```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.14.6-py3-none-any.whl (493 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 493.7/493.7 kB 7.7 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.23.5)
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (9.0.0)
Collecting dill<0.3.8,>=0.3.0 (from datasets)
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 8.9 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.15-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 9.9 MB/s eta 0:00:00
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2023.6
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.8.6)
Collecting huggingface-hub<1.0.0,>=0.14.0 (from datasets)
  Downloading huggingface_hub-0.18.0-py3-none-any.whl (301 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.0/302.0 kB 13.8 MB/s eta 0:00:00
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (23.1.0)
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (3.3
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.0.4)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.9.2)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0.0,>=0.14.0->datasets)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0.0,>=0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (2.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (2023
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->datasets)
Installing collected packages: dill, multiprocess, huggingface-hub, datasets
Successfully installed datasets-2.14.6 dill-0.3.7 huggingface-hub-0.18.0 multiprocess-0.70.15
```

```
# import dataset
from datasets import load_dataset

data = load_dataset(
    "jamescalam/image-text-demo",
    split="train",
    revision="180fdae"
)
data
```

```
Downloading builder script:                    1.72k/1.72k [00:00<00:00,
100%                                           60.0kB/s]
Downloading data:                              37.8M/37.8M [00:01<00:00,
100%                                           26.3MB/s]
Generating train split:      21/0 [00:00<00:00, 38.13 examples/s]
Dataset({
```

```
data[2]['image']
```

To create the patches, we must first convert our PIL image object into a PyTorch tensor. We can do this using torchvision.transforms

```
from torchvision import transforms

# transform the image into tensor
transt = transforms.ToTensor()

img = transt(data[2]["image"])
img.data.shape
```

```
torch.Size([3, 5184, 3456])
```

```
# add batch dimension and shift color channels
patches = img.data.unfold(0,3,3)
patches.shape
```

```
torch.Size([1, 5184, 3456, 3])
```

broke up the image into horizontal patches first. All patches will be square with dimensionalities of 256x256, so the horizontal patch height equals 256 pixels.

```
# break the image into patches (in height dimension)
patch = 256

patches = patches.unfold(1, patch, patch)
patches.shape
```

```
torch.Size([1, 20, 3456, 3, 256])
```

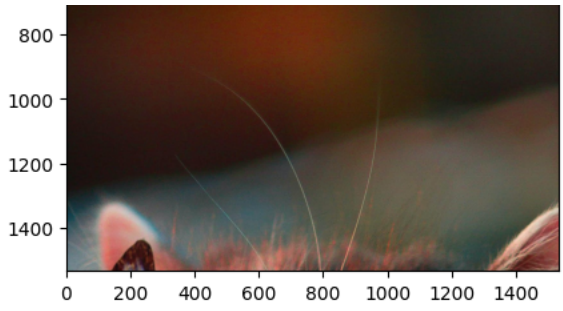
```
# break the image into patches (in width dimension)
patches = patches.unfold(2, patch, patch)
patches.shape
```

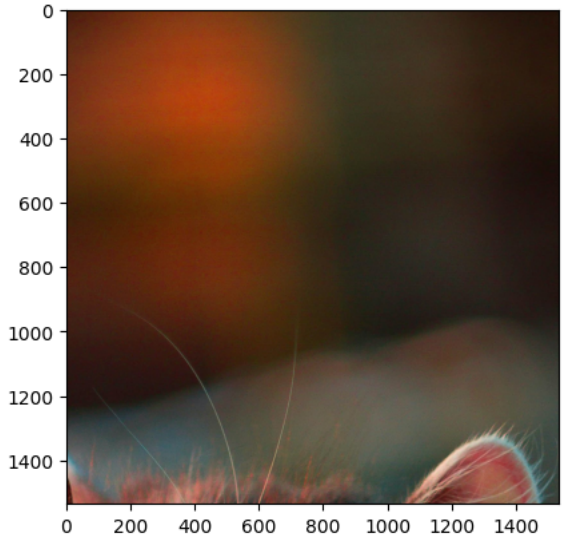
```
torch.Size([1, 20, 13, 3, 256, 256])
```

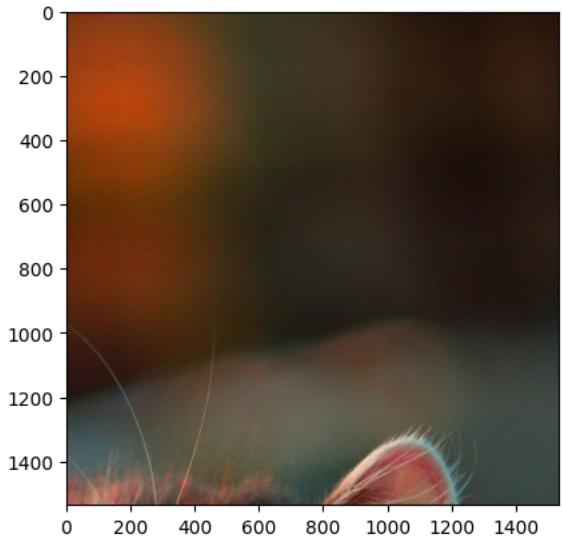
```
import torch
import torchvision
from matplotlib import pyplot as plt
```

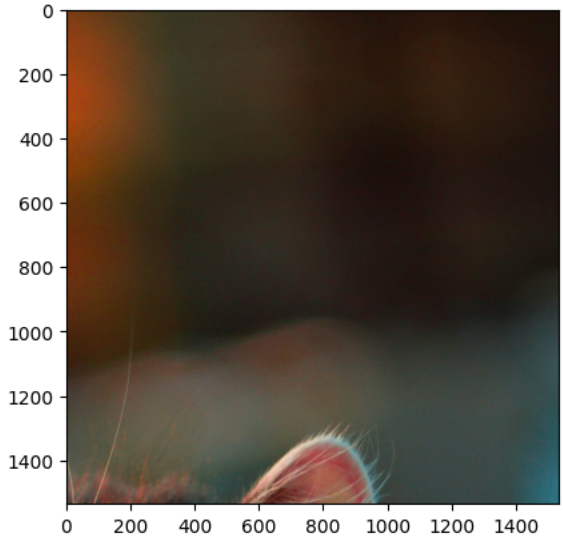
We call this grouping of patches a window. A larger window size captures more global views of the image, whereas a smaller window can produce a more precise map at the risk of missing larger objects. To slide across the image and create a big_batch at each step, we do the following:

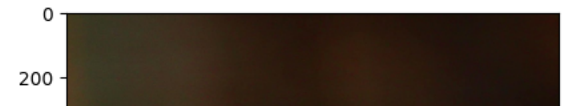
```
window = 6
stride = 1

# window slides from top to bottom
for Y in range(0, patches.shape[1]-window+1, stride):
    # window slides from left to right
    for X in range(0, patches.shape[2]-window+1, stride):
        # initialize an empty big_patch array
        big_patch = torch.zeros(patch*window, patch*window, 3)
        # this gets the current batch of patches that will make big_batch
        patch_batch = patches[0, Y:Y+window, X:X+window]
        # loop through each patch in current batch
        for y in range(patch_batch.shape[1]):
            for x in range(patch_batch.shape[0]):
                # add patch to big_patch
                big_patch[
                    y*patch:(y+1)*patch, x*patch:(x+1)*patch, :
                ] = patch_batch[y, x].permute(1, 2, 0)
        # display current big_patch
        plt.imshow(big_patch)
        plt.show()
```

