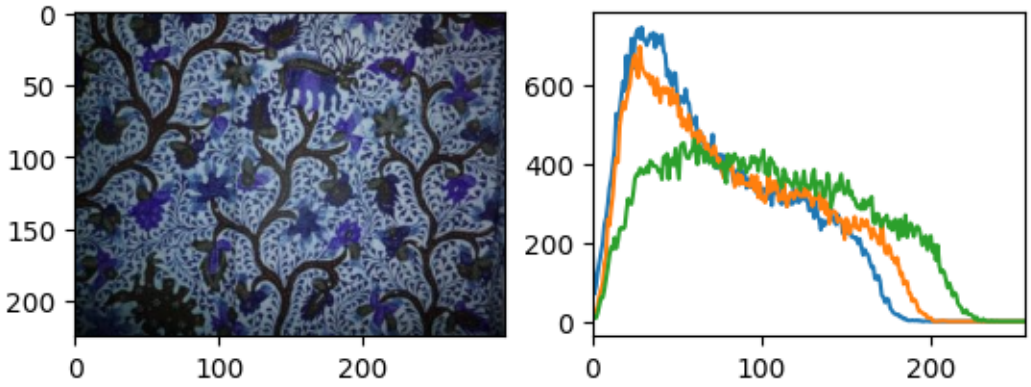
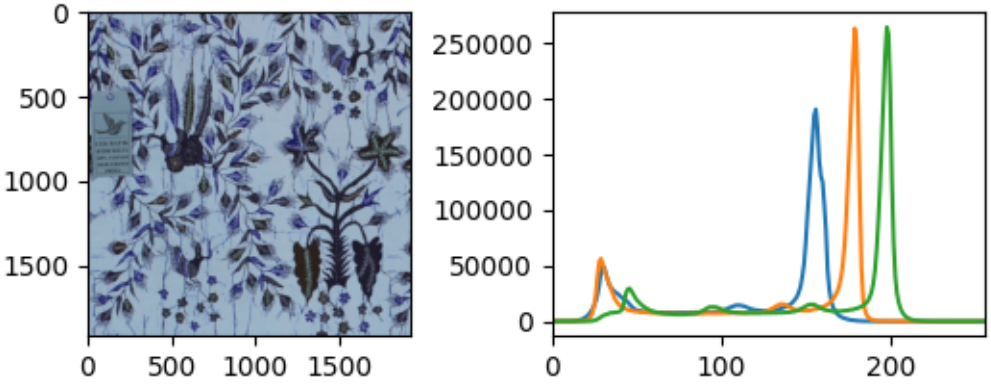
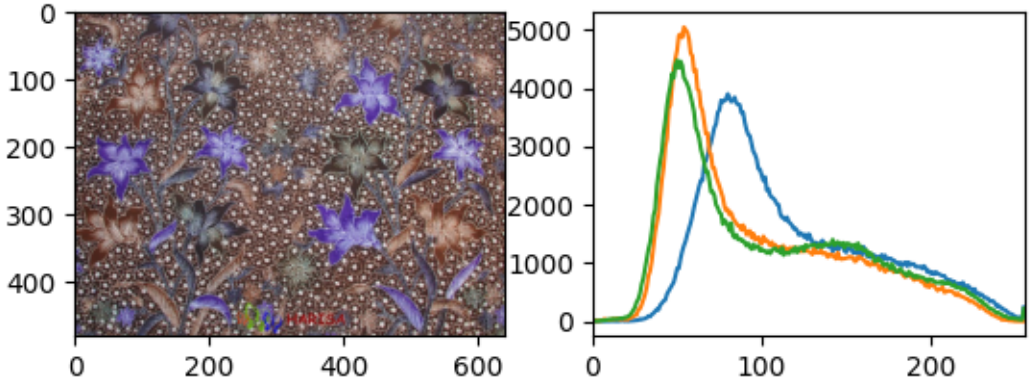

30000
20000
10000
0
0
100
200

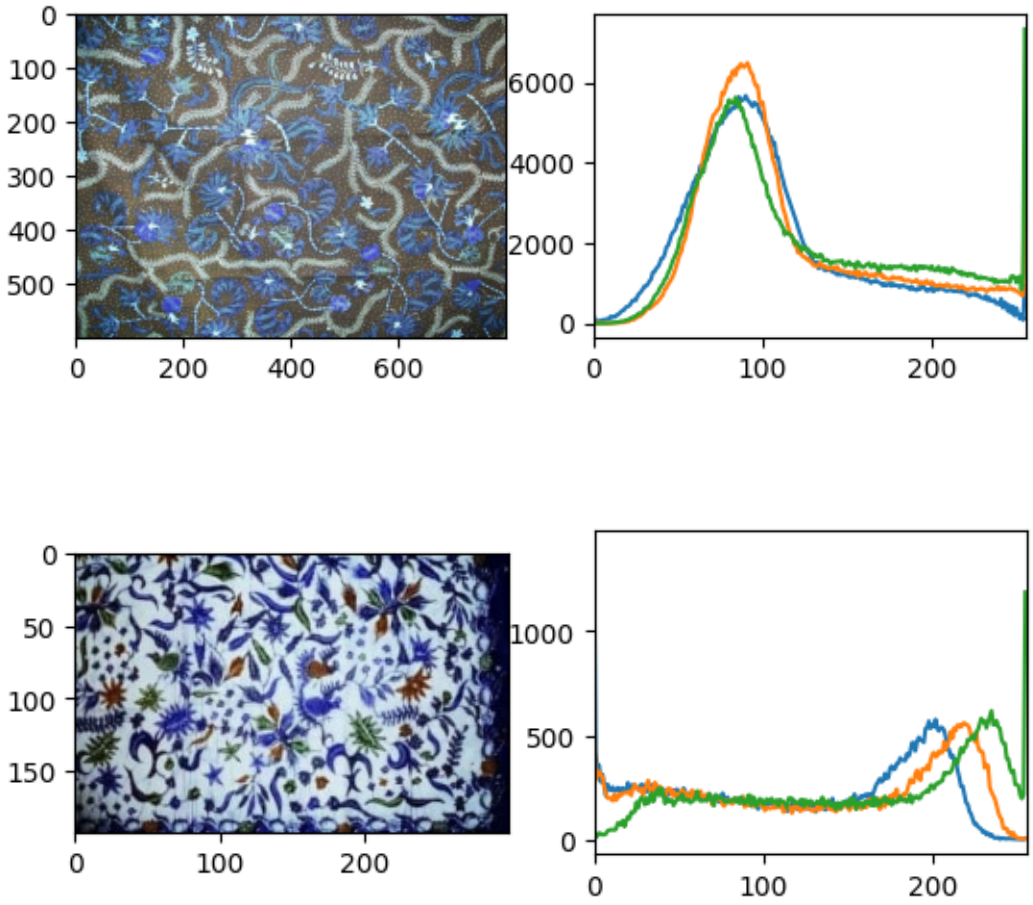

0
200
400
0
200
400
600


10000
8000
6000
4000
2000
0
0
100
200


0
100
200
300
0
200
400

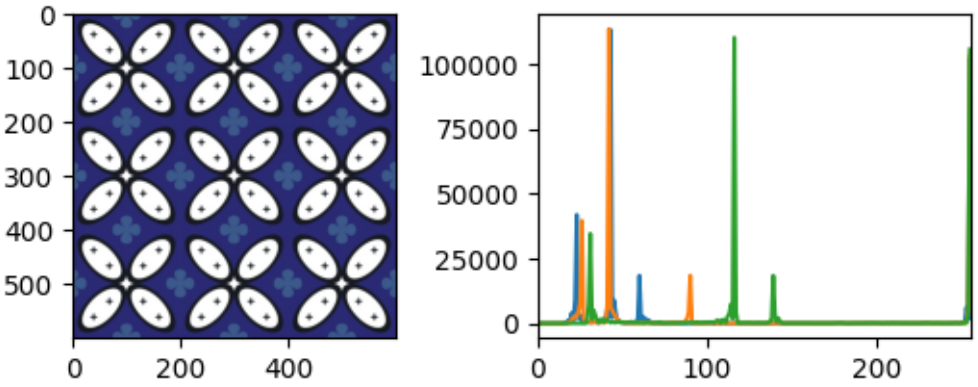

10000
8000
6000
4000
2000
0
0
100
200

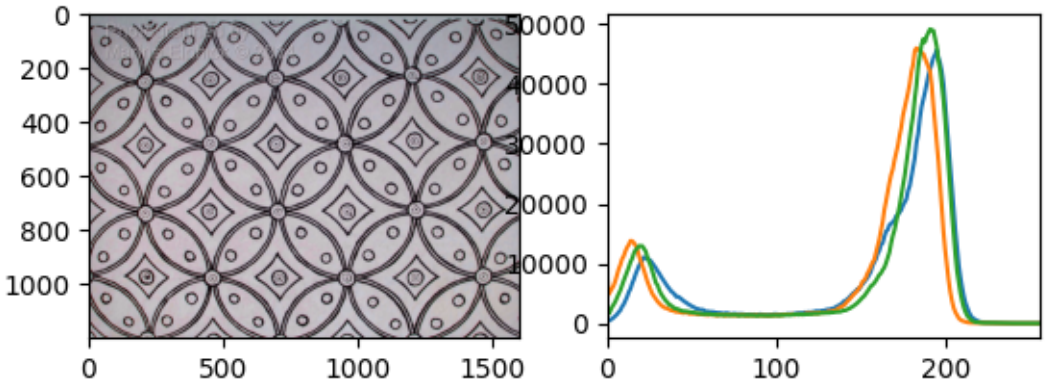




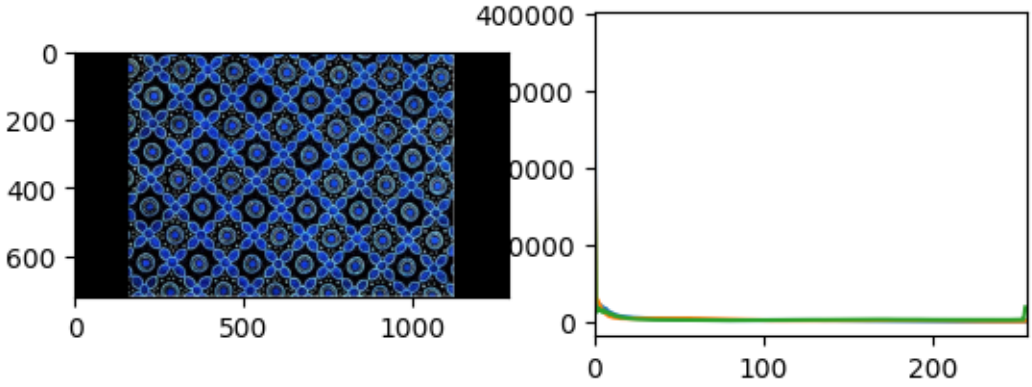


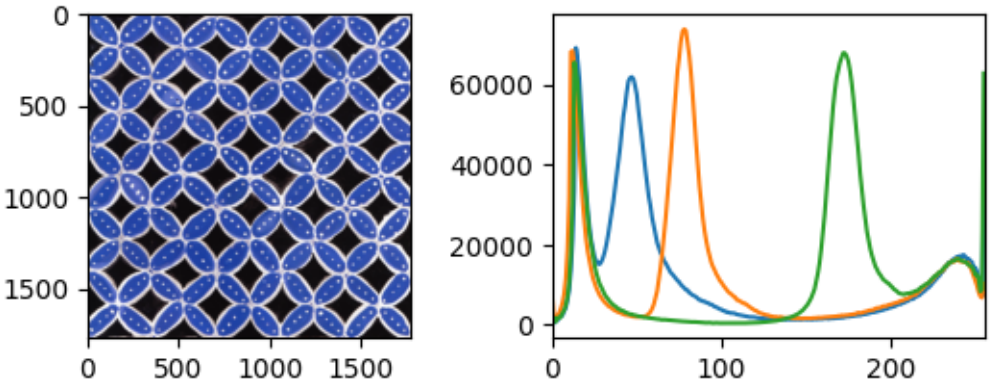
3.1.5 Color Histogram Batik Kawung

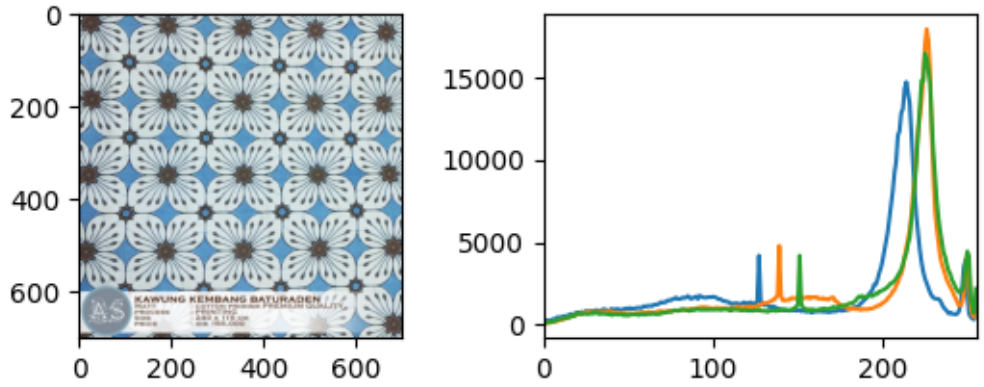
[ ]:
```
for i in os.listdir('/content/gdrive/MyDrive/Dataset2B/batik-kawung/'):
  img = cv2.imread('/content/gdrive/MyDrive/Dataset2B/batik-kawung/'+i)
  hist1 = cv2.calcHist([img],[0],None,[256],[0,256])
  hist2 = cv2.calcHist([img],[1],None,[256],[0,256])
  hist3 = cv2.calcHist([img],[2],None,[256],[0,256])

  plt.subplot(221), plt.imshow(img)
  plt.subplot(222), plt.plot(hist1), plt.plot(hist2),plt.plot(hist3)
  plt.xlim([0,256])

  plt.show()
```

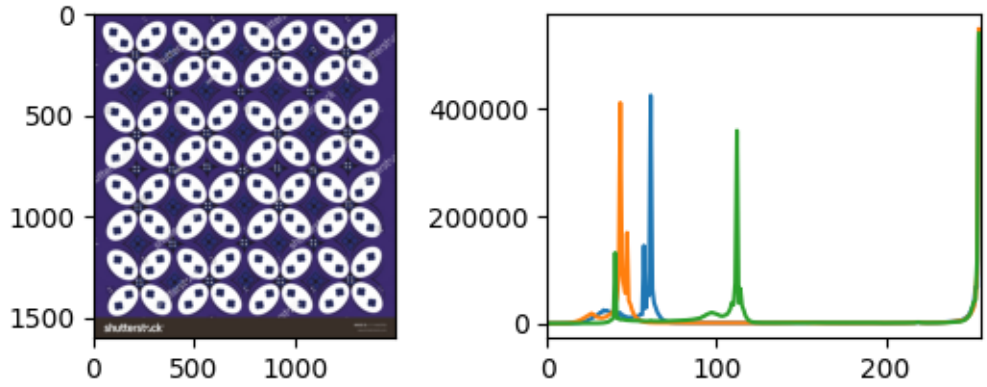

0
100
200
300
400
500
0
200
400
100000
75000
50000
25000
0
0
100
200

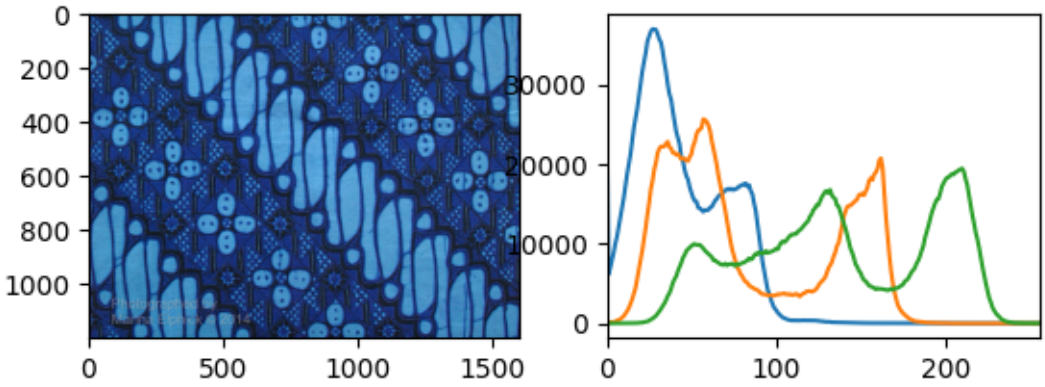

0
200
400
600
800
1000
0
500
1000
1500
50000
40000
30000
20000
10000
0
0
100
200

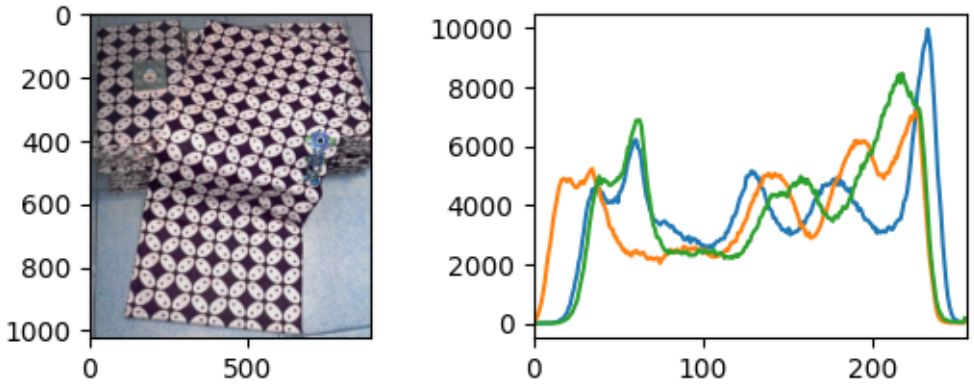

0
200
400
600
0
500
1000
400000
0000
0000
0000
0
0
100
200

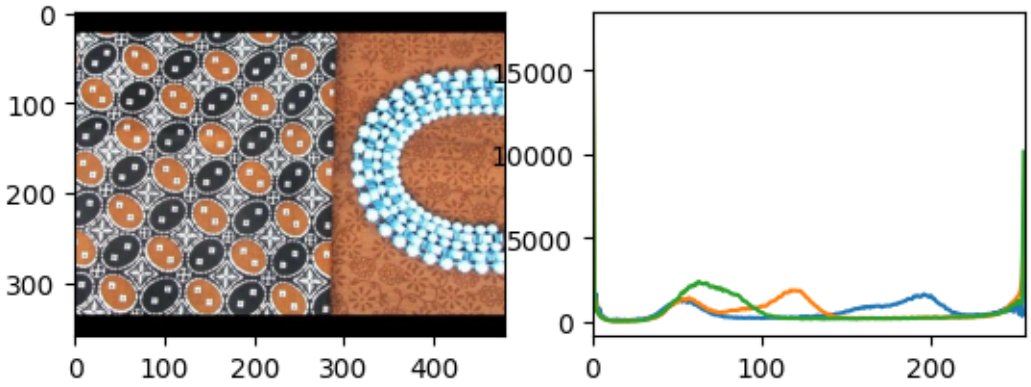

0
500
1000
1500
0
500
1000
1500
60000
40000
20000
0
0
100
200

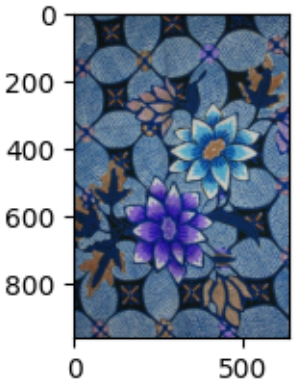

0
200
400
600
0
200
400
600
15000
10000
5000
0
0
100
200

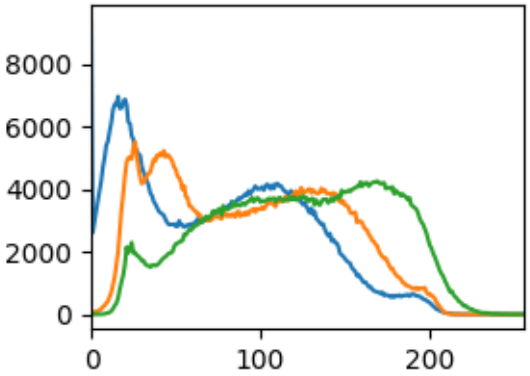

0
500
1000
1500
0
500
1000
400000
200000
0
0
100
200

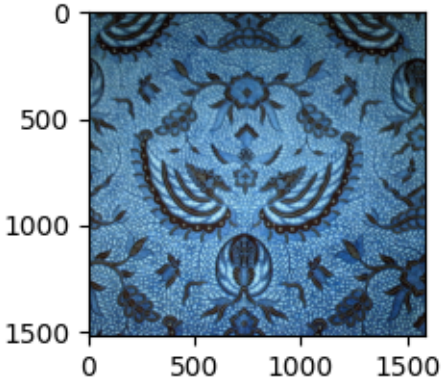

0
200
400
600
800
1000
0
500
1000
1500
30000
20000
10000
0
0
100
200

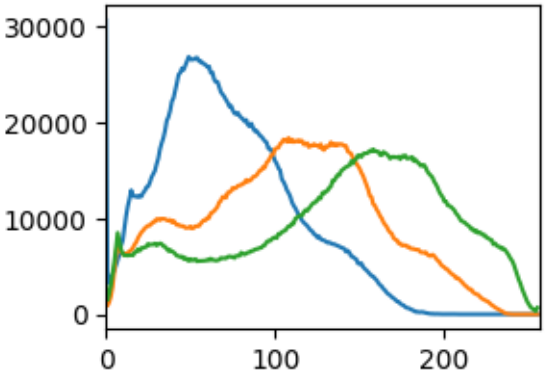

0
200
400
600
800
1000
0
500
10000
8000
6000
4000
2000
0
0
100
200

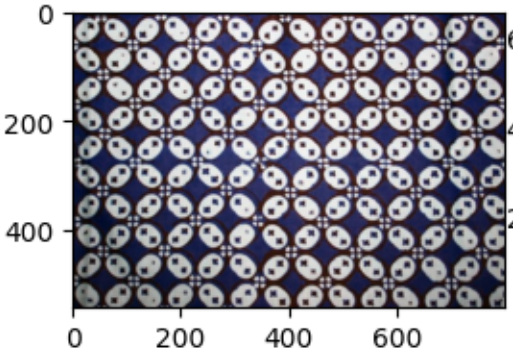

0
100
200
300
0
100
200
300
400
15000
10000
5000
0
0
100
200

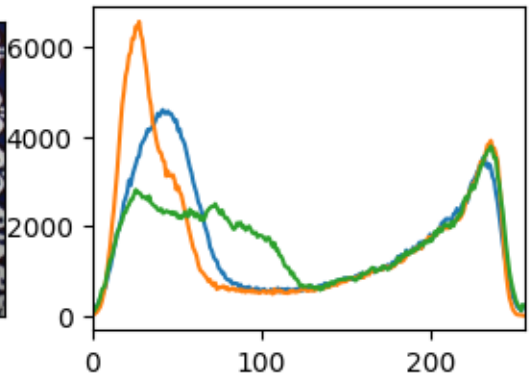

0
200
400
600
800
0
500

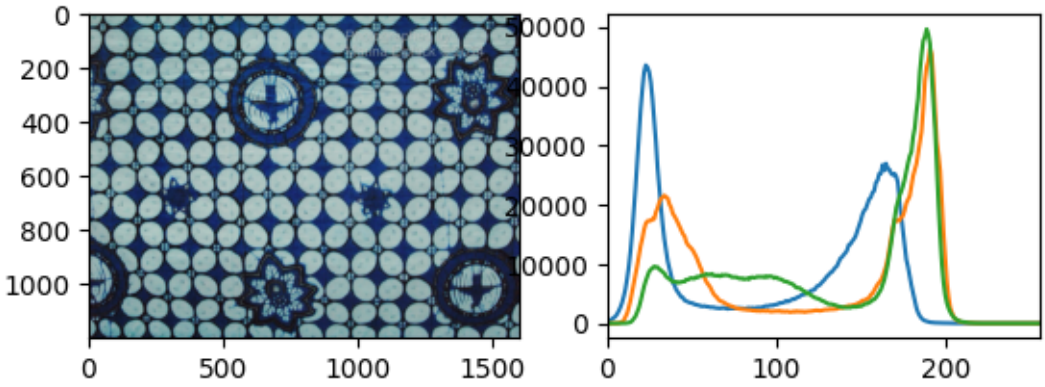

8000
6000
4000
2000
0
0
100
200

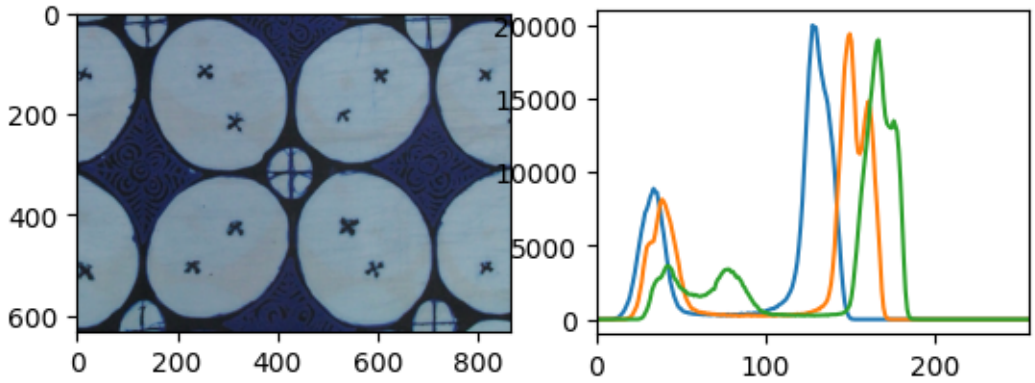

0
500
1000
1500
0
500
1000
1500

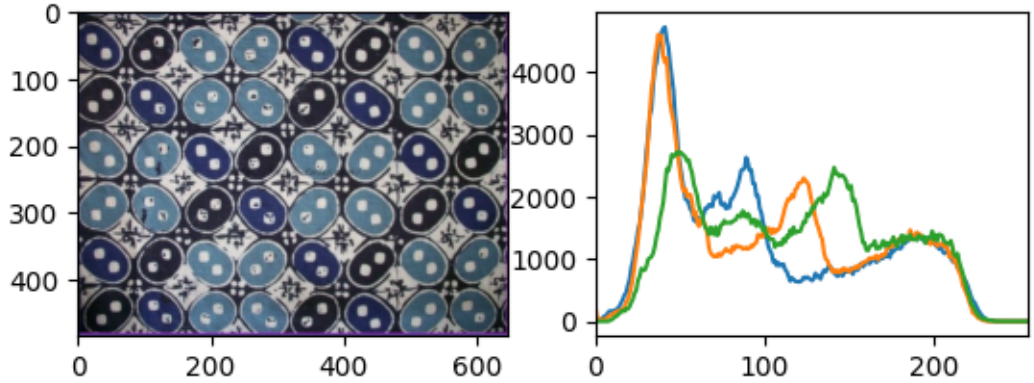

30000
20000
10000
0
0
100
200

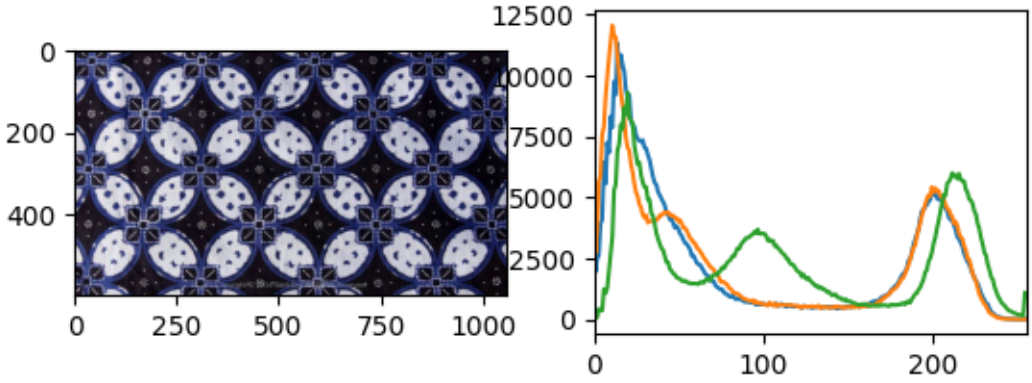

0
200
400
0
200
400
600

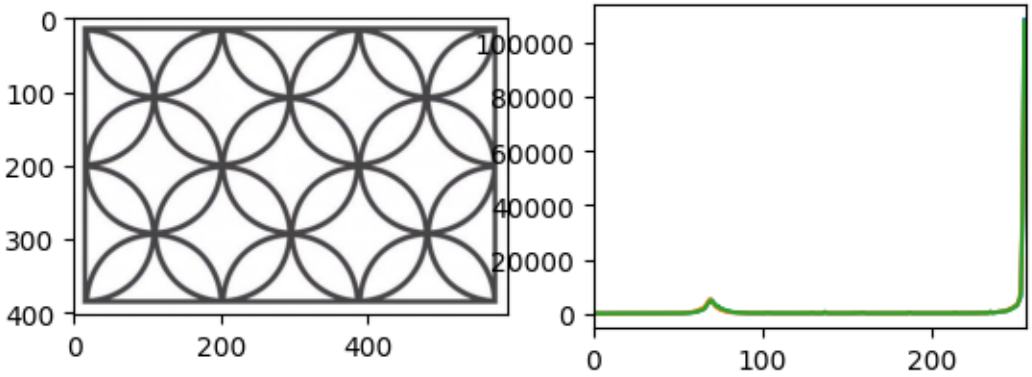

6000
4000
2000
0
0
100
200

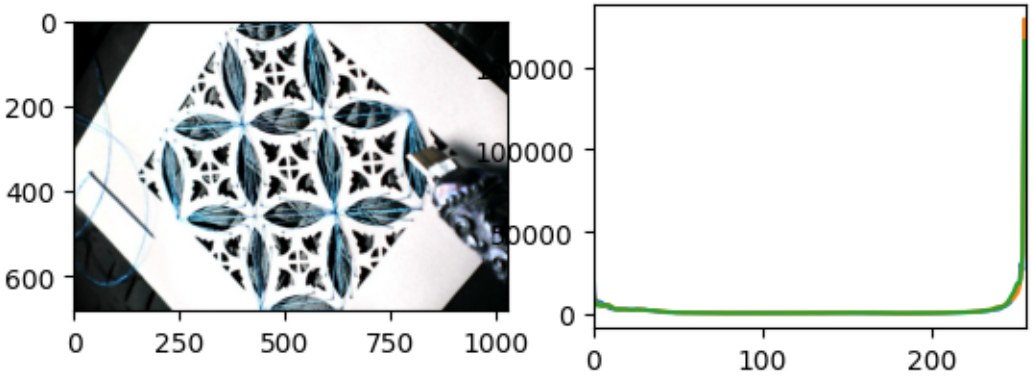

0
200
400
600
800
1000
0
500
1000
1500
50000
40000
30000
20000
10000
0
0
100
200

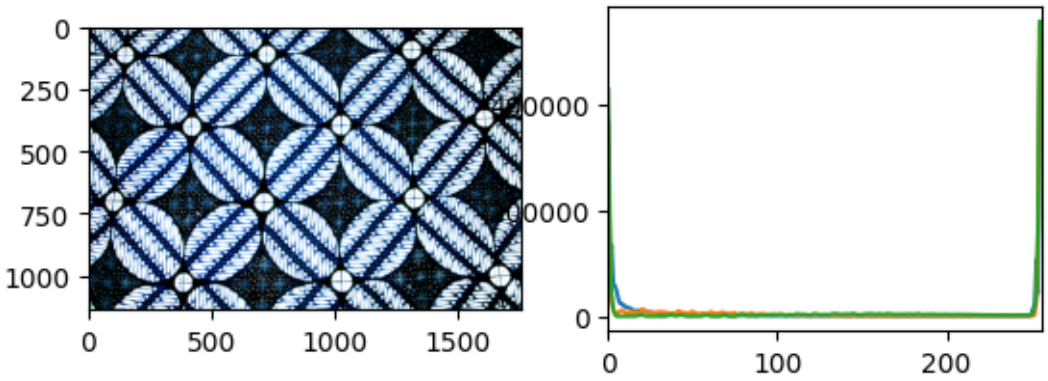

0
200
400
600
0
200
400
600
800
20000
15000
10000
5000
0
0
100
200

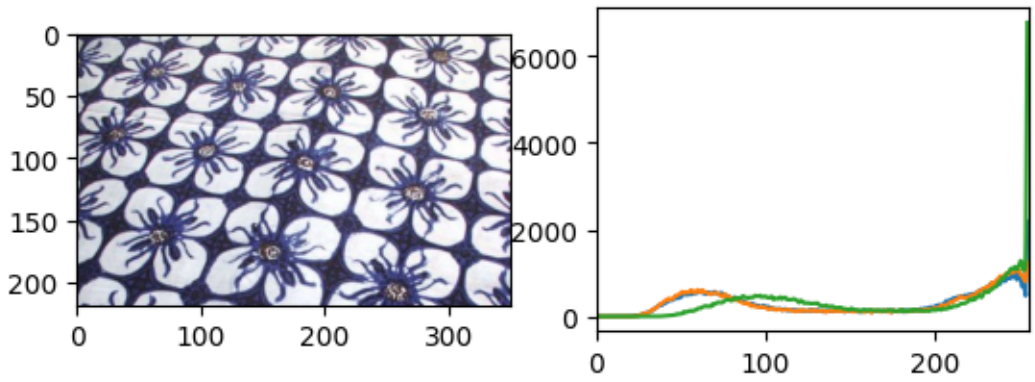

0
100
200
300
400
0
200
400
600
4000
3000
2000
1000
0
0
100
200

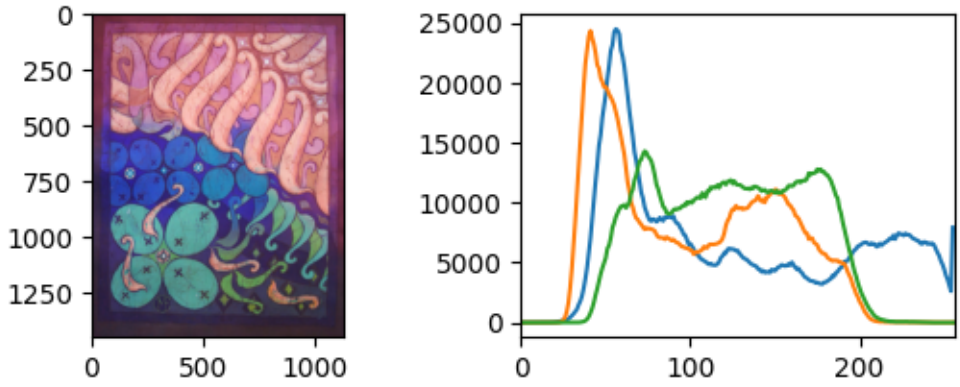

0
200
400
0
250
500
750
1000
12500
10000
7500
5000
2500
0
0
100
200

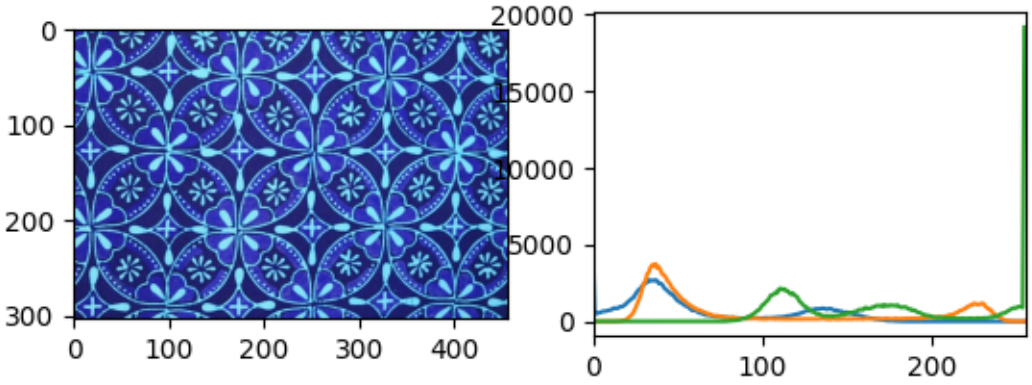

0
100
200
300
400
0
200
400
100000
80000
60000
40000
20000
0
0
100
200

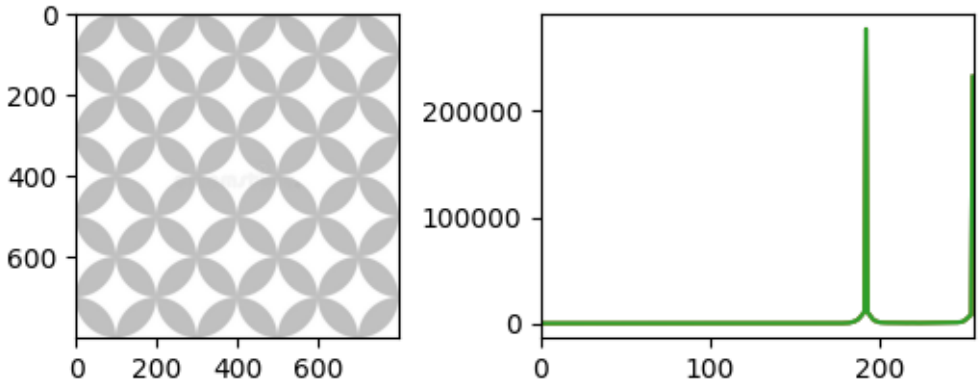

0
200
400
600
0
250
500
750
1000
0
0
100
200

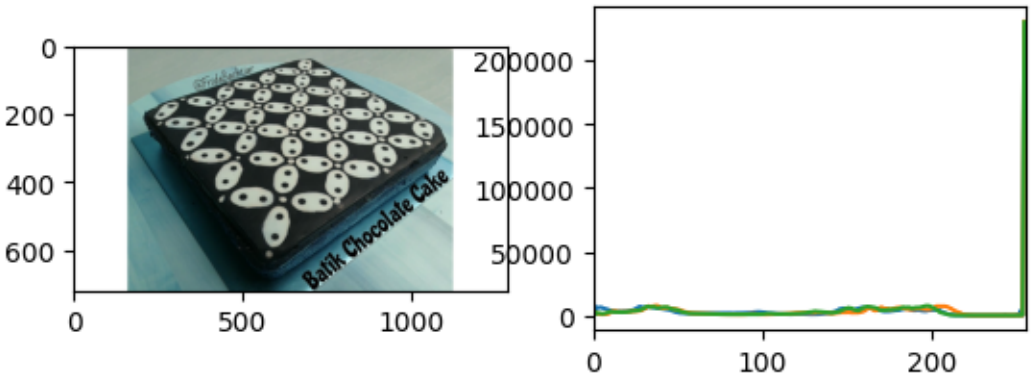

0
250
500
750
1000
0
500
1000
1500
0
100
200

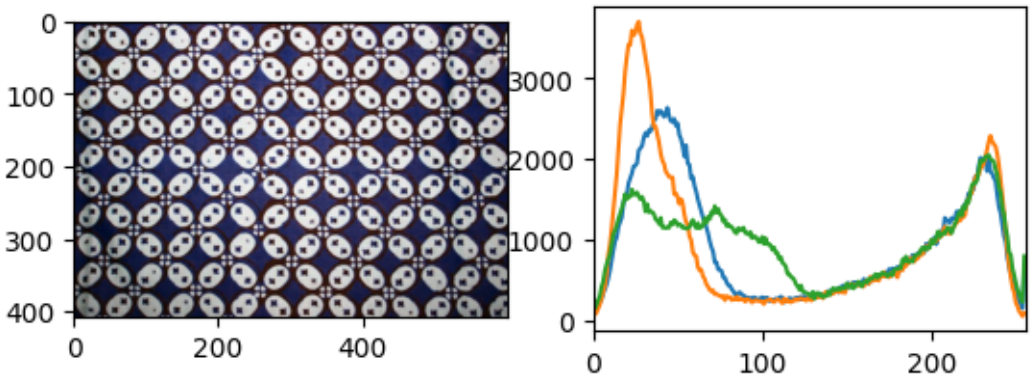

0
50
100
150
200
0
100
200
300
6000
4000
2000
0
0
100
200

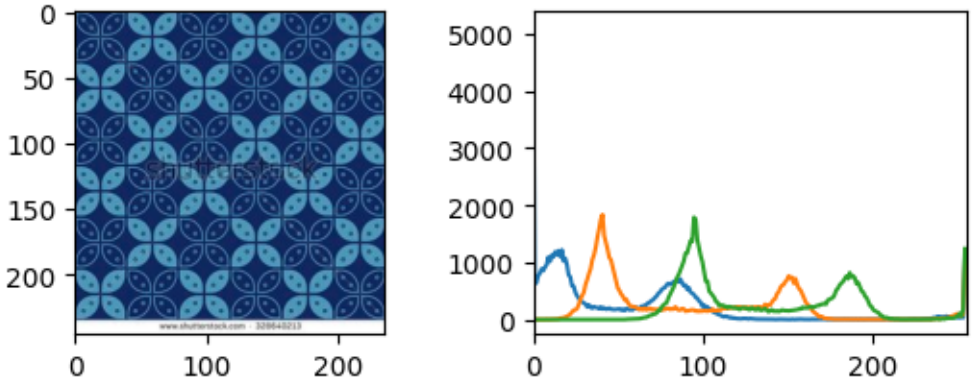

0
250
500
750
1000
1250
0
500
1000
25000
20000
15000
10000
5000
0
0
100
200

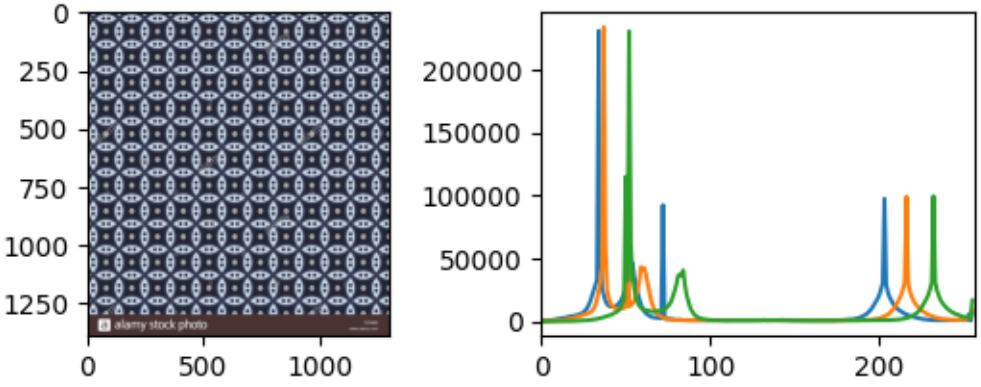

0
100
200
300
0
100
200
300
400
20000
15000
10000
5000
0
0
100
200

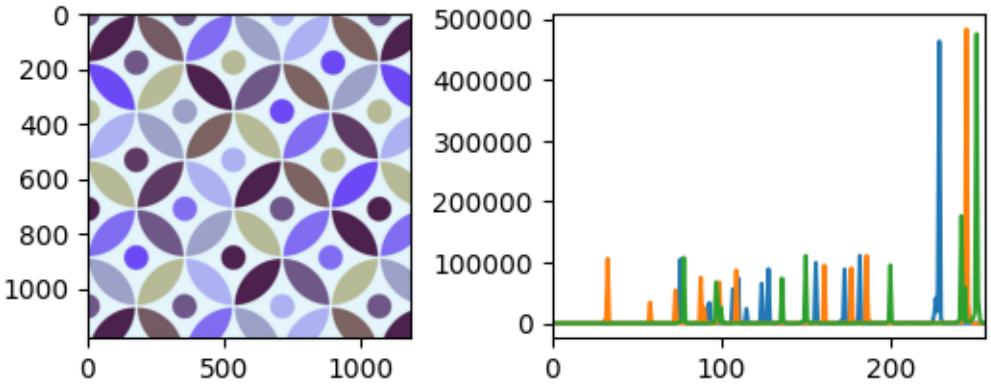

0
200
400
600
0
200
400
600
200000
100000
0
0
100
200

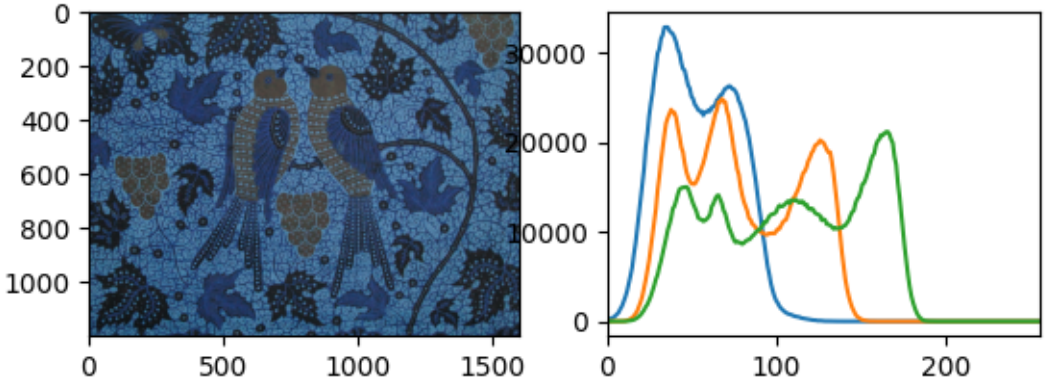

0
200
400
600
0
500
1000
Batik Chocolate Cake
200000
150000
100000
50000
0
0
100
200

