Finding The Original Image

By: Alexander Ogay (6380727)

Project

Get multiple edited images and go through them all.

Returning the original unedited image.

Normal Implementation

Use a for loop each for running through X coordinates and Y coordinates.

Compare pixels.

Add the pixel to an updating dict.

Find the most occurring pixel and return it.

Show the unedited image.

```
# Normal Implementation
start_time = time.time()
for i in range(numX): # Runthrough X
  for j in range(numY): # Runthrough Y
    k = 0
    lst = {} # Make a dict that keeps track of all images[X][Y]
    for image in images:
      tick = 0 # Duplicate Checker
      for check in lst:
        if np.array_equal(lst[check], image[i][j]) == True: # Is there a duplicate
          checker = check
          tick = 1
          continue
      if tick == 1: # Update the dict
        lst[(checker[0], checker[1]+1)] = lst.pop(checker)
      else: # Add a new pixel to the dict
        lst[(k, 1)] = image[i][j]
        k+=1
    max_key = max(lst, key=lambda x: x[1]) # Find the most occuring pixel
    original[i][j] = lst[max_key] # Replace the pixel

end_time = time.time()
elapsed_time = end_time - start_time
print(f"Function took {elapsed_time} seconds to run.")
plt.imshow(original)
plt.figure()
```

Futures

Use Futures in order to add concurrency and help decrease the runtime.

Multiple implementations:

X

```
# Use Futures to run through X
def process(pictures, x, y):
  a = []
  for j in range(y):
    k = 0
    lst = {}
    for picture in pictures:
      tick = 0
      for check in lst:
        if np.array_equal(lst[check], picture[x][j]) == True:
          checker = check
          tick = 1
          continue
      if tick == 1:
        lst[(checker[0], checker[1]+1)] = lst.pop(checker)
      else:
        lst[(k, 1)] = picture[x][j]
        k+=1
    max_key = max(lst, key=lambda x: x[1])
    a.append((lst[max_key], x, j))
  return a

with ThreadPoolExecutor() as executor:
  start_time = time.time()
  futures = [executor.submit(process, images, i, numY) for i in range(numX)]
  for future in as_completed(futures):
      fut = future.result()
      for f in fut:
        original[f[1]][f[2]] = f[0]

end_time = time.time()
elapsed_time = end_time - start_time
print(f"Function took {elapsed_time} seconds to run.")
plt.imshow(original)
plt.figure()
```

Y

```
# Use Futures to run through Y
def process(pictures, x, y):
  k = 0
  lst = {}
  for picture in pictures:
    tick = 0
    for check in lst:
      if np.array_equal(lst[check], picture[x][y]) == True:
        checker = check
        tick = 1
        continue
    if tick == 1:
      lst[(checker[0], checker[1]+1)] = lst.pop(checker)
    else:
      lst[(k, 1)] = picture[x][y]
      k+=1
  max_key = max(lst, key=lambda x: x[1])
  return (lst[max_key], x, y)

with ThreadPoolExecutor() as executor:
  start_time = time.time()
  for i in range(numX):
      futures = [executor.submit(process, images, i, j) for j in range(numY)]
      for future in as_completed(futures):
        a = future.result()
        original[a[1]][a[2]] = a[0]

end_time = time.time()
elapsed_time = end_time - start_time
print(f"Function took {elapsed_time} seconds to run.")
plt.imshow(original)
plt.figure()
```