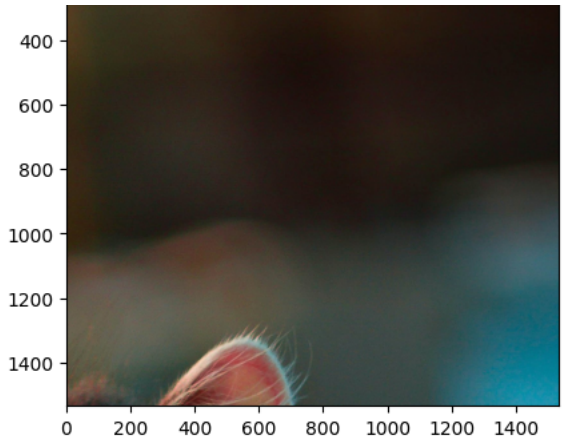

800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

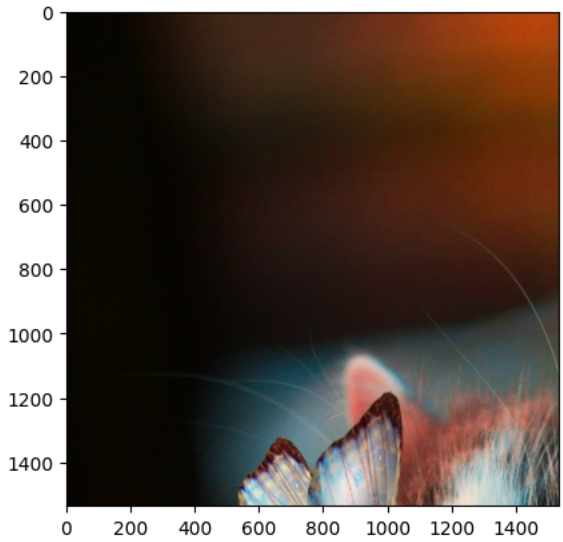

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

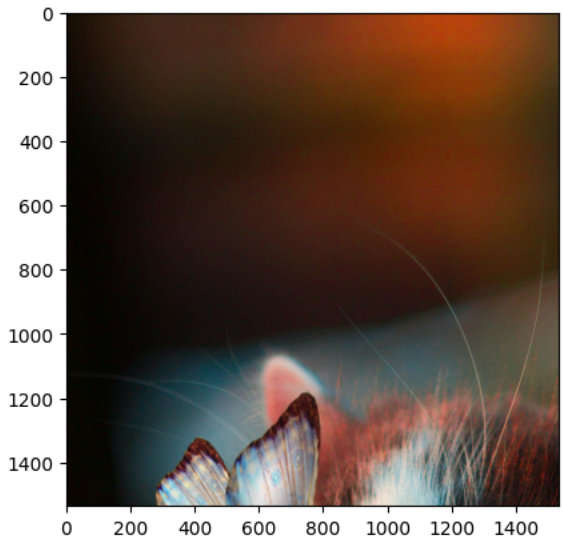

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

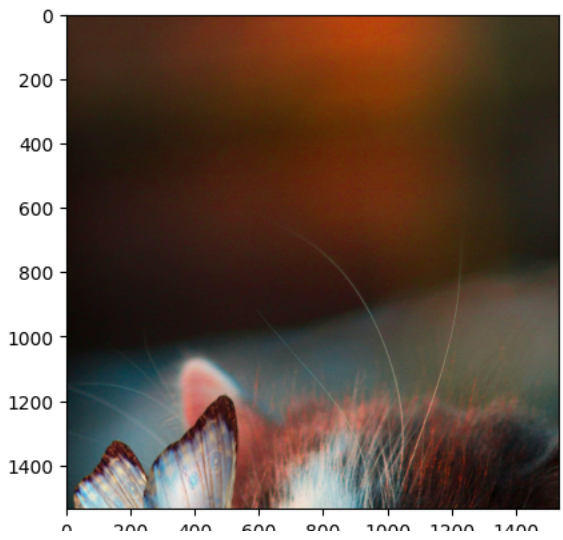

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

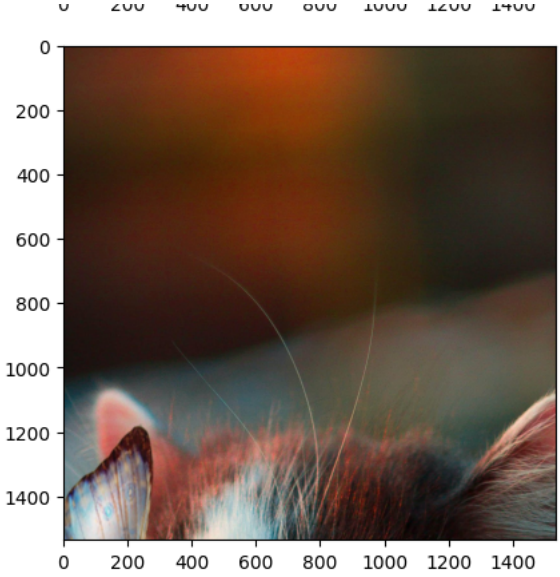

0
200

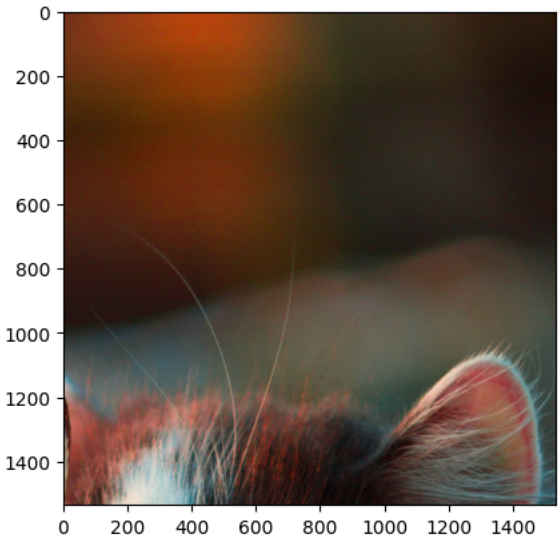

400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

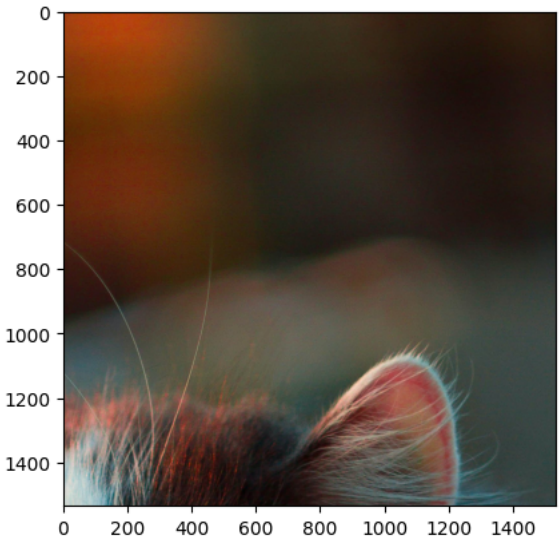

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

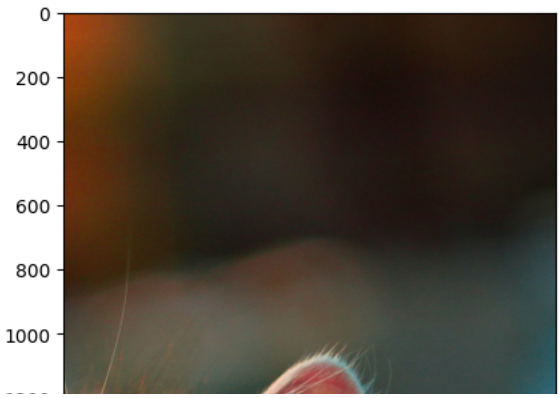

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

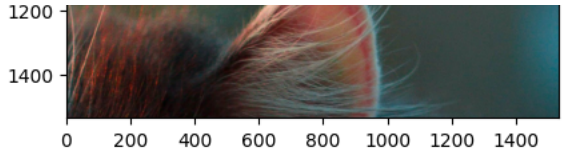

0
200
400
600
800
1000
1200
1400

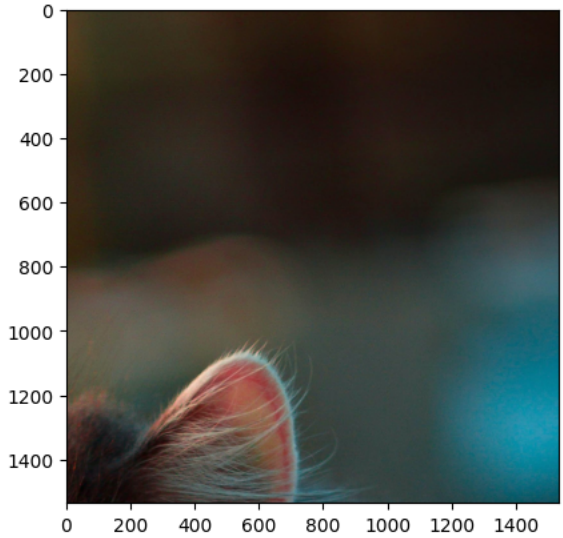