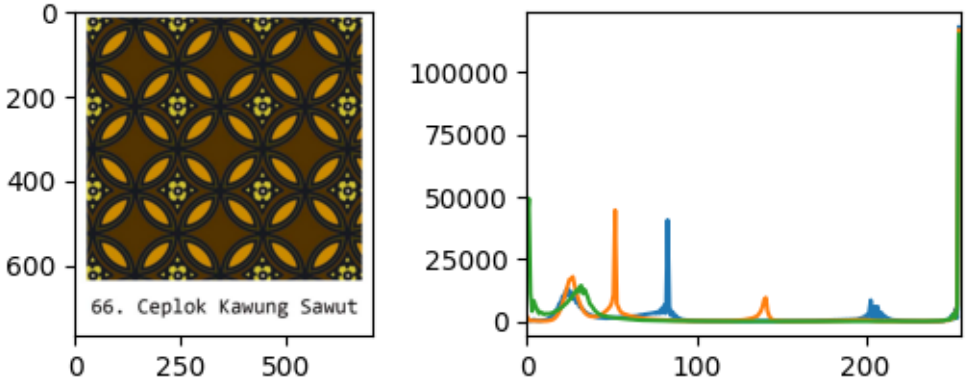

0
100
200
300
400
0
200
400
3000
2000
1000
0
0
100
200

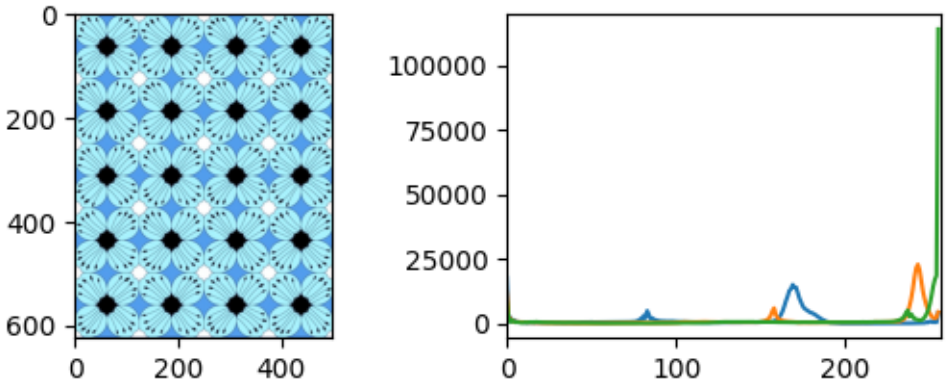

0
50
100
150
200
0
100
200
5000
4000
3000
2000
1000
0
0
100
200

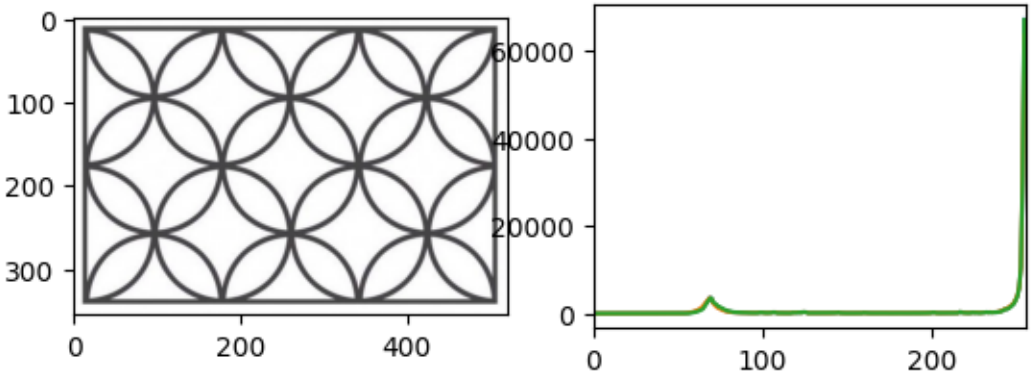

0
250
500
750
1000
1250
0
500
1000
200000
150000
100000
50000
0
0
100
200

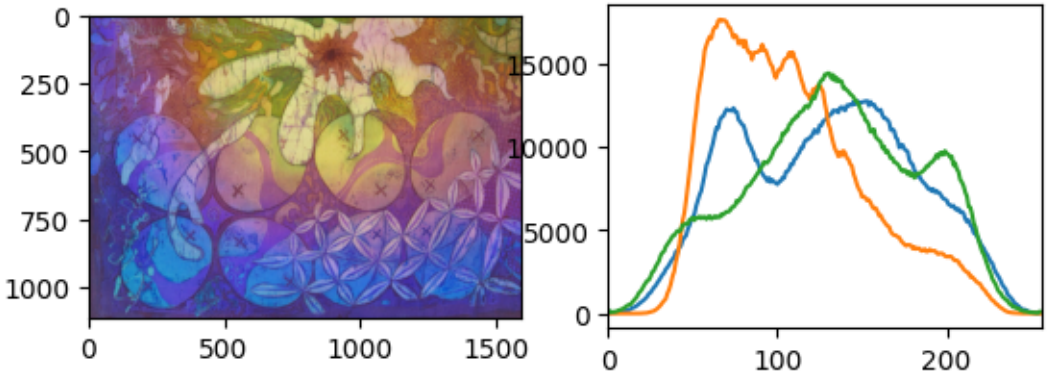

0
200
400
600
800
1000
0
500
1000
500000
400000
300000
200000
100000
0
0
100
200

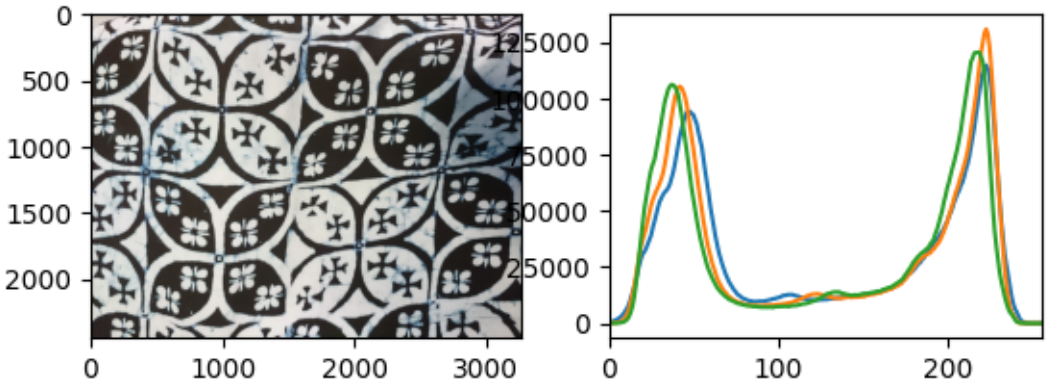

0
200
400
600
800
1000
0
500
1000
1500
30000
20000
10000
0
0
100
200

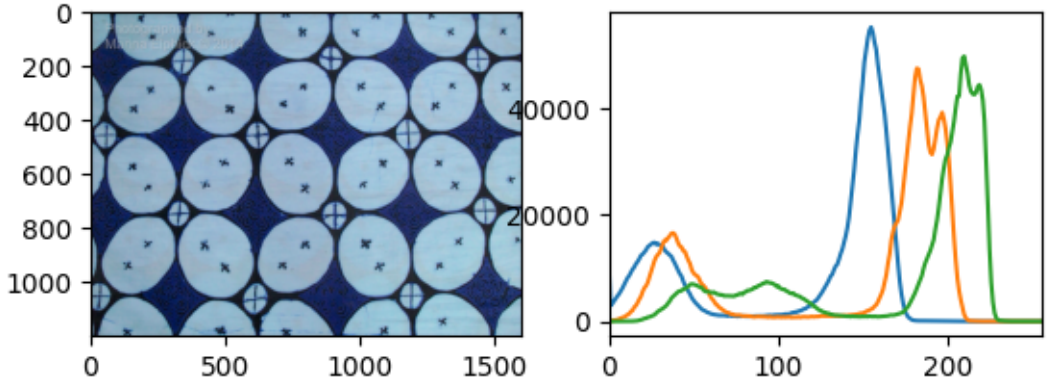

0
200
400
600
66. Ceplok Kawung Sawut
0
250
500
100000
75000
50000
25000
0
0
100
200

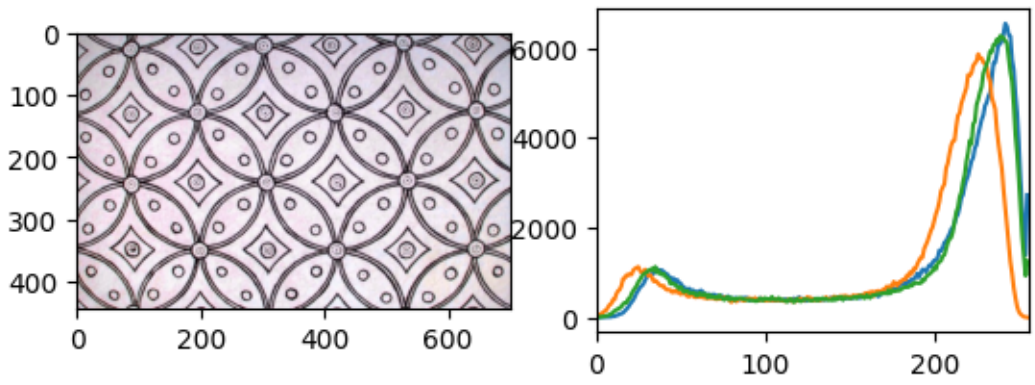

0
200
400
600
0
200
400
100000
75000
50000
25000
0
0
100
200

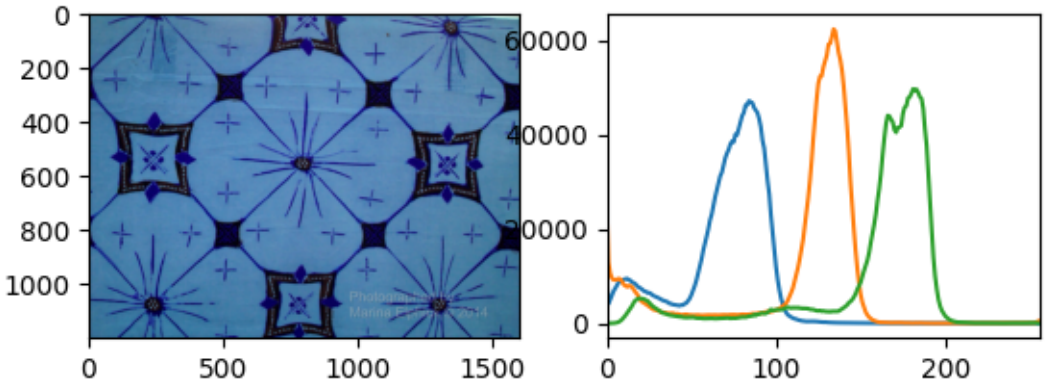

0
100
200
300
0
200
400
60000
40000
20000
0
0
100
200

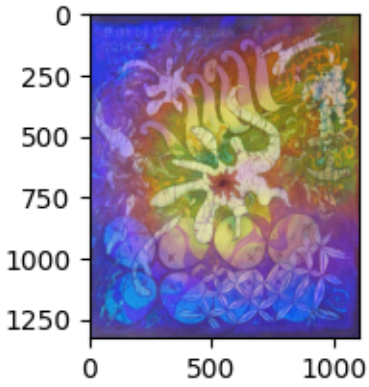

0
250
500
750
1000
0
500
1000
1500
15000
10000
5000
0
0
100
200

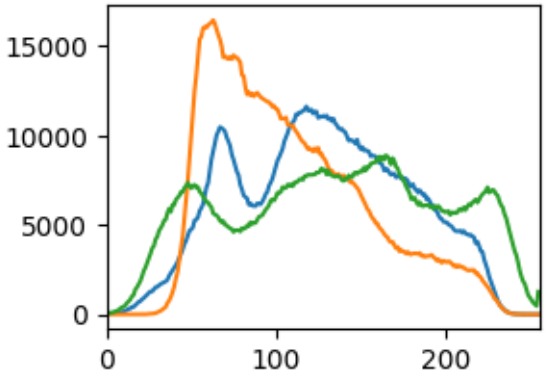

0
500
1000
1500
2000
0
1000
2000
3000
125000
100000
75000
50000
25000
0
0
100
200

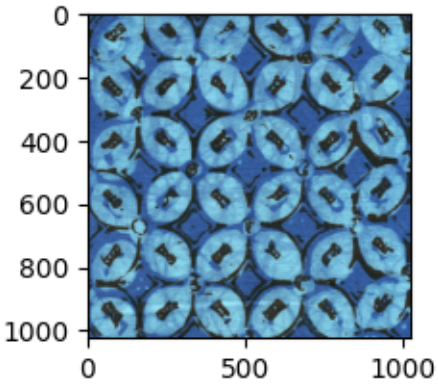

0
200
400
600
800
1000
0
500
1000
1500
40000
20000
0
0
100
200

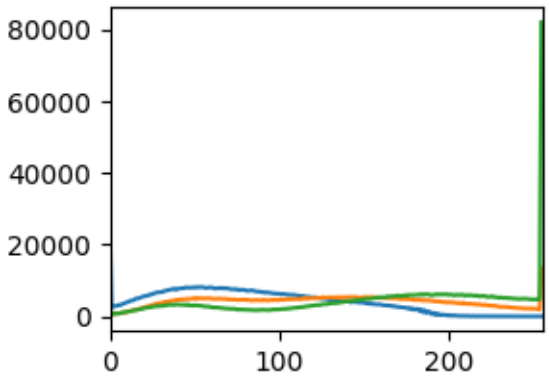

0
100
200
300
400
0
200
400
600
6000
4000
2000
0
0
100
200

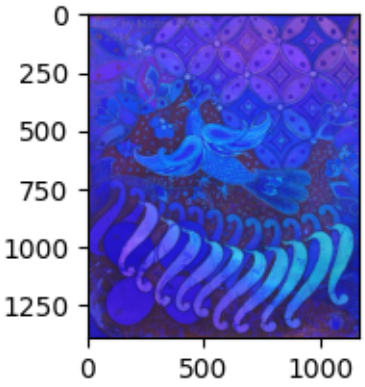

0
200
400
600
800
1000
0
500
1000
1500
60000
40000
20000
0
0
100
200

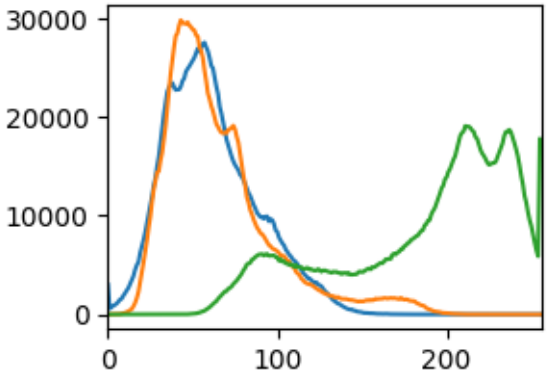

0
250
500
750
1000
1250
0
500
1000

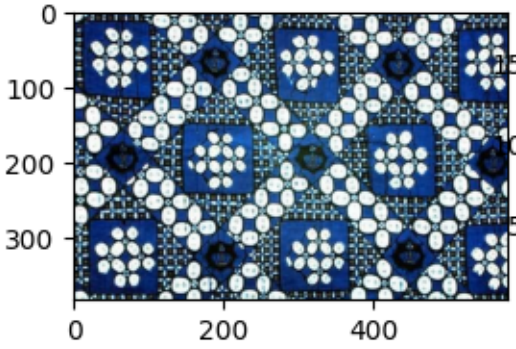

15000
10000
5000
0
0
100
200

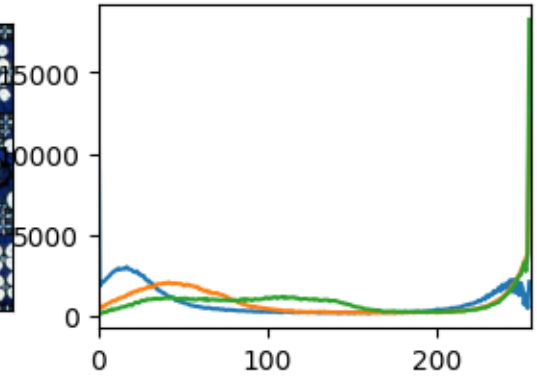

0
200
400
600
800
1000
0
500
1000

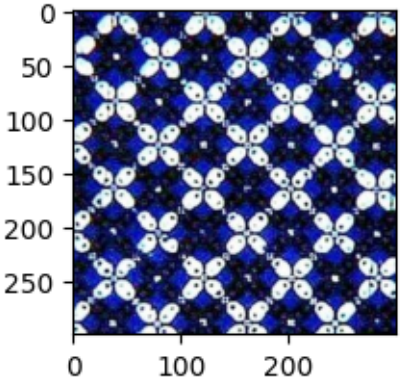

80000
60000
40000
20000
0
0
100
200

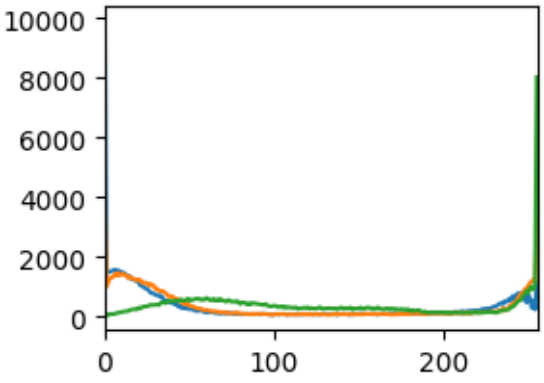

0
250
500
750
1000
1250
0
500
1000

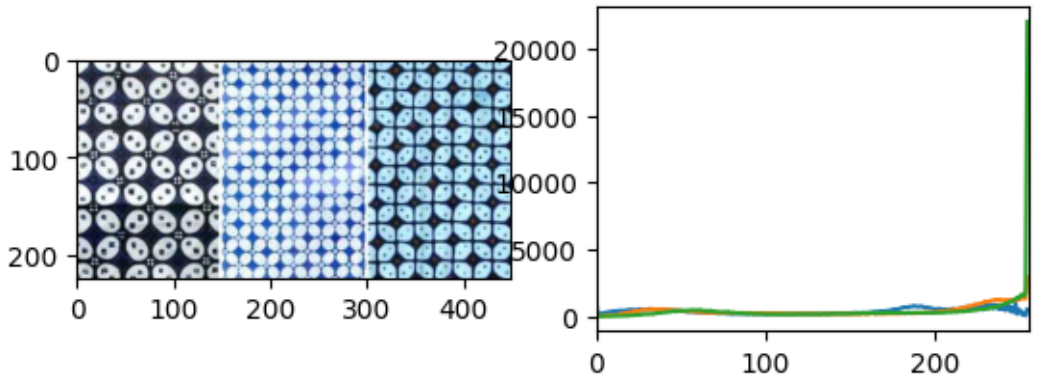

30000
20000
10000
0
0
100
200

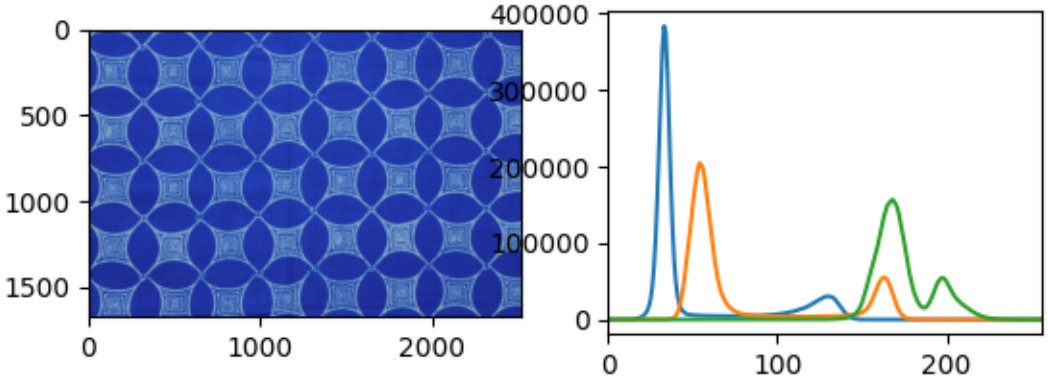

0
100
200
300
0
200
400

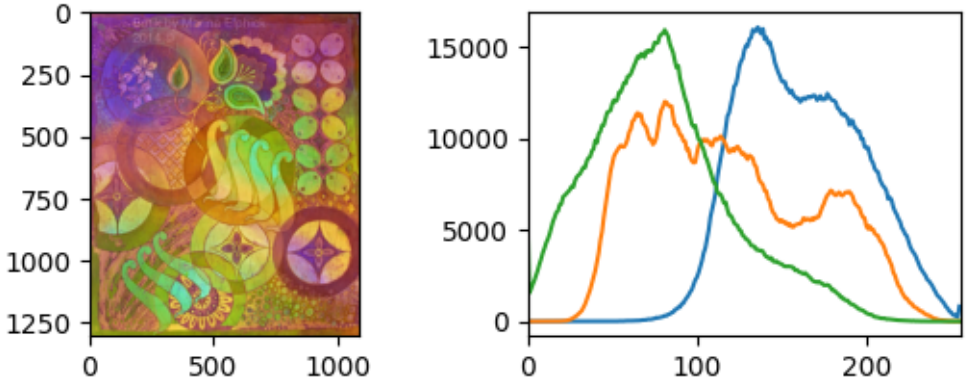

15000
10000
5000
0
0
100
200


0
50
100
150
200
250
0
100
200


10000
8000
6000
4000
2000
0
0
100
200

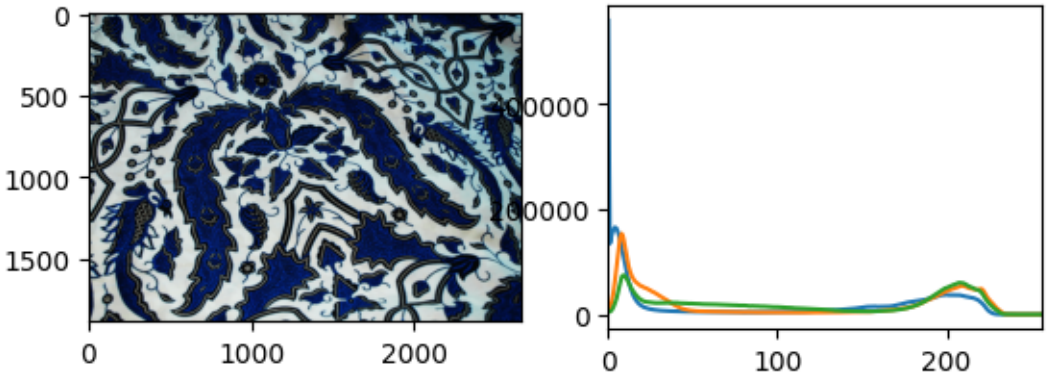

0
100
200
0
100
200
300
400
20000
15000
10000
5000
0
0
100
200

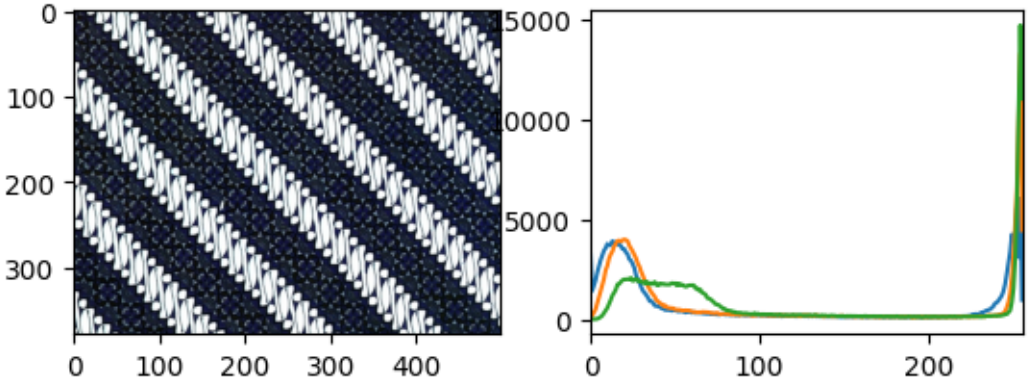

0
500
1000
1500
0
1000
2000
400000
300000
200000
100000
0
0
100
200

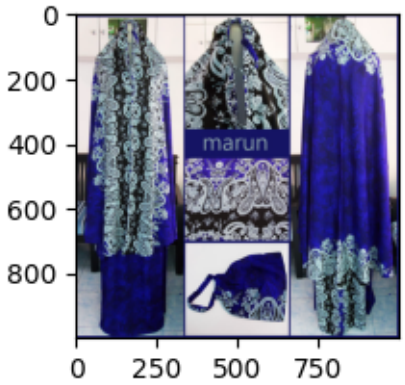

0
250
500
750
1000
1250
0
500
1000
15000
10000
5000
0
0
100
200

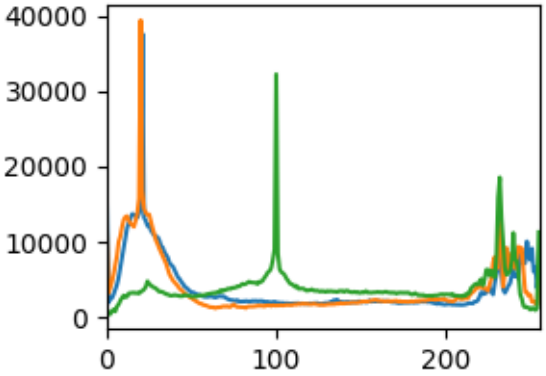
3.1.6 Color Histogram Batik Keraton

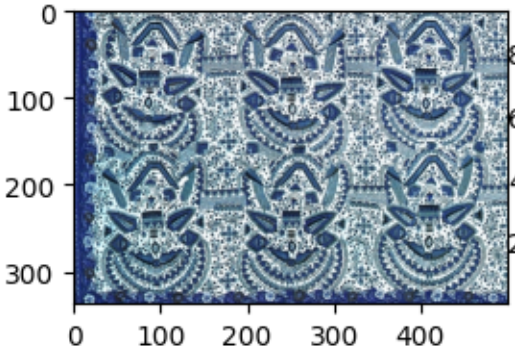
```
[ ]: for i in os.listdir('/content/gdrive/MyDrive/Dataset2B/batik-keraton/'):
       img = cv2.imread('/content/gdrive/MyDrive/Dataset2B/batik-keraton/'+i)
       hist1 = cv2.calcHist([img],[0],None,[256],[0,256])
       hist2 = cv2.calcHist([img],[1],None,[256],[0,256])
       hist3 = cv2.calcHist([img],[2],None,[256],[0,256])

       plt.subplot(221), plt.imshow(img)
       plt.subplot(222), plt.plot(hist1), plt.plot(hist2),plt.plot(hist3)
       plt.xlim([0,256])

       plt.show()
```