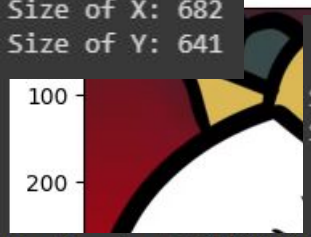
Comparison

```
# Use Futures for image comparison
def process(pictures, x, y):
  k = 0
  lst = {}
  for picture in pictures:
    tick = 0
    for check in lst:
      if np.array_equal(lst[check], picture[x][y]) == True:
        checker = check
        tick = 1
        continue
    if tick == 1:
      lst[(checker[0], checker[1]+1)] = lst.pop(checker)
    else:
      lst[(k, 1)] = picture[x][y]
      k+=1
  max_key = max(lst, key=lambda x: x[1])
  return (lst[max_key], x, y)

with ThreadPoolExecutor() as executor:
  start_time = time.time()
  for i in range(numX):
    for j in range(numY):
      future = executor.submit(process, images, i, j)
      a = future.result()
      original[a[1]][a[2]] = a[0]

end_time = time.time()
elapsed_time = end_time - start_time
print(f"Function took {elapsed_time} seconds to run.")
plt.imshow(original)
plt.figure()
```

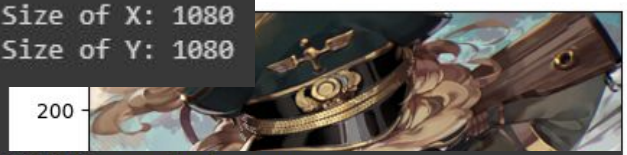
Conclusion

The runtimes of different images show that depending on the image, runtime can be very close between not using futures and using it. Therefore the choice of using futures depends on size of the image and the number of images used.

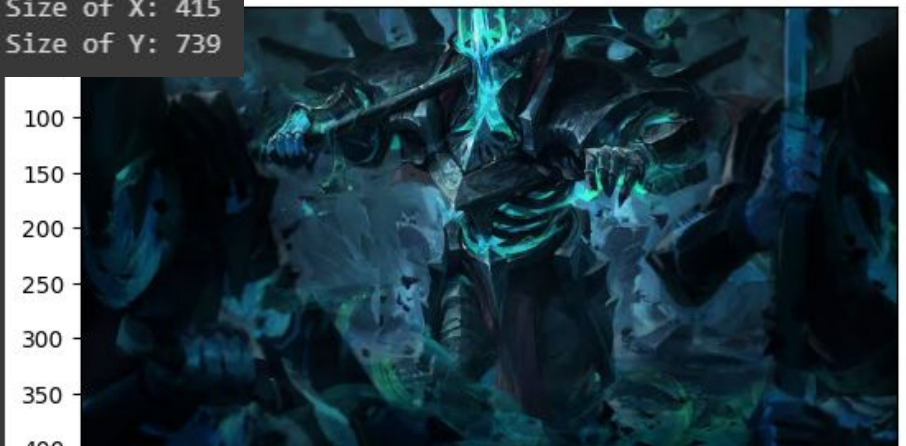
```
Function took 10.938900232315063 seconds to run.
<Figure size 640x480 with 0 Axes>
Size of X: 682
Size of Y: 641
```



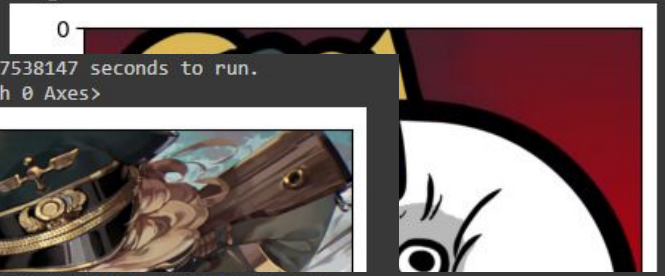

```
Function took 22.28145742416382 seconds to run.
<Figure size 640x480 with 0 Axes>
Size of X: 1080
Size of Y: 1080
```




```
Function took 7.713499069213867 seconds to run.
<Figure size 640x480 with 0 Axes>
Size of X: 415
Size of Y: 739
```




```
Function took 10.625714778900146 seconds to run.
<Figure size 640x480 with 0 Axes>
```




```
Function took 20.07608437538147 seconds to run.
<Figure size 640x480 with 0 Axes>
```



```
Function took 8.042617321014404 seconds to run.
<Figure size 640x480 with 0 Axes>
```