0
200
400
600
800
1000
1200
1400

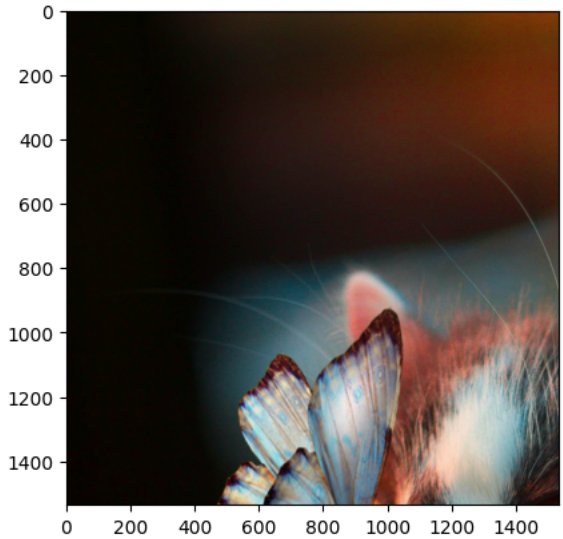

0
200
400
600
800
1000
1200
1400

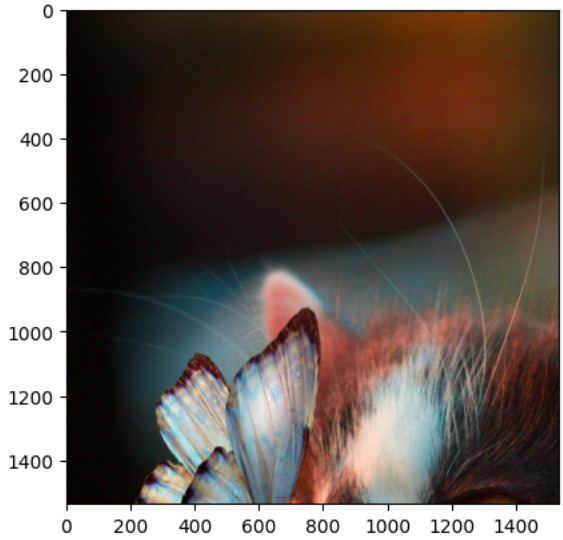

0
200
400
600
800
1000
1200
1400

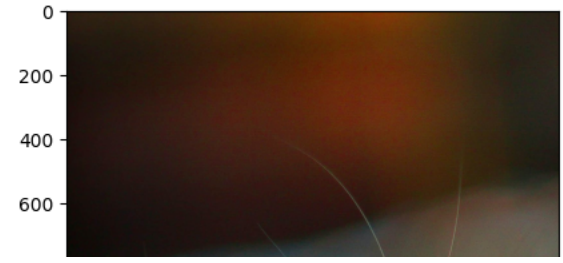

0
200
400
600
800
1000

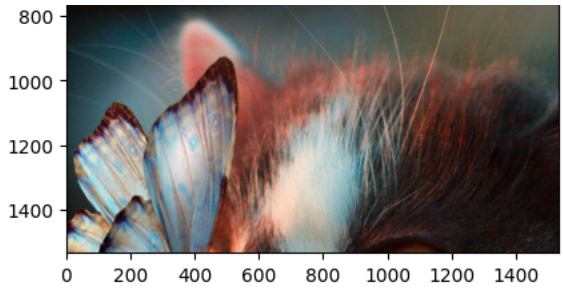

1200
1400
0 200 400 600 800 1000 1200 1400

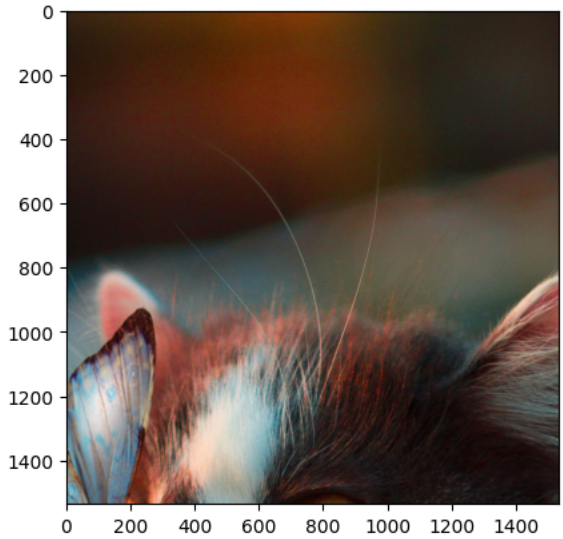

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

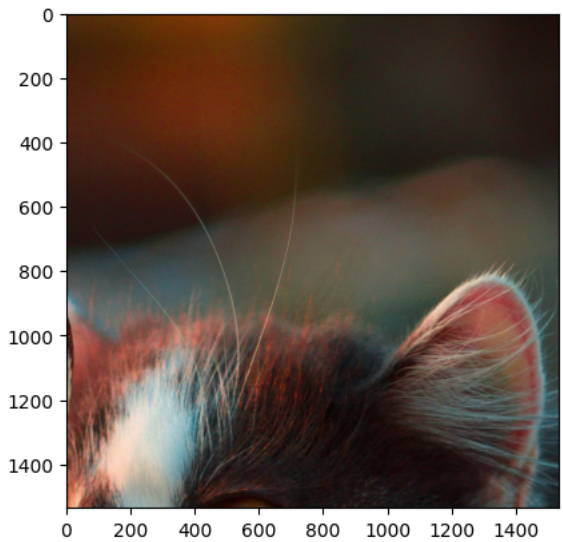

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

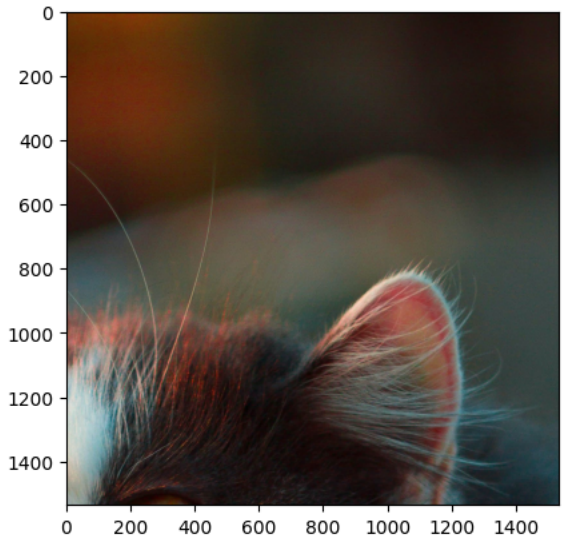

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

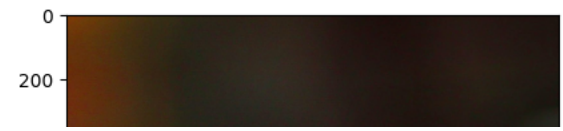

0
200
400
600

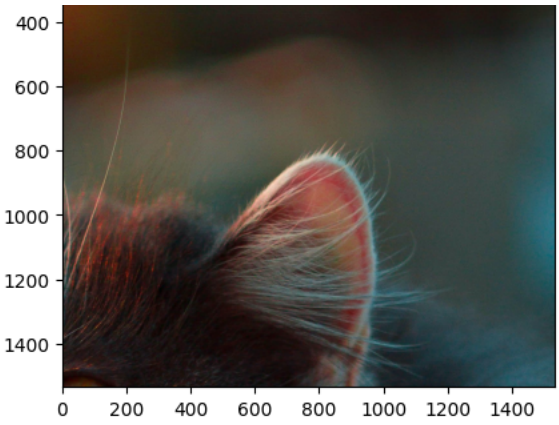

800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

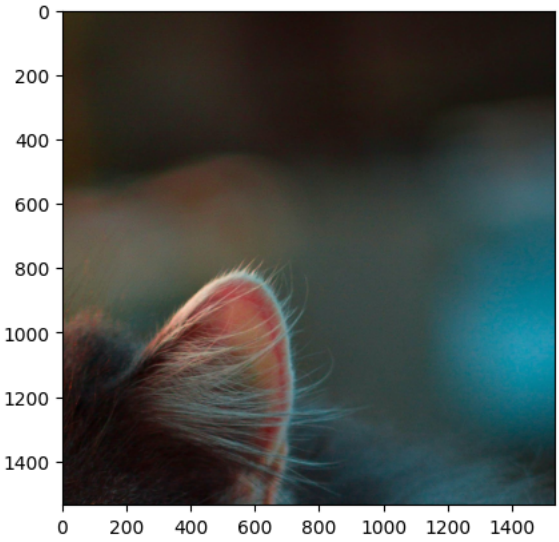