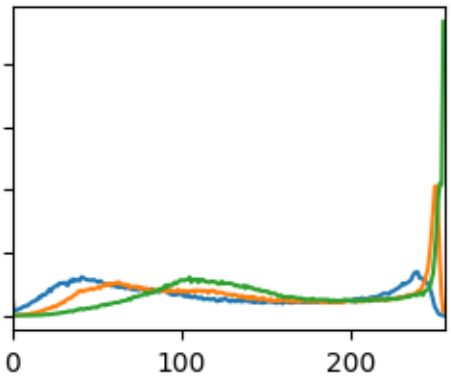


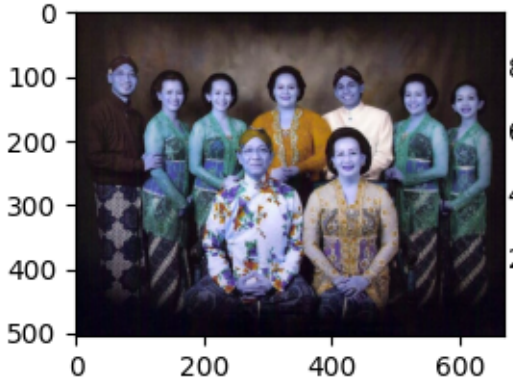


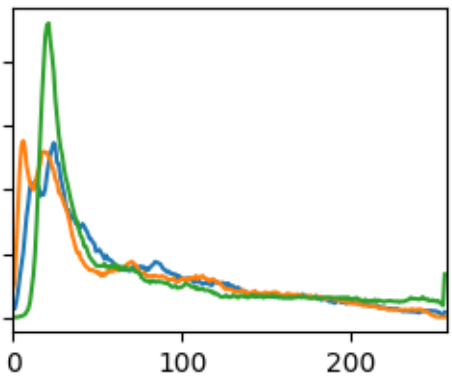

marun
0
200
400
600
800
0
250
500
750

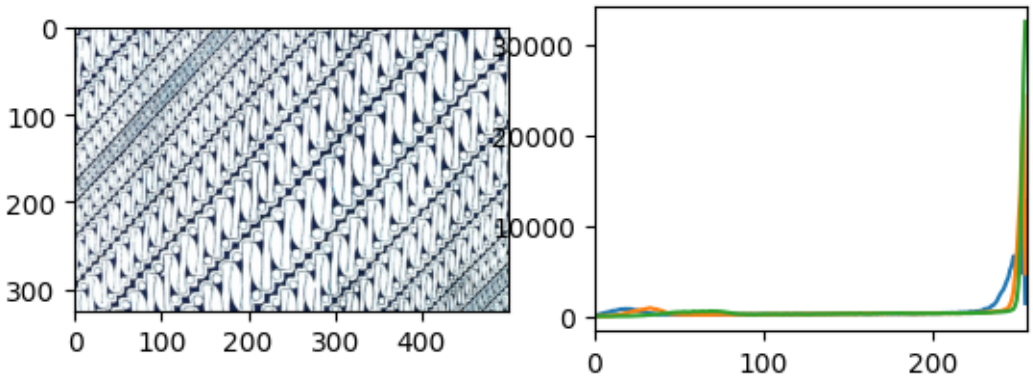

40000
30000
20000
10000
0
0
100
200

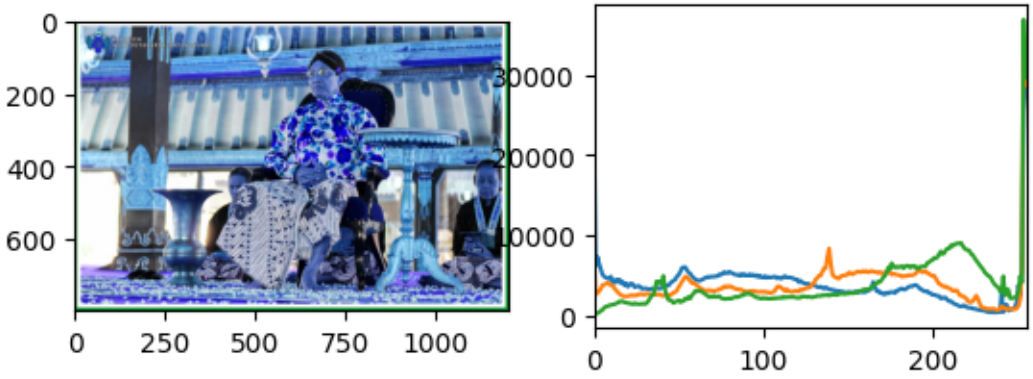

0
100
200
300
0
100
200
300
400

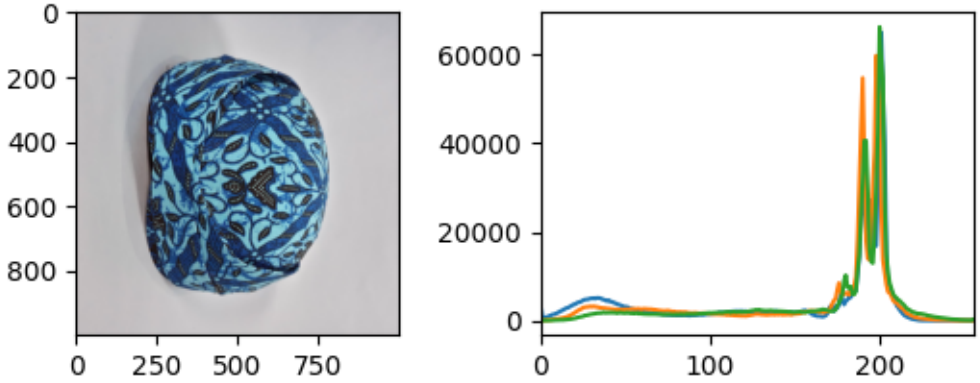

8000
6000
4000
2000
0
0
100
200

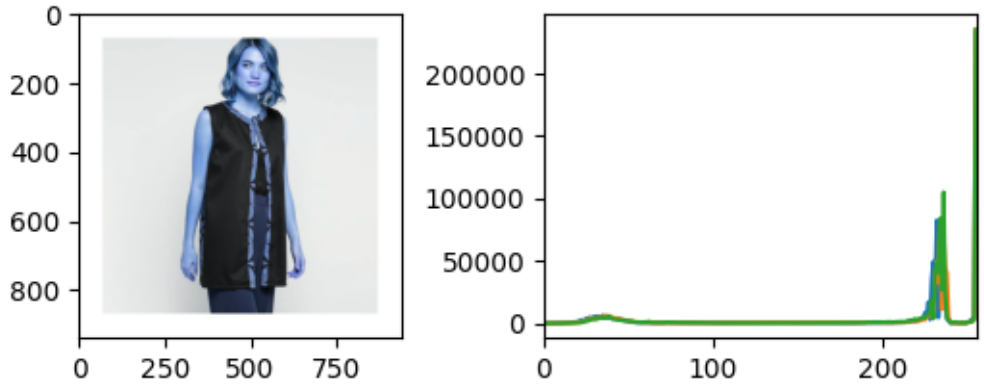

0
100
200
300
400
500
0
200
400
600

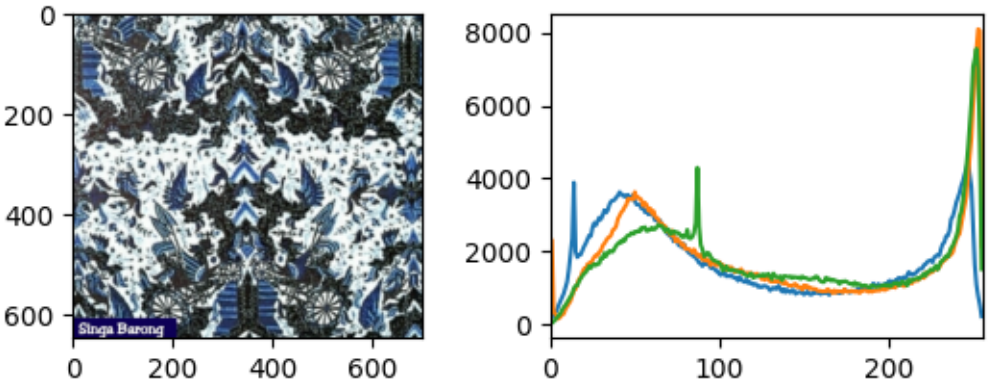

8000
6000
4000
2000
0
0
100
200

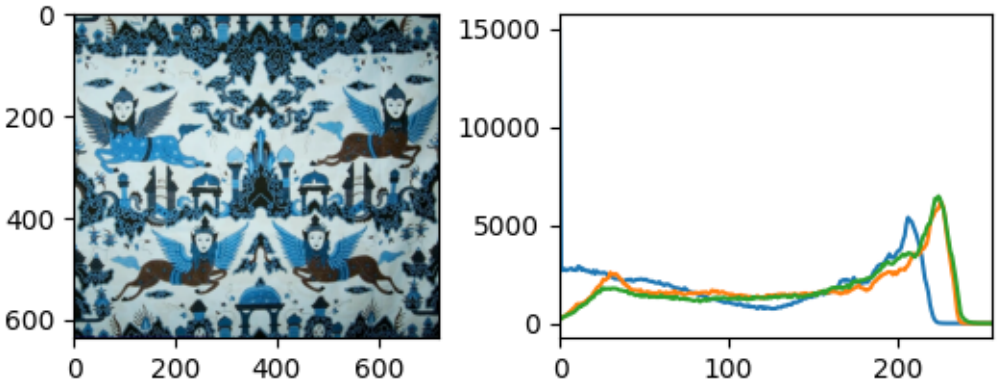

0
100
200
300
0
100
200
300
400
30000
20000
10000
0
0
100
200

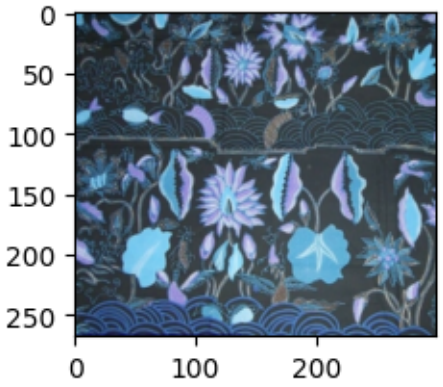

0
200
400
600
0
250
500
750
1000
30000
20000
10000
0
0
100
200

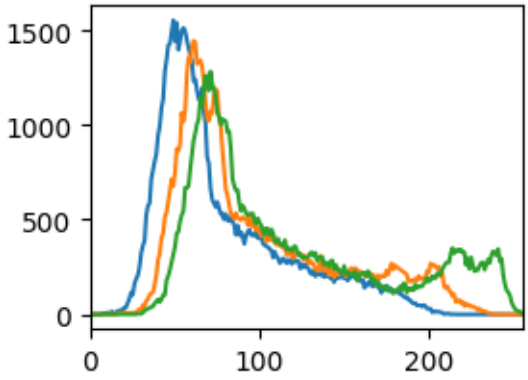

0
200
400
600
800
0
250
500
750
60000
40000
20000
0
0
100
200

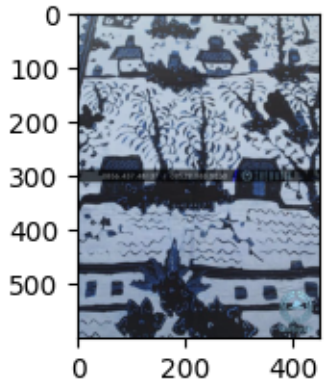

0
200
400
600
800
0
250
500
750
200000
150000
100000
50000
0
0
100
200

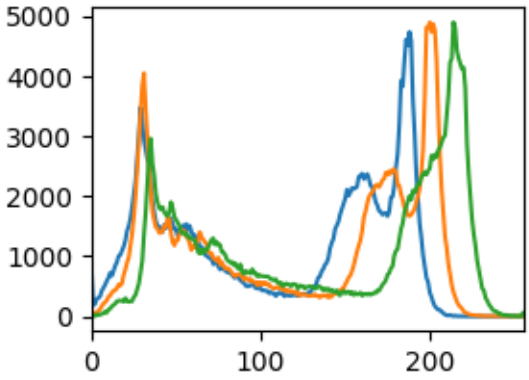

0
200
400
600
Singa Barong
0
200
400
600
8000
6000
4000
2000
0
0
100
200

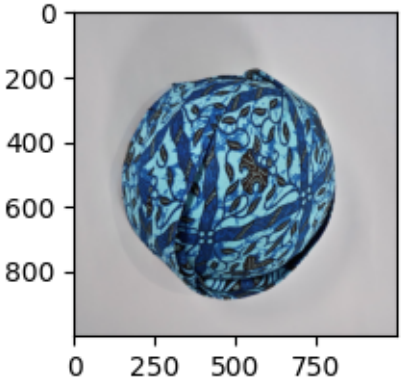

0
200
400
600
0
200
400
600
15000
10000
5000
0
0
100
200

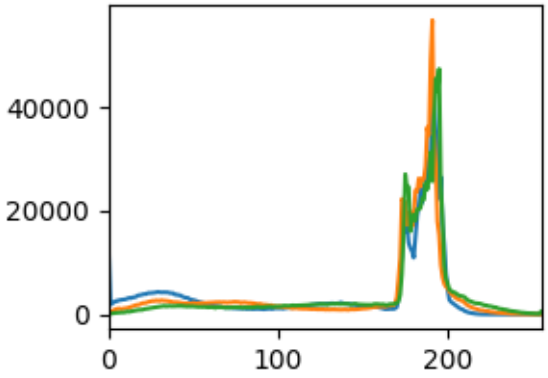

0
50
100
150
200
250
0
100
200

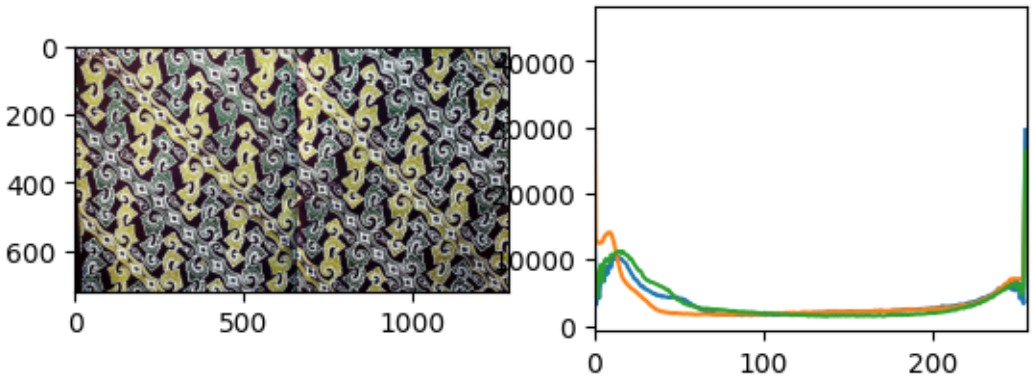

1500
1000
500
0
0
100
200

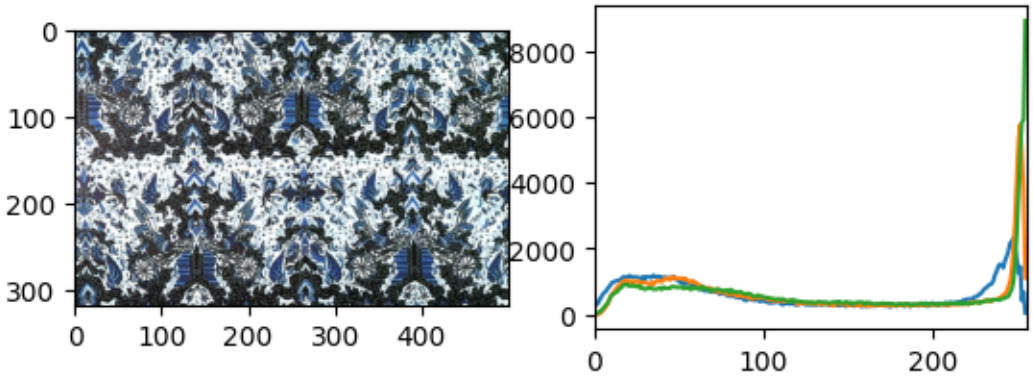

0
100
200
300
400
500
0
200
400

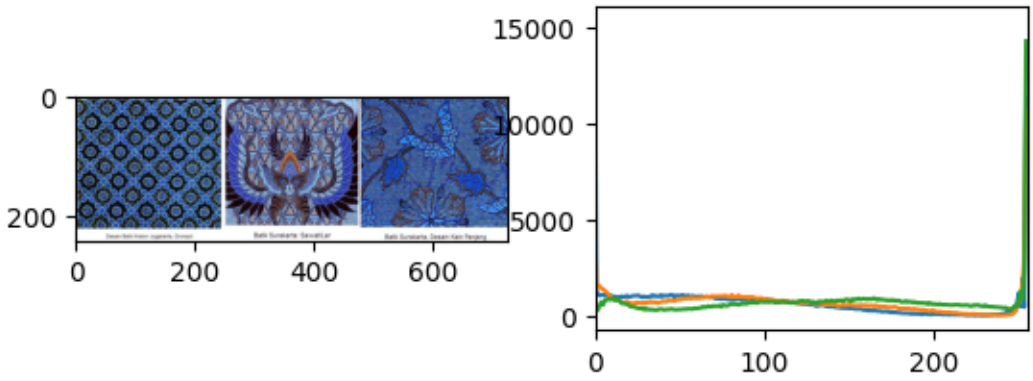

5000
4000
3000
2000
1000
0
0
100
200

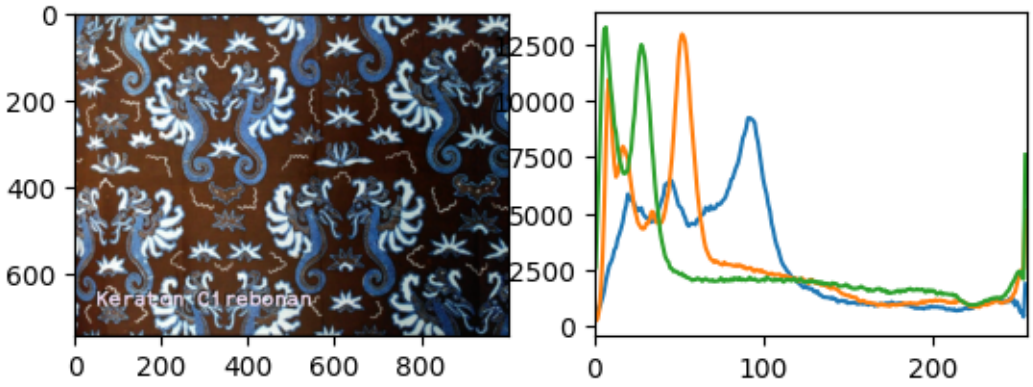

0
200
400
600
800
0
250
500
750

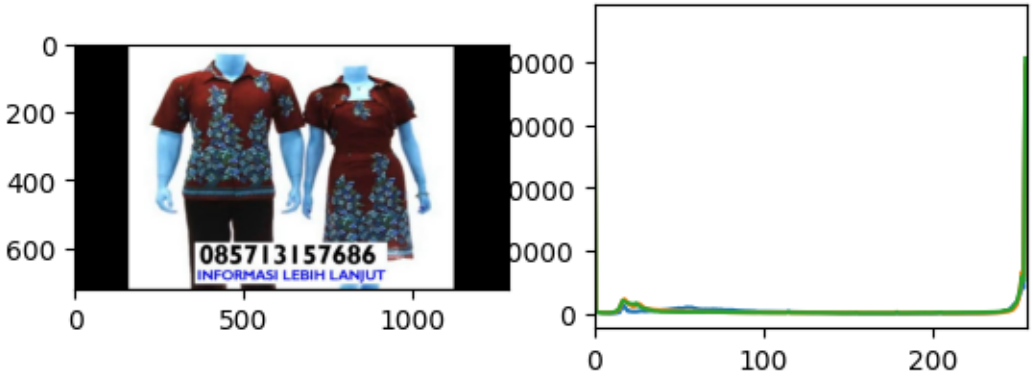

40000
20000
0
0
100
200

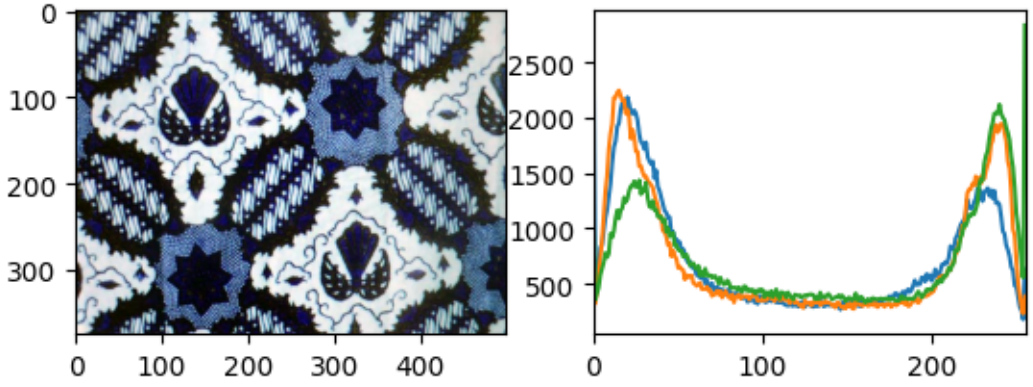

0
200
400
600
0
500
1000
0
100
200

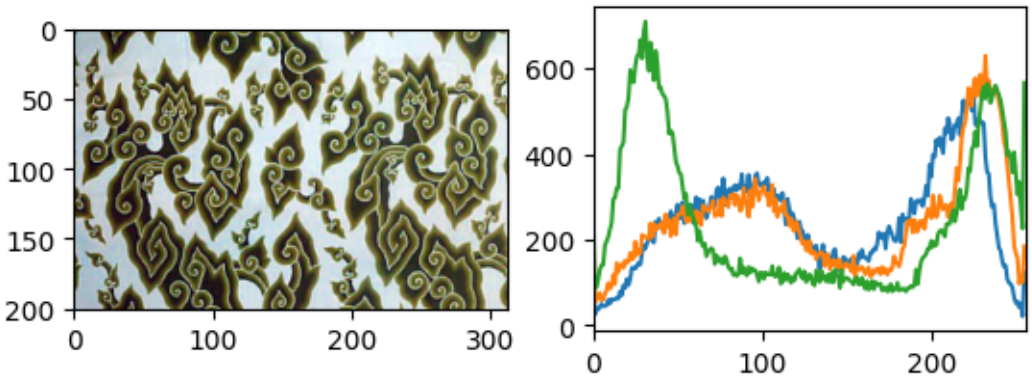

0
100
200
300
0
100
200
300
400
8000
6000
4000
2000
0
0
100
200

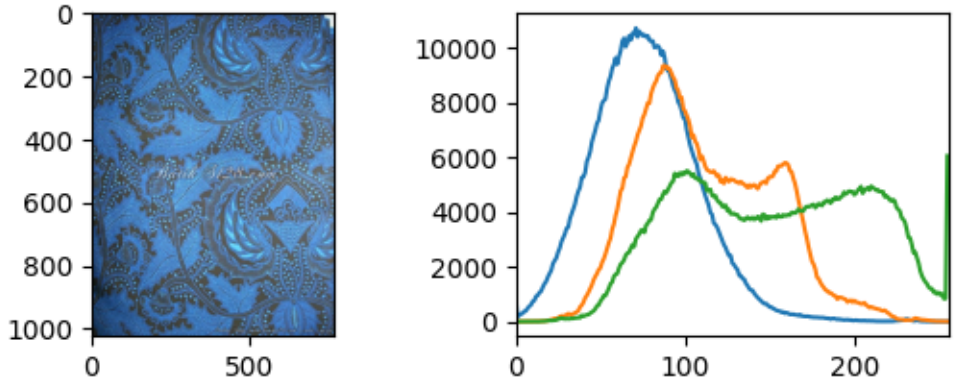

0
200
0
200
400
600
15000
10000
5000
0
0
100
200

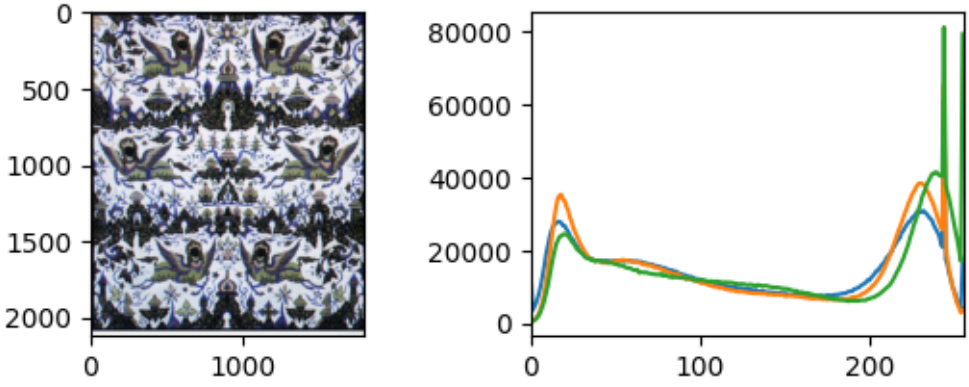

Keraton C1 rebonan

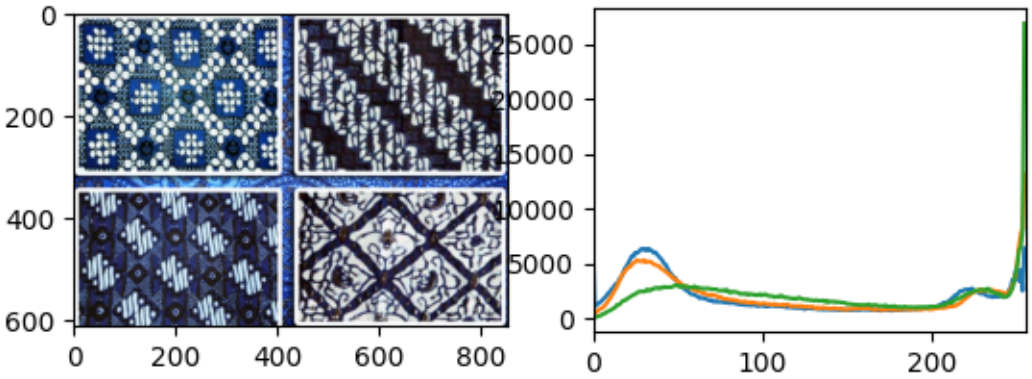

085713157686
INFORMASI LEBIH LANJUT

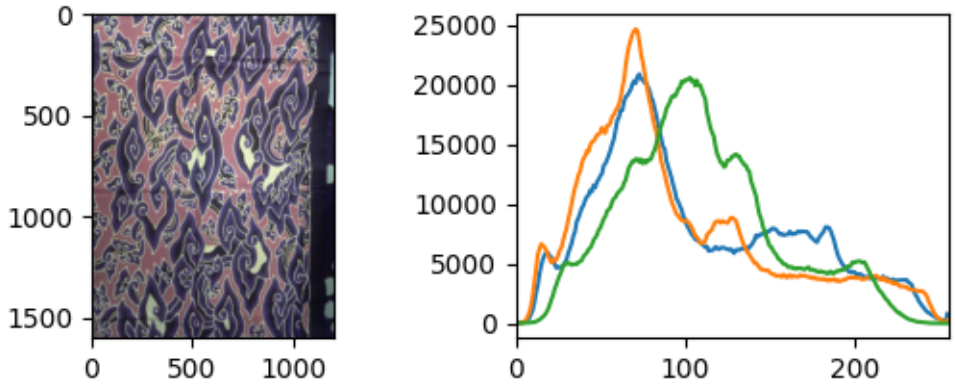


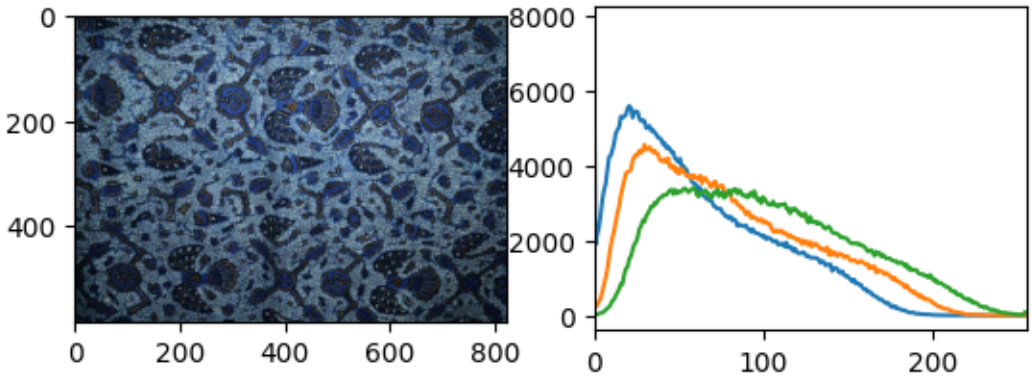

0
50
100
150
200
0
100
200
300
600
400
200
0
0
100
200

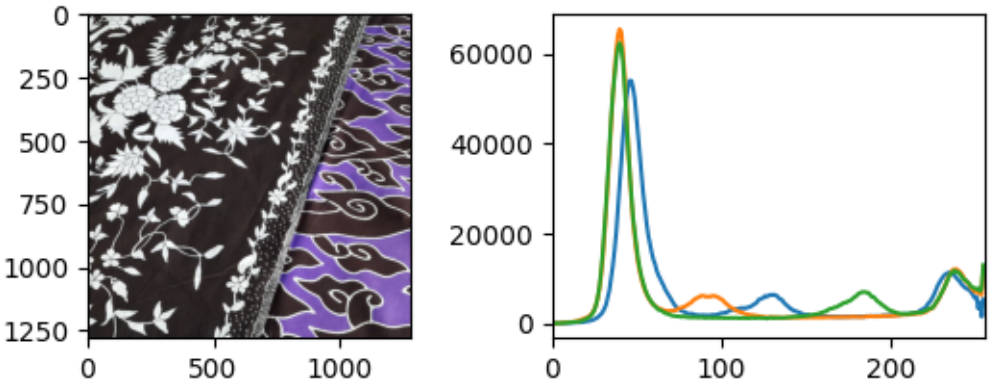

0
200
400
600
800
1000
0
500
10000
8000
6000
4000
2000
0
0
100
200

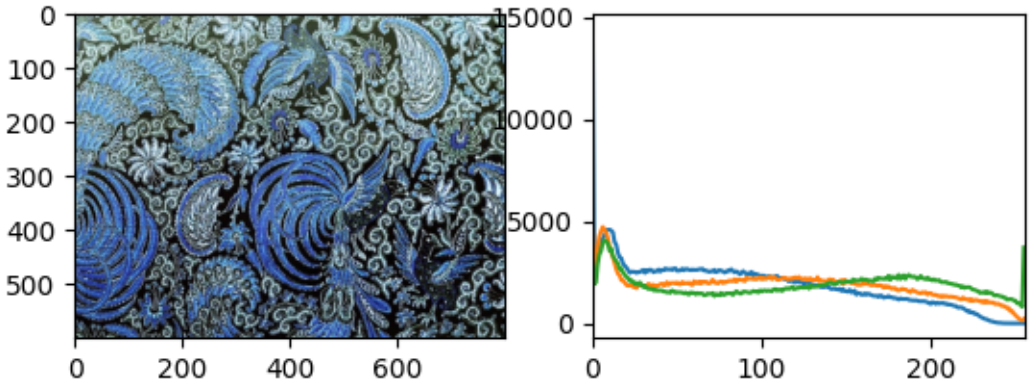

0
500
1000
1500
2000
0
1000
80000
60000
40000
20000
0
0
100
200

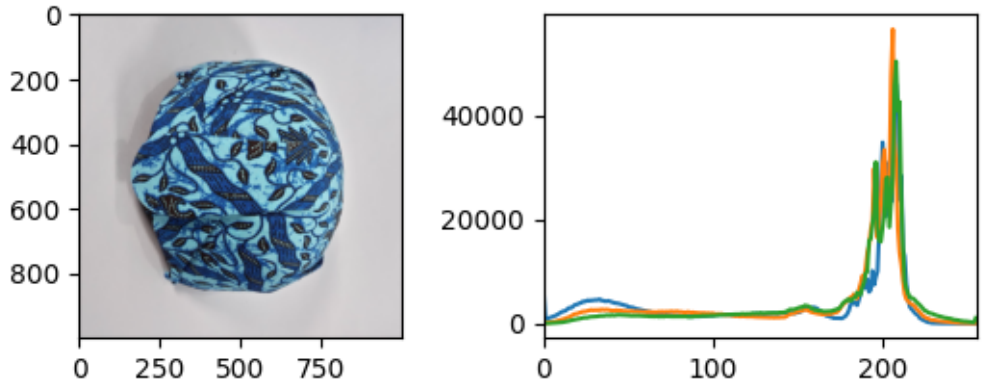

0
200
400
600
0
200
400
600
800
25000
20000
15000
10000
5000
0
0
100
200

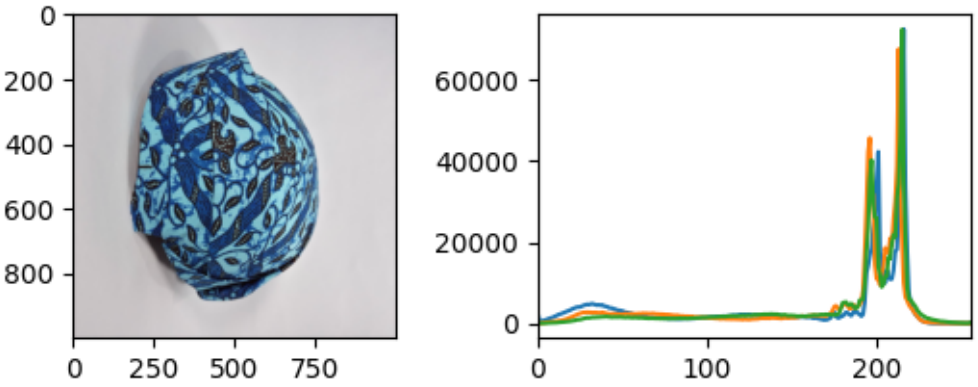

0
500
1000
1500
0
500
1000
25000
20000
15000
10000
5000
0
0
100
200

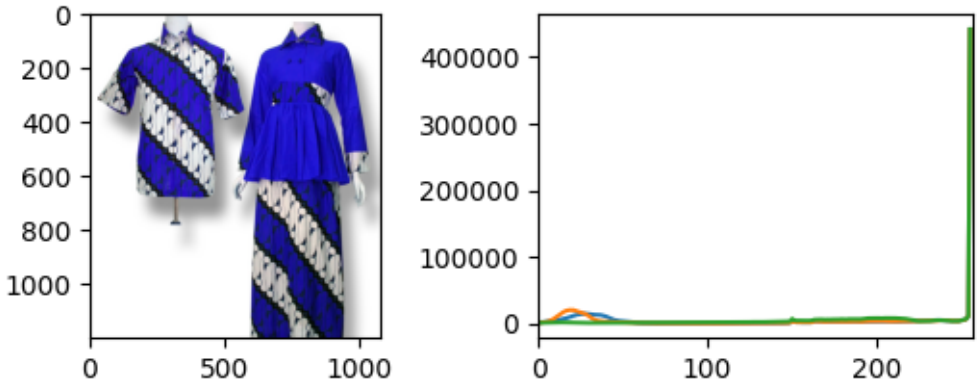

0
200
400
0
200
400
600
800
8000
6000
4000
2000
0
0
100
200

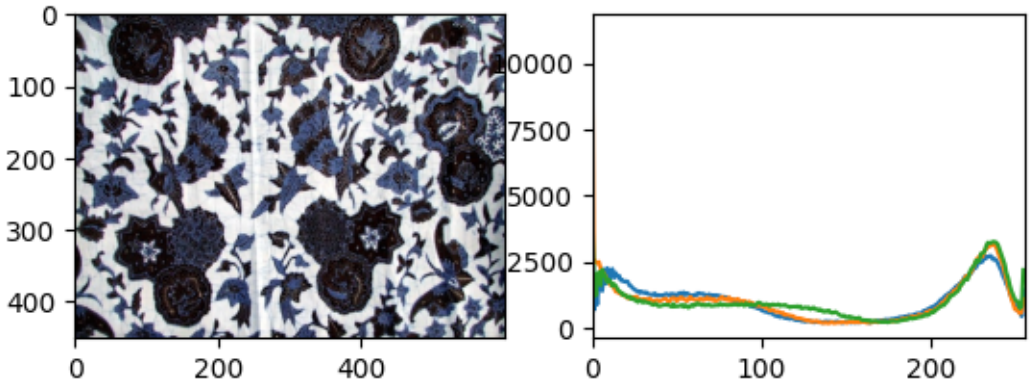

0
250
500
750
1000
1250
0
500
1000
60000
40000
20000
0
0
100
200

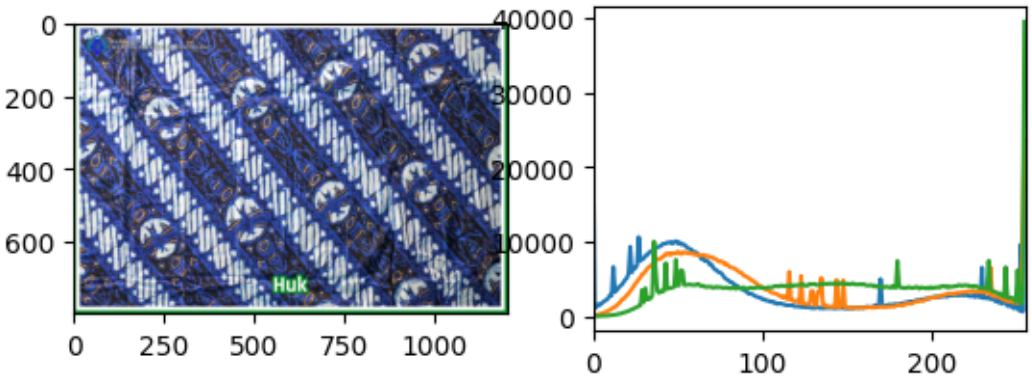

0
100
200
300
400
500
0
200
400
600
15000
10000
5000
0
0
100
200

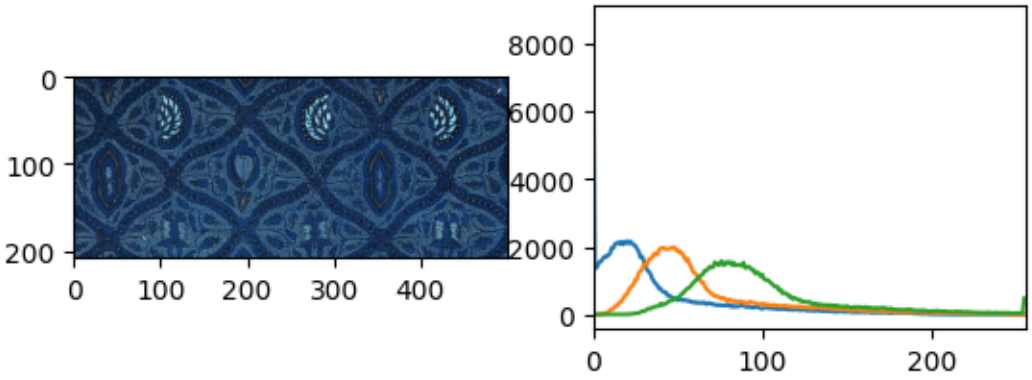

0
200
400
600
800
0
250
500
750
40000
20000
0
0
100
200

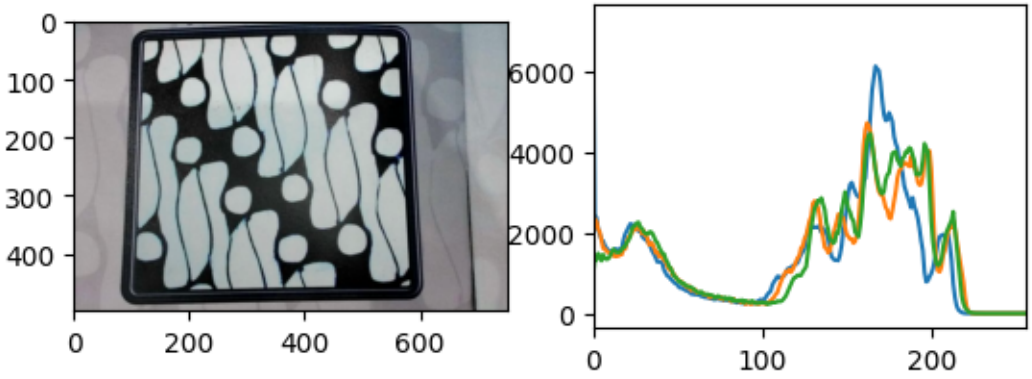

0
200
400
600
800
0
250
500
750
60000
40000
20000
0
0
100
200

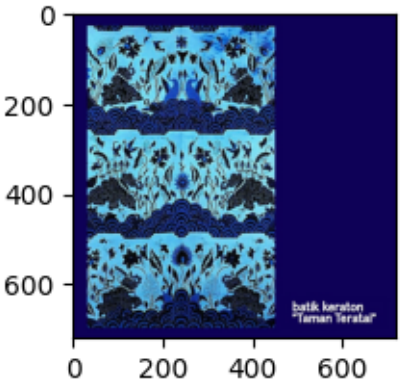

0
200
400
600
800
1000
0
500
1000
400000
300000
200000
100000
0
0
100
200

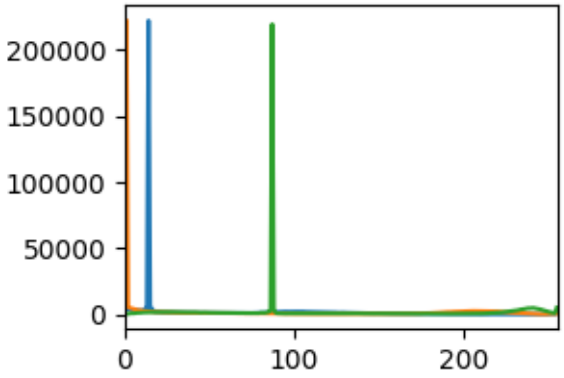

0
100
200
300
400
0
200
400
0000
7500
5000
2500
0
0
100
200

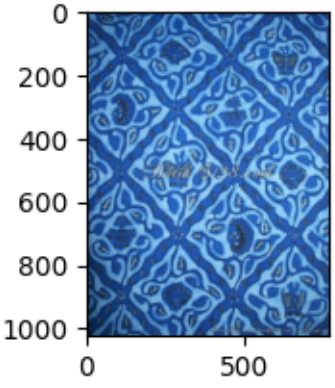

0
200
400
600
0
250
500
750
1000
Huk
40000
30000
20000
10000
0
0
100
200

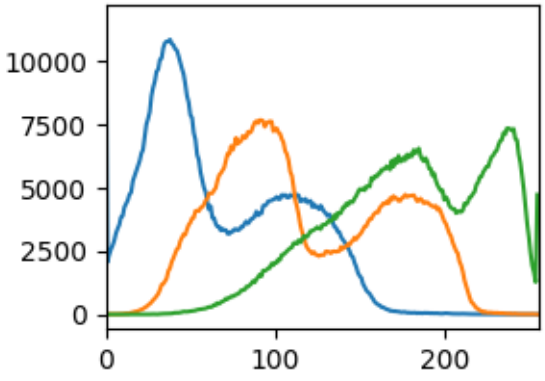

0
100
200
0
100
200
300
400
8000
6000
4000
2000
0
0
100
200

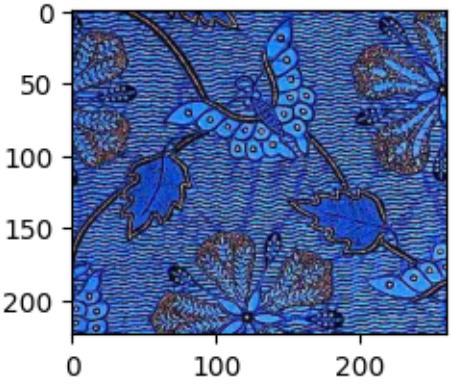

0
100
200
300
400
0
200
400
600
6000
4000
2000
0
0
100
200

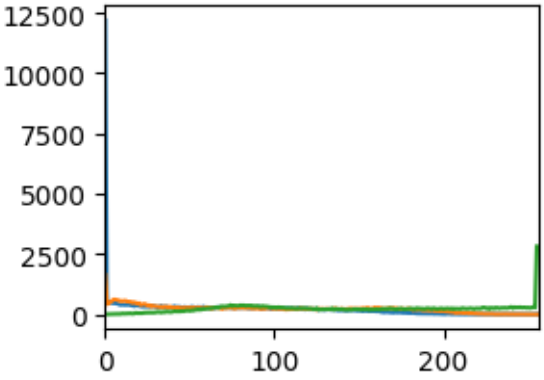

batik keraton
"Taman Teratai"
0
200
400
600
0
200
400
600

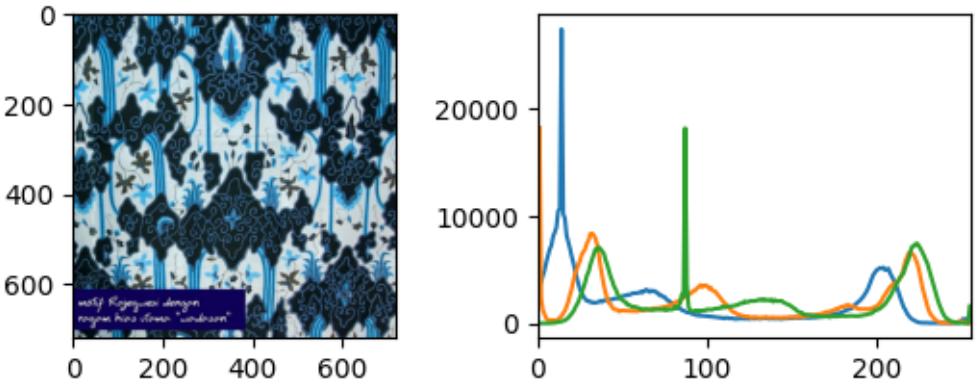

200000
150000
100000
50000
0
0
100
200

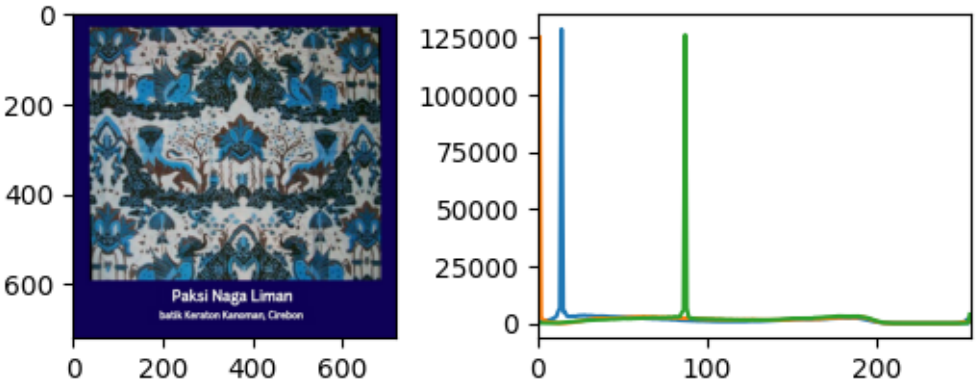

0
200
400
600
800
1000
0
500

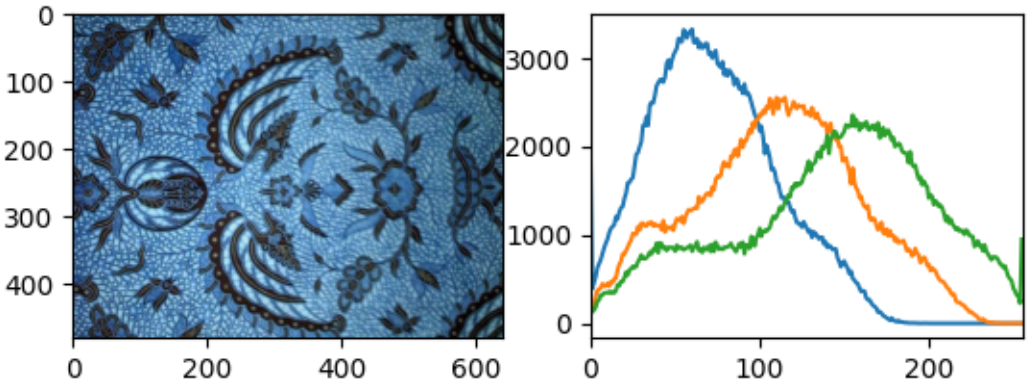

10000
7500
5000
2500
0
0
100
200

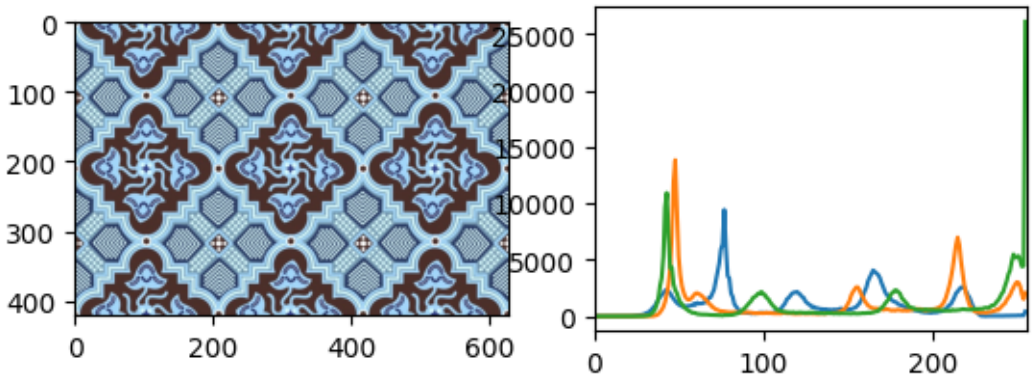

0
50
100
150
200
0
100
200

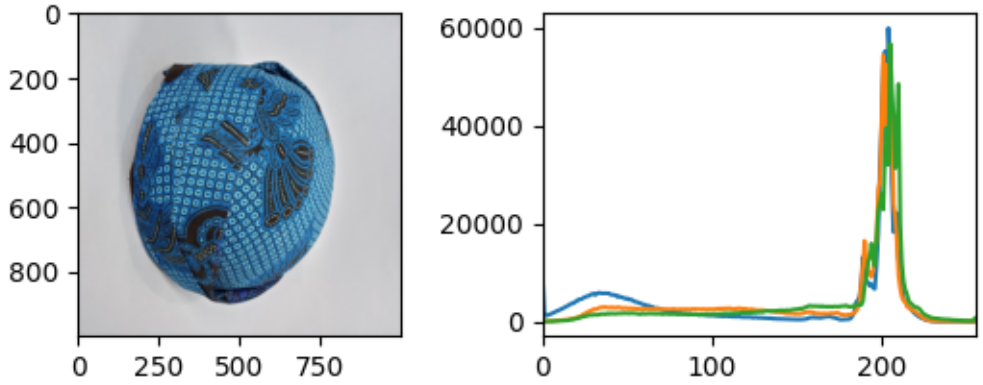

12500
10000
7500
5000
2500
0
0
100
200

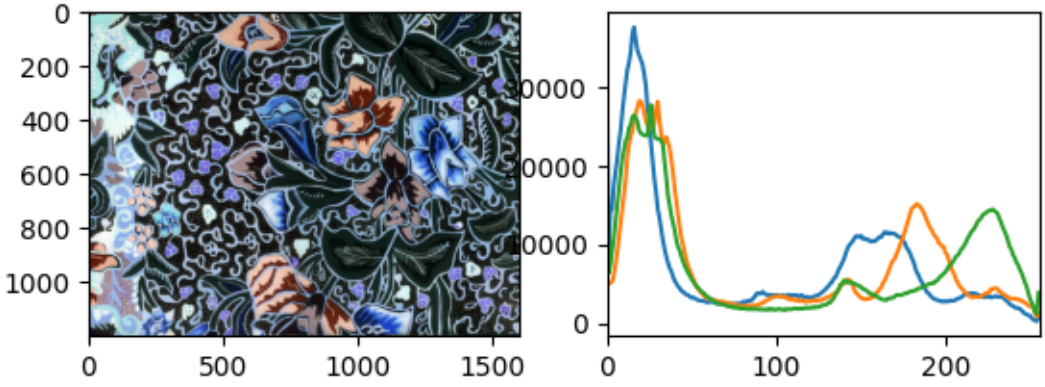

0
200
400
600
0
200
400
600
20000
10000
0
0
100
200

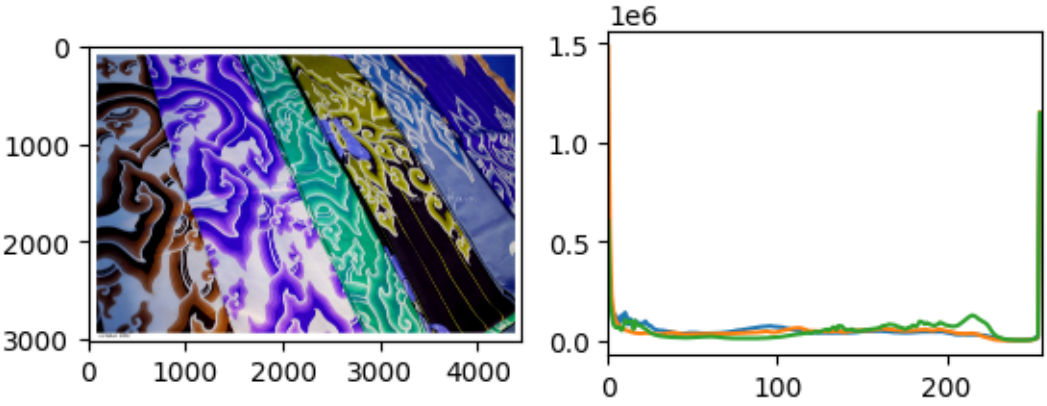

Paksi Naga Liman
batik Keraton Kanoman, Cirebon
0
200
400
600
0
200
400
600
125000
100000
75000
50000
25000
0
0
100
200

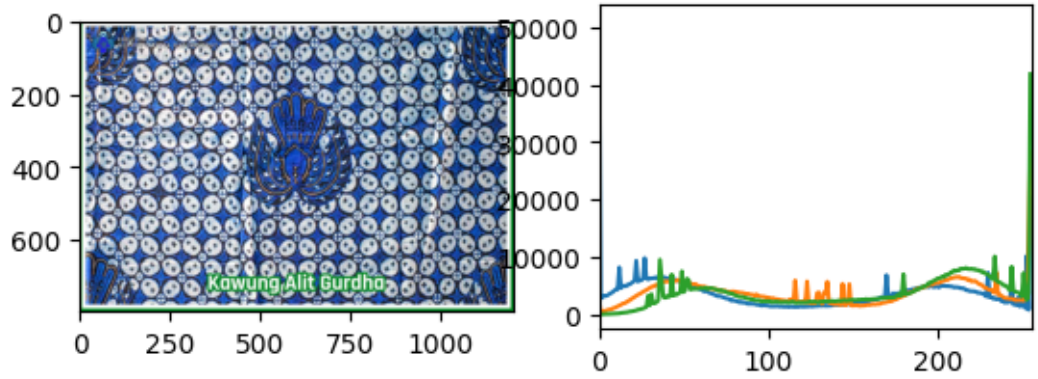

0
100
200
300
400
0
200
400
600
3000
2000
1000
0
0
100
200

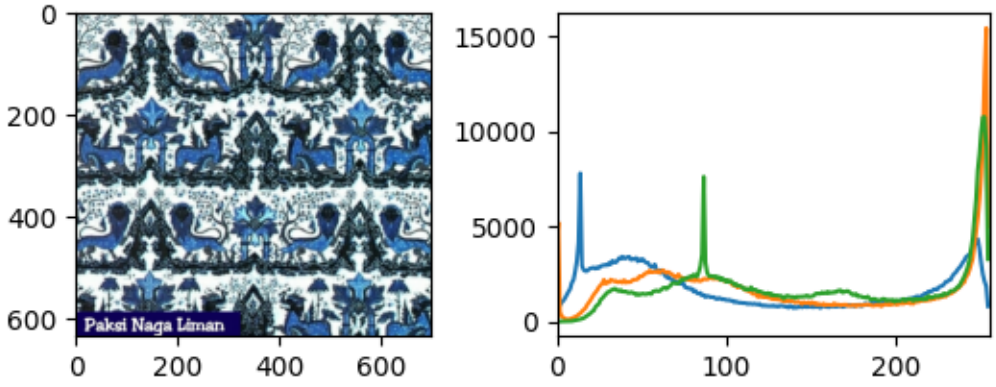

0
100
200
300
400
0
200
400
600
25000
20000
15000
10000
5000
0
0
100
200


0
200
400
600
800
0
250
500
750
60000
40000
20000
0
0
100
200

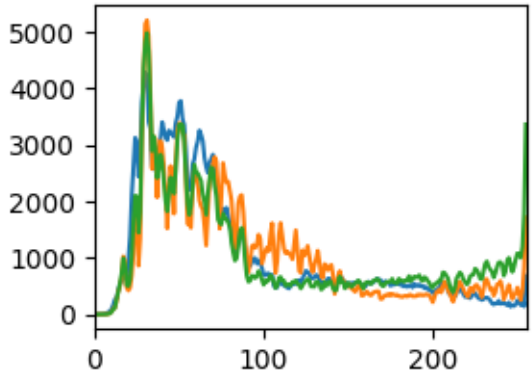

0
200
400
600
800
1000
0
500
1000
1500
0
100
200

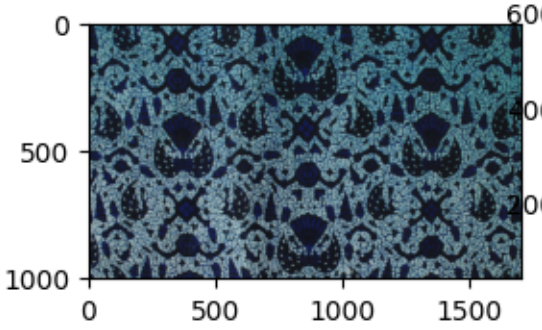

0
1000
2000
3000
0
1000
2000
3000
4000
1e6
1.5
1.0
0.5
0.0
0
100
200

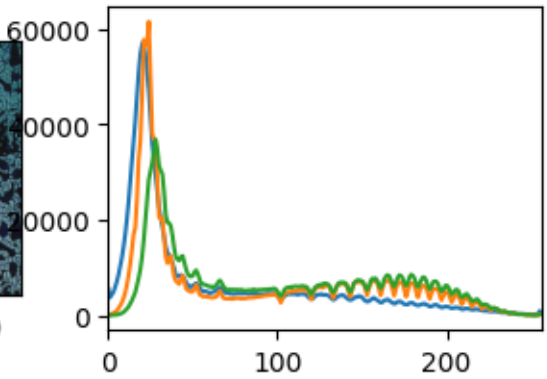

0
200
400
600
0
250
500
750
1000
Kawung Alit Gurdha
50000
40000
30000
20000
10000
0
0
100
200

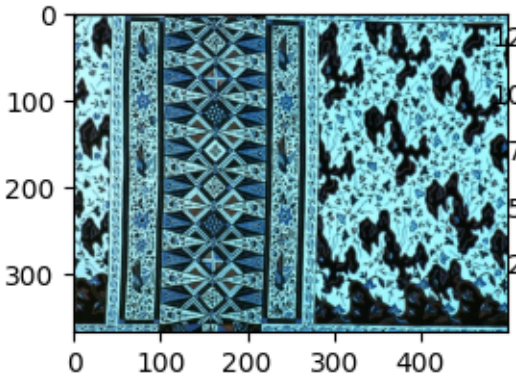

0
200
400
600
0
200
400
600
Paksi Naga Liman
15000
10000
5000
0
0
100
200

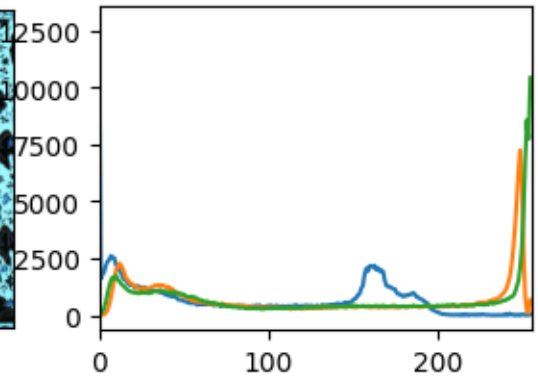

0
100
200
300
400
500
0
200
400


5000
4000
3000
2000
1000
0
0
100
200


0
500
1000
0
500
1000
1500


60000
40000
20000
0
0
100
200


0
100
200
300
0
100
200
300
400


12500
10000
7500
5000
2500
0
0
100
200

3.1.7 Batik Ciamis

[ ]:
```
batikciamis_dir = '/content/gdrive/MyDrive/Dataset2/Dataset2B' # image folder

# get the list of jpegs from sub image class folders
batikciamis_imgs = [fn for fn in os.listdir(f'{batikciamis_dir}/batik-ciamis')␣
 ↪if fn.endswith('.jpg')]

# randomly select 3 of each
select_norm = np.random.choice(batikciamis_imgs, 3, replace = False)

# plotting 2 x 3 image matrix
fig = plt.figure(figsize = (8,6))
for i in range(3):
    if i < 3:
        fp = f'{batikciamis_dir}/batik-ciamis/{select_norm[i]}'
        label = 'batik ciamis'
    ax = fig.add_subplot(2, 3, i+1)

    # to plot without rescaling, remove target_size
    fn = image.load_img(fp, target_size = (100,100), color_mode="rgb")
    plt.imshow(fn, cmap='Greys_r')
    plt.title(label)
    plt.axis('off')
plt.show()

# also check the number of files here
len(batikciamis_imgs)
```

[ ]:
```
50
```

3 gambar diatas merupakan sample dari data btik-ciamis.

```
[ ]: # making n X m matrix
def img2np(path, list_of_filename, size = (64, 64)):
    # iterating through each file
    for fn in list_of_filename:
        fp = path + fn
        current_image = image.load_img(fp, target_size = size,
                                       color_mode = 'grayscale')
        # covert image to a matrix
        img_ts = image.img_to_array(current_image)
        # turn that into a vector / 1D array
        img_ts = [img_ts.ravel()]
        try:
            # concatenate different images
            full_mat = np.concatenate((full_mat, img_ts))
        except UnboundLocalError:
            # if not assigned yet, assign one
            full_mat = img_ts
    return full_mat

# run it on our folders
batikciamis_images = img2np(f'{batikciamis_dir}/batik-ciamis/',␣
 ↪batikciamis_imgs)
```

```
[ ]: def find_mean_img(full_mat, title, size = (64, 64)):
    # calculate the average
    mean_img = np.mean(full_mat, axis = 0)
    # reshape it back to a matrix
    mean_img = mean_img.reshape(size)
    plt.imshow(mean_img, vmin=0, vmax=255, cmap='Greys_r')
    plt.title(f'Average {title}')
    plt.axis('off')
    plt.show()
    return mean_img
```

Fungsi di atas akan melakukan looping untuk setiap file gambar dan mengubahnya menjadi matriks (n,m). n adalah jumlh observasi dan m adalah jumlah pixel. Langkah selanjutnya, matriks (n,m) akan diubah menjadi sebuah array satu dimensi. array atau vektor-vektor tersebut akan digabungkan menjadi matrix yang besar. Fungsi diatas akan dijalankan pada kelima folder data batik untuk menghasilkan matrix gambar dari setiap folder tersebut

```
[ ]: from sklearn.decomposition import PCA
from math import ceil

def eigenimages(full_mat, title, n_comp = 0.7, size = (64, 64)):
    # fit PCA to describe n_comp * variability in the class
    pca = PCA(n_components = n_comp, whiten = True)
    pca.fit(full_mat)
```

```
    print('Number of PC: ', pca.n_components_)
    return pca

def plot_pca(pca, size = (64, 64)):
    # plot eigenimages in a grid
    n = pca.n_components_
    fig = plt.figure(figsize=(8, 8))
    r = int(n**.5)
    c = ceil(n/ r)
    for i in range(n):
        ax = fig.add_subplot(r, c, i + 1, xticks = [], yticks = [])
        ax.imshow(pca.components_[i].reshape(size),
                  cmap='gist_ncar')
    plt.axis('off')
    plt.show()
```

3.1.8 Batik Garutan

[ ]:
```
batikgarutan_dir = '/content/gdrive/MyDrive/Dataset2/Dataset2B' # image folder

# get the list of jpegs from sub image class folders
batikgarutan_imgs = [fn for fn in os.listdir(f'{batikgarutan_dir}/
 ↪batik-garutan') if fn.endswith('.jpg')]

# randomly select 3 of each
select_norm = np.random.choice(batikgarutan_imgs, 3, replace = False)

# plotting 2 x 3 image matrix
fig = plt.figure(figsize = (8,6))
for i in range(3):
    if i < 3:
        fp = f'{batikgarutan_dir}/batik-garutan/{select_norm[i]}'
        label = 'batik garutan'
    ax = fig.add_subplot(2, 3, i+1)

    # to plot without rescaling, remove target_size
    fn = image.load_img(fp, target_size = (100,100), color_mode="rgb")
    plt.imshow(fn, cmap='Greys_r')
    plt.title(label)
    plt.axis('off')
plt.show()

# also check the number of files here
len(batikgarutan_imgs)
```

[ ]:
```
50
```

[ ]:
```
# run it on our folders
batikgarutan_images = img2np(f'{batikgarutan_dir}/batik-garutan/',␣
 ↪batikgarutan_imgs)
```

3.1.9 Batik Gentongan

[ ]:
```
batikgentongan_dir = '/content/gdrive/MyDrive/Dataset2/Dataset2B' # image folder

# get the list of jpegs from sub image class folders
batikgentongan_imgs = [fn for fn in os.listdir(f'{batikgentongan_dir}/
 ↪batik-gentongan') if fn.endswith('.jpg')]

# randomly select 3 of each
select_norm = np.random.choice(batikgentongan_imgs, 3, replace = False)

# plotting 2 x 3 image matrix
fig = plt.figure(figsize = (8,6))
for i in range(3):
    if i < 3:
        fp = f'{batikgentongan_dir}/batik-gentongan/{select_norm[i]}'
        label = 'batik gentongan'
    ax = fig.add_subplot(2, 3, i+1)

    # to plot without rescaling, remove target_size
    fn = image.load_img(fp, target_size = (100,100), color_mode="rgb")
    plt.imshow(fn, cmap='Greys_r')
    plt.title(label)
    plt.axis('off')
plt.show()

# also check the number of files here
```

```
len(batikgentongan_imgs)
```

[ ]: 50

[ ]:
```
# run it on our folders
batikgentongan_images = img2np(f'{batikgentongan_dir}/batik-gentongan/',␣
 ↪batikgentongan_imgs)
```

3.1.10 Batik Kawung

[ ]:
```
batikkawung_dir = '/content/gdrive/MyDrive/Dataset2/Dataset2B' # image folder

# get the list of jpegs from sub image class folders
batikkawung_imgs = [fn for fn in os.listdir(f'{batikkawung_dir}/batik-kawung')␣
 ↪if fn.endswith('.jpg')]

# randomly select 3 of each
select_norm = np.random.choice(batikkawung_imgs, 3, replace = False)

# plotting 2 x 3 image matrix
fig = plt.figure(figsize = (8,6))
for i in range(3):
    if i < 3:
        fp = f'{batikkawung_dir}/batik-kawung/{select_norm[i]}'
        label = 'batik kawung'
    ax = fig.add_subplot(2, 3, i+1)

    # to plot without rescaling, remove target_size
    fn = image.load_img(fp, target_size = (100,100), color_mode="rgb")
    plt.imshow(fn, cmap='Greys_r')
    plt.title(label)
    plt.axis('off')
plt.show()
```

```
# also check the number of files here
len(batikkawung_imgs)
```

[ ]: 45

[ ]:
```
# run it on our folders
batikkawung_images = img2np(f'{batikkawung_dir}/batik-kawung/',␣
 ↪batikkawung_imgs)
```

3.1.11 Batik Keraton

[ ]:
```
batikkeraton_dir = '/content/gdrive/MyDrive/Dataset2/Dataset2B' # image folder

# get the list of jpegs from sub image class folders
batikkeraton_imgs = [fn for fn in os.listdir(f'{batikkeraton_dir}/
 ↪batik-keraton') if fn.endswith('.jpg')]

# randomly select 3 of each
select_norm = np.random.choice(batikkeraton_imgs, 3, replace = False)

# plotting 2 x 3 image matrix
fig = plt.figure(figsize = (8,6))
for i in range(3):
    if i < 3:
        fp = f'{batikkeraton_dir}/batik-keraton/{select_norm[i]}'
        label = 'batik keraton'
    ax = fig.add_subplot(2, 3, i+1)

    # to plot without rescaling, remove target_size
    fn = image.load_img(fp, target_size = (100,100), color_mode="rgb")
    plt.imshow(fn, cmap='Greys_r')
    plt.title(label)
```

```
    plt.axis('off')
plt.show()

# also check the number of files here
len(batikkeraton_imgs)
```

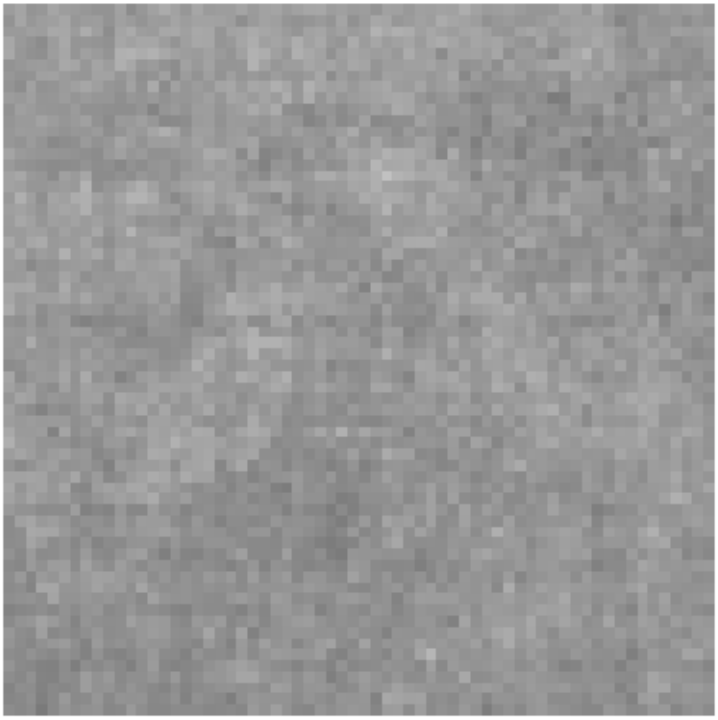
[ ]:
```
50
```

[ ]:
```
# run it on our folders
batikkeraton_images = img2np(f'{batikkeraton_dir}/batik-keraton/',␣
 ↪batikkeraton_imgs)
```

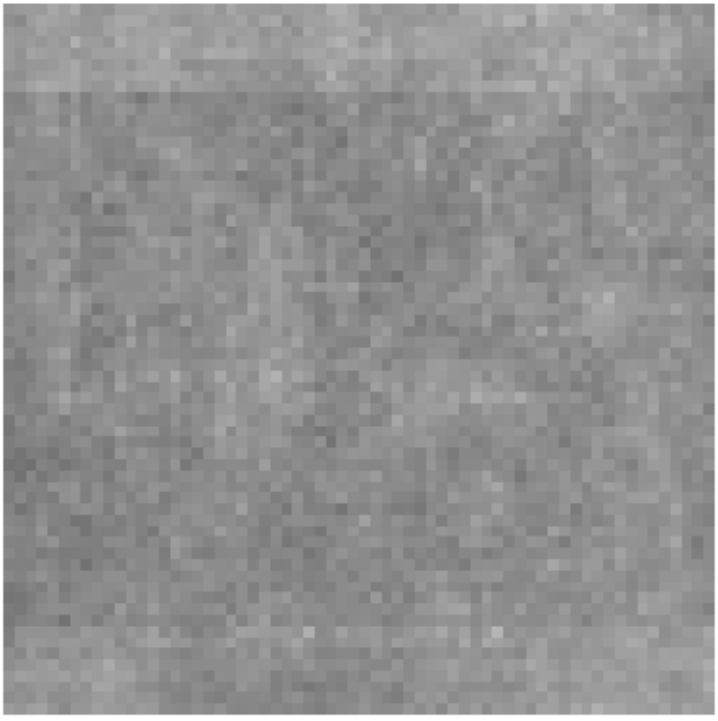
3.1.12 Komparasi per kelas

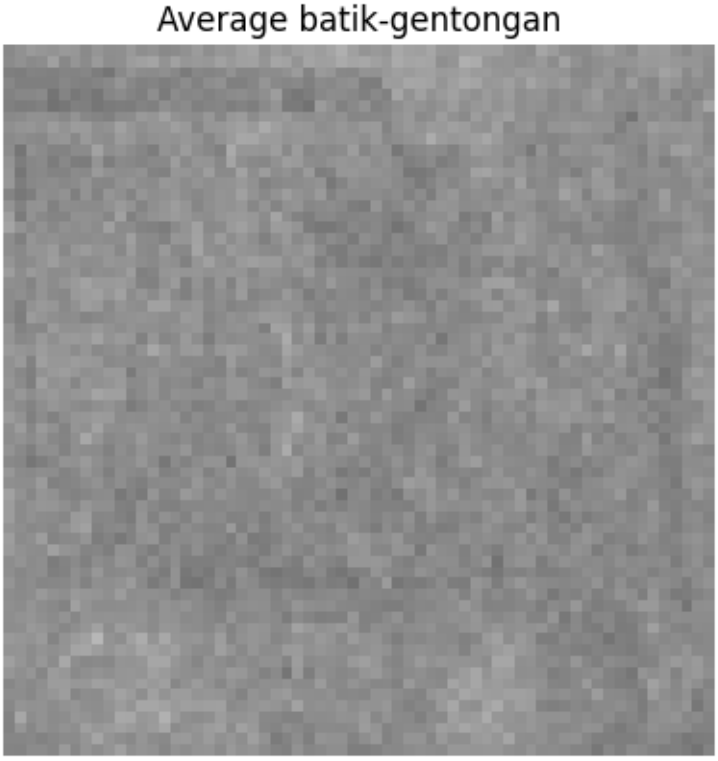
[ ]:
```
batikciamis_mean = find_mean_img(batikciamis_images, 'batik-ciamis')
batikgarutan_mean = find_mean_img(batikgarutan_images, 'batik-garutan')
batikgentongan_mean = find_mean_img(batikgentongan_images, 'batik-gentongan')
batikkawung_mean = find_mean_img(batikkawung_images, 'batik-kawung')
batikkeraton_mean = find_mean_img(batikkeraton_images, 'batik-keraton')
```

Average batik-ciamis

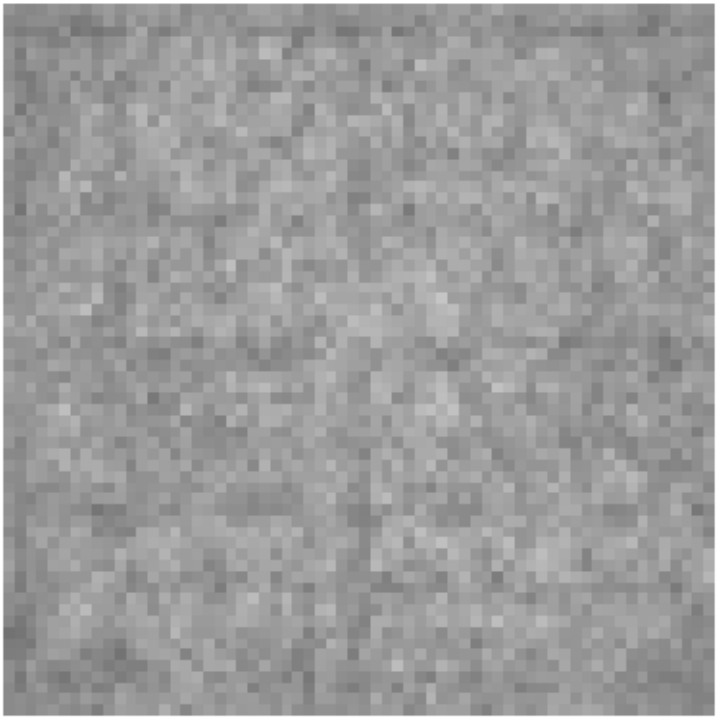


Average batik-garutan

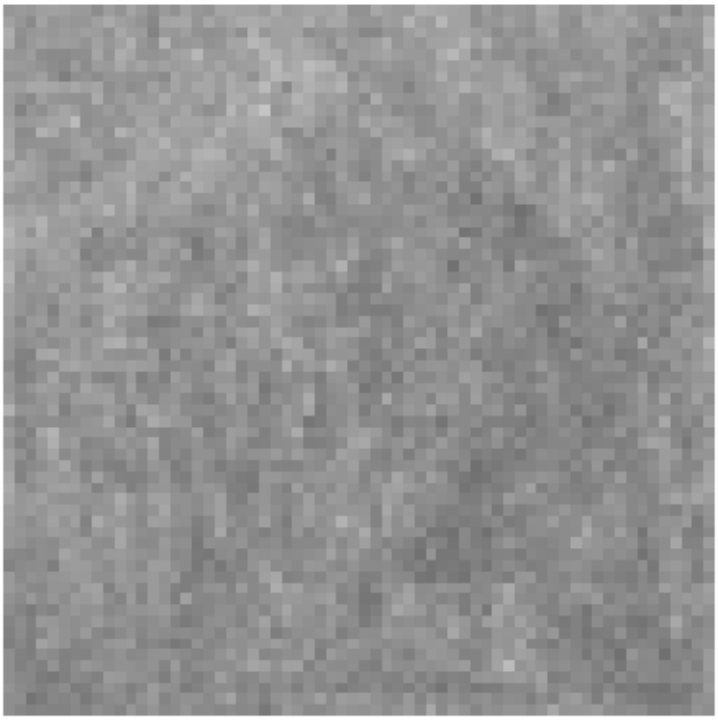



Average batik-gentongan

Average batik-kawung



Average batik-keraton

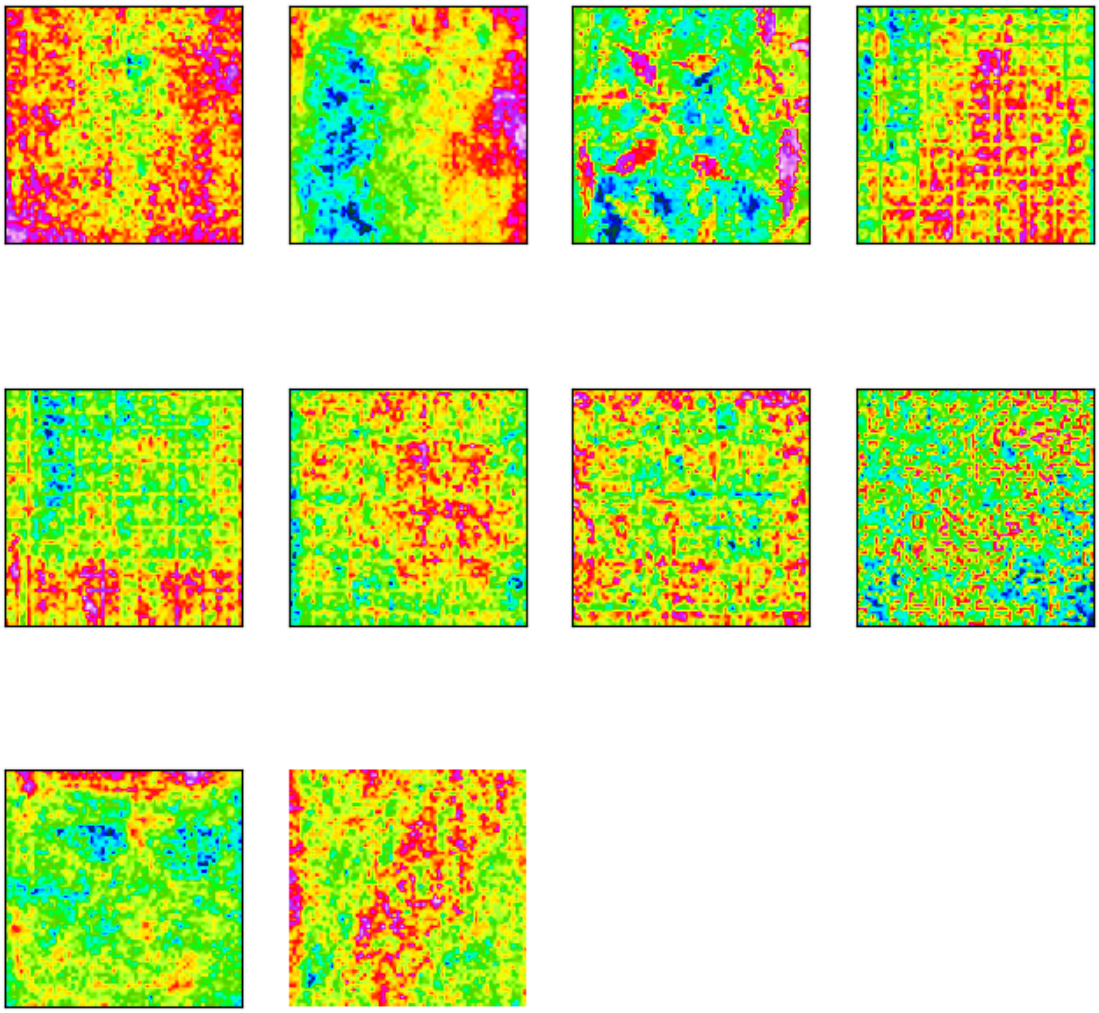


Gambar diatas merupakan average image dari masing-masing kelas batik. Average image digunakan untuk memberikan representasi visual dari masing-masing kelas dalam bentuk suatu gambar. Dalam kasus ini, kita dapat melihat karakteristik khas setiap gambar berupa warna, tekstur, bentuk, dan sebagainya. Karakteristik ini dapat bermanfaat untuk mengenai pola khas suatu kelas. Output diatas merupaka rata-rata gambar dari masing-masing kelas, tidak terlalu jelas perbedaannya secara kasat mata. Hanya batik kawung saja yang cukup memiliki perbedaan pola dari average image kelas lainnya.

Ada beberapa kmungkinan yang menyebabkan hal ini terjadi: 1. Ukuran dataset yang terlalu kecil sehingga perbedaan antar kelas kurang begitu terlihat secara signifikan pada hasil average image ini. 2. Fitur yang kurang representatif untuk dapat membedakan antar kelas. 3. Kualitas gambar yang buruk, kualitas gambar yang buruk mungkin menyebabkan sistem tidak dapat menunjukan perbedaan yang signifikan antar kelas.

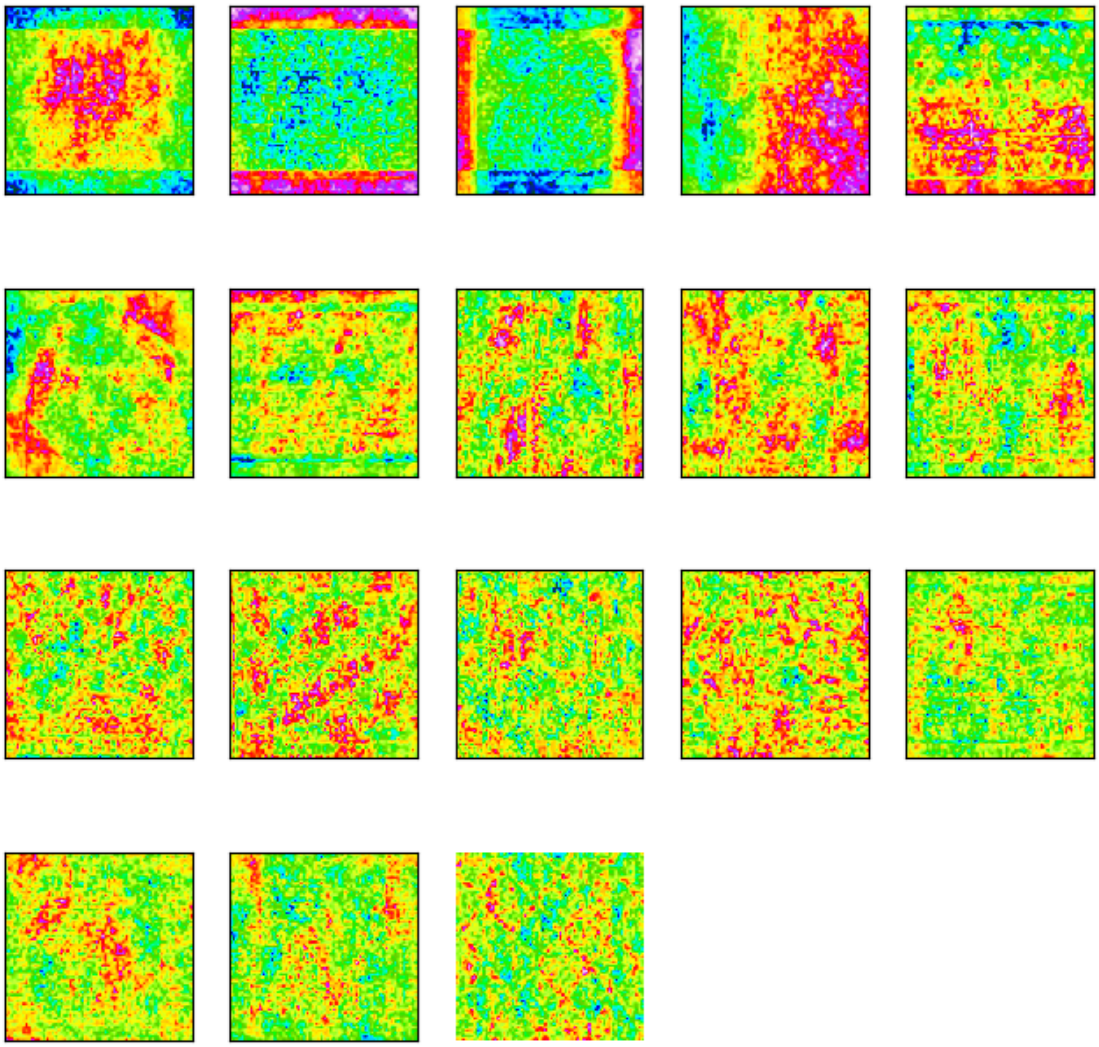
[ ]:
```
print("Batik Ciamis\n")
plot_pca(eigenimages(batikciamis_images, 'Batik Ciamis'))
print("Batik Garutan\n")
plot_pca(eigenimages(batikgarutan_images, 'Batik garutan'))
print("Batik Gentongan\n")
plot_pca(eigenimages(batikgentongan_images, 'Batik gentongan'))
print("Batik Kawung\n")
plot_pca(eigenimages(batikkawung_images, 'Batik kawung'))
print("Batik Keraton\n")
plot_pca(eigenimages(batikkeraton_images, 'Batik keraton'))
```

```
Batik Ciamis

Number of PC:  10
```



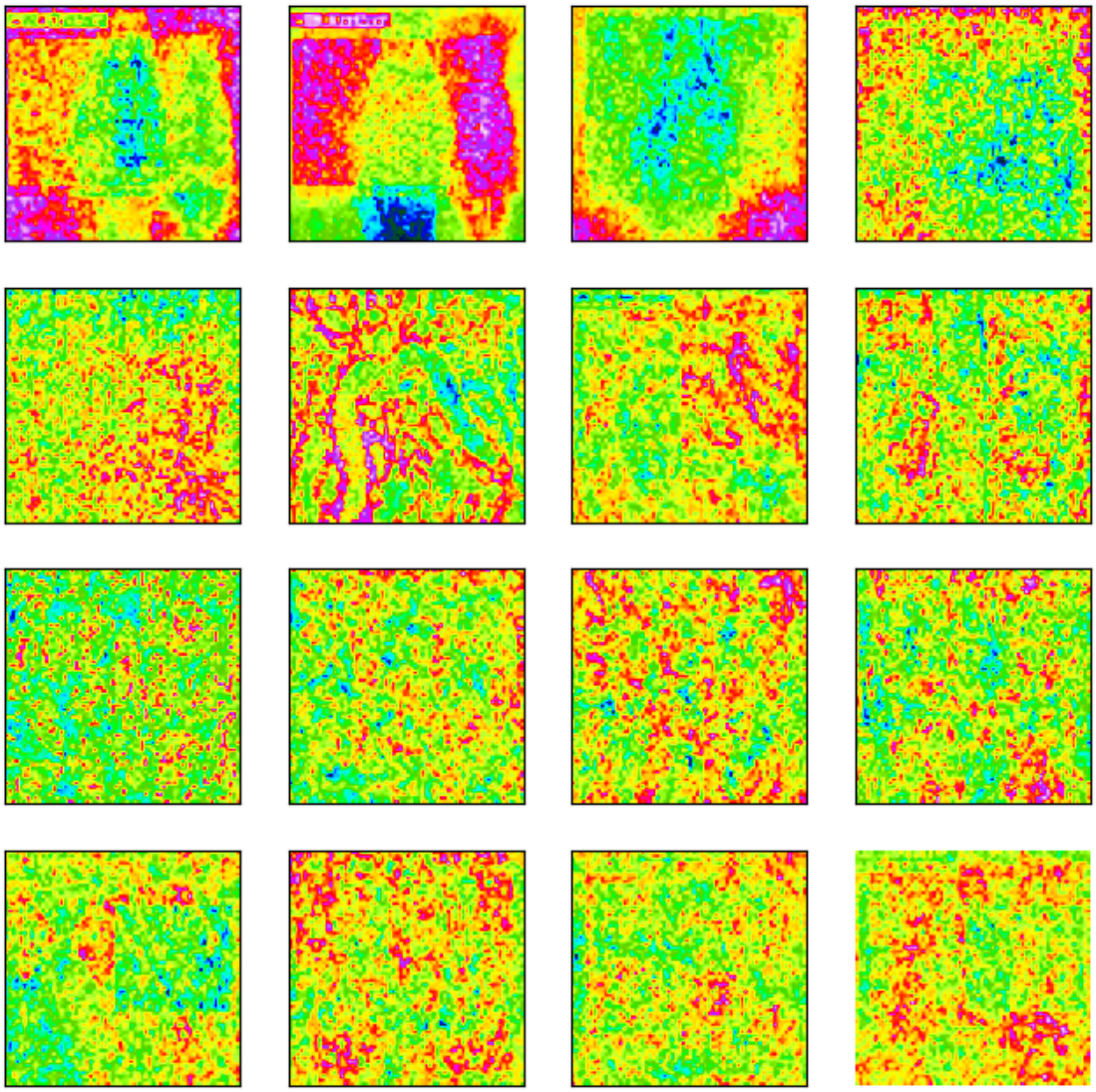
Batik Garutan

Number of PC: 18



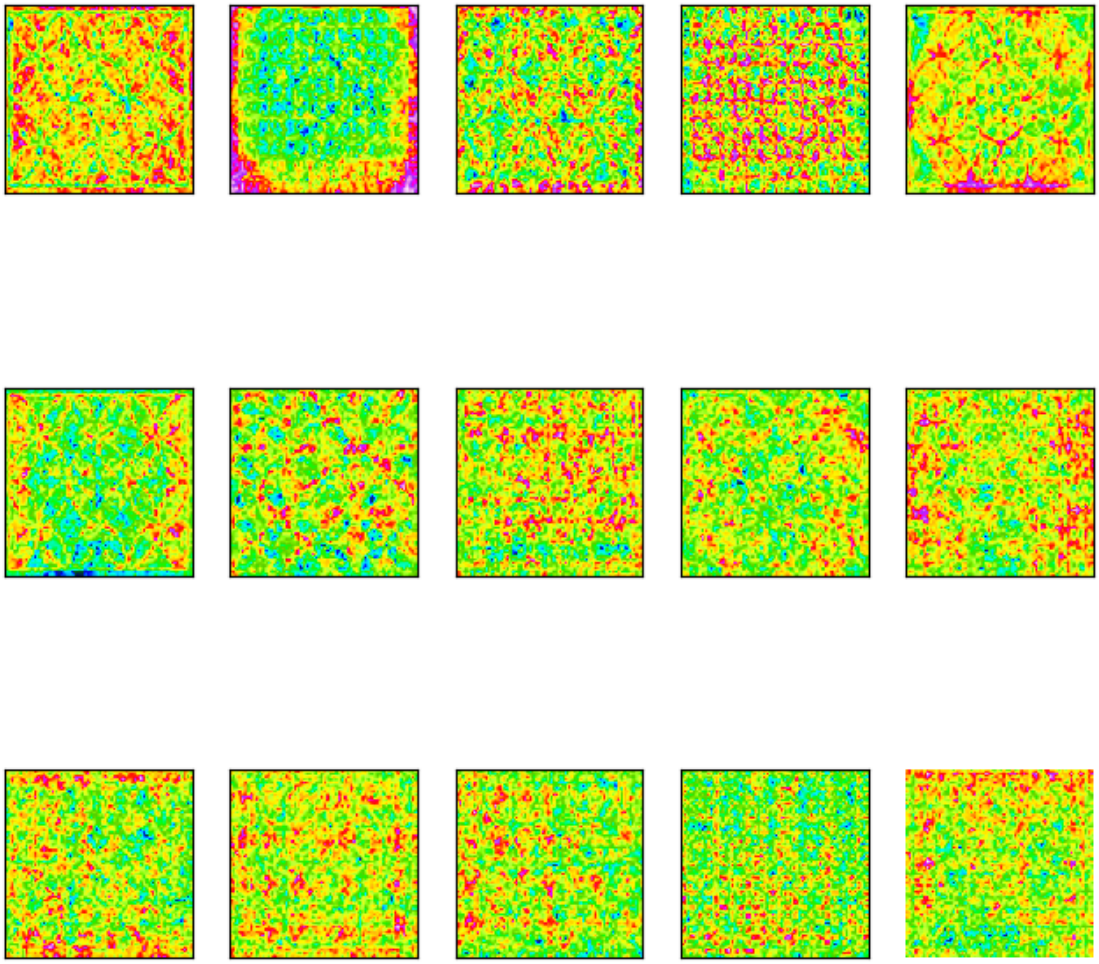
Batik Gentongan

Number of PC: 16



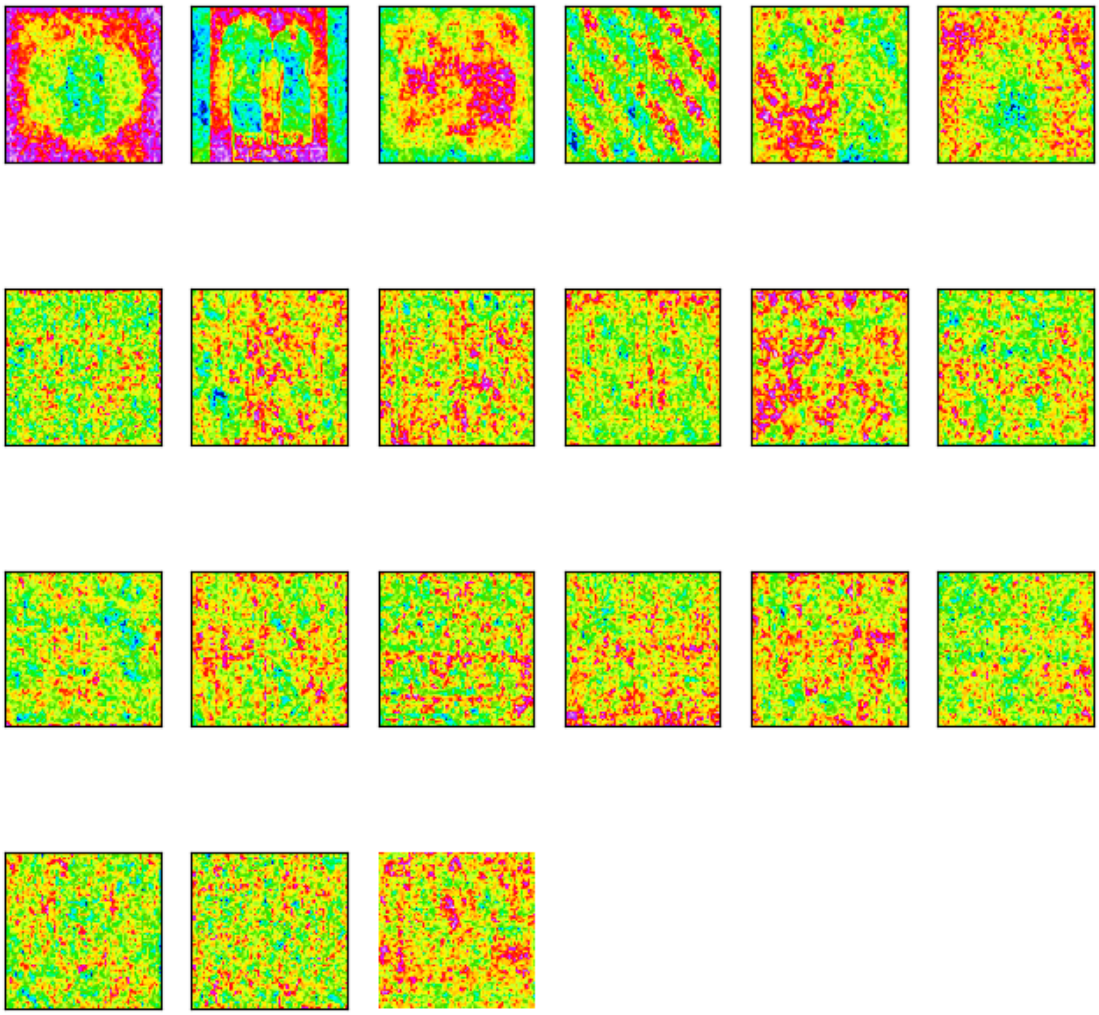
Batik Kawung

Number of PC: 15



Batik Keraton

Number of PC: 21



Output diatas merupakan Eugenimages dari masing-masing kelas batik. Eigenimages dapat digunakan untuk memvisualisaskan komponen yang paling merepresentasikan setaip kelas. Komponen yang dimaksud adalah komponen yang paling mampu membedakan kelas tersebut dari kelas lainnya. Output diats merupakan komponen-komponen utama yang menjelaskan 70% dari variabilitas untuk setiap kelas. Kelas dari batik kawung nampaknya memmiliki perbedaan yang cukup jelas ketimbang kelas batik lainnya, yakni bentuknya yang menyerupai buang kawung yang ditata secara geometris.