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

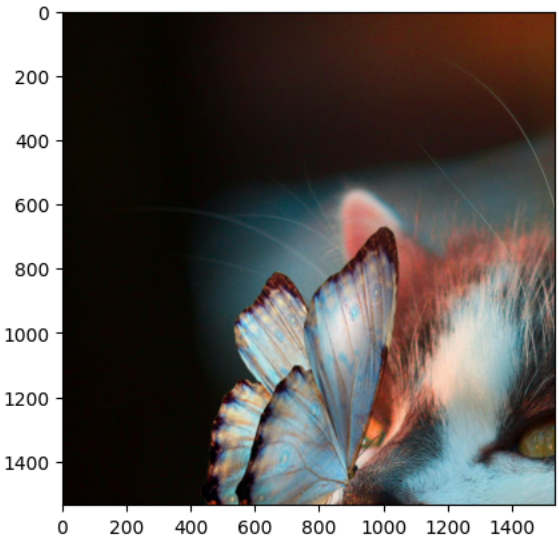

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

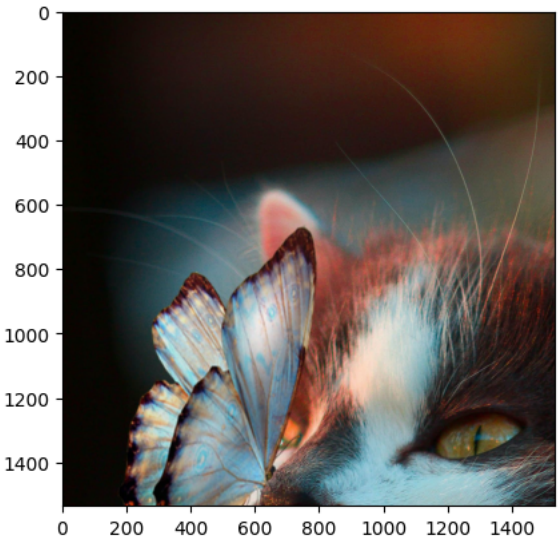

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

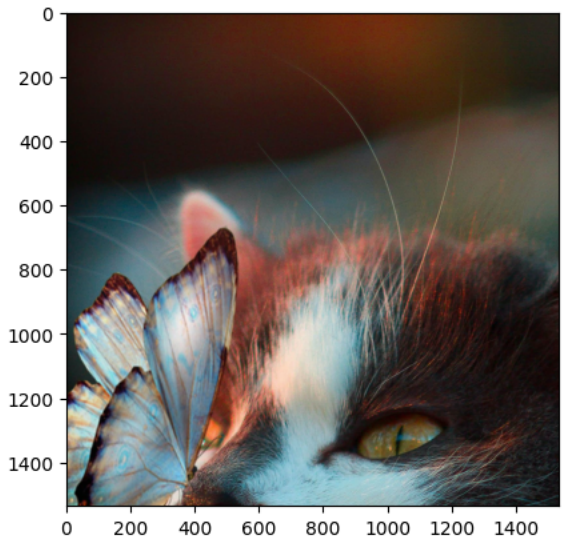

0
200

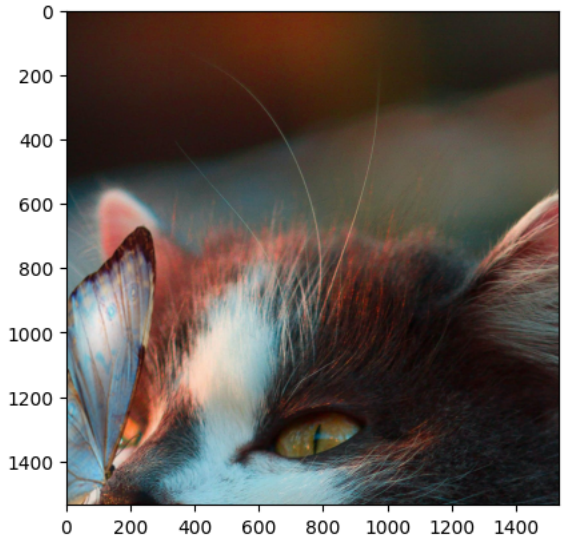

400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

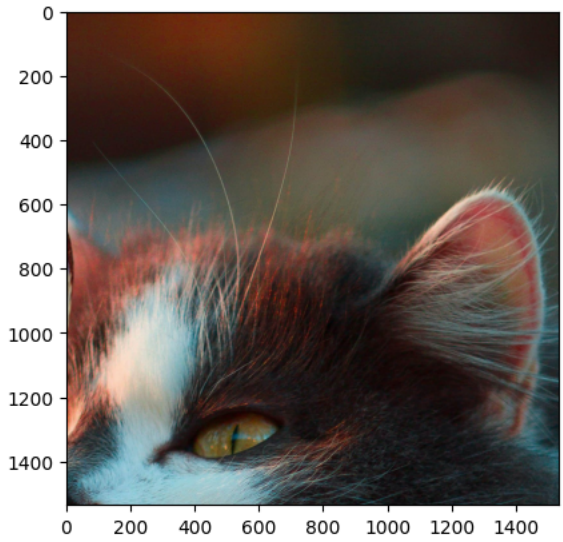

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

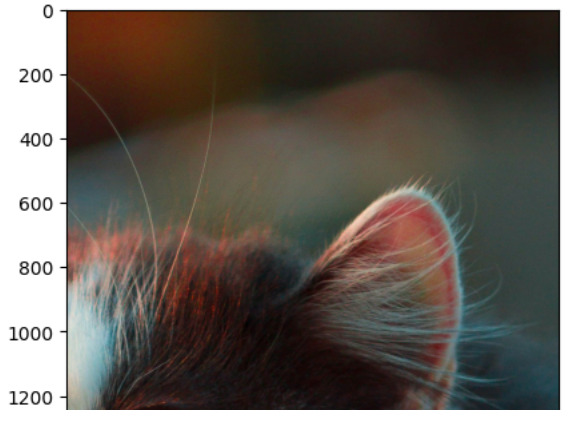

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

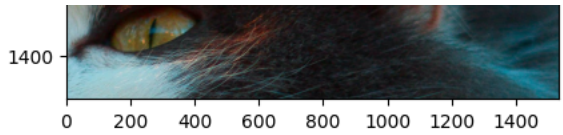

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

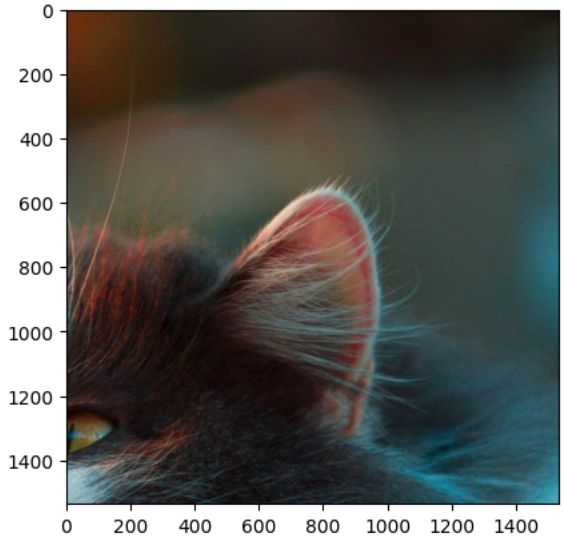

0
200
400
600
800
1000
1200
1400

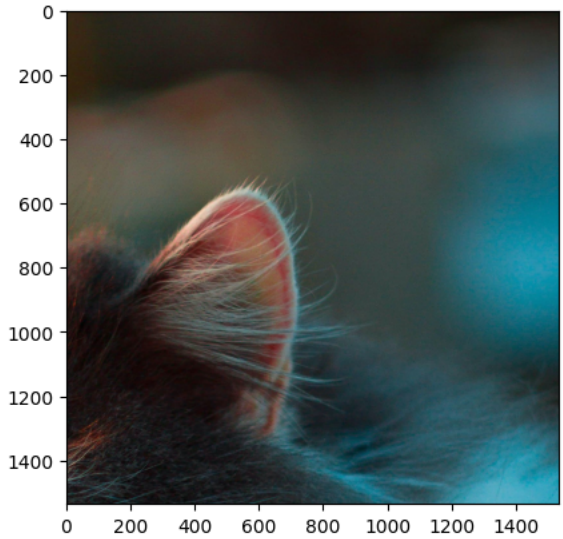

0
200
400
600
800
1000
1200
1400