3.2 Mengimport Data dari Google Drive

[ ]:
```
data_path = '/content/gdrive/MyDrive/Dataset2B/'

class_names = sorted(os.listdir(data_path))
num_classes = len(class_names)
```

```
img_size = (64, 64, 3)

print('classes: ', class_names)
```

```
classes:  ['batik-ciamis', 'batik-garutan', 'batik-gentongan', 'batik-kawung',
'batik-keraton']
```

Kode diatas digunakan untuk mengambil kelima kelas dalam folder Dataset2B

[ ]:
```
### untuk menyimpan data class dan path image
labels = []
images = []
J=1
for cl in class_names:
    print(cl, end=' -> ')
    for img in os.listdir(data_path + cl ):
        print(data_path + cl+img)
        label = np.zeros(num_classes)
        label[class_names.index(cl)] = 1
        labels.append(label)

        image = np.asarray(cv2.resize(cv2.imread(data_path + cl + '/'+ img, cv2.
 ↪IMREAD_COLOR), img_size[0:2]))
        images.append(image)
        J=J+1
    print(cl + 'done')
    print("\n")
```

```
batik-ciamis -> /content/gdrive/MyDrive/Dataset2B/batik-ciamis30.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis46.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis22.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis40.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis44.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis43.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis1.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis14.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis28.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis20.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis8.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis2.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis25.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis13.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis15.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis18.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis38.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis45.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis36.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis19.jpg
```

```
/content/gdrive/MyDrive/Dataset2B/batik-ciamis49.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis17.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis3.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis6.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis5.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis21.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis16.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis7.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis37.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis39.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis4.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis26.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis42.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis32.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis23.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis47.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis48.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis11.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis9.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis41.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis10.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis27.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis29.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis33.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis35.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis31.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis24.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis34.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis12.jpg
/content/gdrive/MyDrive/Dataset2B/batik-ciamis50.jpg
batik-ciamisdone


batik-garutan -> /content/gdrive/MyDrive/Dataset2B/batik-garutan30.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan18.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan2.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan12.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan4.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan35.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan25.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan3.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan32.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan16.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan20.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan38.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan1.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan34.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan10.jpg
```

```
/content/gdrive/MyDrive/Dataset2B/batik-garutan39.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan21.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan37.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan24.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan19.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan13.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan36.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan31.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan29.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan28.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan23.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan22.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan15.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan33.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan17.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan27.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan11.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan26.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan14.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan50.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan47.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan9.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan40.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan8.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan49.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan43.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan6.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan7.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan46.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan41.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan42.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan5.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan44.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan45.jpg
/content/gdrive/MyDrive/Dataset2B/batik-garutan48.jpg
batik-garutandone


batik-gentongan -> /content/gdrive/MyDrive/Dataset2B/batik-gentongan47.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan1.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan28.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan9.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan45.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan5.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan34.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan14.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan50.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan27.jpg
```

```
/content/gdrive/MyDrive/Dataset2B/batik-gentongan12.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan30.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan26.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan3.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan39.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan13.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan2.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan11.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan42.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan8.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan20.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan31.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan49.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan33.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan29.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan48.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan38.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan18.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan37.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan32.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan19.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan16.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan44.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan17.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan41.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan7.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan35.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan25.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan40.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan22.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan36.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan23.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan21.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan15.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan46.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan24.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan10.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan4.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan43.jpg
/content/gdrive/MyDrive/Dataset2B/batik-gentongan6.jpg
batik-gentongandone


batik-kawung -> /content/gdrive/MyDrive/Dataset2B/batik-kawung14.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung13.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung12.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung1.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung15.jpg
```

```
/content/gdrive/MyDrive/Dataset2B/batik-kawung10.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung18.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung11.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung17.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung23.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung21.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung19.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung7.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung42.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung6.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung5.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung3.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung28.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung26.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung34.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung33.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung31.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung25.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung43.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung24.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung36.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung20.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung35.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung32.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung4.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung9.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung37.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung45.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung46.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung8.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung47.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung38.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung40.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung41.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung44.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung49.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung30.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung50.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung22.jpg
/content/gdrive/MyDrive/Dataset2B/batik-kawung39.jpg
batik-kawungdone


batik-keraton -> /content/gdrive/MyDrive/Dataset2B/batik-keraton1.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton11.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton10.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton13.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton19.jpg
```

```
/content/gdrive/MyDrive/Dataset2B/batik-keraton12.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton16.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton20.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton17.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton2.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton18.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton14.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton15.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton33.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton49.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton5.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton31.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton48.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton22.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton6.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton32.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton34.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton29.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton4.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton45.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton38.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton43.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton37.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton21.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton41.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton3.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton47.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton35.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton28.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton25.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton27.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton50.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton7.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton9.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton23.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton42.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton30.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton36.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton8.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton39.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton24.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton44.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton40.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton46.jpg
/content/gdrive/MyDrive/Dataset2B/batik-keraton26.jpg
batik-keratondone
```

Dari kode diatas dapat dilihat bahwa kita telah sukses mengimport data dari google drive ke goolge colab

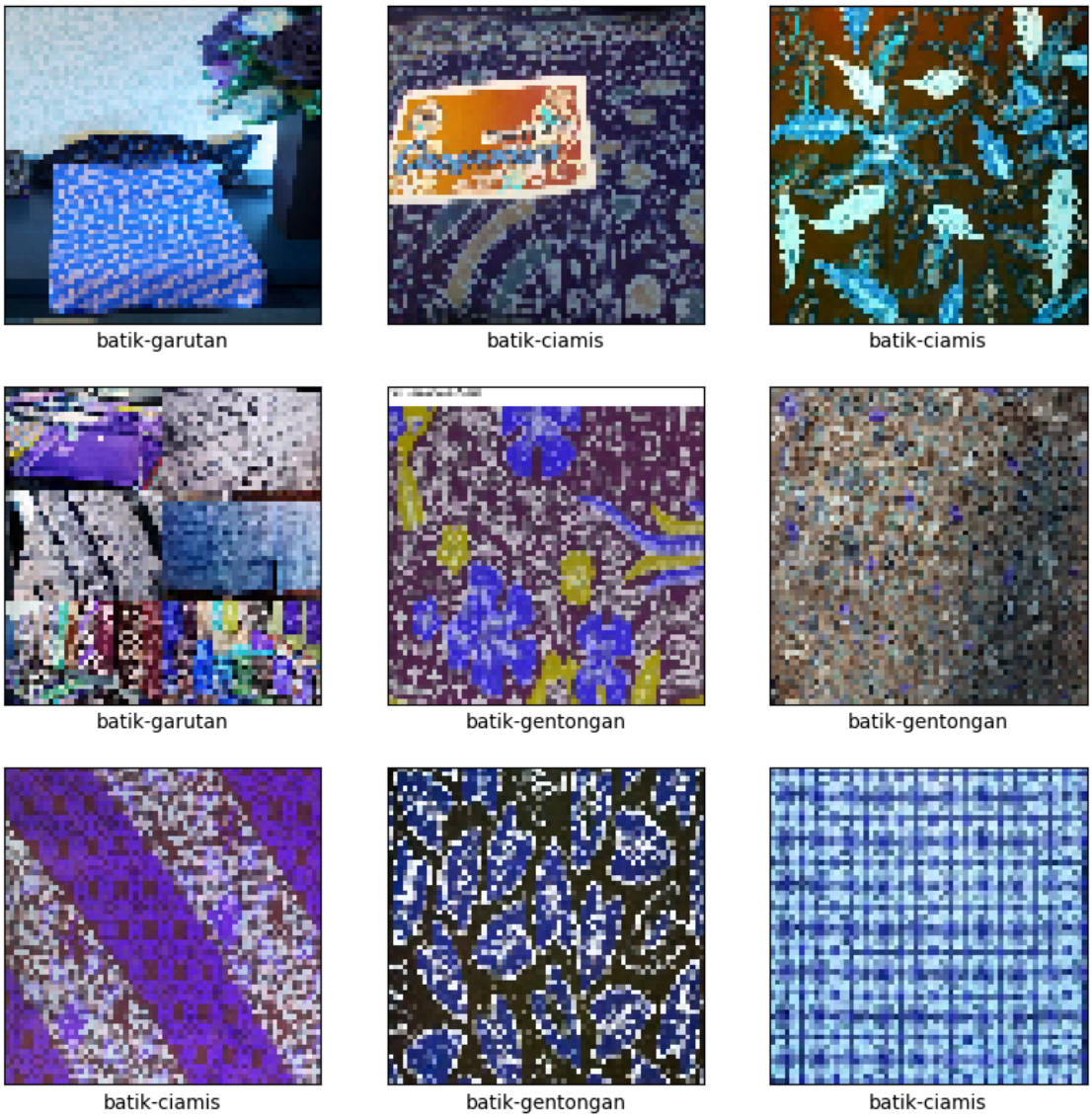
[ ]:
```
labels = np.asarray(labels)
images = np.asarray(images)

print(f'\n\nlabels shape: {labels.shape}')
print(f'images shape: {images.shape}')
```

```
labels shape: (245, 5)
images shape: (245, 64, 64, 3)
```

Meskipun secara kasat mata kita mengetahui bahwa jumlah masing-masing kelas adalah 50 gambar, setelah dicari tahu, batik kawung hanya memiliki 45 gambar. Gambar pada batik kawung yang tidak ada ialah nomor 2, 16, 27, 29, dan 48.

Dari output di atas, menandakan bahwa dataset ini memiliki 5 kelas berbeda dengan jumlah total gambar 245 dimana setiap gambar memiliki resolusi 64**64** 3

[ ]:
```
fig, axs = plt.subplots(3, 3, figsize=(10, 10))

for x in range(3):
    for y in range(3):
        i = randint(0, len(images)*0.8)

        axs[x][y].imshow(images[i])

        axs[x][y].set_xticks([])
        axs[x][y].set_yticks([])
        axs[x][y].set_xlabel(class_names[np.argmax(labels[i])])

plt.show()
```



Output diats merupakan contoh sample random dataset images setelah di resize menjadi 64** 64* *3

4 Nomor 2b

4.1 Membagi data menjadi 80% Training, 10% Validation, dan 10% Testing Set

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.
↪1, random_state=1, stratify=labels)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
↪1, random_state=1, stratify=y_train)
```

```
print(f'train images shape: {X_train.shape}\ntrain labels shape: {y_train.
  ↪shape}\n\nvalidation images shape: {X_val.shape}\nvalidation labels shape:␣
  ↪{y_val.shape}\n\ntest images shape: {X_test.shape}\ntest labels shape:␣
  ↪{y_test.shape}\n')
```

```
train images shape: (198, 64, 64, 3)
train labels shape: (198, 5)

validation images shape: (22, 64, 64, 3)
validation labels shape: (22, 5)

test images shape: (25, 64, 64, 3)
test labels shape: (25, 5)
```

Hingga tahap ini kita telah membagi data menjadi 3 bagian: train, test, dan validation.

5 Data Augmentation

[ ]:
```
train_images_generator = tf.keras.preprocessing.image.
  ↪ImageDataGenerator(shear_range=0.2,
                                                                    ␣
  ↪zoom_range=0.2,
                                                                    ␣
  ↪vertical_flip=True)
train_images_generator = train_images_generator.flow(X_train, y=y_train)
```

Tujuan data augmentation adalah proses untuk memanipulasi data yang ada untuk mendapatkan data baru. Biasanya augentation dilakukan dengan cara memutar gambar, membalik gambar, memperkecil gambar, serta merotasi gambar. Data augmentation dilakukan pada training data dan bukan kepada validation ataupun testing set data. Hal ini disebabkan karena kedua data set tersebut (validation dan test set) digunakan untuk menguji model yang belum pernah dilihat sebelumnya. Jika data augmentation dilakukan pada validation dan test data, maka kedua set data tersebut sudah tidak lagi merepresentasikan data asli dan dapat menngarahkan pada overfitting.

Pada kesempatan kali in saya melakukan data augmentation dengan cara shear, zoom, dan vertical flip. Shear digunakan untuk memberikan efek distorsi, zoom untuk memperbesr atau memperkecil gambar secaar acak.

6 Nomor 2c

[LO 3, 15 poin] Buatlah arsitektur baseline sesuai dengan gambar arsitektur AlexNet berikut ini: (Catatan: Activation function tiap hidden layer menggunakan ReLU)

6.1 Membuat Arsitektur AlexNet Baseline

Gambar disoal merupakan arsitektur alexnet dengan sedikit modifikasi. Jumlah layer terdiri dari 8 yaitu 5 convolutional layer (3 diantaranya diikuti dengan metode pooling) dan 3 layer fully connected. Berikut merupakan arstektur AlexNet:

[ ]:
```
import keras

from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

import tensorflow as tf
from tensorflow import keras
import keras.layers as layers

model = keras.Sequential()
model.add(layers.Conv2D(filters=64, kernel_size=(5, 5),
                        strides=(1, 1), activation="relu", padding="valid",
                        input_shape=(64, 64, 3)))
model.add(layers.MaxPool2D(pool_size=(13, 13), strides= (2, 2),␣
 ↪padding="valid"))

model.add(layers.Conv2D(filters=256, kernel_size=(5, 5),
                        strides=(1, 1), activation="relu",
                        padding="same"))

model.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))

model.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))

model.add(layers.Conv2D(filters=192, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))

model.add(layers.MaxPool2D(pool_size=(1, 1), strides=(1, 1)))
model.add(layers.Flatten())
model.add(layers.Dense(4096, activation="relu"))

model.add(layers.Dense(4096, activation="relu"))

model.add(layers.Dense(5, activation="softmax"))
```

```
optimizer = keras.optimizers.Adam(learning_rate=0.1)
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 60, 60, 64)        4864

 max_pooling2d (MaxPooling2D  (None, 24, 24, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 24, 24, 256)       409856

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 384)       885120

 conv2d_3 (Conv2D)           (None, 12, 12, 384)       1327488

 conv2d_4 (Conv2D)           (None, 12, 12, 192)       663744

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 192)      0
 2D)

 flatten (Flatten)           (None, 27648)             0

 dense (Dense)               (None, 4096)              113250304

 dense_1 (Dense)             (None, 4096)              16781312

 dense_2 (Dense)             (None, 5)                 20485

=================================================================
Total params: 133,343,173
Trainable params: 133,343,173
Non-trainable params: 0
_________________________________________________________________
```

Setelah membuat desain arsitektur AlexNet, sekarang waktunya untuk melatih data yang ada.

[ ]:
```
history = model.fit(((train_images_generator)), epochs=50, verbose=1,␣
 ↪validation_data=(X_val, y_val))
```

```
Epoch 1/50
7/7 [==============================] - 15s 191ms/step - loss: 349207776.0000 -
accuracy: 0.1919 - val_loss: 256.9140 - val_accuracy: 0.1818
Epoch 2/50
7/7 [==============================] - 0s 55ms/step - loss: 951.7314 - accuracy:
0.1919 - val_loss: 15.9465 - val_accuracy: 0.2273
Epoch 3/50
7/7 [==============================] - 0s 48ms/step - loss: 11.6211 - accuracy:
0.1869 - val_loss: 4.2462 - val_accuracy: 0.2273
Epoch 4/50
7/7 [==============================] - 0s 48ms/step - loss: 5.1619 - accuracy:
0.1869 - val_loss: 5.0611 - val_accuracy: 0.2273
Epoch 5/50
7/7 [==============================] - 0s 48ms/step - loss: 7.2352 - accuracy:
0.1465 - val_loss: 7.5827 - val_accuracy: 0.1818
Epoch 6/50
7/7 [==============================] - 0s 50ms/step - loss: 4.7543 - accuracy:
0.1970 - val_loss: 2.4938 - val_accuracy: 0.1818
Epoch 7/50
7/7 [==============================] - 0s 47ms/step - loss: 2.0453 - accuracy:
0.2071 - val_loss: 1.9480 - val_accuracy: 0.2273
Epoch 8/50
7/7 [==============================] - 0s 50ms/step - loss: 1.7182 - accuracy:
0.1919 - val_loss: 1.6748 - val_accuracy: 0.2273
Epoch 9/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6669 - accuracy:
0.2071 - val_loss: 1.6539 - val_accuracy: 0.1818
Epoch 10/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6522 - accuracy:
0.2020 - val_loss: 1.6660 - val_accuracy: 0.1818
Epoch 11/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6379 - accuracy:
0.1515 - val_loss: 1.6293 - val_accuracy: 0.2273
Epoch 12/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6505 - accuracy:
0.1667 - val_loss: 1.6446 - val_accuracy: 0.1818
Epoch 13/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6201 - accuracy:
0.2071 - val_loss: 1.6201 - val_accuracy: 0.1818
Epoch 14/50
7/7 [==============================] - 0s 49ms/step - loss: 1.6307 - accuracy:
0.1515 - val_loss: 1.6394 - val_accuracy: 0.1818
Epoch 15/50
7/7 [==============================] - 0s 52ms/step - loss: 1.6450 - accuracy:
0.1717 - val_loss: 1.6257 - val_accuracy: 0.1818
Epoch 16/50
7/7 [==============================] - 0s 51ms/step - loss: 1.6439 - accuracy:
0.1970 - val_loss: 1.7108 - val_accuracy: 0.1818
```

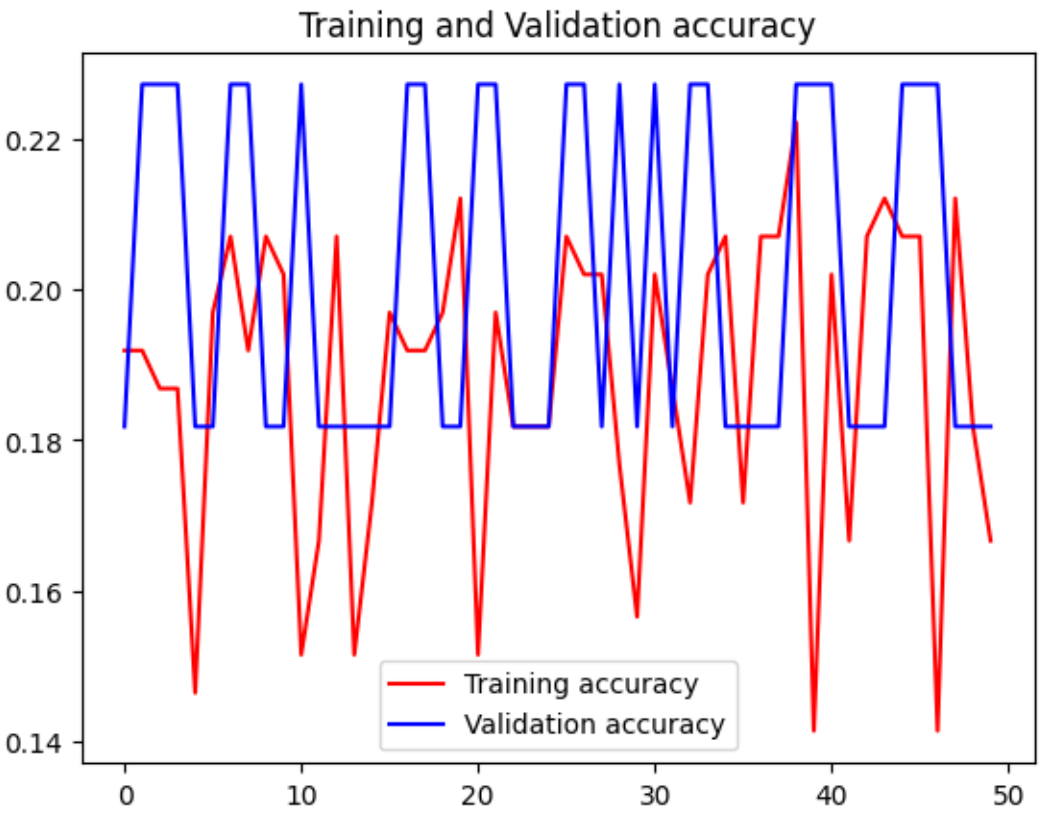
```
Epoch 17/50
7/7 [==============================] - 0s 50ms/step - loss: 1.6612 - accuracy:
0.1919 - val_loss: 1.6108 - val_accuracy: 0.2273
Epoch 18/50
7/7 [==============================] - 0s 55ms/step - loss: 1.6357 - accuracy:
0.1919 - val_loss: 1.6232 - val_accuracy: 0.2273
Epoch 19/50
7/7 [==============================] - 0s 57ms/step - loss: 1.6276 - accuracy:
0.1970 - val_loss: 1.6086 - val_accuracy: 0.1818
Epoch 20/50
7/7 [==============================] - 0s 54ms/step - loss: 1.6252 - accuracy:
0.2121 - val_loss: 1.6126 - val_accuracy: 0.1818
Epoch 21/50
7/7 [==============================] - 0s 52ms/step - loss: 1.6184 - accuracy:
0.1515 - val_loss: 1.6220 - val_accuracy: 0.2273
Epoch 22/50
7/7 [==============================] - 0s 49ms/step - loss: 1.6514 - accuracy:
0.1970 - val_loss: 1.6185 - val_accuracy: 0.2273
Epoch 23/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6165 - accuracy:
0.1818 - val_loss: 1.6133 - val_accuracy: 0.1818
Epoch 24/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6250 - accuracy:
0.1818 - val_loss: 1.6096 - val_accuracy: 0.1818
Epoch 25/50
7/7 [==============================] - 0s 49ms/step - loss: 1.6204 - accuracy:
0.1818 - val_loss: 1.6107 - val_accuracy: 0.1818
Epoch 26/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6147 - accuracy:
0.2071 - val_loss: 1.6087 - val_accuracy: 0.2273
Epoch 27/50
7/7 [==============================] - 0s 52ms/step - loss: 1.6444 - accuracy:
0.2020 - val_loss: 1.6121 - val_accuracy: 0.2273
Epoch 28/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6367 - accuracy:
0.2020 - val_loss: 1.6629 - val_accuracy: 0.1818
Epoch 29/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6549 - accuracy:
0.1768 - val_loss: 1.6106 - val_accuracy: 0.2273
Epoch 30/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6326 - accuracy:
0.1566 - val_loss: 1.6271 - val_accuracy: 0.1818
Epoch 31/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6238 - accuracy:
0.2020 - val_loss: 1.6076 - val_accuracy: 0.2273
Epoch 32/50
7/7 [==============================] - 0s 49ms/step - loss: 1.6176 - accuracy:
0.1869 - val_loss: 1.6140 - val_accuracy: 0.1818
```

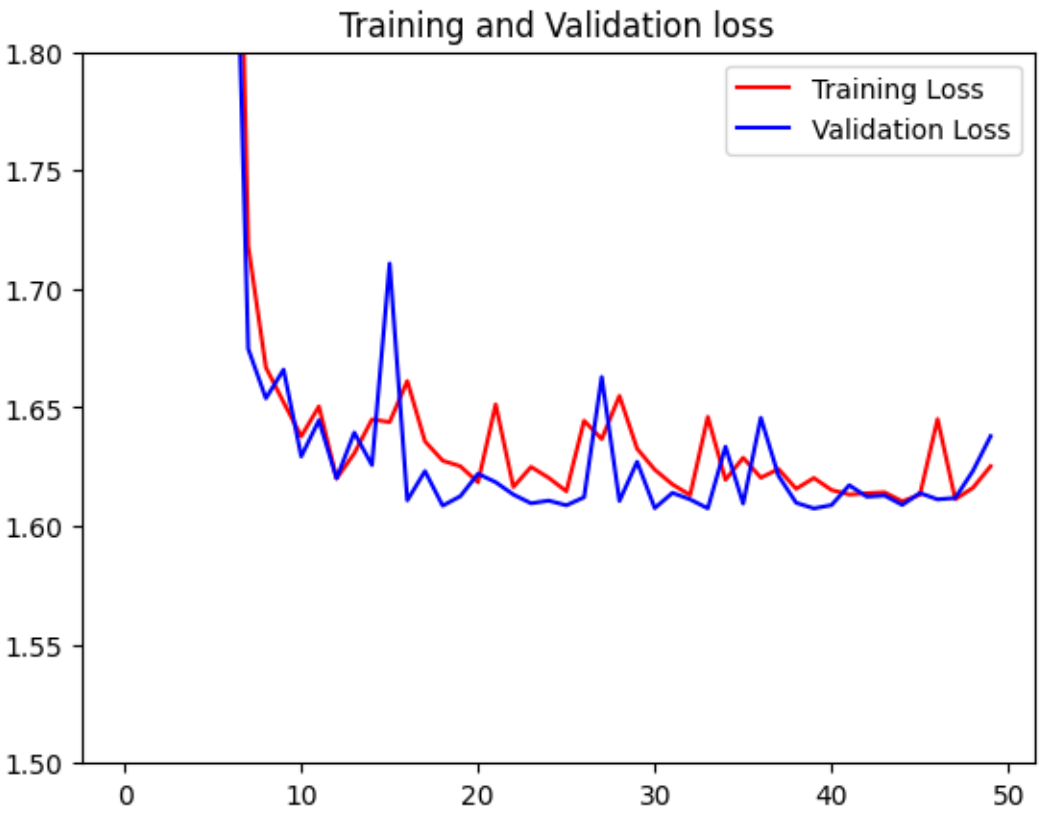
```
Epoch 33/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6131 - accuracy:
0.1717 - val_loss: 1.6113 - val_accuracy: 0.2273
Epoch 34/50
7/7 [==============================] - 0s 51ms/step - loss: 1.6461 - accuracy:
0.2020 - val_loss: 1.6075 - val_accuracy: 0.2273
Epoch 35/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6195 - accuracy:
0.2071 - val_loss: 1.6335 - val_accuracy: 0.1818
Epoch 36/50
7/7 [==============================] - 0s 49ms/step - loss: 1.6288 - accuracy:
0.1717 - val_loss: 1.6095 - val_accuracy: 0.1818
Epoch 37/50
7/7 [==============================] - 0s 50ms/step - loss: 1.6204 - accuracy:
0.2071 - val_loss: 1.6457 - val_accuracy: 0.1818
Epoch 38/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6241 - accuracy:
0.2071 - val_loss: 1.6212 - val_accuracy: 0.1818
Epoch 39/50
7/7 [==============================] - 0s 50ms/step - loss: 1.6157 - accuracy:
0.2222 - val_loss: 1.6098 - val_accuracy: 0.2273
Epoch 40/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6203 - accuracy:
0.1414 - val_loss: 1.6074 - val_accuracy: 0.2273
Epoch 41/50
7/7 [==============================] - 0s 50ms/step - loss: 1.6152 - accuracy:
0.2020 - val_loss: 1.6088 - val_accuracy: 0.2273
Epoch 42/50
7/7 [==============================] - 0s 49ms/step - loss: 1.6133 - accuracy:
0.1667 - val_loss: 1.6172 - val_accuracy: 0.1818
Epoch 43/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6138 - accuracy:
0.2071 - val_loss: 1.6124 - val_accuracy: 0.1818
Epoch 44/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6142 - accuracy:
0.2121 - val_loss: 1.6129 - val_accuracy: 0.1818
Epoch 45/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6104 - accuracy:
0.2071 - val_loss: 1.6089 - val_accuracy: 0.2273
Epoch 46/50
7/7 [==============================] - 0s 50ms/step - loss: 1.6132 - accuracy:
0.2071 - val_loss: 1.6139 - val_accuracy: 0.2273
Epoch 47/50
7/7 [==============================] - 0s 59ms/step - loss: 1.6452 - accuracy:
0.1414 - val_loss: 1.6113 - val_accuracy: 0.2273
Epoch 48/50
7/7 [==============================] - 0s 54ms/step - loss: 1.6113 - accuracy:
0.2121 - val_loss: 1.6119 - val_accuracy: 0.1818
```

```
Epoch 49/50
7/7 [==============================] - 0s 58ms/step - loss: 1.6161 - accuracy:
0.1818 - val_loss: 1.6233 - val_accuracy: 0.1818
Epoch 50/50
7/7 [==============================] - 0s 54ms/step - loss: 1.6252 - accuracy:
0.1667 - val_loss: 1.6379 - val_accuracy: 0.1818
```

[ ]:
```python
import matplotlib.pyplot as plt
# Plot the chart for accuracy and loss on both training and validation
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()

plt.ylim([1.5,1.8])
plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation loss')
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x7f36a06918a0>
```


Training and Validation accuracy
0.22
0.20
0.18
0.16
0.14
0
10
20
30
40
50
Training accuracy
Validation accuracy



[ ]:
```
model.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 401ms/step - loss: 1.6247 - accuracy:
0.2000
```

[ ]:
```
[1.624703049659729, 0.20000000298023224]
```

Output diatas menunjukan hasil pelatihan pada training set. Ketika dibandingkan dengan hasil validation set, nampak terjadi perbedaan akurasi yang cukup signifikan dan tidak beraturan, akurasi maksimum pada training set hanya menyentuh angka 22% dan validation set hanya sedikit diatas 22%. Hal ini menandakan bahwa model belum mempelajari pola-pola yang terdapat di training maupun di validation set. Oleh karena itu sangat dibutuhkan perbaikan arsitektur dan parameter dari model tersebut.

Beberapa faktor yang mungkin mejadi penyebab performa model yang kurang yaitu: * Ukurang input image yang semakin kecil dapat membuat hilangnya informasi yang tersimpan dalam gambar tersebut, dan ini tentu akan mempersulit tugas klasifikasi. * Tidak adanya batch normalization. Batch normaliztion memainkan peran yang penting untuk mempercepat proses kovergensi dan mengurangi resiko overfitting. Batch normalization merupakan teknik regularisasi yang dapat mengurangi ketergantungan model terhdapa bobot tertentu dalam layer dengan cara menormalisasi nilai input ke stiap layer. * Tidak ada drop out. Tidak ada dropout dalam fungsi tersebut, padahal keuntungan dari menggunakan dropout adalah untuk mencegah overfitting selama pelatihan.

Dengan adanya dropout, neuron akan dihilangkan secara acar untuk menghindari memorisasi data training. * Ketika batch normalization dan drop out tidak ada, model akan mengingat model training denga sangat baik sehingga tidak dapat mengeneralisasi, akibatnya data baru menjadi tidak dapat di kenali. * Jumlah iterasi atau epoch yang kurang. Epoch adalah banyaknya model melihat keseluruhan data seama training. Epoch yang kecil mungkin membuat model kurang terlatih dengan baik, model menjadi kurang maksimal dalam mempelajari data secara sepenuhnya. Akan tetapi sebaliknya, epoch yang terlalu banyak akan memakan waktu yang sangat lama dan menyebabkan overfitting. * Pemilihan optimizer juga merupakan parameter yang penting dan dapat mempengaruhi performa model. Optimizer bekerja untuk mengatur bagaimana parameter akan diupdate berdasarkan loss function. * Pemilihan tingkat learning rate, hal ini menjadi sesuatu yang penting karena learning rate menentukan besar update yang dilakukan setiap iterasi. Iterasi yang terlalu besar dapat menjerumuskan pada permasalahan divergen sedangkan iterasi yang terlalu kecil akan memakan waktu yang sangat lama untuk mencapai titik optimal

Berdasarkan point-point diatas, saya mencoba beberapa perubaha pada arsitektur dan parameter.

7 Nomor 2d

[LO 3, 25 poin] Modifikasi arsitektur AlexNet di atas agar mendapatkan hasil klasifikasi yang optimal. Kalian dapat menambahkan atau mengurangi arsitektur tersebutdan melakukan mengubah arsitektur pada nomor 2c dengan menggunakan dropout, batch normalization dan lain-lainnya. Dan selanjutnya lakukan proses tuning hyperparameter agar akurasi klasifikasinya meningkat. Berikan alasan mengapa modifikasi arsitektur dan metode tuning hyperparameter kalian lebih baik.

7.1 Tunning Final

Berikut merupakan hasil modifikasi yang terbaik yang bisa saya lakukan. Tidak hanya melihat dari persentase akurasi, namun saya juga melihat nilai lossnya. Pada modifikasi arsitektur ini ada beberapa perubahan yang saya lakukan yaitu: 1. Mengurangi fully connected layer,dari semula yang bejumlah 2 menjadi 1. 2. Menambah dropout sebanyak 0.5 pada fully connected layer. 3. Mengubah teknik optimasi menggunakan SGD method. 4. Menurunkan learning rate dari 0.1 menjadi 0.001

Berikut merupakan arsitekturnya:

[ ]:
```
import keras

from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

import tensorflow as tf
from tensorflow import keras
import keras.layers as layers
from keras.layers import SpatialDropout2D

model2 = keras.Sequential()
model2.add(layers.Conv2D(filters=64, kernel_size=(5, 5),
                         strides=(1, 1), activation="relu", padding="valid",
                         input_shape=(64, 64, 3)))
```

```
model2.add(layers.MaxPool2D(pool_size=(13, 13), strides= (2, 2),␣
 ↪padding="valid"))

model2.add(layers.Conv2D(filters=256, kernel_size=(5, 5),
                         strides=(1, 1), activation="relu",
                         padding="same"))


model2.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model2.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model2.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model2.add(layers.Conv2D(filters=192, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model2.add(layers.MaxPool2D(pool_size=(1, 1), strides=(1, 1)))
model2.add(layers.Flatten())

model2.add(layers.Dense(4096, activation="relu"))
model2.add(Dropout(0.5))

model2.add(layers.Dense(5, activation="softmax"))

optimizer = keras.optimizers.SGD(learning_rate=0.001)
model2.compile(loss='categorical_crossentropy',
               optimizer=optimizer,
               metrics=['accuracy'])
model2.summary()
```

```
Model: "sequential_27"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_121 (Conv2D)         (None, 60, 60, 64)        4864

 max_pooling2d_71 (MaxPoolin  (None, 24, 24, 64)       0
 g2D)

 conv2d_122 (Conv2D)         (None, 24, 24, 256)       409856
```

```
 max_pooling2d_72 (MaxPoolin  (None, 12, 12, 256)      0
 g2D)

 conv2d_123 (Conv2D)         (None, 12, 12, 384)       885120

 conv2d_124 (Conv2D)         (None, 12, 12, 384)       1327488

 conv2d_125 (Conv2D)         (None, 12, 12, 192)       663744

 max_pooling2d_73 (MaxPoolin  (None, 12, 12, 192)      0
 g2D)

 flatten_23 (Flatten)        (None, 27648)             0

 dense_53 (Dense)            (None, 4096)              113250304

 dropout_16 (Dropout)        (None, 4096)              0

 dense_54 (Dense)            (None, 5)                 20485

=================================================================
Total params: 116,561,861
Trainable params: 116,561,861
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
history = model2.fit(X_train,y_train, epochs=50, verbose=1,␣
↪validation_data=(X_val, y_val))
```

```
Epoch 1/50
7/7 [==============================] - 2s 85ms/step - loss: 186.6296 - accuracy:
0.2374 - val_loss: 1.5965 - val_accuracy: 0.2273
Epoch 2/50
7/7 [==============================] - 0s 41ms/step - loss: 1.6275 - accuracy:
0.2677 - val_loss: 1.6134 - val_accuracy: 0.3636
Epoch 3/50
7/7 [==============================] - 0s 42ms/step - loss: 1.6158 - accuracy:
0.3030 - val_loss: 1.6100 - val_accuracy: 0.3636
Epoch 4/50
7/7 [==============================] - 0s 42ms/step - loss: 1.6147 - accuracy:
0.2980 - val_loss: 1.5938 - val_accuracy: 0.3636
Epoch 5/50
7/7 [==============================] - 0s 42ms/step - loss: 1.6108 - accuracy:
0.2929 - val_loss: 1.6071 - val_accuracy: 0.3182
Epoch 6/50
7/7 [==============================] - 0s 41ms/step - loss: 1.6083 - accuracy:
0.2980 - val_loss: 1.6225 - val_accuracy: 0.3182
```

```
Epoch 7/50
7/7 [==============================] - 0s 41ms/step - loss: 1.6083 - accuracy:
0.2879 - val_loss: 1.6164 - val_accuracy: 0.3182
Epoch 8/50
7/7 [==============================] - 0s 43ms/step - loss: 1.6029 - accuracy:
0.2879 - val_loss: 1.6107 - val_accuracy: 0.2727
Epoch 9/50
7/7 [==============================] - 0s 41ms/step - loss: 1.6011 - accuracy:
0.2980 - val_loss: 1.5975 - val_accuracy: 0.2727
Epoch 10/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5979 - accuracy:
0.2525 - val_loss: 1.6119 - val_accuracy: 0.2273
Epoch 11/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5927 - accuracy:
0.2677 - val_loss: 1.6027 - val_accuracy: 0.2727
Epoch 12/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5917 - accuracy:
0.2727 - val_loss: 1.6199 - val_accuracy: 0.2273
Epoch 13/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5954 - accuracy:
0.2626 - val_loss: 1.6169 - val_accuracy: 0.2727
Epoch 14/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5955 - accuracy:
0.2828 - val_loss: 1.6057 - val_accuracy: 0.2727
Epoch 15/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5897 - accuracy:
0.2879 - val_loss: 1.6169 - val_accuracy: 0.2273
Epoch 16/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5916 - accuracy:
0.2778 - val_loss: 1.6013 - val_accuracy: 0.2727
Epoch 17/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5887 - accuracy:
0.2576 - val_loss: 1.6107 - val_accuracy: 0.2727
Epoch 18/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5852 - accuracy:
0.2677 - val_loss: 1.6238 - val_accuracy: 0.2273
Epoch 19/50
7/7 [==============================] - 0s 42ms/step - loss: 1.5922 - accuracy:
0.2828 - val_loss: 1.6173 - val_accuracy: 0.2273
Epoch 20/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5862 - accuracy:
0.2727 - val_loss: 1.6020 - val_accuracy: 0.2273
Epoch 21/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5779 - accuracy:
0.2727 - val_loss: 1.6124 - val_accuracy: 0.2273
Epoch 22/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5803 - accuracy:
0.2677 - val_loss: 1.6049 - val_accuracy: 0.2273
```

```
Epoch 23/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5766 - accuracy:
0.2677 - val_loss: 1.6077 - val_accuracy: 0.2273
Epoch 24/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5760 - accuracy:
0.2727 - val_loss: 1.5846 - val_accuracy: 0.2727
Epoch 25/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5730 - accuracy:
0.2778 - val_loss: 1.6136 - val_accuracy: 0.2273
Epoch 26/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5753 - accuracy:
0.2727 - val_loss: 1.6194 - val_accuracy: 0.2273
Epoch 27/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5803 - accuracy:
0.2727 - val_loss: 1.6026 - val_accuracy: 0.2273
Epoch 28/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5714 - accuracy:
0.2626 - val_loss: 1.5853 - val_accuracy: 0.2727
Epoch 29/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5734 - accuracy:
0.2576 - val_loss: 1.5936 - val_accuracy: 0.2273
Epoch 30/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5693 - accuracy:
0.2677 - val_loss: 1.6091 - val_accuracy: 0.2273
Epoch 31/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5733 - accuracy:
0.2626 - val_loss: 1.6177 - val_accuracy: 0.2273
Epoch 32/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5783 - accuracy:
0.2778 - val_loss: 1.6000 - val_accuracy: 0.2273
Epoch 33/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5668 - accuracy:
0.2727 - val_loss: 1.5890 - val_accuracy: 0.2727
Epoch 34/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5696 - accuracy:
0.2677 - val_loss: 1.5966 - val_accuracy: 0.2273
Epoch 35/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5614 - accuracy:
0.2677 - val_loss: 1.5788 - val_accuracy: 0.2727
Epoch 36/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5619 - accuracy:
0.2677 - val_loss: 1.5936 - val_accuracy: 0.2273
Epoch 37/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5596 - accuracy:
0.2677 - val_loss: 1.6145 - val_accuracy: 0.2273
Epoch 38/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5671 - accuracy:
0.2475 - val_loss: 1.5977 - val_accuracy: 0.2273
```

```
Epoch 39/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5522 - accuracy:
0.2626 - val_loss: 1.5714 - val_accuracy: 0.3182
Epoch 40/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5549 - accuracy:
0.2727 - val_loss: 1.6055 - val_accuracy: 0.2273
Epoch 41/50
7/7 [==============================] - 0s 42ms/step - loss: 1.5574 - accuracy:
0.2727 - val_loss: 1.5742 - val_accuracy: 0.3182
Epoch 42/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5536 - accuracy:
0.2677 - val_loss: 1.5941 - val_accuracy: 0.2273
Epoch 43/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5570 - accuracy:
0.2677 - val_loss: 1.6145 - val_accuracy: 0.2273
Epoch 44/50
7/7 [==============================] - 0s 42ms/step - loss: 1.5516 - accuracy:
0.2778 - val_loss: 1.5934 - val_accuracy: 0.2273
Epoch 45/50
7/7 [==============================] - 0s 42ms/step - loss: 1.5528 - accuracy:
0.2778 - val_loss: 1.5535 - val_accuracy: 0.2727
Epoch 46/50
7/7 [==============================] - 0s 42ms/step - loss: 1.5516 - accuracy:
0.2576 - val_loss: 1.5284 - val_accuracy: 0.2727
Epoch 47/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5553 - accuracy:
0.2677 - val_loss: 1.6127 - val_accuracy: 0.2273
Epoch 48/50
7/7 [==============================] - 0s 42ms/step - loss: 1.5492 - accuracy:
0.2727 - val_loss: 1.5598 - val_accuracy: 0.2727
Epoch 49/50
7/7 [==============================] - 0s 43ms/step - loss: 1.5448 - accuracy:
0.2576 - val_loss: 1.5927 - val_accuracy: 0.2273
Epoch 50/50
7/7 [==============================] - 0s 41ms/step - loss: 1.5434 - accuracy:
0.2677 - val_loss: 1.5970 - val_accuracy: 0.2273
```

[ ]:
```
model2.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 34ms/step - loss: 1.5909 - accuracy:
0.3200
```

[ ]:
```
[1.5909435749053955, 0.3199999928474426]
```

[ ]:
```
loss_val1, accuracy_val1 = model2.evaluate(X_val,y_val)
loss_train1, accuracy_train1 = model2.evaluate(X_train,y_train)
loss_test1, accuracy_test1 = model2.evaluate(X_test, y_test)
print(f"Validation loss:{loss_val1:.2f}")
```

```
print(f"Training loss:{loss_train1:.2f}")
print(f"Test loss:{loss_test1:.2f}")

print(f"Validation Acc:{accuracy_val1:.2f}")
print(f"Training Acc:{accuracy_train1:.2f}")
print(f"Test Acc:{accuracy_test1:.2f}")
```

```
1/1 [==============================] - 0s 37ms/step - loss: 1.5970 - accuracy:
0.2273
7/7 [==============================] - 0s 14ms/step - loss: 1.5407 - accuracy:
0.2727
1/1 [==============================] - 0s 34ms/step - loss: 1.5909 - accuracy:
0.3200
Validation loss:1.60
Training loss:1.54
Test loss:1.59
Validation Acc:0.23
Training Acc:0.27
Test Acc:0.32
```

7.2 Pembahasan

Hasil diatas merupakan hasil terbaik dari modifikasi arsitektur AlexNet dan tunning hyperparameter yang saya lakukan. Meskipun tidak memberikan akurasi yang tinggi, akan tetapi model tersebut berhasil menaikan akurasi dan menekan nilai loss dengan tetap menjauhkan model dari peristiwa "overfit". perbedaan antara validation loss dengan training loss hanya 0.06 dan perbedaan training dengan akurasi hanya sebesar 0.05. Kemudian hasil akurasi dan loss pada fase training tidak memiliki perbedaan yang jauh ketika dilakukan pada test set. Hal ini berarti bahwa model telah mampu melakukan generalisasi yang cukup baik. Model mampu memberikan prediksi yang baik pada data baru yang tidak trmasuk pada set pelatihan. Akan tetapi satu hal yang menjadi catatan bahwa model ini sangat butuh modifikasi tambahan dan pengembangan lanjut karena nilai akurasinya yang belum cukup baik.

Adanya dropout dan pengurangan fully connected layer membantu model untuk mengurangi overfitting. Dengan mengurangi 1 layer, kita dapat membantu model dalam mengurangi parameter yang sangat besar dan melibatkan operasi yang kompleks. Sementara itu dropout berfungsi untuk menghindari atau meminimalkan resiko model dari mmorisasi data training. Saya juga mengubah teknik optimasi sebab optimizer ini cocok digunakan pada data yang memiliki banyak noise (dalam hal ini misalnya data yang kurang pencahayaan, buram, dan sebagainya).

Dengan adanya peningkatan akurasi, penurunan nilai loss, dan proses generalisasi yang baik, model diatas dapat dikatakan lebih baik dari model baseline.

8 Nomor 2e

[ ]:
```
loss_val1, accuracy_val1 = model2.evaluate(X_val,y_val)
loss_train1, accuracy_train1 = model2.evaluate(X_train,y_train)
loss_test1, accuracy_test1 = model2.evaluate(X_test, y_test)
print(f"Validation loss:{loss_val1:.2f}")
print(f"Training loss:{loss_train1:.2f}")
print(f"Test loss:{loss_test1:.2f}")

print(f"Validation Acc:{accuracy_val1:.2f}")
print(f"Training Acc:{accuracy_train1:.2f}")
print(f"Test Acc:{accuracy_test1:.2f}")
```

```
1/1 [==============================] - 0s 37ms/step - loss: 1.5970 - accuracy:
0.2273
7/7 [==============================] - 0s 14ms/step - loss: 1.5407 - accuracy:
0.2727
1/1 [==============================] - 0s 34ms/step - loss: 1.5909 - accuracy:
0.3200
Validation loss:1.60
Training loss:1.54
Test loss:1.59
Validation Acc:0.23
Training Acc:0.27
Test Acc:0.32
```

[ ]:
```
predicted = model2.predict(X_test)
test_predictions = np.array(predicted)
test_predictions = np.argmax(np.array(test_predictions), axis=1)
print ("Predicted Class")
test_predictions
```

```
1/1 [==============================] - 0s 95ms/step
Predicted Class
```

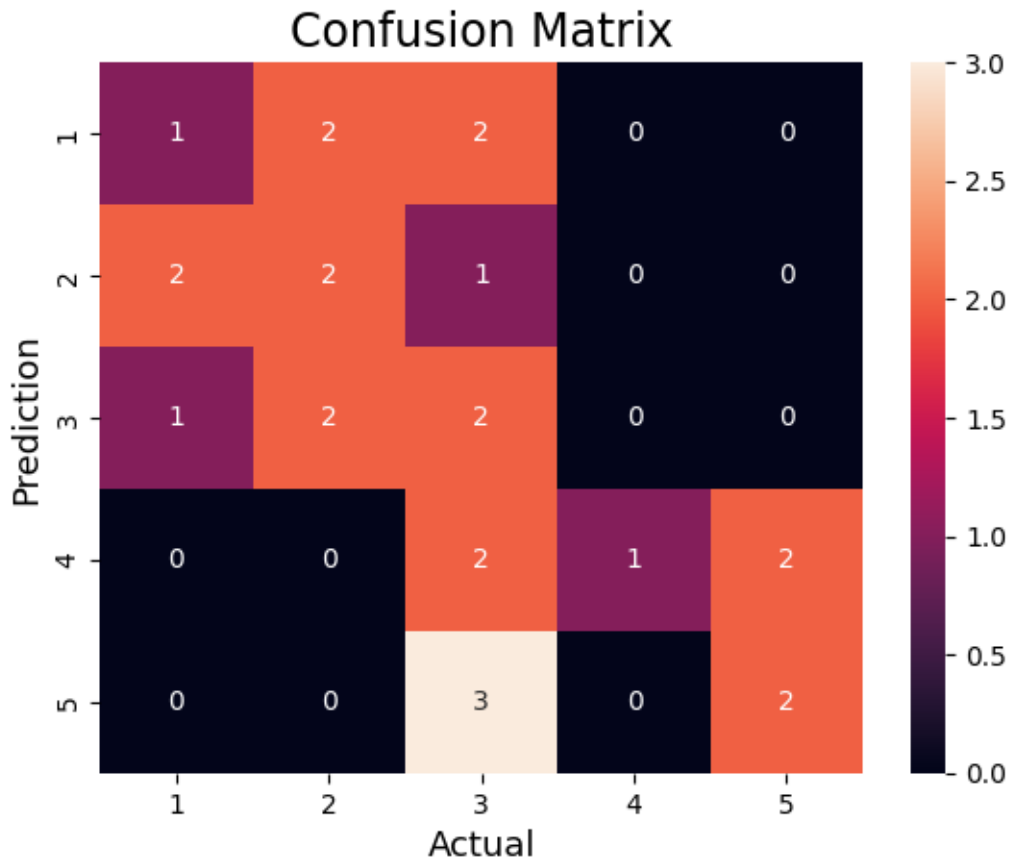
[ ]:
```
array([2, 3, 1, 4, 0, 4, 2, 1, 0, 4, 2, 2, 2, 2, 0, 2, 2, 1, 0, 4, 1, 1,
       2, 1, 2])
```

[ ]:
```
y_label = np.array(y_test)
y_label = np.argmax(np.array(y_label), axis=1)
print ("Ground Truth")
y_label
```

```
Ground Truth
```

[ ]:
```
array([0, 3, 2, 3, 1, 4, 3, 1, 1, 3, 4, 2, 2, 1, 0, 0, 3, 0, 2, 4, 1, 0,
       4, 2, 4])
```

[ ]:
```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

actual = np.array([0, 3, 2, 3, 1, 4, 3, 1, 1, 3, 4, 2, 2, 1, 0, 0, 3, 0, 2, 4,␣
 ↪1, 0,
       4, 2, 4])
predicted = np.array([2, 3, 1, 4, 0, 4, 2, 1, 0, 4, 2, 2, 2, 2, 0, 2, 2, 1, 0,␣
 ↪4, 1, 1,
       2, 1, 2])
cm = confusion_matrix(actual,predicted)
sns.heatmap(cm,
            annot=True,
            fmt='g',
            xticklabels=['1','2','3', '4','5'],
            yticklabels=['1','2','3', '4','5'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```



```
[ ]: print(classification_report(actual, predicted))
```

```
              precision    recall  f1-score   support

           0       0.25      0.20      0.22         5
           1       0.33      0.40      0.36         5
           2       0.20      0.40      0.27         5
           3       1.00      0.20      0.33         5
           4       0.50      0.40      0.44         5

    accuracy                           0.32        25
   macro avg       0.46      0.32      0.33        25
weighted avg       0.46      0.32      0.33        25
```

Kode diatas merupakan hasil evaluasi dari arsitektur modifikasi final. Model yang terbaik sejauh ini adalah model final sebab selain melihat nilai akurasi, model tersebut juga memiliki nilai loss terkecil tanpa adanya peristiwa overfitting. Tunning 1, meskipun memiliki nilai akurasi yag lebih tinggi yakni 40%, namun model tersebut memiliki perbedaan yang cukup besar dari sisi validasi dan training. Kemudiann Tunning 3 tidak dipilih karena hanya memiliki performa yang baik pada fase training, sedangkan pada fase testing akurasi hanya mencapai 28% (berbeda 12% dari fase traiing). Serta tunning nomor 4 tidak dipilih karena selama fase training, model mengalami overfit antara training dengan validation set, selain itu nilai loss pada model tersebut belum cukup optimal jika dibandinga dengan tunning final.

Secara keseluruhan model terbaik terlihat pada final tunning sebab model tersebut dapat memberikan generalisasi yang lebih baik ketimbang model tunning lainnya. Selain itu model tunning final memiliki nilai akurasi yang lebih tinggi dan berhasil menurunkan loss jika dibandingkan dengan model baseline. Akan tetapi tidak bisa dipungkin bahwa model ini belum dapat digunakan sepenuhnya untuk mendeteksi lima macam batik. Perlu dilakukan pengaturan dan pembuatan arsitektur yang lebih baik agar mencapai hasil yang optimum.

Hasil klasifikasi model ialah 32% akurasi, 32% recal, 32% f1-score, dan 46% presisi. Hal ini berarti ketika model memprediksi suatu kelas, hal tersebut tepat sebesar 46% (presisi). Kemudian ketika batik kedalam suatu kelas, model dapat memprediksi 32% tepat. Dan dari data yang ada, model berhasil mengklasifikasikan data secara tepat sebesar 32%.

Berkaitan dengan tunning 2, tunning 2 memiliki hasil akurasi yang paling tinggi, tunning 2 dibuat dengan menggurangi jumlah layer, menambah drop out, menggunakan batch normalization, dan mengubah kernel size, dan jumlah filter. Meski memiliki akurasi paling tinggi, namun nilai loss nya tidak lebih rendah dari baseline. Nilai loss merupakan nilai yang patut kita lihat juga sebab nilai akurasi hanya mengukur persetase prediksi yang benar, akan tetapi tidak memberikan informasi tentang tingkat kesalahan yang dilakukan. Oleh karena itu saya tetap memilih tunning final sebagai model yang lebih baik dari model baseline.

9 Nomor 2F

https://drive.google.com/drive/folders/1iIxpxIgVLqB1G-cWYxDCVfliwRd05f-k?usp=sharing

9.1 Tunning 1

[ ]:
```
import keras

from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

import tensorflow as tf
from tensorflow import keras
import keras.layers as layers

model1 = keras.Sequential()
model1.add(layers.Conv2D(filters=64, kernel_size=(5, 5),
                         strides=(1, 1), activation="relu", padding="valid",
                         input_shape=(64, 64, 3)))
model1.add(layers.MaxPool2D(pool_size=(13, 13), strides= (2, 2),␣
 ↪padding="valid"))

model1.add(layers.Conv2D(filters=256, kernel_size=(5, 5),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model1.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model1.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model1.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model1.add(layers.Conv2D(filters=192, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model1.add(layers.MaxPool2D(pool_size=(1, 1), strides=(1, 1)))
model1.add(layers.Flatten())
model1.add(layers.Dense(4096, activation="relu"))

model1.add(layers.Dense(4096, activation="relu"))

model1.add(layers.Dense(5, activation="softmax"))
```

```
model.add(layers.Dropout(0.5))

optimizer = keras.optimizers.Adam(learning_rate=0.001)
model1.compile(loss='categorical_crossentropy',
               optimizer=optimizer,
               metrics=['accuracy'])
model1.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_30 (Conv2D)          (None, 60, 60, 64)        4864

 max_pooling2d_18 (MaxPoolin  (None, 24, 24, 64)       0
 g2D)

 conv2d_31 (Conv2D)          (None, 24, 24, 256)       409856

 max_pooling2d_19 (MaxPoolin  (None, 12, 12, 256)      0
 g2D)

 conv2d_32 (Conv2D)          (None, 12, 12, 384)       885120

 conv2d_33 (Conv2D)          (None, 12, 12, 384)       1327488

 conv2d_34 (Conv2D)          (None, 12, 12, 192)       663744

 max_pooling2d_20 (MaxPoolin  (None, 12, 12, 192)      0
 g2D)

 flatten_6 (Flatten)         (None, 27648)             0

 dense_18 (Dense)            (None, 4096)              113250304

 dense_19 (Dense)            (None, 4096)              16781312

 dense_20 (Dense)            (None, 5)                 20485

=================================================================
Total params: 133,343,173
Trainable params: 133,343,173
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
history = model1.fit(X_train,y_train, epochs=50, verbose=1,␣
↪validation_data=(X_val, y_val))
```

```
Epoch 1/50
7/7 [==============================] - 3s 82ms/step - loss: 2725.7219 -
accuracy: 0.2222 - val_loss: 22.9488 - val_accuracy: 0.2273
Epoch 2/50
7/7 [==============================] - 0s 47ms/step - loss: 30.1901 - accuracy:
0.1970 - val_loss: 2.4877 - val_accuracy: 0.2273
Epoch 3/50
7/7 [==============================] - 0s 47ms/step - loss: 2.4823 - accuracy:
0.1414 - val_loss: 1.7023 - val_accuracy: 0.2273
Epoch 4/50
7/7 [==============================] - 0s 47ms/step - loss: 1.7068 - accuracy:
0.1818 - val_loss: 1.5840 - val_accuracy: 0.2273
Epoch 5/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6576 - accuracy:
0.2020 - val_loss: 1.5860 - val_accuracy: 0.4091
Epoch 6/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6352 - accuracy:
0.2020 - val_loss: 1.5793 - val_accuracy: 0.1364
Epoch 7/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6081 - accuracy:
0.1869 - val_loss: 1.5906 - val_accuracy: 0.3182
Epoch 8/50
7/7 [==============================] - 0s 47ms/step - loss: 1.5979 - accuracy:
0.2323 - val_loss: 1.5759 - val_accuracy: 0.2727
Epoch 9/50
7/7 [==============================] - 0s 47ms/step - loss: 1.5923 - accuracy:
0.2374 - val_loss: 1.5540 - val_accuracy: 0.2273
Epoch 10/50
7/7 [==============================] - 0s 53ms/step - loss: 1.6162 - accuracy:
0.2374 - val_loss: 1.6351 - val_accuracy: 0.3182
Epoch 11/50
7/7 [==============================] - 0s 47ms/step - loss: 1.8178 - accuracy:
0.2576 - val_loss: 1.6745 - val_accuracy: 0.2273
Epoch 12/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6933 - accuracy:
0.2273 - val_loss: 1.5594 - val_accuracy: 0.3636
Epoch 13/50
7/7 [==============================] - 0s 57ms/step - loss: 1.6110 - accuracy:
0.2727 - val_loss: 1.4598 - val_accuracy: 0.4091
Epoch 14/50
7/7 [==============================] - 0s 47ms/step - loss: 1.7242 - accuracy:
0.2677 - val_loss: 1.6340 - val_accuracy: 0.3182
Epoch 15/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6420 - accuracy:
0.2323 - val_loss: 1.6258 - val_accuracy: 0.1364
Epoch 16/50
7/7 [==============================] - 0s 49ms/step - loss: 1.6367 - accuracy:
0.2222 - val_loss: 1.5780 - val_accuracy: 0.2273
```

```
Epoch 17/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6324 - accuracy:
0.2374 - val_loss: 1.6538 - val_accuracy: 0.3182
Epoch 18/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6114 - accuracy:
0.2323 - val_loss: 1.6182 - val_accuracy: 0.1364
Epoch 19/50
7/7 [==============================] - 0s 47ms/step - loss: 1.5995 - accuracy:
0.2374 - val_loss: 1.6021 - val_accuracy: 0.2273
Epoch 20/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6092 - accuracy:
0.2374 - val_loss: 1.6113 - val_accuracy: 0.2273
Epoch 21/50
7/7 [==============================] - 0s 47ms/step - loss: 1.6207 - accuracy:
0.2626 - val_loss: 1.9570 - val_accuracy: 0.1818
Epoch 22/50
7/7 [==============================] - 0s 47ms/step - loss: 1.7567 - accuracy:
0.2121 - val_loss: 1.6159 - val_accuracy: 0.2727
Epoch 23/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6946 - accuracy:
0.2626 - val_loss: 1.7662 - val_accuracy: 0.2273
Epoch 24/50
7/7 [==============================] - 0s 49ms/step - loss: 1.6333 - accuracy:
0.2576 - val_loss: 1.6601 - val_accuracy: 0.1818
Epoch 25/50
7/7 [==============================] - 0s 48ms/step - loss: 1.6651 - accuracy:
0.2626 - val_loss: 1.6494 - val_accuracy: 0.1364
Epoch 26/50
7/7 [==============================] - 0s 47ms/step - loss: 1.5913 - accuracy:
0.2475 - val_loss: 1.6756 - val_accuracy: 0.1364
Epoch 27/50
7/7 [==============================] - 0s 48ms/step - loss: 1.5921 - accuracy:
0.2525 - val_loss: 1.6152 - val_accuracy: 0.2727
Epoch 28/50
7/7 [==============================] - 0s 48ms/step - loss: 1.5852 - accuracy:
0.2778 - val_loss: 1.5976 - val_accuracy: 0.3182
Epoch 29/50
7/7 [==============================] - 0s 47ms/step - loss: 1.5819 - accuracy:
0.3030 - val_loss: 1.6795 - val_accuracy: 0.2273
Epoch 30/50
7/7 [==============================] - 0s 47ms/step - loss: 1.5864 - accuracy:
0.2929 - val_loss: 1.8015 - val_accuracy: 0.2727
Epoch 31/50
7/7 [==============================] - 0s 48ms/step - loss: 1.5195 - accuracy:
0.2980 - val_loss: 1.7309 - val_accuracy: 0.1818
Epoch 32/50
7/7 [==============================] - 0s 47ms/step - loss: 1.5521 - accuracy:
0.2525 - val_loss: 1.6701 - val_accuracy: 0.2273
```

```
Epoch 33/50
7/7 [==============================] - 0s 47ms/step - loss: 1.5193 - accuracy:
0.3131 - val_loss: 1.8216 - val_accuracy: 0.2273
Epoch 34/50
7/7 [==============================] - 0s 48ms/step - loss: 1.4723 - accuracy:
0.3333 - val_loss: 1.9522 - val_accuracy: 0.2727
Epoch 35/50
7/7 [==============================] - 0s 47ms/step - loss: 1.4903 - accuracy:
0.3232 - val_loss: 1.7719 - val_accuracy: 0.2727
Epoch 36/50
7/7 [==============================] - 0s 47ms/step - loss: 1.4803 - accuracy:
0.3434 - val_loss: 1.8160 - val_accuracy: 0.1818
Epoch 37/50
7/7 [==============================] - 0s 48ms/step - loss: 1.5147 - accuracy:
0.3182 - val_loss: 1.7904 - val_accuracy: 0.1818
Epoch 38/50
7/7 [==============================] - 0s 47ms/step - loss: 1.4369 - accuracy:
0.3687 - val_loss: 1.9657 - val_accuracy: 0.1818
Epoch 39/50
7/7 [==============================] - 0s 47ms/step - loss: 1.5009 - accuracy:
0.3333 - val_loss: 2.0009 - val_accuracy: 0.3182
Epoch 40/50
7/7 [==============================] - 0s 48ms/step - loss: 1.4791 - accuracy:
0.3030 - val_loss: 2.0276 - val_accuracy: 0.2727
Epoch 41/50
7/7 [==============================] - 0s 47ms/step - loss: 1.3850 - accuracy:
0.4293 - val_loss: 2.1398 - val_accuracy: 0.2727
Epoch 42/50
7/7 [==============================] - 0s 47ms/step - loss: 1.3681 - accuracy:
0.4242 - val_loss: 1.8345 - val_accuracy: 0.2273
Epoch 43/50
7/7 [==============================] - 0s 48ms/step - loss: 1.2765 - accuracy:
0.4848 - val_loss: 2.2127 - val_accuracy: 0.2273
Epoch 44/50
7/7 [==============================] - 0s 48ms/step - loss: 1.3025 - accuracy:
0.4495 - val_loss: 2.3047 - val_accuracy: 0.1818
Epoch 45/50
7/7 [==============================] - 0s 47ms/step - loss: 1.2705 - accuracy:
0.4495 - val_loss: 2.1340 - val_accuracy: 0.2273
Epoch 46/50
7/7 [==============================] - 0s 48ms/step - loss: 1.1649 - accuracy:
0.5455 - val_loss: 2.0835 - val_accuracy: 0.2727
Epoch 47/50
7/7 [==============================] - 0s 48ms/step - loss: 1.1845 - accuracy:
0.5455 - val_loss: 2.2983 - val_accuracy: 0.2727
Epoch 48/50
7/7 [==============================] - 0s 47ms/step - loss: 1.0617 - accuracy:
0.6010 - val_loss: 2.6698 - val_accuracy: 0.2273
```

```
Epoch 49/50
7/7 [==============================] - 0s 51ms/step - loss: 1.0331 - accuracy:
0.5505 - val_loss: 2.7772 - val_accuracy: 0.1818
Epoch 50/50
7/7 [==============================] - 0s 48ms/step - loss: 0.9675 - accuracy:
0.5909 - val_loss: 2.5702 - val_accuracy: 0.3182
```

[ ]:
```
model1.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 38ms/step - loss: 2.2111 - accuracy:
0.4000
```

[ ]:
```
[2.2111382484436035, 0.4000000059604645]
```

[ ]:
```
loss_val1, accuracy_val1 = model1.evaluate(X_val,y_val)
loss_train1, accuracy_train1 = model1.evaluate(X_train,y_train)
loss_test1, accuracy_test1 = model1.evaluate(X_test, y_test)
print(f"Validation loss:{loss_val1:.2f}")
print(f"Training loss:{loss_train1:.2f}")
print(f"Test loss:{loss_test1:.2f}")

print(f"Validation Acc:{accuracy_val1:.2f}")
print(f"Training Acc:{accuracy_train1:.2f}")
print(f"Test Acc:{accuracy_test1:.2f}")
```

```
1/1 [==============================] - 0s 37ms/step - loss: 2.5702 - accuracy:
0.3182
7/7 [==============================] - 0s 15ms/step - loss: 1.0215 - accuracy:
0.6061
1/1 [==============================] - 0s 39ms/step - loss: 2.2111 - accuracy:
0.4000
Validation loss:2.57
Training loss:1.02
Test loss:2.21
Validation Acc:0.32
Training Acc:0.61
Test Acc:0.40
```

9.2 Tunning 2

[ ]:
```
import keras

from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

import tensorflow as tf
from tensorflow import keras
import keras.layers as layers
```

```
model5 = keras.Sequential()
model5.add(layers.Conv2D(filters=69, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu", padding="same",
                         input_shape=(64, 64, 3)))
model5.add(layers.BatchNormalization())
model5.add(layers.MaxPool2D(pool_size=(2, 2), strides= (2, 2), padding="valid"))

model5.add(layers.Conv2D(filters=16, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))
model5.add(layers.BatchNormalization())
model5.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model5.add(layers.Conv2D(filters=32, kernel_size=(2, 2),
                         strides=(2, 2), activation="relu",
                         padding="same"))
model5.add(layers.BatchNormalization())
model5.add(layers.Flatten())

model5.add(layers.Dense(1000, activation="relu"))
model5.add(layers.Dropout(0.5))

model5.add(layers.Dense(1000, activation="relu"))
model5.add(layers.Dropout(0.5))

model5.add(layers.Dense(5, activation="softmax"))


optimizer = keras.optimizers.SGD(learning_rate=0.0001)
model5.compile(loss='categorical_crossentropy',
               optimizer=optimizer,
               metrics=['accuracy'])
model5.summary()
```

```
Model: "sequential_30"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_96 (Conv2D)          (None, 64, 64, 69)        1932

 batch_normalization_40 (Bat  (None, 64, 64, 69)       276
 chNormalization)

 max_pooling2d_60 (MaxPoolin  (None, 32, 32, 69)       0
 g2D)
```

```
conv2d_97 (Conv2D)          (None, 32, 32, 16)        9952

batch_normalization_41 (Bat  (None, 32, 32, 16)       64
chNormalization)

max_pooling2d_61 (MaxPoolin  (None, 16, 16, 16)       0
g2D)

conv2d_98 (Conv2D)          (None, 8, 8, 32)          2080

batch_normalization_42 (Bat  (None, 8, 8, 32)         128
chNormalization)

flatten_26 (Flatten)        (None, 2048)              0

dense_78 (Dense)            (None, 1000)              2049000

dropout_47 (Dropout)        (None, 1000)              0

dense_79 (Dense)            (None, 1000)              1001000

dropout_48 (Dropout)        (None, 1000)              0

dense_80 (Dense)            (None, 5)                 5005

=================================================================
Total params: 3,069,437
Trainable params: 3,069,203
Non-trainable params: 234
_________________________________________________________________
```

[ ]:
```
history = model5.fit(X_train,y_train, epochs=100, batch_size=9,verbose=1,␣
↪validation_data=(X_val, y_val))
```

```
Epoch 1/100
22/22 [==============================] - 2s 19ms/step - loss: 2.2850 - accuracy:
0.1818 - val_loss: 3.3099 - val_accuracy: 0.0909
Epoch 2/100
22/22 [==============================] - 0s 7ms/step - loss: 2.2413 - accuracy:
0.1768 - val_loss: 2.4594 - val_accuracy: 0.1364
Epoch 3/100
22/22 [==============================] - 0s 9ms/step - loss: 2.2336 - accuracy:
0.2222 - val_loss: 2.1339 - val_accuracy: 0.1818
Epoch 4/100
22/22 [==============================] - 0s 8ms/step - loss: 2.1989 - accuracy:
0.2020 - val_loss: 1.9170 - val_accuracy: 0.1818
Epoch 5/100
22/22 [==============================] - 0s 9ms/step - loss: 2.2076 - accuracy:
```

```
0.2222 - val_loss: 1.8024 - val_accuracy: 0.1364
Epoch 6/100
22/22 [==============================] - 0s 10ms/step - loss: 2.2726 - accuracy:
0.2172 - val_loss: 1.7349 - val_accuracy: 0.0909
Epoch 7/100
22/22 [==============================] - 0s 9ms/step - loss: 2.0249 - accuracy:
0.2677 - val_loss: 1.6851 - val_accuracy: 0.2273
Epoch 8/100
22/22 [==============================] - 0s 8ms/step - loss: 2.2480 - accuracy:
0.1667 - val_loss: 1.6698 - val_accuracy: 0.2273
Epoch 9/100
22/22 [==============================] - 0s 8ms/step - loss: 2.0884 - accuracy:
0.1768 - val_loss: 1.6506 - val_accuracy: 0.1818
Epoch 10/100
22/22 [==============================] - 0s 8ms/step - loss: 2.1686 - accuracy:
0.2121 - val_loss: 1.6604 - val_accuracy: 0.1364
Epoch 11/100
22/22 [==============================] - 0s 8ms/step - loss: 2.2839 - accuracy:
0.1818 - val_loss: 1.6413 - val_accuracy: 0.1818
Epoch 12/100
22/22 [==============================] - 0s 9ms/step - loss: 2.0898 - accuracy:
0.2374 - val_loss: 1.6204 - val_accuracy: 0.3182
Epoch 13/100
22/22 [==============================] - 0s 9ms/step - loss: 2.1722 - accuracy:
0.1869 - val_loss: 1.6112 - val_accuracy: 0.3182
Epoch 14/100
22/22 [==============================] - 0s 10ms/step - loss: 2.1458 - accuracy:
0.2424 - val_loss: 1.6015 - val_accuracy: 0.3636
Epoch 15/100
22/22 [==============================] - 0s 8ms/step - loss: 2.1084 - accuracy:
0.2071 - val_loss: 1.6075 - val_accuracy: 0.3182
Epoch 16/100
22/22 [==============================] - 0s 8ms/step - loss: 2.1338 - accuracy:
0.1768 - val_loss: 1.5846 - val_accuracy: 0.3636
Epoch 17/100
22/22 [==============================] - 0s 8ms/step - loss: 2.0342 - accuracy:
0.2222 - val_loss: 1.5862 - val_accuracy: 0.2727
Epoch 18/100
22/22 [==============================] - 0s 15ms/step - loss: 2.1799 - accuracy:
0.1869 - val_loss: 1.6370 - val_accuracy: 0.3182
Epoch 19/100
22/22 [==============================] - 0s 11ms/step - loss: 2.1572 - accuracy:
0.1970 - val_loss: 1.6126 - val_accuracy: 0.2727
Epoch 20/100
22/22 [==============================] - 0s 8ms/step - loss: 2.0344 - accuracy:
0.2222 - val_loss: 1.5577 - val_accuracy: 0.3636
Epoch 21/100
22/22 [==============================] - 0s 8ms/step - loss: 1.9380 - accuracy:
```

```
0.2222 - val_loss: 1.5638 - val_accuracy: 0.1364
Epoch 22/100
22/22 [==============================] - 0s 8ms/step - loss: 2.0230 - accuracy:
0.2525 - val_loss: 1.5636 - val_accuracy: 0.2273
Epoch 23/100
22/22 [==============================] - 0s 7ms/step - loss: 2.0795 - accuracy:
0.2071 - val_loss: 1.5591 - val_accuracy: 0.1818
Epoch 24/100
22/22 [==============================] - 0s 8ms/step - loss: 1.9069 - accuracy:
0.2929 - val_loss: 1.5826 - val_accuracy: 0.2273
Epoch 25/100
22/22 [==============================] - 0s 7ms/step - loss: 1.9380 - accuracy:
0.2374 - val_loss: 1.5604 - val_accuracy: 0.1818
Epoch 26/100
22/22 [==============================] - 0s 7ms/step - loss: 1.8587 - accuracy:
0.2929 - val_loss: 1.5509 - val_accuracy: 0.2273
Epoch 27/100
22/22 [==============================] - 0s 6ms/step - loss: 1.9933 - accuracy:
0.2980 - val_loss: 1.5299 - val_accuracy: 0.2727
Epoch 28/100
22/22 [==============================] - 0s 7ms/step - loss: 1.8026 - accuracy:
0.3081 - val_loss: 1.5323 - val_accuracy: 0.3182
Epoch 29/100
22/22 [==============================] - 0s 6ms/step - loss: 1.8485 - accuracy:
0.2525 - val_loss: 1.5146 - val_accuracy: 0.2273
Epoch 30/100
22/22 [==============================] - 0s 6ms/step - loss: 1.8438 - accuracy:
0.2879 - val_loss: 1.5091 - val_accuracy: 0.1818
Epoch 31/100
22/22 [==============================] - 0s 7ms/step - loss: 1.8815 - accuracy:
0.2222 - val_loss: 1.4586 - val_accuracy: 0.4091
Epoch 32/100
22/22 [==============================] - 0s 7ms/step - loss: 1.7918 - accuracy:
0.3232 - val_loss: 1.4619 - val_accuracy: 0.3636
Epoch 33/100
22/22 [==============================] - 0s 6ms/step - loss: 1.7781 - accuracy:
0.2828 - val_loss: 1.4735 - val_accuracy: 0.2727
Epoch 34/100
22/22 [==============================] - 0s 12ms/step - loss: 1.7637 - accuracy:
0.3232 - val_loss: 1.4868 - val_accuracy: 0.2727
Epoch 35/100
22/22 [==============================] - 0s 11ms/step - loss: 1.8892 - accuracy:
0.2525 - val_loss: 1.4714 - val_accuracy: 0.3182
Epoch 36/100
22/22 [==============================] - 0s 7ms/step - loss: 1.7591 - accuracy:
0.3333 - val_loss: 1.4878 - val_accuracy: 0.2727
Epoch 37/100
22/22 [==============================] - 0s 6ms/step - loss: 1.8566 - accuracy:
```

```
0.3131 - val_loss: 1.4797 - val_accuracy: 0.2727
Epoch 38/100
22/22 [==============================] - 0s 7ms/step - loss: 1.8416 - accuracy:
0.2677 - val_loss: 1.4867 - val_accuracy: 0.2273
Epoch 39/100
22/22 [==============================] - 0s 6ms/step - loss: 1.8063 - accuracy:
0.2727 - val_loss: 1.4731 - val_accuracy: 0.2273
Epoch 40/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6261 - accuracy:
0.3333 - val_loss: 1.4778 - val_accuracy: 0.3182
Epoch 41/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6724 - accuracy:
0.3182 - val_loss: 1.5182 - val_accuracy: 0.2727
Epoch 42/100
22/22 [==============================] - 0s 7ms/step - loss: 1.8235 - accuracy:
0.3586 - val_loss: 1.4980 - val_accuracy: 0.2273
Epoch 43/100
22/22 [==============================] - 0s 6ms/step - loss: 1.7312 - accuracy:
0.3232 - val_loss: 1.5265 - val_accuracy: 0.1818
Epoch 44/100
22/22 [==============================] - 0s 6ms/step - loss: 1.7443 - accuracy:
0.3131 - val_loss: 1.5515 - val_accuracy: 0.2273
Epoch 45/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6901 - accuracy:
0.3232 - val_loss: 1.5495 - val_accuracy: 0.2273
Epoch 46/100
22/22 [==============================] - 0s 7ms/step - loss: 1.7356 - accuracy:
0.3636 - val_loss: 1.5196 - val_accuracy: 0.2727
Epoch 47/100
22/22 [==============================] - 0s 6ms/step - loss: 1.7527 - accuracy:
0.3838 - val_loss: 1.4961 - val_accuracy: 0.3182
Epoch 48/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6849 - accuracy:
0.3182 - val_loss: 1.4787 - val_accuracy: 0.3182
Epoch 49/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6501 - accuracy:
0.3485 - val_loss: 1.5015 - val_accuracy: 0.3182
Epoch 50/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5929 - accuracy:
0.3788 - val_loss: 1.4930 - val_accuracy: 0.3182
Epoch 51/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6214 - accuracy:
0.3485 - val_loss: 1.4923 - val_accuracy: 0.3636
Epoch 52/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6339 - accuracy:
0.3384 - val_loss: 1.4609 - val_accuracy: 0.3636
Epoch 53/100
22/22 [==============================] - 0s 7ms/step - loss: 1.7482 - accuracy:
```

```
0.3283 - val_loss: 1.4702 - val_accuracy: 0.4091
Epoch 54/100
22/22 [==============================] - 0s 7ms/step - loss: 1.6353 - accuracy:
0.3485 - val_loss: 1.4569 - val_accuracy: 0.3182
Epoch 55/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6229 - accuracy:
0.3586 - val_loss: 1.4726 - val_accuracy: 0.4091
Epoch 56/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4772 - accuracy:
0.4242 - val_loss: 1.4769 - val_accuracy: 0.3182
Epoch 57/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5281 - accuracy:
0.4040 - val_loss: 1.4612 - val_accuracy: 0.4091
Epoch 58/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5988 - accuracy:
0.3384 - val_loss: 1.4656 - val_accuracy: 0.3636
Epoch 59/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5272 - accuracy:
0.3687 - val_loss: 1.4814 - val_accuracy: 0.2273
Epoch 60/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5885 - accuracy:
0.3687 - val_loss: 1.4598 - val_accuracy: 0.3182
Epoch 61/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6376 - accuracy:
0.3586 - val_loss: 1.4577 - val_accuracy: 0.3182
Epoch 62/100
22/22 [==============================] - 0s 6ms/step - loss: 1.7218 - accuracy:
0.3232 - val_loss: 1.4449 - val_accuracy: 0.3636
Epoch 63/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5553 - accuracy:
0.3687 - val_loss: 1.4301 - val_accuracy: 0.4091
Epoch 64/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6278 - accuracy:
0.3939 - val_loss: 1.4517 - val_accuracy: 0.3636
Epoch 65/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5164 - accuracy:
0.3889 - val_loss: 1.4442 - val_accuracy: 0.4091
Epoch 66/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5082 - accuracy:
0.3687 - val_loss: 1.4492 - val_accuracy: 0.4091
Epoch 67/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5231 - accuracy:
0.4040 - val_loss: 1.4657 - val_accuracy: 0.2727
Epoch 68/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6141 - accuracy:
0.3939 - val_loss: 1.4595 - val_accuracy: 0.4091
Epoch 69/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5207 - accuracy:
```

```
0.3889 - val_loss: 1.4527 - val_accuracy: 0.3636
Epoch 70/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4451 - accuracy:
0.4091 - val_loss: 1.4422 - val_accuracy: 0.4091
Epoch 71/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6711 - accuracy:
0.3535 - val_loss: 1.4520 - val_accuracy: 0.3636
Epoch 72/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4634 - accuracy:
0.3889 - val_loss: 1.4455 - val_accuracy: 0.4091
Epoch 73/100
22/22 [==============================] - 0s 6ms/step - loss: 1.6521 - accuracy:
0.2879 - val_loss: 1.4185 - val_accuracy: 0.3636
Epoch 74/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5630 - accuracy:
0.3586 - val_loss: 1.4515 - val_accuracy: 0.3636
Epoch 75/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4861 - accuracy:
0.3485 - val_loss: 1.4581 - val_accuracy: 0.3182
Epoch 76/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4399 - accuracy:
0.4596 - val_loss: 1.4807 - val_accuracy: 0.2727
Epoch 77/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5587 - accuracy:
0.4040 - val_loss: 1.4796 - val_accuracy: 0.3182
Epoch 78/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4721 - accuracy:
0.4091 - val_loss: 1.4809 - val_accuracy: 0.2727
Epoch 79/100
22/22 [==============================] - 0s 6ms/step - loss: 1.5193 - accuracy:
0.3485 - val_loss: 1.4993 - val_accuracy: 0.2727
Epoch 80/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4511 - accuracy:
0.4293 - val_loss: 1.4704 - val_accuracy: 0.3182
Epoch 81/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4379 - accuracy:
0.3990 - val_loss: 1.4691 - val_accuracy: 0.2273
Epoch 82/100
22/22 [==============================] - 0s 6ms/step - loss: 1.3775 - accuracy:
0.4495 - val_loss: 1.4792 - val_accuracy: 0.2727
Epoch 83/100
22/22 [==============================] - 0s 6ms/step - loss: 1.3368 - accuracy:
0.4242 - val_loss: 1.4866 - val_accuracy: 0.2727
Epoch 84/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4822 - accuracy:
0.4141 - val_loss: 1.4872 - val_accuracy: 0.3636
Epoch 85/100
22/22 [==============================] - 0s 7ms/step - loss: 1.3671 - accuracy:
```

```
0.4242 - val_loss: 1.4596 - val_accuracy: 0.3182
Epoch 86/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4978 - accuracy:
0.4040 - val_loss: 1.4523 - val_accuracy: 0.3636
Epoch 87/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4148 - accuracy:
0.4343 - val_loss: 1.4213 - val_accuracy: 0.4545
Epoch 88/100
22/22 [==============================] - 0s 6ms/step - loss: 1.3577 - accuracy:
0.4596 - val_loss: 1.4368 - val_accuracy: 0.4545
Epoch 89/100
22/22 [==============================] - 0s 6ms/step - loss: 1.2405 - accuracy:
0.5253 - val_loss: 1.4175 - val_accuracy: 0.4091
Epoch 90/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4194 - accuracy:
0.3586 - val_loss: 1.4210 - val_accuracy: 0.4091
Epoch 91/100
22/22 [==============================] - 0s 6ms/step - loss: 1.4086 - accuracy:
0.4141 - val_loss: 1.4184 - val_accuracy: 0.4545
Epoch 92/100
22/22 [==============================] - 0s 6ms/step - loss: 1.2735 - accuracy:
0.4697 - val_loss: 1.4095 - val_accuracy: 0.4091
Epoch 93/100
22/22 [==============================] - 0s 8ms/step - loss: 1.2964 - accuracy:
0.4899 - val_loss: 1.4167 - val_accuracy: 0.4545
Epoch 94/100
22/22 [==============================] - 0s 7ms/step - loss: 1.4326 - accuracy:
0.4293 - val_loss: 1.4262 - val_accuracy: 0.4091
Epoch 95/100
22/22 [==============================] - 0s 8ms/step - loss: 1.4655 - accuracy:
0.4141 - val_loss: 1.4379 - val_accuracy: 0.3182
Epoch 96/100
22/22 [==============================] - 0s 7ms/step - loss: 1.4513 - accuracy:
0.4192 - val_loss: 1.4617 - val_accuracy: 0.3182
Epoch 97/100
22/22 [==============================] - 0s 7ms/step - loss: 1.5020 - accuracy:
0.4293 - val_loss: 1.4852 - val_accuracy: 0.3636
Epoch 98/100
22/22 [==============================] - 0s 8ms/step - loss: 1.3954 - accuracy:
0.4141 - val_loss: 1.4794 - val_accuracy: 0.3636
Epoch 99/100
22/22 [==============================] - 0s 7ms/step - loss: 1.3689 - accuracy:
0.4444 - val_loss: 1.4736 - val_accuracy: 0.3636
Epoch 100/100
22/22 [==============================] - 0s 7ms/step - loss: 1.4036 - accuracy:
0.4040 - val_loss: 1.4686 - val_accuracy: 0.4091
```