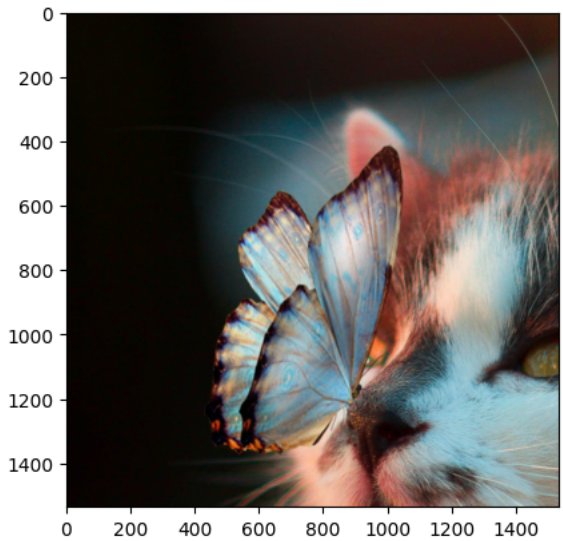

0
200
400
600
800
1000
1200
1400

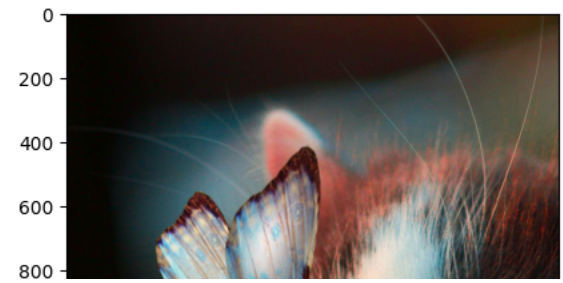

0
200
400
600
800
1000
1200

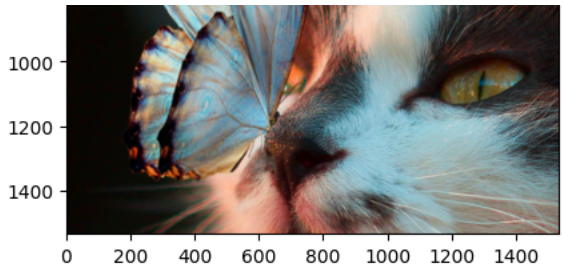

1400
0 200 400 600 800 1000 1200 1400

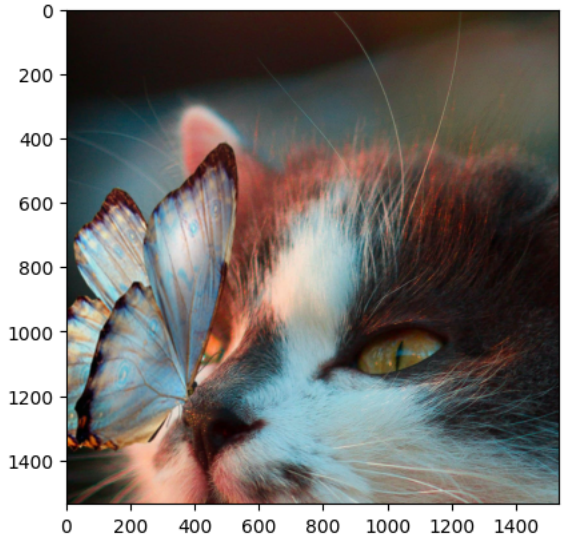

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

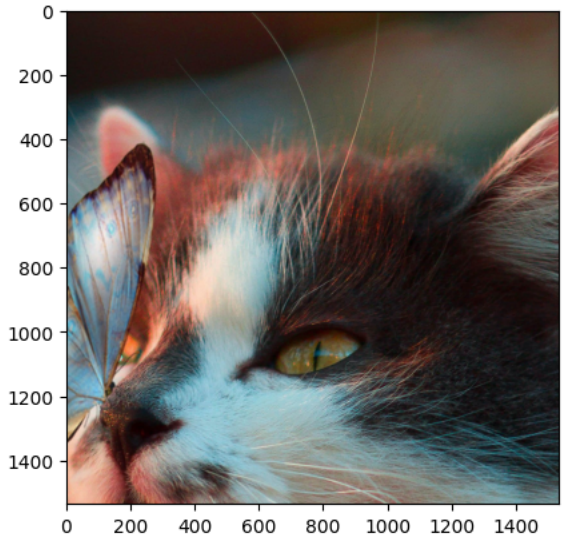

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

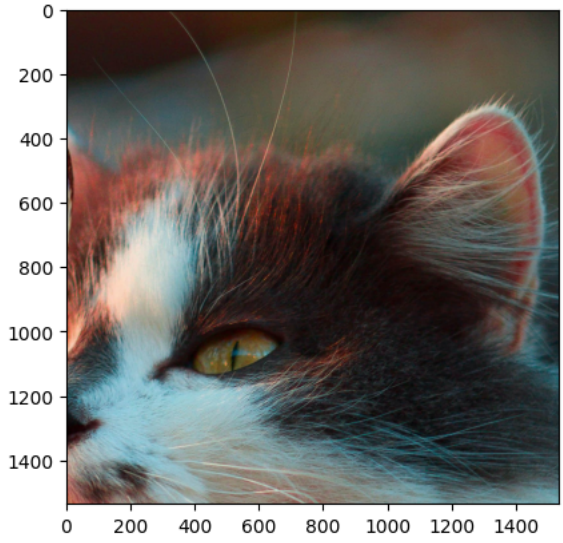

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

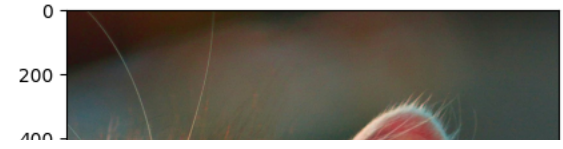

0
200
400
600
800


1000
1200
1400
0 200 400 600 800 1000 1200 1400

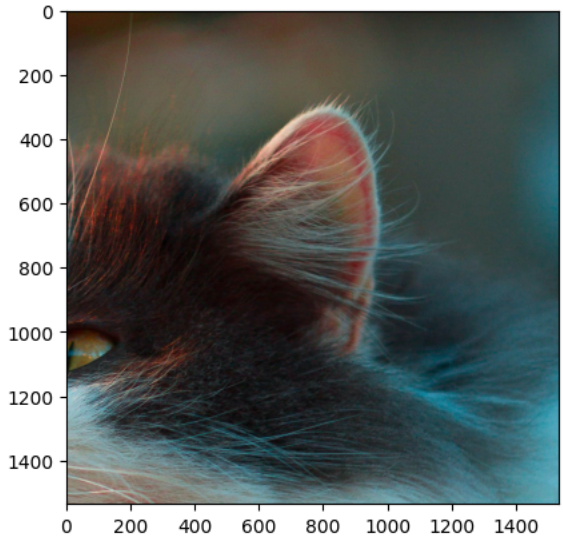

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

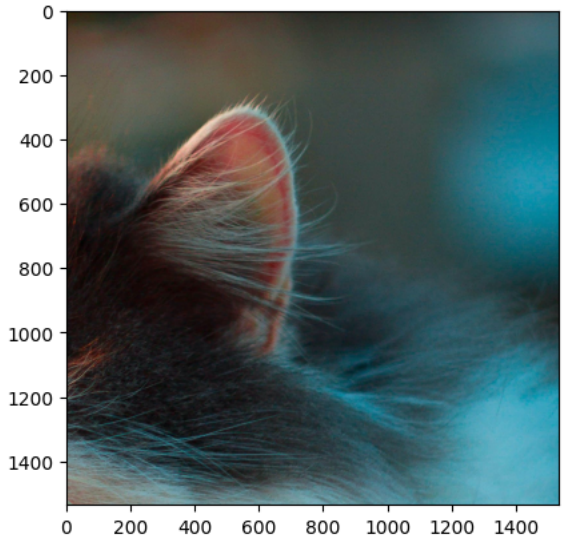

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

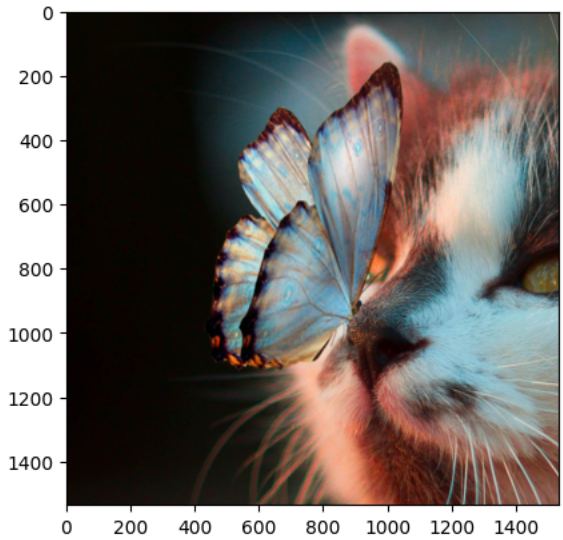

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200
400

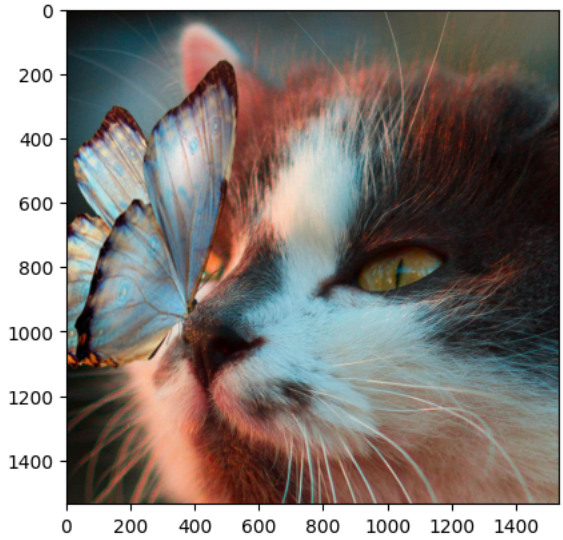

600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

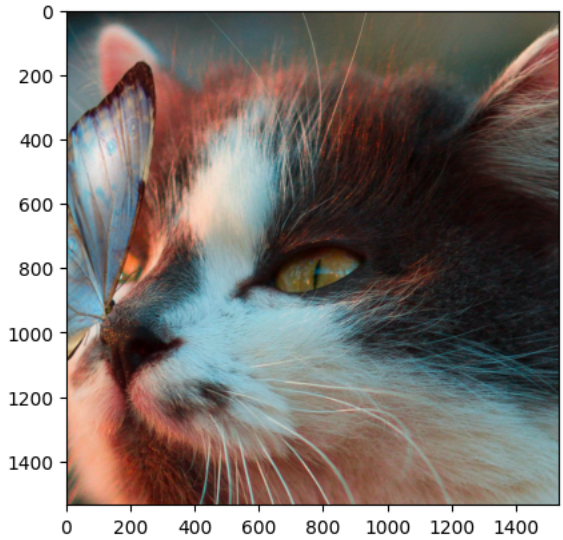

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

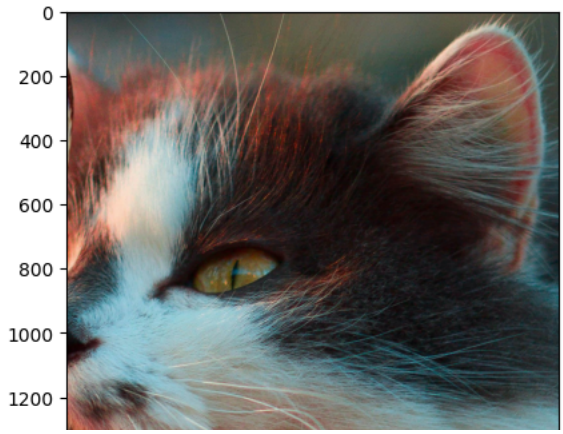

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

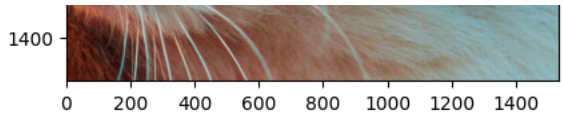

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

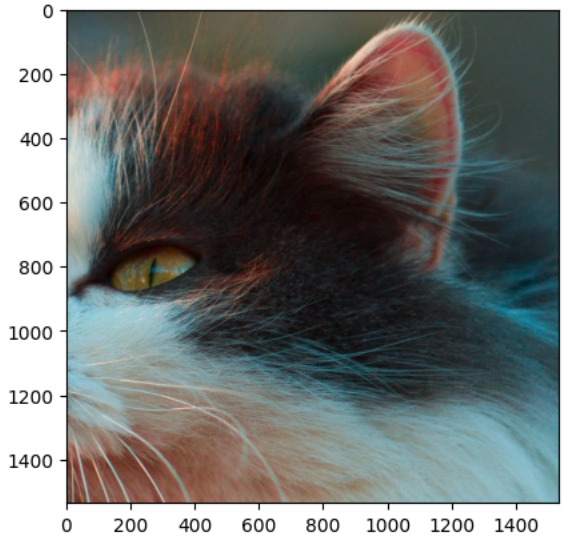

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

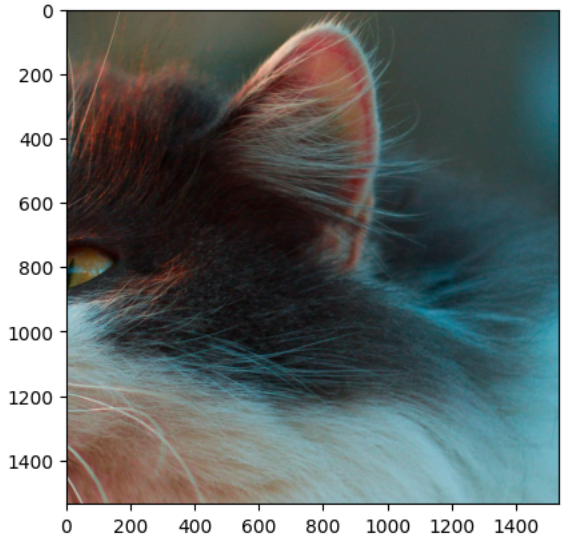

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

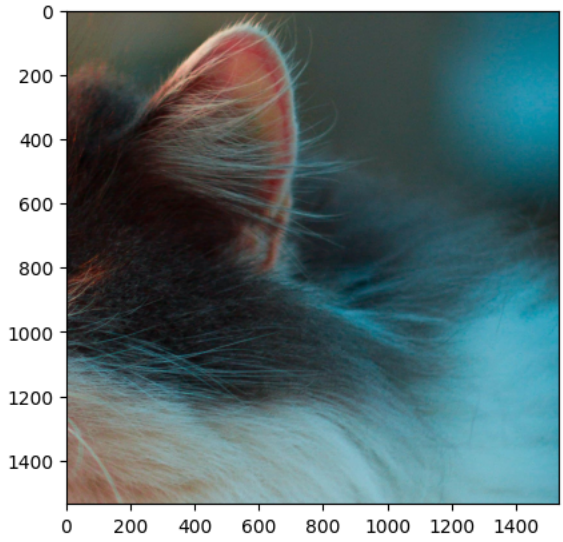

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200
400
600
800
1000
1200

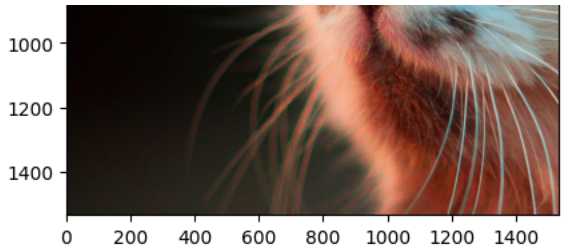