[ ]:
```
model.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 387ms/step - loss: 1.9768 - accuracy:
0.4583
```

[ ]:
```
[1.9767512083053589, 0.4583333432674408]
```

9.3 Tunning 3

[ ]:
```
import keras

from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

import tensorflow as tf
from tensorflow import keras
import keras.layers as layers
from keras.layers import SpatialDropout2D

model3 = keras.Sequential()
model3.add(layers.Conv2D(filters=64, kernel_size=(5, 5),
                         strides=(1, 1), activation="relu", padding="valid",
                         input_shape=(64, 64, 3)))

model3.add(layers.MaxPool2D(pool_size=(13, 13), strides= (2, 2),␣
 ↪padding="valid"))

model3.add(layers.Conv2D(filters=256, kernel_size=(5, 5),
                         strides=(1, 1), activation="relu",
                         padding="same"))


model3.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model3.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model3.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model3.add(layers.Conv2D(filters=192, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu",
                         padding="same"))

model3.add(layers.MaxPool2D(pool_size=(1, 1), strides=(1, 1)))
```

```
model3.add(layers.Flatten())

model3.add(layers.Dense(4096, activation="relu"))
# model3.add(Dropout(0.5))

# model3.add(layers.Dense(4096, activation="relu"))
# model3.add(Dropout(0.2))

model3.add(layers.Dense(5, activation="softmax"))

optimizer = keras.optimizers.SGD(learning_rate=0.001)
model3.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
model3.summary()
```

```
Model: "sequential_67"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_317 (Conv2D)         (None, 60, 60, 64)        4864

 max_pooling2d_188 (MaxPooli  (None, 24, 24, 64)       0
 ng2D)

 conv2d_318 (Conv2D)         (None, 24, 24, 256)       409856

 max_pooling2d_189 (MaxPooli  (None, 12, 12, 256)      0
 ng2D)

 conv2d_319 (Conv2D)         (None, 12, 12, 384)       885120

 conv2d_320 (Conv2D)         (None, 12, 12, 384)       1327488

 conv2d_321 (Conv2D)         (None, 12, 12, 192)       663744

 max_pooling2d_190 (MaxPooli  (None, 12, 12, 192)      0
 ng2D)

 flatten_62 (Flatten)        (None, 27648)             0

 dense_150 (Dense)           (None, 4096)              113250304

 dense_151 (Dense)           (None, 5)                 20485

=================================================================
Total params: 116,561,861
```

```
Trainable params: 116,561,861
Non-trainable params: 0
_________________________________________________________________
```

```
[ ]: history = model3.fit(X_train,y_train, epochs=30, batch_size=10, verbose=1,␣
      ↪validation_data=(X_val, y_val))
```

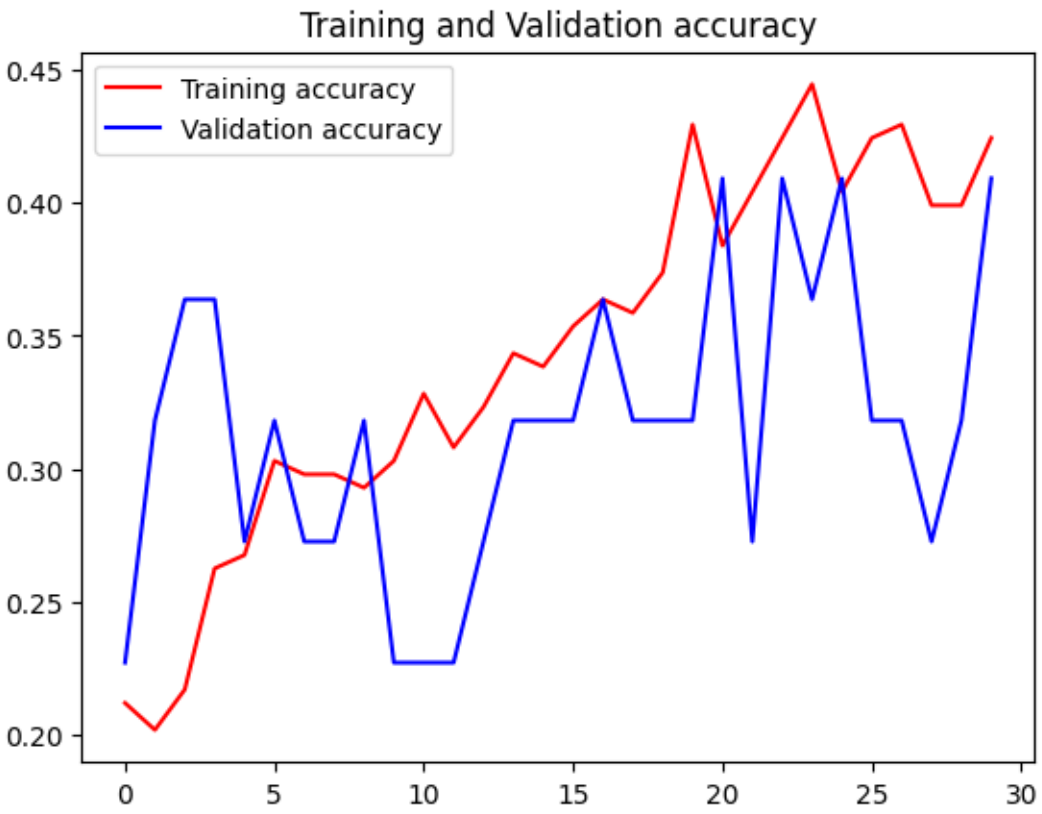
```
Epoch 1/30
20/20 [==============================] - 2s 41ms/step - loss: 58.2248 -
accuracy: 0.2121 - val_loss: 1.6070 - val_accuracy: 0.2273
Epoch 2/30
20/20 [==============================] - 1s 30ms/step - loss: 1.6089 - accuracy:
0.2020 - val_loss: 1.6042 - val_accuracy: 0.3182
Epoch 3/30
20/20 [==============================] - 1s 31ms/step - loss: 1.5962 - accuracy:
0.2172 - val_loss: 1.6029 - val_accuracy: 0.3636
Epoch 4/30
20/20 [==============================] - 1s 30ms/step - loss: 1.5860 - accuracy:
0.2626 - val_loss: 1.6034 - val_accuracy: 0.3636
Epoch 5/30
20/20 [==============================] - 1s 30ms/step - loss: 1.5781 - accuracy:
0.2677 - val_loss: 1.6044 - val_accuracy: 0.2727
Epoch 6/30
20/20 [==============================] - 1s 30ms/step - loss: 1.5730 - accuracy:
0.3030 - val_loss: 1.6043 - val_accuracy: 0.3182
Epoch 7/30
20/20 [==============================] - 1s 30ms/step - loss: 1.5620 - accuracy:
0.2980 - val_loss: 1.6006 - val_accuracy: 0.2727
Epoch 8/30
20/20 [==============================] - 1s 30ms/step - loss: 1.5565 - accuracy:
0.2980 - val_loss: 1.5907 - val_accuracy: 0.2727
Epoch 9/30
20/20 [==============================] - 1s 30ms/step - loss: 1.5487 - accuracy:
0.2929 - val_loss: 1.5919 - val_accuracy: 0.3182
Epoch 10/30
20/20 [==============================] - 1s 31ms/step - loss: 1.5373 - accuracy:
0.3030 - val_loss: 1.5913 - val_accuracy: 0.2273
Epoch 11/30
20/20 [==============================] - 1s 31ms/step - loss: 1.5299 - accuracy:
0.3283 - val_loss: 1.5818 - val_accuracy: 0.2273
Epoch 12/30
20/20 [==============================] - 1s 31ms/step - loss: 1.5237 - accuracy:
0.3081 - val_loss: 1.5805 - val_accuracy: 0.2273
Epoch 13/30
20/20 [==============================] - 1s 31ms/step - loss: 1.5029 - accuracy:
0.3232 - val_loss: 1.5725 - val_accuracy: 0.2727
Epoch 14/30
20/20 [==============================] - 1s 31ms/step - loss: 1.4982 - accuracy:
```

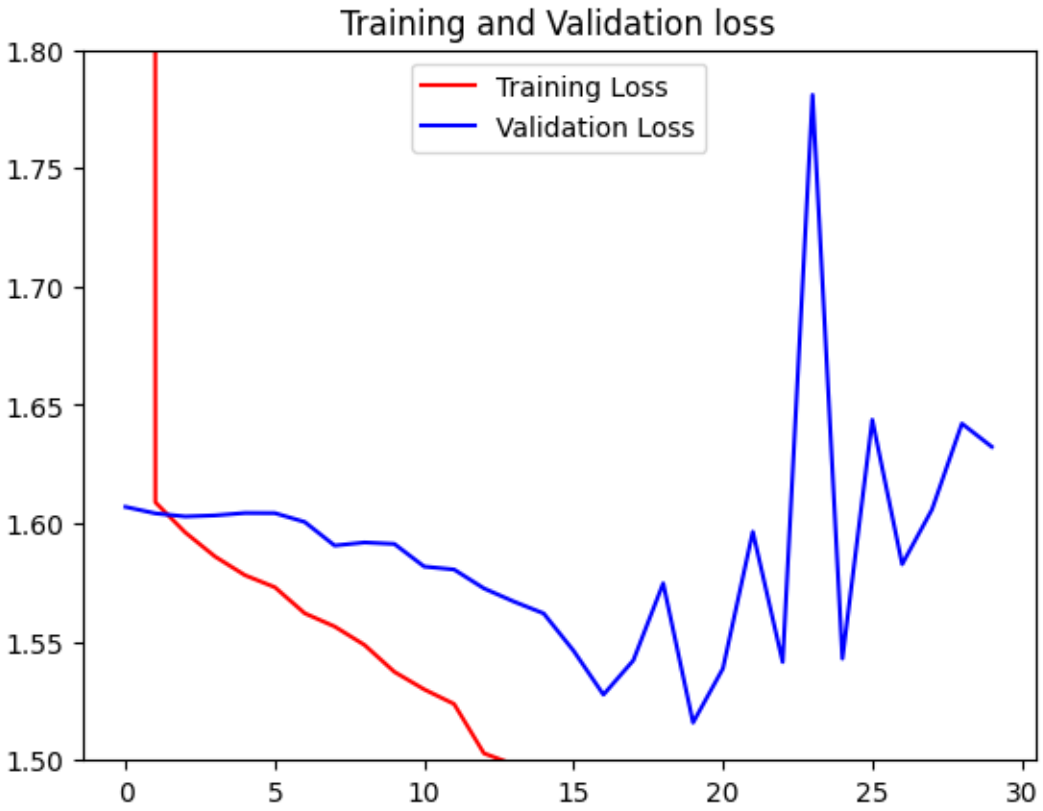
```
0.3434 - val_loss: 1.5670 - val_accuracy: 0.3182
Epoch 15/30
20/20 [==============================] - 1s 30ms/step - loss: 1.4668 - accuracy:
0.3384 - val_loss: 1.5620 - val_accuracy: 0.3182
Epoch 16/30
20/20 [==============================] - 1s 30ms/step - loss: 1.4552 - accuracy:
0.3535 - val_loss: 1.5462 - val_accuracy: 0.3182
Epoch 17/30
20/20 [==============================] - 1s 31ms/step - loss: 1.4373 - accuracy:
0.3636 - val_loss: 1.5277 - val_accuracy: 0.3636
Epoch 18/30
20/20 [==============================] - 1s 31ms/step - loss: 1.4145 - accuracy:
0.3586 - val_loss: 1.5421 - val_accuracy: 0.3182
Epoch 19/30
20/20 [==============================] - 1s 31ms/step - loss: 1.4042 - accuracy:
0.3737 - val_loss: 1.5747 - val_accuracy: 0.3182
Epoch 20/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3766 - accuracy:
0.4293 - val_loss: 1.5159 - val_accuracy: 0.3182
Epoch 21/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3766 - accuracy:
0.3838 - val_loss: 1.5387 - val_accuracy: 0.4091
Epoch 22/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3700 - accuracy:
0.4040 - val_loss: 1.5965 - val_accuracy: 0.2727
Epoch 23/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3255 - accuracy:
0.4242 - val_loss: 1.5415 - val_accuracy: 0.4091
Epoch 24/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3122 - accuracy:
0.4444 - val_loss: 1.7810 - val_accuracy: 0.3636
Epoch 25/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3145 - accuracy:
0.4040 - val_loss: 1.5429 - val_accuracy: 0.4091
Epoch 26/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3264 - accuracy:
0.4242 - val_loss: 1.6438 - val_accuracy: 0.3182
Epoch 27/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3415 - accuracy:
0.4293 - val_loss: 1.5827 - val_accuracy: 0.3182
Epoch 28/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3492 - accuracy:
0.3990 - val_loss: 1.6059 - val_accuracy: 0.2727
Epoch 29/30
20/20 [==============================] - 1s 31ms/step - loss: 1.3316 - accuracy:
0.3990 - val_loss: 1.6422 - val_accuracy: 0.3182
Epoch 30/30
20/20 [==============================] - 1s 31ms/step - loss: 1.2705 - accuracy:
```

```
0.4242 - val_loss: 1.6324 - val_accuracy: 0.4091
```

[ ]:
```
import matplotlib.pyplot as plt
# Plot the chart for accuracy and loss on both training and validation
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()

plt.ylim([1.5,1.8])
plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation loss')
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x7f48b45606d0>
```





[ ]:
```
model3.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 36ms/step - loss: 2.0355 - accuracy:
0.2800
```

[ ]:
```
[2.0355262756347656, 0.2800000011920929]
```

[ ]:
```
loss_val1, accuracy_val1 = model3.evaluate(X_val,y_val)
loss_train1, accuracy_train1 = model3.evaluate(X_train,y_train)
loss_test1, accuracy_test1 = model3.evaluate(X_test, y_test)
print(f"Validation loss:{loss_val1:.2f}")
print(f"Training loss:{loss_train1:.2f}")
print(f"Test loss:{loss_test1:.2f}")

print(f"Validation Acc:{accuracy_val1:.2f}")
print(f"Training Acc:{accuracy_train1:.2f}")
print(f"Test Acc:{accuracy_test1:.2f}")
```

```
1/1 [==============================] - 0s 36ms/step - loss: 1.6324 - accuracy:
```

```
0.4091
7/7 [==============================] - 0s 14ms/step - loss: 1.2204 - accuracy:
0.4545
1/1 [==============================] - 0s 35ms/step - loss: 2.0355 - accuracy:
0.2800
Validation loss:1.63
Training loss:1.22
Test loss:2.04
Validation Acc:0.41
Training Acc:0.45
Test Acc:0.28
```

9.4 Tuning 4

[ ]:
```
model = keras.Sequential()
model.add(layers.Conv2D(filters=64, kernel_size=(5, 5),
                        strides=(1, 1), activation="relu", padding="valid",
                        input_shape=(64, 64, 3)))

model.add(layers.BatchNormalization())
model.add(layers.MaxPool2D(pool_size=(13, 13), strides= (2, 2),␣
 ↪padding="valid"))

model.add(layers.Conv2D(filters=256, kernel_size=(5, 5),
                        strides=(1, 1), activation="relu",
                        padding="same"))

model.add(layers.BatchNormalization())

model.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))
model.add(SpatialDropout2D(0.5))
model.add(layers.BatchNormalization())

model.add(layers.Conv2D(filters=384, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))
model.add(layers.BatchNormalization())

model.add(layers.Conv2D(filters=192, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))
model.add(SpatialDropout2D(0.5))
model.add(layers.BatchNormalization())
```

```
model.add(layers.MaxPool2D(pool_size=(1, 1), strides=(1, 1)))
model.add(layers.Flatten())
model.add(layers.Dense(4096, activation="relu"))
model.add(layers.Dropout(0.3))

model.add(layers.Dense(4096, activation="relu"))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(5, activation="softmax"))

optimizer = keras.optimizers.SGD(learning_rate=0.001)
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_53 (Conv2D)          (None, 60, 60, 64)        4864

 batch_normalization_53 (Bat  (None, 60, 60, 64)       256
 chNormalization)

 max_pooling2d_33 (MaxPoolin  (None, 24, 24, 64)       0
 g2D)

 conv2d_54 (Conv2D)          (None, 24, 24, 256)       409856

 batch_normalization_54 (Bat  (None, 24, 24, 256)      1024
 chNormalization)

 max_pooling2d_34 (MaxPoolin  (None, 12, 12, 256)      0
 g2D)

 conv2d_55 (Conv2D)          (None, 12, 12, 384)       885120

 spatial_dropout2d_17 (Spati  (None, 12, 12, 384)      0
 alDropout2D)

 batch_normalization_55 (Bat  (None, 12, 12, 384)      1536
 chNormalization)

 conv2d_56 (Conv2D)          (None, 12, 12, 384)       1327488
```

```
 batch_normalization_56 (Bat  (None, 12, 12, 384)      1536
 chNormalization)

 conv2d_57 (Conv2D)          (None, 12, 12, 192)       663744

 spatial_dropout2d_18 (Spati  (None, 12, 12, 192)      0
 alDropout2D)

 batch_normalization_57 (Bat  (None, 12, 12, 192)      768
 chNormalization)

 max_pooling2d_35 (MaxPoolin  (None, 12, 12, 192)      0
 g2D)

 flatten_11 (Flatten)        (None, 27648)             0

 dense_33 (Dense)            (None, 4096)              113250304

 dropout_22 (Dropout)        (None, 4096)              0

 dense_34 (Dense)            (None, 4096)              16781312

 dropout_23 (Dropout)        (None, 4096)              0

 dense_35 (Dense)            (None, 5)                 20485

=================================================================
Total params: 133,348,293
Trainable params: 133,345,733
Non-trainable params: 2,560
_________________________________________________________________
```

[ ]:
```
history = model.fit(X_train, y_train,
                    epochs=300,
                    verbose=1,
                    batch_size = 35,
                    validation_data=(X_val, y_val))
```

```
Epoch 1/300
7/7 [==============================] - 2s 115ms/step - loss: 2.3570 - accuracy:
0.2045 - val_loss: 2.6025 - val_accuracy: 0.1600
Epoch 2/300
7/7 [==============================] - 0s 67ms/step - loss: 2.1291 - accuracy:
0.2182 - val_loss: 1.9719 - val_accuracy: 0.1200
Epoch 3/300
7/7 [==============================] - 0s 67ms/step - loss: 2.3317 - accuracy:
0.2000 - val_loss: 1.9126 - val_accuracy: 0.2800
Epoch 4/300
```

```
7/7 [==============================] - 0s 66ms/step - loss: 2.2465 - accuracy:
0.2318 - val_loss: 1.6188 - val_accuracy: 0.2400
Epoch 5/300
7/7 [==============================] - 0s 67ms/step - loss: 2.1711 - accuracy:
0.2227 - val_loss: 1.7075 - val_accuracy: 0.1600
Epoch 6/300
7/7 [==============================] - 0s 67ms/step - loss: 2.0980 - accuracy:
0.2318 - val_loss: 1.6727 - val_accuracy: 0.2000
Epoch 7/300
7/7 [==============================] - 0s 67ms/step - loss: 2.2184 - accuracy:
0.1682 - val_loss: 1.6724 - val_accuracy: 0.2400
Epoch 8/300
7/7 [==============================] - 0s 67ms/step - loss: 2.0439 - accuracy:
0.2455 - val_loss: 1.7615 - val_accuracy: 0.1200
Epoch 9/300
7/7 [==============================] - 0s 67ms/step - loss: 2.0561 - accuracy:
0.2409 - val_loss: 1.7416 - val_accuracy: 0.1600
Epoch 10/300
7/7 [==============================] - 0s 67ms/step - loss: 2.0431 - accuracy:
0.3000 - val_loss: 1.6453 - val_accuracy: 0.2400
Epoch 11/300
7/7 [==============================] - 0s 67ms/step - loss: 2.1663 - accuracy:
0.2045 - val_loss: 1.6874 - val_accuracy: 0.2400
Epoch 12/300
7/7 [==============================] - 0s 66ms/step - loss: 2.1455 - accuracy:
0.2500 - val_loss: 1.5925 - val_accuracy: 0.2400
Epoch 13/300
7/7 [==============================] - 0s 67ms/step - loss: 1.9395 - accuracy:
0.2682 - val_loss: 1.6051 - val_accuracy: 0.2000
Epoch 14/300
7/7 [==============================] - 0s 67ms/step - loss: 2.0149 - accuracy:
0.2227 - val_loss: 1.5743 - val_accuracy: 0.2000
Epoch 15/300
7/7 [==============================] - 0s 67ms/step - loss: 2.0813 - accuracy:
0.2727 - val_loss: 1.5810 - val_accuracy: 0.3600
Epoch 16/300
7/7 [==============================] - 0s 67ms/step - loss: 2.0929 - accuracy:
0.2455 - val_loss: 1.5793 - val_accuracy: 0.3600
Epoch 17/300
7/7 [==============================] - 0s 67ms/step - loss: 1.9926 - accuracy:
0.2364 - val_loss: 1.5418 - val_accuracy: 0.4000
Epoch 18/300
7/7 [==============================] - 0s 68ms/step - loss: 2.0028 - accuracy:
0.2182 - val_loss: 1.6461 - val_accuracy: 0.3200
Epoch 19/300
7/7 [==============================] - 0s 67ms/step - loss: 2.1468 - accuracy:
0.2273 - val_loss: 1.5566 - val_accuracy: 0.3200
Epoch 20/300
```

```
7/7 [==============================] - 0s 67ms/step - loss: 1.9645 - accuracy:
0.2864 - val_loss: 1.5232 - val_accuracy: 0.2800
Epoch 21/300
7/7 [==============================] - 0s 68ms/step - loss: 1.8889 - accuracy:
0.2682 - val_loss: 1.5448 - val_accuracy: 0.2800
Epoch 22/300
7/7 [==============================] - 0s 67ms/step - loss: 1.7587 - accuracy:
0.2545 - val_loss: 1.5689 - val_accuracy: 0.2400
Epoch 23/300
7/7 [==============================] - 0s 67ms/step - loss: 2.0010 - accuracy:
0.2455 - val_loss: 1.5232 - val_accuracy: 0.2400
Epoch 24/300
7/7 [==============================] - 0s 68ms/step - loss: 1.8509 - accuracy:
0.2909 - val_loss: 1.5258 - val_accuracy: 0.2000
Epoch 25/300
7/7 [==============================] - 0s 68ms/step - loss: 1.8221 - accuracy:
0.3091 - val_loss: 1.5104 - val_accuracy: 0.2000
Epoch 26/300
7/7 [==============================] - 0s 68ms/step - loss: 2.0333 - accuracy:
0.2273 - val_loss: 1.5102 - val_accuracy: 0.2800
Epoch 27/300
7/7 [==============================] - 0s 67ms/step - loss: 1.8647 - accuracy:
0.3045 - val_loss: 1.5094 - val_accuracy: 0.3600
Epoch 28/300
7/7 [==============================] - 0s 68ms/step - loss: 1.9312 - accuracy:
0.2864 - val_loss: 1.5200 - val_accuracy: 0.3200
Epoch 29/300
7/7 [==============================] - 0s 67ms/step - loss: 1.8246 - accuracy:
0.2682 - val_loss: 1.5348 - val_accuracy: 0.2800
Epoch 30/300
7/7 [==============================] - 0s 67ms/step - loss: 1.8828 - accuracy:
0.3045 - val_loss: 1.5460 - val_accuracy: 0.2800
Epoch 31/300
7/7 [==============================] - 0s 67ms/step - loss: 1.9104 - accuracy:
0.3136 - val_loss: 1.4983 - val_accuracy: 0.2800
Epoch 32/300
7/7 [==============================] - 0s 68ms/step - loss: 1.9313 - accuracy:
0.2682 - val_loss: 1.4921 - val_accuracy: 0.2800
Epoch 33/300
7/7 [==============================] - 0s 68ms/step - loss: 1.8091 - accuracy:
0.2955 - val_loss: 1.5215 - val_accuracy: 0.2800
Epoch 34/300
7/7 [==============================] - 0s 68ms/step - loss: 1.9096 - accuracy:
0.2955 - val_loss: 1.5326 - val_accuracy: 0.3600
Epoch 35/300
7/7 [==============================] - 0s 68ms/step - loss: 1.9311 - accuracy:
0.2773 - val_loss: 1.5323 - val_accuracy: 0.3600
Epoch 36/300
```

```
7/7 [==============================] - 0s 68ms/step - loss: 1.8615 - accuracy:
0.2818 - val_loss: 1.6511 - val_accuracy: 0.1600
Epoch 37/300
7/7 [==============================] - 0s 67ms/step - loss: 1.9044 - accuracy:
0.3045 - val_loss: 1.6934 - val_accuracy: 0.2000
Epoch 38/300
7/7 [==============================] - 0s 67ms/step - loss: 1.9159 - accuracy:
0.2727 - val_loss: 2.2521 - val_accuracy: 0.1200
Epoch 39/300
7/7 [==============================] - 0s 68ms/step - loss: 1.7801 - accuracy:
0.2909 - val_loss: 1.9547 - val_accuracy: 0.1600
Epoch 40/300
7/7 [==============================] - 0s 68ms/step - loss: 1.7147 - accuracy:
0.3318 - val_loss: 1.6456 - val_accuracy: 0.2400
Epoch 41/300
7/7 [==============================] - 0s 68ms/step - loss: 1.8547 - accuracy:
0.3182 - val_loss: 1.8103 - val_accuracy: 0.1600
Epoch 42/300
7/7 [==============================] - 0s 68ms/step - loss: 1.8423 - accuracy:
0.2909 - val_loss: 1.6702 - val_accuracy: 0.1600
Epoch 43/300
7/7 [==============================] - 0s 68ms/step - loss: 1.7453 - accuracy:
0.3545 - val_loss: 1.6214 - val_accuracy: 0.1600
Epoch 44/300
7/7 [==============================] - 0s 68ms/step - loss: 1.8565 - accuracy:
0.3000 - val_loss: 1.6085 - val_accuracy: 0.2400
Epoch 45/300
7/7 [==============================] - 0s 67ms/step - loss: 1.8656 - accuracy:
0.2773 - val_loss: 1.6978 - val_accuracy: 0.2000
Epoch 46/300
7/7 [==============================] - 0s 68ms/step - loss: 1.6370 - accuracy:
0.3409 - val_loss: 1.6434 - val_accuracy: 0.2000
Epoch 47/300
7/7 [==============================] - 0s 68ms/step - loss: 1.7221 - accuracy:
0.3136 - val_loss: 1.6128 - val_accuracy: 0.3200
Epoch 48/300
7/7 [==============================] - 0s 69ms/step - loss: 1.6612 - accuracy:
0.3409 - val_loss: 1.6459 - val_accuracy: 0.2400
Epoch 49/300
7/7 [==============================] - 0s 70ms/step - loss: 1.6631 - accuracy:
0.3182 - val_loss: 1.6781 - val_accuracy: 0.2000
Epoch 50/300
7/7 [==============================] - 0s 69ms/step - loss: 1.8079 - accuracy:
0.3182 - val_loss: 1.5887 - val_accuracy: 0.2400
Epoch 51/300
7/7 [==============================] - 0s 69ms/step - loss: 1.7729 - accuracy:
0.3500 - val_loss: 1.5740 - val_accuracy: 0.1600
Epoch 52/300
```

```
7/7 [==============================] - 0s 69ms/step - loss: 1.7555 - accuracy:
0.3273 - val_loss: 1.5707 - val_accuracy: 0.2000
Epoch 53/300
7/7 [==============================] - 0s 68ms/step - loss: 1.7056 - accuracy:
0.3045 - val_loss: 1.5874 - val_accuracy: 0.2800
Epoch 54/300
7/7 [==============================] - 0s 69ms/step - loss: 1.6792 - accuracy:
0.3409 - val_loss: 1.6299 - val_accuracy: 0.2400
Epoch 55/300
7/7 [==============================] - 0s 68ms/step - loss: 1.6560 - accuracy:
0.3727 - val_loss: 1.7253 - val_accuracy: 0.2400
Epoch 56/300
7/7 [==============================] - 0s 69ms/step - loss: 1.6255 - accuracy:
0.3727 - val_loss: 1.6714 - val_accuracy: 0.2000
Epoch 57/300
7/7 [==============================] - 0s 69ms/step - loss: 1.5996 - accuracy:
0.4045 - val_loss: 1.6279 - val_accuracy: 0.2000
Epoch 58/300
7/7 [==============================] - 0s 69ms/step - loss: 1.7305 - accuracy:
0.3409 - val_loss: 1.6512 - val_accuracy: 0.2400
Epoch 59/300
7/7 [==============================] - 0s 69ms/step - loss: 1.5610 - accuracy:
0.4182 - val_loss: 1.7093 - val_accuracy: 0.2400
Epoch 60/300
7/7 [==============================] - 0s 69ms/step - loss: 1.6067 - accuracy:
0.3455 - val_loss: 1.7391 - val_accuracy: 0.2400
Epoch 61/300
7/7 [==============================] - 0s 69ms/step - loss: 1.7087 - accuracy:
0.3545 - val_loss: 1.7036 - val_accuracy: 0.2800
Epoch 62/300
7/7 [==============================] - 0s 69ms/step - loss: 1.5913 - accuracy:
0.4136 - val_loss: 1.6962 - val_accuracy: 0.2400
Epoch 63/300
7/7 [==============================] - 0s 69ms/step - loss: 1.7206 - accuracy:
0.3409 - val_loss: 1.7315 - val_accuracy: 0.1600
Epoch 64/300
7/7 [==============================] - 0s 69ms/step - loss: 1.6637 - accuracy:
0.3045 - val_loss: 1.6367 - val_accuracy: 0.2400
Epoch 65/300
7/7 [==============================] - 0s 69ms/step - loss: 1.6914 - accuracy:
0.2955 - val_loss: 1.6304 - val_accuracy: 0.2800
Epoch 66/300
7/7 [==============================] - 0s 69ms/step - loss: 1.6809 - accuracy:
0.3818 - val_loss: 1.6351 - val_accuracy: 0.2000
Epoch 67/300
7/7 [==============================] - 0s 69ms/step - loss: 1.5386 - accuracy:
0.3682 - val_loss: 1.6603 - val_accuracy: 0.2400
Epoch 68/300
```

```
7/7 [==============================] - 0s 68ms/step - loss: 1.5540 - accuracy:
0.3818 - val_loss: 1.6874 - val_accuracy: 0.2400
Epoch 69/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4593 - accuracy:
0.4000 - val_loss: 1.7226 - val_accuracy: 0.2800
Epoch 70/300
7/7 [==============================] - 0s 68ms/step - loss: 1.5472 - accuracy:
0.3727 - val_loss: 1.8096 - val_accuracy: 0.2000
Epoch 71/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4765 - accuracy:
0.3909 - val_loss: 1.7789 - val_accuracy: 0.2400
Epoch 72/300
7/7 [==============================] - 0s 68ms/step - loss: 1.5171 - accuracy:
0.3455 - val_loss: 1.7485 - val_accuracy: 0.2400
Epoch 73/300
7/7 [==============================] - 0s 69ms/step - loss: 1.5810 - accuracy:
0.4045 - val_loss: 1.7635 - val_accuracy: 0.2400
Epoch 74/300
7/7 [==============================] - 0s 69ms/step - loss: 1.6197 - accuracy:
0.3909 - val_loss: 1.6926 - val_accuracy: 0.2400
Epoch 75/300
7/7 [==============================] - 0s 69ms/step - loss: 1.5321 - accuracy:
0.3773 - val_loss: 1.6873 - val_accuracy: 0.2400
Epoch 76/300
7/7 [==============================] - 0s 69ms/step - loss: 1.6015 - accuracy:
0.4136 - val_loss: 1.7379 - val_accuracy: 0.2400
Epoch 77/300
7/7 [==============================] - 0s 70ms/step - loss: 1.7133 - accuracy:
0.3045 - val_loss: 1.6980 - val_accuracy: 0.2400
Epoch 78/300
7/7 [==============================] - 0s 70ms/step - loss: 1.4407 - accuracy:
0.3955 - val_loss: 1.6004 - val_accuracy: 0.2400
Epoch 79/300
7/7 [==============================] - 0s 68ms/step - loss: 1.6788 - accuracy:
0.3591 - val_loss: 1.6418 - val_accuracy: 0.2800
Epoch 80/300
7/7 [==============================] - 0s 69ms/step - loss: 1.5527 - accuracy:
0.3955 - val_loss: 1.6182 - val_accuracy: 0.2400
Epoch 81/300
7/7 [==============================] - 0s 68ms/step - loss: 1.5617 - accuracy:
0.3727 - val_loss: 1.6087 - val_accuracy: 0.2800
Epoch 82/300
7/7 [==============================] - 0s 68ms/step - loss: 1.6421 - accuracy:
0.3773 - val_loss: 1.6478 - val_accuracy: 0.2800
Epoch 83/300
7/7 [==============================] - 0s 68ms/step - loss: 1.5943 - accuracy:
0.3682 - val_loss: 1.7879 - val_accuracy: 0.2000
Epoch 84/300
```

```
7/7 [==============================] - 0s 68ms/step - loss: 1.4953 - accuracy:
0.3818 - val_loss: 1.8247 - val_accuracy: 0.2000
Epoch 85/300
7/7 [==============================] - 0s 68ms/step - loss: 1.5558 - accuracy:
0.3864 - val_loss: 1.8040 - val_accuracy: 0.2000
Epoch 86/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4805 - accuracy:
0.3909 - val_loss: 1.8053 - val_accuracy: 0.2400
Epoch 87/300
7/7 [==============================] - 0s 68ms/step - loss: 1.5226 - accuracy:
0.4045 - val_loss: 1.5593 - val_accuracy: 0.2400
Epoch 88/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4907 - accuracy:
0.4045 - val_loss: 1.5444 - val_accuracy: 0.2800
Epoch 89/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4636 - accuracy:
0.4000 - val_loss: 1.5812 - val_accuracy: 0.3200
Epoch 90/300
7/7 [==============================] - 0s 68ms/step - loss: 1.5038 - accuracy:
0.3955 - val_loss: 1.5520 - val_accuracy: 0.2800
Epoch 91/300
7/7 [==============================] - 0s 67ms/step - loss: 1.4493 - accuracy:
0.4364 - val_loss: 1.5874 - val_accuracy: 0.2000
Epoch 92/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4992 - accuracy:
0.3955 - val_loss: 1.6673 - val_accuracy: 0.2000
Epoch 93/300
7/7 [==============================] - 0s 68ms/step - loss: 1.5824 - accuracy:
0.3682 - val_loss: 1.5895 - val_accuracy: 0.2800
Epoch 94/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4693 - accuracy:
0.4136 - val_loss: 1.5558 - val_accuracy: 0.2400
Epoch 95/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4326 - accuracy:
0.4273 - val_loss: 1.6457 - val_accuracy: 0.2800
Epoch 96/300
7/7 [==============================] - 0s 68ms/step - loss: 1.5781 - accuracy:
0.3591 - val_loss: 1.5603 - val_accuracy: 0.3200
Epoch 97/300
7/7 [==============================] - 0s 67ms/step - loss: 1.4746 - accuracy:
0.4182 - val_loss: 1.6087 - val_accuracy: 0.2800
Epoch 98/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4553 - accuracy:
0.4227 - val_loss: 1.5862 - val_accuracy: 0.2800
Epoch 99/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4023 - accuracy:
0.4364 - val_loss: 1.5834 - val_accuracy: 0.2800
Epoch 100/300
```

```
7/7 [==============================] - 0s 68ms/step - loss: 1.3655 - accuracy:
0.4500 - val_loss: 1.6707 - val_accuracy: 0.2400
Epoch 101/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4379 - accuracy:
0.4455 - val_loss: 1.6264 - val_accuracy: 0.2800
Epoch 102/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4039 - accuracy:
0.3909 - val_loss: 1.5873 - val_accuracy: 0.3200
Epoch 103/300
7/7 [==============================] - 0s 68ms/step - loss: 1.3262 - accuracy:
0.4864 - val_loss: 1.5260 - val_accuracy: 0.2400
Epoch 104/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4729 - accuracy:
0.4091 - val_loss: 1.5380 - val_accuracy: 0.3200
Epoch 105/300
7/7 [==============================] - 0s 67ms/step - loss: 1.3355 - accuracy:
0.4545 - val_loss: 1.5124 - val_accuracy: 0.3200
Epoch 106/300
7/7 [==============================] - 0s 67ms/step - loss: 1.3600 - accuracy:
0.4500 - val_loss: 1.5358 - val_accuracy: 0.2800
Epoch 107/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2989 - accuracy:
0.4545 - val_loss: 1.6268 - val_accuracy: 0.2400
Epoch 108/300
7/7 [==============================] - 0s 67ms/step - loss: 1.3120 - accuracy:
0.4818 - val_loss: 1.7713 - val_accuracy: 0.2800
Epoch 109/300
7/7 [==============================] - 0s 68ms/step - loss: 1.4539 - accuracy:
0.4091 - val_loss: 1.6643 - val_accuracy: 0.3200
Epoch 110/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2914 - accuracy:
0.4773 - val_loss: 1.6394 - val_accuracy: 0.3200
Epoch 111/300
7/7 [==============================] - 0s 67ms/step - loss: 1.3359 - accuracy:
0.4864 - val_loss: 1.6957 - val_accuracy: 0.3200
Epoch 112/300
7/7 [==============================] - 0s 67ms/step - loss: 1.4438 - accuracy:
0.4000 - val_loss: 1.6964 - val_accuracy: 0.2400
Epoch 113/300
7/7 [==============================] - 0s 67ms/step - loss: 1.3986 - accuracy:
0.4273 - val_loss: 2.1158 - val_accuracy: 0.2400
Epoch 114/300
7/7 [==============================] - 0s 67ms/step - loss: 1.3443 - accuracy:
0.4864 - val_loss: 1.8158 - val_accuracy: 0.3600
Epoch 115/300
7/7 [==============================] - 0s 67ms/step - loss: 1.4229 - accuracy:
0.3773 - val_loss: 1.8561 - val_accuracy: 0.2400
Epoch 116/300
```

```
7/7 [==============================] - 0s 67ms/step - loss: 1.3018 - accuracy:
0.5045 - val_loss: 1.7219 - val_accuracy: 0.2400
Epoch 117/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2522 - accuracy:
0.4773 - val_loss: 1.8914 - val_accuracy: 0.2000
Epoch 118/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2122 - accuracy:
0.5045 - val_loss: 1.7148 - val_accuracy: 0.2400
Epoch 119/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2073 - accuracy:
0.4955 - val_loss: 1.8927 - val_accuracy: 0.2400
Epoch 120/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1862 - accuracy:
0.5364 - val_loss: 1.8413 - val_accuracy: 0.2000
Epoch 121/300
7/7 [==============================] - 0s 67ms/step - loss: 1.3877 - accuracy:
0.4545 - val_loss: 1.7433 - val_accuracy: 0.2800
Epoch 122/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1750 - accuracy:
0.5364 - val_loss: 1.7715 - val_accuracy: 0.2800
Epoch 123/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1559 - accuracy:
0.5636 - val_loss: 1.8298 - val_accuracy: 0.2400
Epoch 124/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1893 - accuracy:
0.4773 - val_loss: 1.6295 - val_accuracy: 0.3600
Epoch 125/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2762 - accuracy:
0.4955 - val_loss: 1.6107 - val_accuracy: 0.2800
Epoch 126/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1954 - accuracy:
0.5364 - val_loss: 1.8861 - val_accuracy: 0.2000
Epoch 127/300
7/7 [==============================] - 0s 68ms/step - loss: 1.2517 - accuracy:
0.5182 - val_loss: 1.6752 - val_accuracy: 0.2800
Epoch 128/300
7/7 [==============================] - 0s 67ms/step - loss: 1.3279 - accuracy:
0.4955 - val_loss: 1.7134 - val_accuracy: 0.2800
Epoch 129/300
7/7 [==============================] - 0s 69ms/step - loss: 1.2905 - accuracy:
0.4727 - val_loss: 2.5357 - val_accuracy: 0.1600
Epoch 130/300
7/7 [==============================] - 0s 68ms/step - loss: 1.2584 - accuracy:
0.5273 - val_loss: 1.8351 - val_accuracy: 0.2000
Epoch 131/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1599 - accuracy:
0.5318 - val_loss: 1.8226 - val_accuracy: 0.2400
Epoch 132/300
```

```
7/7 [==============================] - 0s 67ms/step - loss: 1.3043 - accuracy:
0.4500 - val_loss: 1.7480 - val_accuracy: 0.2400
Epoch 133/300
7/7 [==============================] - 0s 66ms/step - loss: 1.1232 - accuracy:
0.5409 - val_loss: 1.9201 - val_accuracy: 0.2800
Epoch 134/300
7/7 [==============================] - 0s 67ms/step - loss: 1.0812 - accuracy:
0.5409 - val_loss: 2.0513 - val_accuracy: 0.2000
Epoch 135/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2186 - accuracy:
0.5227 - val_loss: 1.8628 - val_accuracy: 0.2400
Epoch 136/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2281 - accuracy:
0.5000 - val_loss: 1.7447 - val_accuracy: 0.2800
Epoch 137/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1313 - accuracy:
0.5591 - val_loss: 1.7820 - val_accuracy: 0.3200
Epoch 138/300
7/7 [==============================] - 0s 67ms/step - loss: 1.0865 - accuracy:
0.5773 - val_loss: 1.7008 - val_accuracy: 0.2800
Epoch 139/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2094 - accuracy:
0.5000 - val_loss: 1.5449 - val_accuracy: 0.3200
Epoch 140/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1113 - accuracy:
0.5591 - val_loss: 1.6280 - val_accuracy: 0.3200
Epoch 141/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1560 - accuracy:
0.5455 - val_loss: 1.5506 - val_accuracy: 0.2400
Epoch 142/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1811 - accuracy:
0.4636 - val_loss: 1.6899 - val_accuracy: 0.2400
Epoch 143/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1336 - accuracy:
0.5273 - val_loss: 1.5505 - val_accuracy: 0.3200
Epoch 144/300
7/7 [==============================] - 0s 67ms/step - loss: 1.2123 - accuracy:
0.5273 - val_loss: 1.6040 - val_accuracy: 0.3200
Epoch 145/300
7/7 [==============================] - 0s 66ms/step - loss: 1.1939 - accuracy:
0.5409 - val_loss: 1.6172 - val_accuracy: 0.2800
Epoch 146/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1710 - accuracy:
0.5455 - val_loss: 1.6363 - val_accuracy: 0.2800
Epoch 147/300
7/7 [==============================] - 0s 66ms/step - loss: 1.0833 - accuracy:
0.5136 - val_loss: 1.5906 - val_accuracy: 0.3200
Epoch 148/300
```

```
7/7 [==============================] - 0s 67ms/step - loss: 1.1343 - accuracy:
0.5409 - val_loss: 1.6144 - val_accuracy: 0.2800
Epoch 149/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1715 - accuracy:
0.5182 - val_loss: 1.6996 - val_accuracy: 0.3200
Epoch 150/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1377 - accuracy:
0.5818 - val_loss: 1.8955 - val_accuracy: 0.3200
Epoch 151/300
7/7 [==============================] - 0s 67ms/step - loss: 1.0336 - accuracy:
0.5455 - val_loss: 1.8322 - val_accuracy: 0.2800
Epoch 152/300
7/7 [==============================] - 0s 66ms/step - loss: 1.1554 - accuracy:
0.5364 - val_loss: 1.8019 - val_accuracy: 0.3200
Epoch 153/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1022 - accuracy:
0.5409 - val_loss: 1.9292 - val_accuracy: 0.1600
Epoch 154/300
7/7 [==============================] - 0s 67ms/step - loss: 1.0582 - accuracy:
0.6045 - val_loss: 1.6914 - val_accuracy: 0.2400
Epoch 155/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9945 - accuracy:
0.6136 - val_loss: 1.8249 - val_accuracy: 0.3200
Epoch 156/300
7/7 [==============================] - 0s 68ms/step - loss: 1.0918 - accuracy:
0.5273 - val_loss: 2.9915 - val_accuracy: 0.1200
Epoch 157/300
7/7 [==============================] - 0s 68ms/step - loss: 1.1156 - accuracy:
0.5455 - val_loss: 2.1586 - val_accuracy: 0.1600
Epoch 158/300
7/7 [==============================] - 0s 67ms/step - loss: 1.0762 - accuracy:
0.5773 - val_loss: 2.1472 - val_accuracy: 0.1600
Epoch 159/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1511 - accuracy:
0.5455 - val_loss: 3.1667 - val_accuracy: 0.1200
Epoch 160/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1516 - accuracy:
0.4818 - val_loss: 1.8324 - val_accuracy: 0.2000
Epoch 161/300
7/7 [==============================] - 0s 67ms/step - loss: 1.0369 - accuracy:
0.5955 - val_loss: 1.8436 - val_accuracy: 0.2800
Epoch 162/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9851 - accuracy:
0.5909 - val_loss: 1.6386 - val_accuracy: 0.3200
Epoch 163/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9730 - accuracy:
0.5909 - val_loss: 1.8088 - val_accuracy: 0.2800
Epoch 164/300
```

```
7/7 [==============================] - 0s 66ms/step - loss: 0.9591 - accuracy:
0.6045 - val_loss: 2.1451 - val_accuracy: 0.2400
Epoch 165/300
7/7 [==============================] - 0s 67ms/step - loss: 1.1298 - accuracy:
0.5500 - val_loss: 2.0379 - val_accuracy: 0.2400
Epoch 166/300
7/7 [==============================] - 0s 66ms/step - loss: 1.0407 - accuracy:
0.6227 - val_loss: 1.5304 - val_accuracy: 0.3600
Epoch 167/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9225 - accuracy:
0.6227 - val_loss: 1.6654 - val_accuracy: 0.2800
Epoch 168/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9320 - accuracy:
0.6227 - val_loss: 1.9323 - val_accuracy: 0.2400
Epoch 169/300
7/7 [==============================] - 0s 67ms/step - loss: 1.0028 - accuracy:
0.6000 - val_loss: 1.6803 - val_accuracy: 0.2400
Epoch 170/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9841 - accuracy:
0.5773 - val_loss: 1.9734 - val_accuracy: 0.2400
Epoch 171/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9159 - accuracy:
0.5955 - val_loss: 1.7085 - val_accuracy: 0.2400
Epoch 172/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9843 - accuracy:
0.6136 - val_loss: 1.9240 - val_accuracy: 0.2800
Epoch 173/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9016 - accuracy:
0.6409 - val_loss: 1.5782 - val_accuracy: 0.3200
Epoch 174/300
7/7 [==============================] - 0s 66ms/step - loss: 0.9327 - accuracy:
0.6500 - val_loss: 1.7439 - val_accuracy: 0.3200
Epoch 175/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9567 - accuracy:
0.6455 - val_loss: 1.8791 - val_accuracy: 0.2800
Epoch 176/300
7/7 [==============================] - 0s 66ms/step - loss: 0.8572 - accuracy:
0.7045 - val_loss: 1.8683 - val_accuracy: 0.3600
Epoch 177/300
7/7 [==============================] - 0s 66ms/step - loss: 0.8623 - accuracy:
0.6727 - val_loss: 1.9761 - val_accuracy: 0.3200
Epoch 178/300
7/7 [==============================] - 0s 68ms/step - loss: 0.7943 - accuracy:
0.7000 - val_loss: 1.9867 - val_accuracy: 0.3200
Epoch 179/300
7/7 [==============================] - 0s 68ms/step - loss: 0.8684 - accuracy:
0.6409 - val_loss: 1.5786 - val_accuracy: 0.2800
Epoch 180/300
```

```
7/7 [==============================] - 0s 67ms/step - loss: 0.8899 - accuracy:
0.6636 - val_loss: 1.6726 - val_accuracy: 0.3200
Epoch 181/300
7/7 [==============================] - 0s 68ms/step - loss: 0.8265 - accuracy:
0.6636 - val_loss: 1.7219 - val_accuracy: 0.2800
Epoch 182/300
7/7 [==============================] - 0s 67ms/step - loss: 0.9050 - accuracy:
0.6273 - val_loss: 1.9595 - val_accuracy: 0.2400
Epoch 183/300
7/7 [==============================] - 0s 68ms/step - loss: 0.8234 - accuracy:
0.6727 - val_loss: 1.9054 - val_accuracy: 0.2800
Epoch 184/300
7/7 [==============================] - 0s 68ms/step - loss: 0.9069 - accuracy:
0.6182 - val_loss: 1.9870 - val_accuracy: 0.2400
Epoch 185/300
7/7 [==============================] - 0s 66ms/step - loss: 0.8807 - accuracy:
0.6364 - val_loss: 1.8462 - val_accuracy: 0.3600
Epoch 186/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7736 - accuracy:
0.6591 - val_loss: 1.5769 - val_accuracy: 0.3600
Epoch 187/300
7/7 [==============================] - 0s 67ms/step - loss: 0.8800 - accuracy:
0.6500 - val_loss: 1.5445 - val_accuracy: 0.3200
Epoch 188/300
7/7 [==============================] - 0s 66ms/step - loss: 0.8896 - accuracy:
0.6682 - val_loss: 1.7870 - val_accuracy: 0.3200
Epoch 189/300
7/7 [==============================] - 0s 67ms/step - loss: 0.8353 - accuracy:
0.6636 - val_loss: 1.8099 - val_accuracy: 0.3200
Epoch 190/300
7/7 [==============================] - 0s 67ms/step - loss: 0.8166 - accuracy:
0.6500 - val_loss: 1.6341 - val_accuracy: 0.3600
Epoch 191/300
7/7 [==============================] - 0s 67ms/step - loss: 0.8801 - accuracy:
0.6182 - val_loss: 1.7659 - val_accuracy: 0.3600
Epoch 192/300
7/7 [==============================] - 0s 67ms/step - loss: 0.8471 - accuracy:
0.6682 - val_loss: 1.5511 - val_accuracy: 0.4400
Epoch 193/300
7/7 [==============================] - 0s 67ms/step - loss: 0.8528 - accuracy:
0.6455 - val_loss: 1.5141 - val_accuracy: 0.3600
Epoch 194/300
7/7 [==============================] - 0s 67ms/step - loss: 0.8086 - accuracy:
0.6773 - val_loss: 1.7489 - val_accuracy: 0.3200
Epoch 195/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7320 - accuracy:
0.6773 - val_loss: 1.9051 - val_accuracy: 0.3600
Epoch 196/300
```

```
7/7 [==============================] - 0s 68ms/step - loss: 0.8643 - accuracy:
0.6227 - val_loss: 2.2192 - val_accuracy: 0.3200
Epoch 197/300
7/7 [==============================] - 0s 67ms/step - loss: 0.8028 - accuracy:
0.6955 - val_loss: 2.1413 - val_accuracy: 0.2800
Epoch 198/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7639 - accuracy:
0.7000 - val_loss: 1.6376 - val_accuracy: 0.4000
Epoch 199/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7567 - accuracy:
0.6727 - val_loss: 2.6906 - val_accuracy: 0.2000
Epoch 200/300
7/7 [==============================] - 0s 67ms/step - loss: 0.8302 - accuracy:
0.7000 - val_loss: 1.8444 - val_accuracy: 0.3600
Epoch 201/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7551 - accuracy:
0.6773 - val_loss: 1.7934 - val_accuracy: 0.3200
Epoch 202/300
7/7 [==============================] - 0s 68ms/step - loss: 0.7991 - accuracy:
0.6636 - val_loss: 2.2709 - val_accuracy: 0.1600
Epoch 203/300
7/7 [==============================] - 0s 66ms/step - loss: 0.7522 - accuracy:
0.7455 - val_loss: 2.6061 - val_accuracy: 0.2800
Epoch 204/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7720 - accuracy:
0.7091 - val_loss: 2.4891 - val_accuracy: 0.2400
Epoch 205/300
7/7 [==============================] - 0s 66ms/step - loss: 0.7549 - accuracy:
0.7136 - val_loss: 2.4484 - val_accuracy: 0.1600
Epoch 206/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7681 - accuracy:
0.7045 - val_loss: 1.7644 - val_accuracy: 0.3600
Epoch 207/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7626 - accuracy:
0.6636 - val_loss: 2.0330 - val_accuracy: 0.2400
Epoch 208/300
7/7 [==============================] - 0s 68ms/step - loss: 0.7391 - accuracy:
0.7091 - val_loss: 1.8179 - val_accuracy: 0.3200
Epoch 209/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6744 - accuracy:
0.7136 - val_loss: 1.7788 - val_accuracy: 0.3600
Epoch 210/300
7/7 [==============================] - 0s 69ms/step - loss: 0.7402 - accuracy:
0.7318 - val_loss: 1.7198 - val_accuracy: 0.4000
Epoch 211/300
7/7 [==============================] - 0s 68ms/step - loss: 0.7195 - accuracy:
0.7045 - val_loss: 2.0915 - val_accuracy: 0.2400
Epoch 212/300
```

```
7/7 [==============================] - 0s 73ms/step - loss: 0.6786 - accuracy:
0.7500 - val_loss: 1.9562 - val_accuracy: 0.2400
Epoch 213/300
7/7 [==============================] - 0s 68ms/step - loss: 0.7289 - accuracy:
0.7091 - val_loss: 1.8398 - val_accuracy: 0.2800
Epoch 214/300
7/7 [==============================] - 0s 67ms/step - loss: 0.6686 - accuracy:
0.7500 - val_loss: 2.2887 - val_accuracy: 0.2400
Epoch 215/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6437 - accuracy:
0.7727 - val_loss: 1.9229 - val_accuracy: 0.3600
Epoch 216/300
7/7 [==============================] - 0s 67ms/step - loss: 0.6857 - accuracy:
0.7000 - val_loss: 3.3676 - val_accuracy: 0.4400
Epoch 217/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6666 - accuracy:
0.7455 - val_loss: 2.5424 - val_accuracy: 0.3600
Epoch 218/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7025 - accuracy:
0.7318 - val_loss: 2.8682 - val_accuracy: 0.4000
Epoch 219/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6998 - accuracy:
0.7227 - val_loss: 3.2045 - val_accuracy: 0.4000
Epoch 220/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5913 - accuracy:
0.7773 - val_loss: 2.4943 - val_accuracy: 0.3600
Epoch 221/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6646 - accuracy:
0.7455 - val_loss: 3.3101 - val_accuracy: 0.2400
Epoch 222/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6696 - accuracy:
0.7409 - val_loss: 2.5283 - val_accuracy: 0.3600
Epoch 223/300
7/7 [==============================] - 0s 67ms/step - loss: 0.6819 - accuracy:
0.7545 - val_loss: 2.7799 - val_accuracy: 0.4000
Epoch 224/300
7/7 [==============================] - 0s 67ms/step - loss: 0.5922 - accuracy:
0.8000 - val_loss: 3.0848 - val_accuracy: 0.3600
Epoch 225/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7104 - accuracy:
0.7227 - val_loss: 3.0037 - val_accuracy: 0.3600
Epoch 226/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6942 - accuracy:
0.7273 - val_loss: 2.6030 - val_accuracy: 0.3600
Epoch 227/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5626 - accuracy:
0.7773 - val_loss: 2.9054 - val_accuracy: 0.4000
Epoch 228/300
```

```
7/7 [==============================] - 0s 67ms/step - loss: 0.5497 - accuracy:
0.8091 - val_loss: 2.5817 - val_accuracy: 0.4000
Epoch 229/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6237 - accuracy:
0.7409 - val_loss: 2.8744 - val_accuracy: 0.2800
Epoch 230/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5856 - accuracy:
0.7636 - val_loss: 2.5052 - val_accuracy: 0.4000
Epoch 231/300
7/7 [==============================] - 0s 67ms/step - loss: 0.6085 - accuracy:
0.7818 - val_loss: 2.0584 - val_accuracy: 0.3200
Epoch 232/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6196 - accuracy:
0.7455 - val_loss: 1.7718 - val_accuracy: 0.3600
Epoch 233/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6067 - accuracy:
0.7591 - val_loss: 2.0745 - val_accuracy: 0.4800
Epoch 234/300
7/7 [==============================] - 0s 70ms/step - loss: 0.5878 - accuracy:
0.7864 - val_loss: 2.1099 - val_accuracy: 0.2800
Epoch 235/300
7/7 [==============================] - 0s 69ms/step - loss: 0.5614 - accuracy:
0.8091 - val_loss: 2.1154 - val_accuracy: 0.3200
Epoch 236/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6089 - accuracy:
0.7909 - val_loss: 2.1661 - val_accuracy: 0.4000
Epoch 237/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5208 - accuracy:
0.8045 - val_loss: 2.8012 - val_accuracy: 0.2400
Epoch 238/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5331 - accuracy:
0.7955 - val_loss: 2.4172 - val_accuracy: 0.2400
Epoch 239/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6436 - accuracy:
0.7682 - val_loss: 2.6685 - val_accuracy: 0.2800
Epoch 240/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6230 - accuracy:
0.7909 - val_loss: 2.7709 - val_accuracy: 0.2800
Epoch 241/300
7/7 [==============================] - 0s 67ms/step - loss: 0.6378 - accuracy:
0.7455 - val_loss: 2.3460 - val_accuracy: 0.4400
Epoch 242/300
7/7 [==============================] - 0s 67ms/step - loss: 0.6000 - accuracy:
0.8182 - val_loss: 2.1380 - val_accuracy: 0.3600
Epoch 243/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5033 - accuracy:
0.8091 - val_loss: 1.9569 - val_accuracy: 0.4000
Epoch 244/300
```

```
7/7 [==============================] - 0s 69ms/step - loss: 0.5091 - accuracy:
0.8136 - val_loss: 2.0847 - val_accuracy: 0.4800
Epoch 245/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5662 - accuracy:
0.7727 - val_loss: 1.8345 - val_accuracy: 0.3600
Epoch 246/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5098 - accuracy:
0.8091 - val_loss: 1.8379 - val_accuracy: 0.4400
Epoch 247/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4531 - accuracy:
0.8773 - val_loss: 1.7674 - val_accuracy: 0.2800
Epoch 248/300
7/7 [==============================] - 0s 73ms/step - loss: 0.4549 - accuracy:
0.8409 - val_loss: 1.8609 - val_accuracy: 0.3600
Epoch 249/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4591 - accuracy:
0.8318 - val_loss: 2.2620 - val_accuracy: 0.3600
Epoch 250/300
7/7 [==============================] - 0s 71ms/step - loss: 0.5449 - accuracy:
0.8227 - val_loss: 1.7957 - val_accuracy: 0.4400
Epoch 251/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5248 - accuracy:
0.8182 - val_loss: 1.9608 - val_accuracy: 0.4400
Epoch 252/300
7/7 [==============================] - 0s 67ms/step - loss: 0.4451 - accuracy:
0.8455 - val_loss: 1.8411 - val_accuracy: 0.4000
Epoch 253/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4286 - accuracy:
0.8545 - val_loss: 1.8063 - val_accuracy: 0.4000
Epoch 254/300
7/7 [==============================] - 0s 67ms/step - loss: 0.5087 - accuracy:
0.8182 - val_loss: 1.8853 - val_accuracy: 0.3200
Epoch 255/300
7/7 [==============================] - 0s 67ms/step - loss: 0.4404 - accuracy:
0.8409 - val_loss: 1.8326 - val_accuracy: 0.4400
Epoch 256/300
7/7 [==============================] - 0s 67ms/step - loss: 0.5049 - accuracy:
0.8273 - val_loss: 1.9749 - val_accuracy: 0.4400
Epoch 257/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4160 - accuracy:
0.8773 - val_loss: 1.9455 - val_accuracy: 0.4400
Epoch 258/300
7/7 [==============================] - 0s 69ms/step - loss: 0.4880 - accuracy:
0.8318 - val_loss: 2.2119 - val_accuracy: 0.2800
Epoch 259/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4159 - accuracy:
0.8409 - val_loss: 2.0618 - val_accuracy: 0.4000
Epoch 260/300
```