1400
0 200 400 600 800 1000 1200 1400

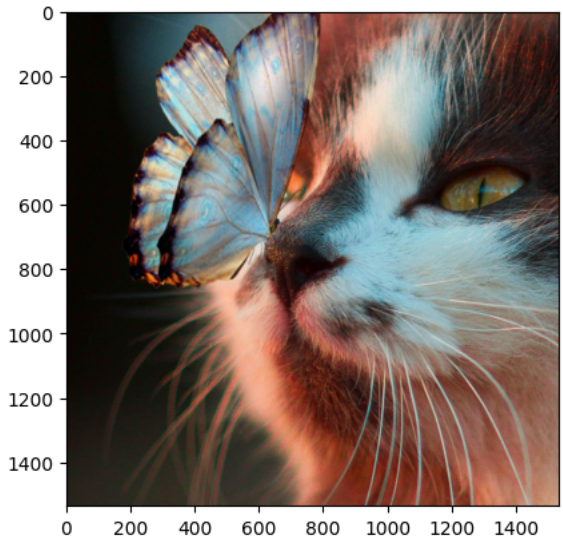

0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400

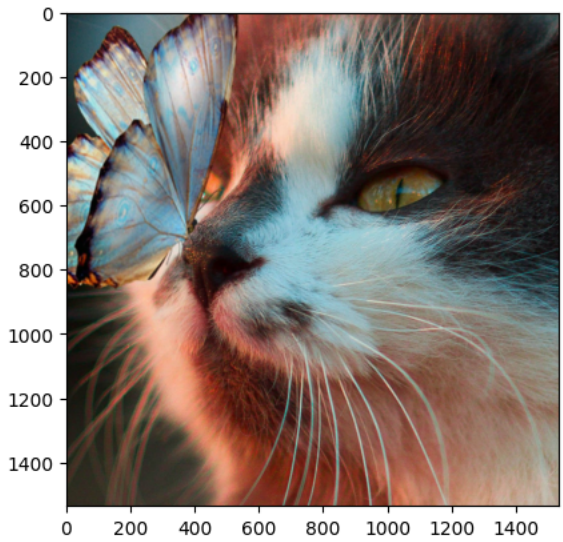

0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400

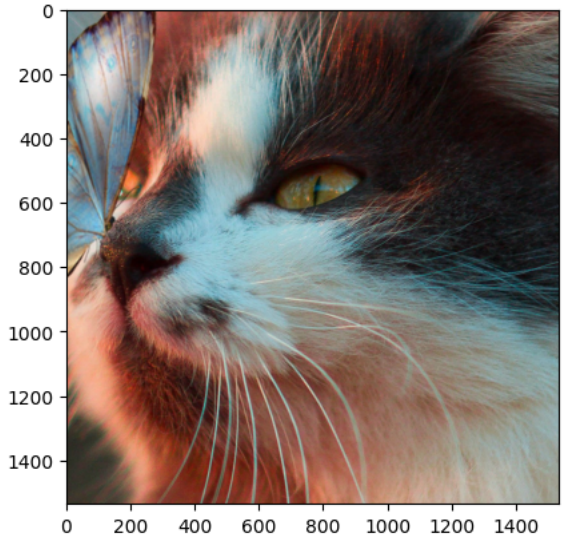

0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400

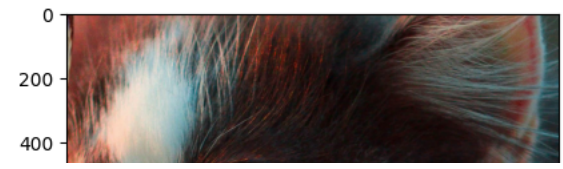

0 200 400 600 800

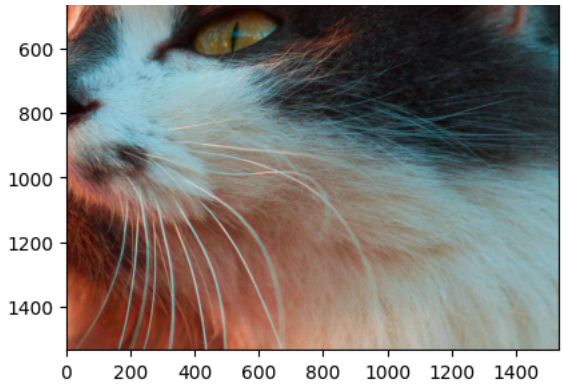

1000
1200
1400
0 200 400 600 800 1000 1200 1400

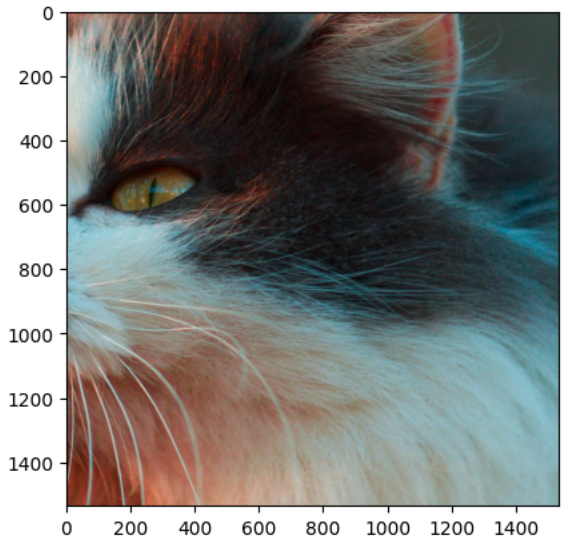

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

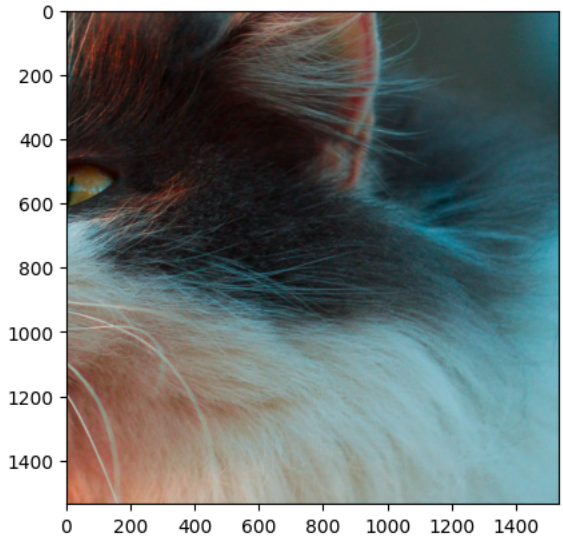

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

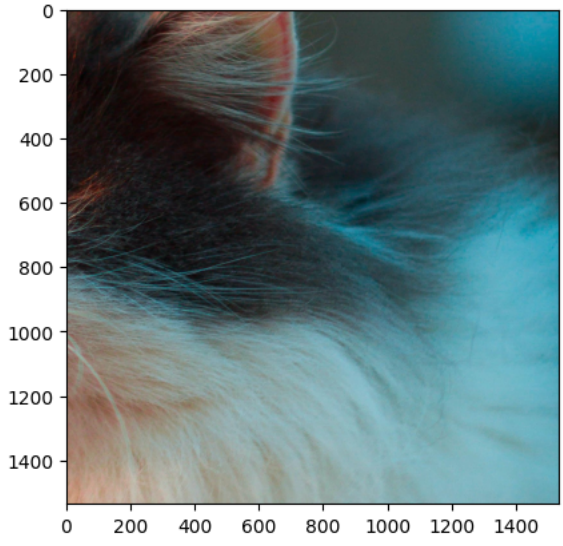

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200
400

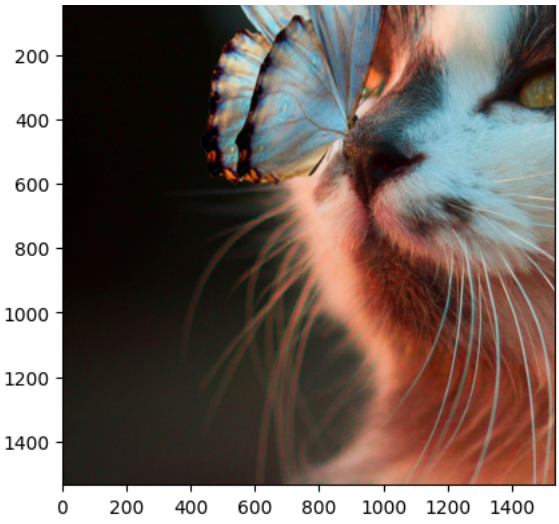