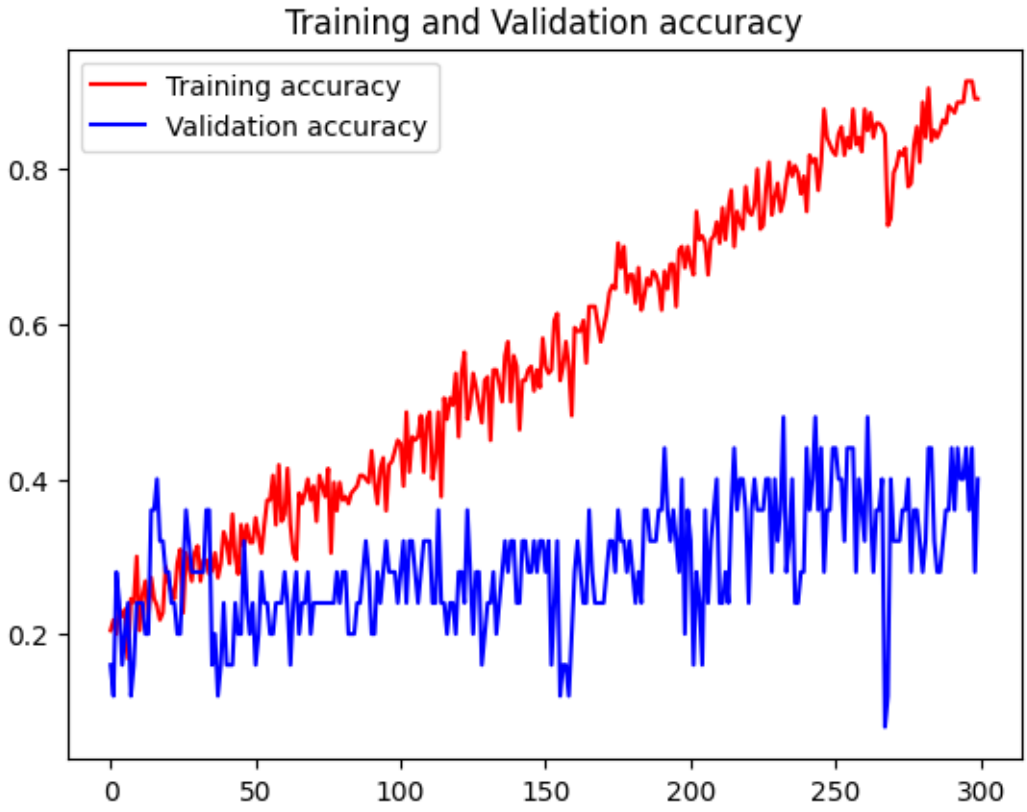
```
7/7 [==============================] - 0s 68ms/step - loss: 0.4770 - accuracy:
0.8227 - val_loss: 2.2216 - val_accuracy: 0.3600
Epoch 261/300
7/7 [==============================] - 0s 69ms/step - loss: 0.3568 - accuracy:
0.8773 - val_loss: 2.0769 - val_accuracy: 0.3200
Epoch 262/300
7/7 [==============================] - 0s 69ms/step - loss: 0.4170 - accuracy:
0.8500 - val_loss: 2.1767 - val_accuracy: 0.4800
Epoch 263/300
7/7 [==============================] - 0s 68ms/step - loss: 0.3794 - accuracy:
0.8727 - val_loss: 2.0623 - val_accuracy: 0.3600
Epoch 264/300
7/7 [==============================] - 0s 67ms/step - loss: 0.4241 - accuracy:
0.8409 - val_loss: 2.6790 - val_accuracy: 0.2800
Epoch 265/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4250 - accuracy:
0.8591 - val_loss: 2.0654 - val_accuracy: 0.3600
Epoch 266/300
7/7 [==============================] - 0s 67ms/step - loss: 0.4189 - accuracy:
0.8591 - val_loss: 2.8043 - val_accuracy: 0.3600
Epoch 267/300
7/7 [==============================] - 0s 67ms/step - loss: 0.4291 - accuracy:
0.8545 - val_loss: 1.8909 - val_accuracy: 0.4000
Epoch 268/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4982 - accuracy:
0.8455 - val_loss: 10.4052 - val_accuracy: 0.0800
Epoch 269/300
7/7 [==============================] - 0s 67ms/step - loss: 0.7255 - accuracy:
0.7273 - val_loss: 4.1760 - val_accuracy: 0.1200
Epoch 270/300
7/7 [==============================] - 0s 68ms/step - loss: 0.6801 - accuracy:
0.7364 - val_loss: 1.7914 - val_accuracy: 0.4000
Epoch 271/300
7/7 [==============================] - 0s 68ms/step - loss: 0.5695 - accuracy:
0.7955 - val_loss: 1.9094 - val_accuracy: 0.3200
Epoch 272/300
7/7 [==============================] - 0s 67ms/step - loss: 0.5331 - accuracy:
0.8045 - val_loss: 1.8920 - val_accuracy: 0.3200
Epoch 273/300
7/7 [==============================] - 0s 67ms/step - loss: 0.5326 - accuracy:
0.8227 - val_loss: 1.9971 - val_accuracy: 0.3200
Epoch 274/300
7/7 [==============================] - 0s 67ms/step - loss: 0.4968 - accuracy:
0.8182 - val_loss: 1.8892 - val_accuracy: 0.3600
Epoch 275/300
7/7 [==============================] - 0s 67ms/step - loss: 0.5387 - accuracy:
0.8273 - val_loss: 1.9728 - val_accuracy: 0.3600
Epoch 276/300
```

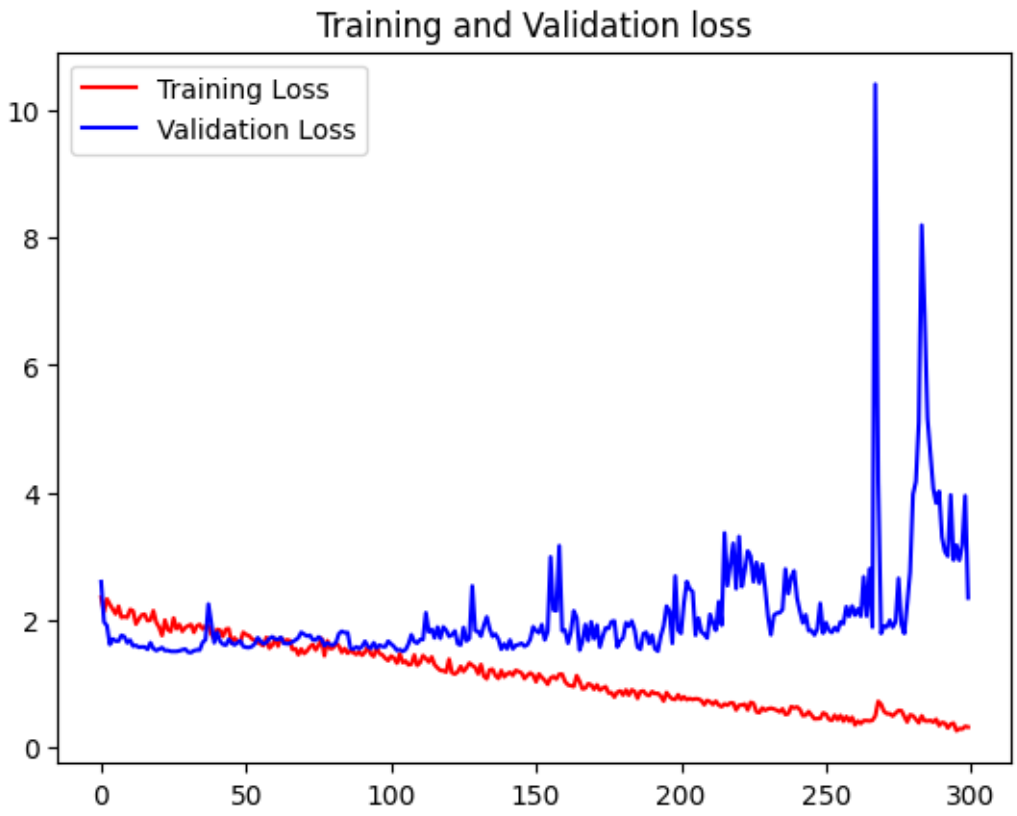
```
7/7 [==============================] - 0s 67ms/step - loss: 0.5840 - accuracy:
0.7773 - val_loss: 2.6558 - val_accuracy: 0.4000
Epoch 277/300
7/7 [==============================] - 0s 67ms/step - loss: 0.5791 - accuracy:
0.7818 - val_loss: 1.9348 - val_accuracy: 0.2800
Epoch 278/300
7/7 [==============================] - 0s 67ms/step - loss: 0.4900 - accuracy:
0.8318 - val_loss: 1.7914 - val_accuracy: 0.3600
Epoch 279/300
7/7 [==============================] - 0s 67ms/step - loss: 0.4033 - accuracy:
0.8545 - val_loss: 2.2553 - val_accuracy: 0.3600
Epoch 280/300
7/7 [==============================] - 0s 67ms/step - loss: 0.5147 - accuracy:
0.8091 - val_loss: 2.7541 - val_accuracy: 0.3200
Epoch 281/300
7/7 [==============================] - 0s 67ms/step - loss: 0.5012 - accuracy:
0.8864 - val_loss: 3.9728 - val_accuracy: 0.2800
Epoch 282/300
7/7 [==============================] - 0s 66ms/step - loss: 0.4454 - accuracy:
0.8409 - val_loss: 4.1742 - val_accuracy: 0.3200
Epoch 283/300
7/7 [==============================] - 0s 68ms/step - loss: 0.3906 - accuracy:
0.9045 - val_loss: 5.1058 - val_accuracy: 0.4400
Epoch 284/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4995 - accuracy:
0.8364 - val_loss: 8.1952 - val_accuracy: 0.4400
Epoch 285/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4228 - accuracy:
0.8500 - val_loss: 6.7980 - val_accuracy: 0.3200
Epoch 286/300
7/7 [==============================] - 0s 69ms/step - loss: 0.4185 - accuracy:
0.8409 - val_loss: 5.1643 - val_accuracy: 0.2800
Epoch 287/300
7/7 [==============================] - 0s 69ms/step - loss: 0.4274 - accuracy:
0.8500 - val_loss: 4.6116 - val_accuracy: 0.2800
Epoch 288/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4011 - accuracy:
0.8636 - val_loss: 4.0635 - val_accuracy: 0.3200
Epoch 289/300
7/7 [==============================] - 0s 68ms/step - loss: 0.4442 - accuracy:
0.8591 - val_loss: 3.8357 - val_accuracy: 0.3600
Epoch 290/300
7/7 [==============================] - 0s 68ms/step - loss: 0.3479 - accuracy:
0.8818 - val_loss: 4.0194 - val_accuracy: 0.3600
Epoch 291/300
7/7 [==============================] - 0s 68ms/step - loss: 0.3948 - accuracy:
0.8773 - val_loss: 3.3023 - val_accuracy: 0.4400
Epoch 292/300
```

```
7/7 [==============================] - 0s 67ms/step - loss: 0.3835 - accuracy:
0.8727 - val_loss: 3.0774 - val_accuracy: 0.3600
Epoch 293/300
7/7 [==============================] - 0s 67ms/step - loss: 0.3091 - accuracy:
0.8864 - val_loss: 3.0040 - val_accuracy: 0.4400
Epoch 294/300
7/7 [==============================] - 0s 67ms/step - loss: 0.3692 - accuracy:
0.8864 - val_loss: 3.9628 - val_accuracy: 0.4000
Epoch 295/300
7/7 [==============================] - 0s 68ms/step - loss: 0.3834 - accuracy:
0.8864 - val_loss: 2.9436 - val_accuracy: 0.4000
Epoch 296/300
7/7 [==============================] - 0s 67ms/step - loss: 0.2650 - accuracy:
0.9136 - val_loss: 3.1776 - val_accuracy: 0.4400
Epoch 297/300
7/7 [==============================] - 0s 68ms/step - loss: 0.2982 - accuracy:
0.9136 - val_loss: 2.9385 - val_accuracy: 0.3600
Epoch 298/300
7/7 [==============================] - 0s 67ms/step - loss: 0.2924 - accuracy:
0.9136 - val_loss: 3.1804 - val_accuracy: 0.4400
Epoch 299/300
7/7 [==============================] - 0s 67ms/step - loss: 0.3337 - accuracy:
0.8909 - val_loss: 3.9510 - val_accuracy: 0.2800
Epoch 300/300
7/7 [==============================] - 0s 67ms/step - loss: 0.3228 - accuracy:
0.8909 - val_loss: 2.3494 - val_accuracy: 0.4000
```

[ ]:
```python
import matplotlib.pyplot as plt
# Plot the chart for accuracy and loss on both training and validation
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation loss')
plt.legend()
```

[ ]: <matplotlib.legend.Legend at 0x7f040f4504f0>





[ ]:
```
model.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 36ms/step - loss: 2.3494 - accuracy:
0.4000
```

[ ]:
```
[2.349395513534546, 0.4000000059604645]
```

9.5 Tunning 5

[ ]:
```
import keras

from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

import tensorflow as tf
from tensorflow import keras
import keras.layers as layers

model1 = keras.Sequential()
model1.add(layers.Conv2D(filters=16, kernel_size=(3, 3),
                         strides=(1, 1), activation="relu", padding="same",
```

```
                        input_shape=(64, 64, 3)))
model1.add(layers.BatchNormalization())
model1.add(layers.MaxPool2D(pool_size=(2, 2), strides= (2, 2), padding="valid"))

model1.add(layers.Conv2D(filters=16, kernel_size=(3, 3),
                        strides=(1, 1), activation="relu",
                        padding="same"))
model1.add(layers.BatchNormalization())
model1.add(layers.MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model1.add(layers.Conv2D(filters=32, kernel_size=(2, 2),
                        strides=(2, 2), activation="sigmoid",
                        padding="same"))
model1.add(layers.BatchNormalization())
model1.add(layers.Flatten())

model1.add(layers.Dense(1000, activation="sigmoid"))
model1.add(layers.Dropout(0.5))

model1.add(layers.Dense(1000, activation="sigmoid"))
model1.add(layers.Dropout(0.5))

model1.add(layers.Dense(5, activation="softmax"))


optimizer = keras.optimizers.SGD(learning_rate=0.0009)
model1.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
model1.summary()
```

```
Model: "sequential_47"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_175 (Conv2D)         (None, 64, 64, 16)        448

 batch_normalization_77 (Bat  (None, 64, 64, 16)       64
 chNormalization)

 max_pooling2d_113 (MaxPooli  (None, 32, 32, 16)       0
 ng2D)

 conv2d_176 (Conv2D)         (None, 32, 32, 16)        2320

 batch_normalization_78 (Bat  (None, 32, 32, 16)       64
 chNormalization)
```

```
max_pooling2d_114 (MaxPooli  (None, 16, 16, 16)       0
ng2D)

conv2d_177 (Conv2D)          (None, 8, 8, 32)         2080

batch_normalization_79 (Bat  (None, 8, 8, 32)         128
chNormalization)

flatten_46 (Flatten)         (None, 2048)             0

dense_98 (Dense)             (None, 1000)             2049000

dropout_39 (Dropout)         (None, 1000)             0

dense_99 (Dense)             (None, 1000)             1001000

dropout_40 (Dropout)         (None, 1000)             0

dense_100 (Dense)            (None, 5)                5005

=================================================================
Total params: 3,060,109
Trainable params: 3,059,981
Non-trainable params: 128
_________________________________________________________________
```

[ ]:
```
history = model1.fit(X_train,y_train, epochs=50, batch_size=10,verbose=1,␣
↪validation_data=(X_val, y_val))
model1.evaluate(X_test, y_test)
```

```
Epoch 1/50
20/20 [==============================] - 0s 8ms/step - loss: 1.7183 - accuracy:
0.2424 - val_loss: 1.5989 - val_accuracy: 0.3182
Epoch 2/50
20/20 [==============================] - 0s 6ms/step - loss: 1.8266 - accuracy:
0.1616 - val_loss: 1.6040 - val_accuracy: 0.2273
Epoch 3/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7929 - accuracy:
0.2020 - val_loss: 1.6002 - val_accuracy: 0.2273
Epoch 4/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7074 - accuracy:
0.2273 - val_loss: 1.5992 - val_accuracy: 0.3636
Epoch 5/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7418 - accuracy:
0.2626 - val_loss: 1.6017 - val_accuracy: 0.3636
Epoch 6/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7242 - accuracy:
```

```
0.1970 - val_loss: 1.5991 - val_accuracy: 0.2273
Epoch 7/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7130 - accuracy:
0.2424 - val_loss: 1.5954 - val_accuracy: 0.2727
Epoch 8/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7572 - accuracy:
0.1970 - val_loss: 1.5937 - val_accuracy: 0.2727
Epoch 9/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6475 - accuracy:
0.3384 - val_loss: 1.6000 - val_accuracy: 0.2273
Epoch 10/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6288 - accuracy:
0.2677 - val_loss: 1.5981 - val_accuracy: 0.3182
Epoch 11/50
20/20 [==============================] - 0s 7ms/step - loss: 1.7607 - accuracy:
0.2020 - val_loss: 1.5938 - val_accuracy: 0.2273
Epoch 12/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6957 - accuracy:
0.2626 - val_loss: 1.5930 - val_accuracy: 0.3182
Epoch 13/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7548 - accuracy:
0.2020 - val_loss: 1.5920 - val_accuracy: 0.4091
Epoch 14/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6618 - accuracy:
0.2778 - val_loss: 1.5975 - val_accuracy: 0.2273
Epoch 15/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6662 - accuracy:
0.2828 - val_loss: 1.5980 - val_accuracy: 0.0909
Epoch 16/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7372 - accuracy:
0.2273 - val_loss: 1.5909 - val_accuracy: 0.2273
Epoch 17/50
20/20 [==============================] - 0s 7ms/step - loss: 1.7522 - accuracy:
0.2071 - val_loss: 1.5912 - val_accuracy: 0.2273
Epoch 18/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7056 - accuracy:
0.2424 - val_loss: 1.5912 - val_accuracy: 0.1818
Epoch 19/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6926 - accuracy:
0.2677 - val_loss: 1.5953 - val_accuracy: 0.3182
Epoch 20/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7055 - accuracy:
0.2778 - val_loss: 1.5945 - val_accuracy: 0.1364
Epoch 21/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7118 - accuracy:
0.2879 - val_loss: 1.5934 - val_accuracy: 0.1818
Epoch 22/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7182 - accuracy:
```

```
0.2222 - val_loss: 1.5962 - val_accuracy: 0.2727
Epoch 23/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7601 - accuracy:
0.2020 - val_loss: 1.5934 - val_accuracy: 0.3182
Epoch 24/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6669 - accuracy:
0.2525 - val_loss: 1.5915 - val_accuracy: 0.4091
Epoch 25/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7562 - accuracy:
0.2475 - val_loss: 1.5881 - val_accuracy: 0.3182
Epoch 26/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6792 - accuracy:
0.2475 - val_loss: 1.5924 - val_accuracy: 0.2727
Epoch 27/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6254 - accuracy:
0.2424 - val_loss: 1.5898 - val_accuracy: 0.4091
Epoch 28/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6254 - accuracy:
0.2677 - val_loss: 1.5918 - val_accuracy: 0.4091
Epoch 29/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6971 - accuracy:
0.2727 - val_loss: 1.5876 - val_accuracy: 0.4091
Epoch 30/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6913 - accuracy:
0.2424 - val_loss: 1.5909 - val_accuracy: 0.1818
Epoch 31/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6502 - accuracy:
0.2475 - val_loss: 1.5863 - val_accuracy: 0.1818
Epoch 32/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6675 - accuracy:
0.2727 - val_loss: 1.5881 - val_accuracy: 0.4091
Epoch 33/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6346 - accuracy:
0.2980 - val_loss: 1.5816 - val_accuracy: 0.3182
Epoch 34/50
20/20 [==============================] - 0s 7ms/step - loss: 1.6351 - accuracy:
0.2879 - val_loss: 1.5849 - val_accuracy: 0.3636
Epoch 35/50
20/20 [==============================] - 0s 7ms/step - loss: 1.7539 - accuracy:
0.1970 - val_loss: 1.5860 - val_accuracy: 0.2273
Epoch 36/50
20/20 [==============================] - 0s 7ms/step - loss: 1.6556 - accuracy:
0.2980 - val_loss: 1.5905 - val_accuracy: 0.2273
Epoch 37/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7038 - accuracy:
0.2374 - val_loss: 1.5847 - val_accuracy: 0.2273
Epoch 38/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6740 - accuracy:
```

```
0.2323 - val_loss: 1.5872 - val_accuracy: 0.1364
Epoch 39/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7163 - accuracy:
0.2374 - val_loss: 1.5817 - val_accuracy: 0.2727
Epoch 40/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6697 - accuracy:
0.2424 - val_loss: 1.5821 - val_accuracy: 0.2727
Epoch 41/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6585 - accuracy:
0.2475 - val_loss: 1.5830 - val_accuracy: 0.0909
Epoch 42/50
20/20 [==============================] - 0s 7ms/step - loss: 1.7227 - accuracy:
0.1970 - val_loss: 1.5840 - val_accuracy: 0.3636
Epoch 43/50
20/20 [==============================] - 0s 7ms/step - loss: 1.6155 - accuracy:
0.3030 - val_loss: 1.5827 - val_accuracy: 0.1818
Epoch 44/50
20/20 [==============================] - 0s 6ms/step - loss: 1.7101 - accuracy:
0.2677 - val_loss: 1.5826 - val_accuracy: 0.3182
Epoch 45/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6248 - accuracy:
0.2626 - val_loss: 1.5791 - val_accuracy: 0.2727
Epoch 46/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6913 - accuracy:
0.2525 - val_loss: 1.5811 - val_accuracy: 0.3182
Epoch 47/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6395 - accuracy:
0.2677 - val_loss: 1.5795 - val_accuracy: 0.1818
Epoch 48/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6164 - accuracy:
0.2374 - val_loss: 1.5768 - val_accuracy: 0.3182
Epoch 49/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6453 - accuracy:
0.2424 - val_loss: 1.5747 - val_accuracy: 0.2727
Epoch 50/50
20/20 [==============================] - 0s 6ms/step - loss: 1.6605 - accuracy:
0.2727 - val_loss: 1.5786 - val_accuracy: 0.3182
1/1 [==============================] - 0s 22ms/step - loss: 1.5658 - accuracy:
0.4000
```

[ ]:
```
[1.565824270248413, 0.4000000059604645]
```

[ ]:
```
loss_val1, accuracy_val1 = model1.evaluate(X_val,y_val)
loss_train1, accuracy_train1 = model1.evaluate(X_train,y_train)
loss_test1, accuracy_test1 = model1.evaluate(X_test, y_test)
print(f"Validation loss:{loss_val1:.2f}")
print(f"Training loss:{loss_train1:.2f}")
```

```
print(f"Test loss:{loss_test1:.2f}")

print(f"Validation Acc:{accuracy_val1:.2f}")
print(f"Training Acc:{accuracy_train1:.2f}")
print(f"Test Acc:{accuracy_test1:.2f}")
```

```
1/1 [==============================] - 0s 26ms/step - loss: 1.5786 - accuracy:
0.3182
7/7 [==============================] - 0s 3ms/step - loss: 1.4589 - accuracy:
0.5707
1/1 [==============================] - 0s 22ms/step - loss: 1.5658 - accuracy:
0.4000
Validation loss:1.58
Training loss:1.46
Test loss:1.57
Validation Acc:0.32
Training Acc:0.57
Test Acc:0.40
```