600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

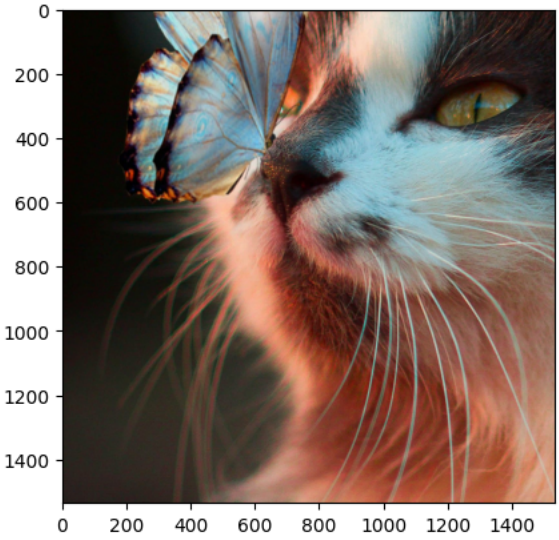

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

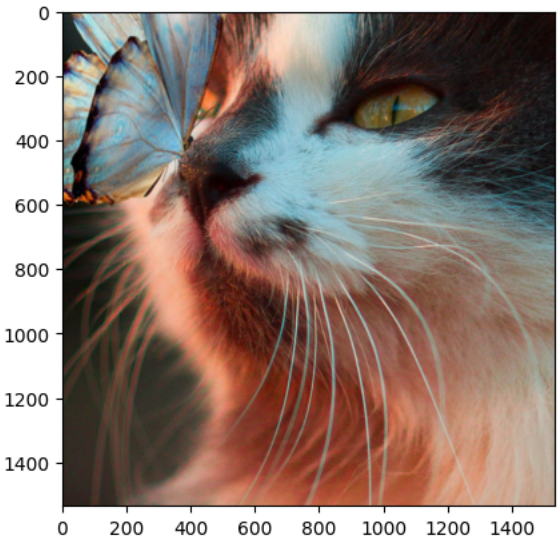

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

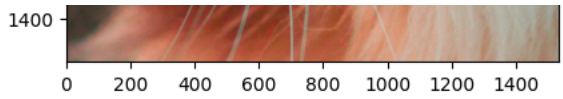

0

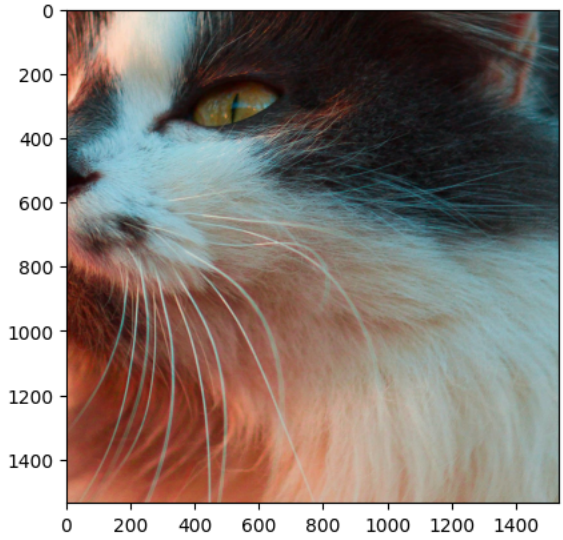

200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

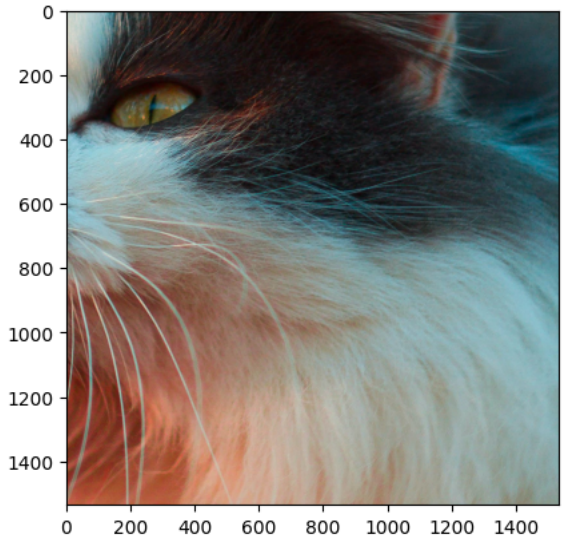

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

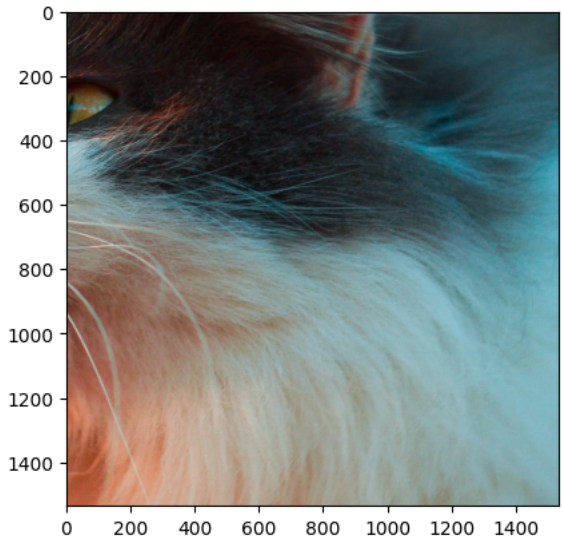

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

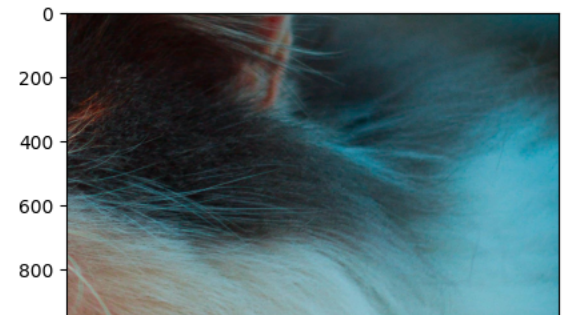

0
200
400
600
800
1000
1200

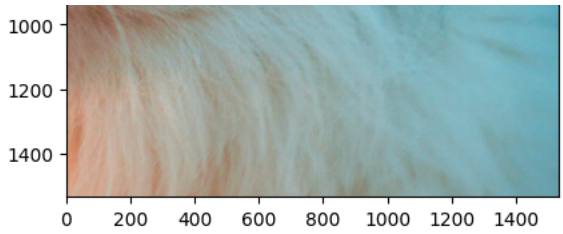

1400
0 200 400 600 800 1000 1200 1400

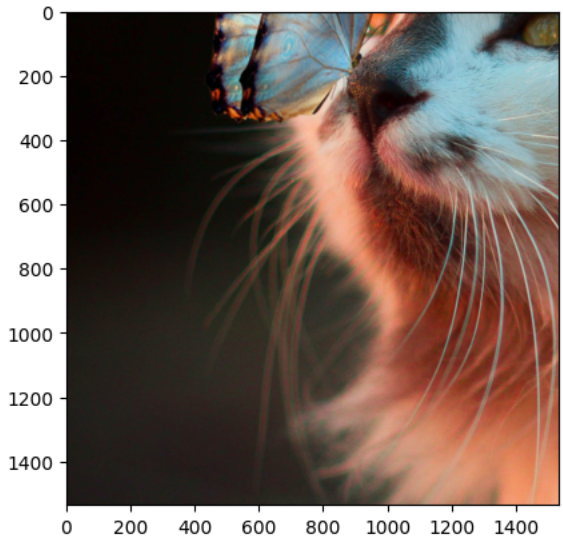

0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400

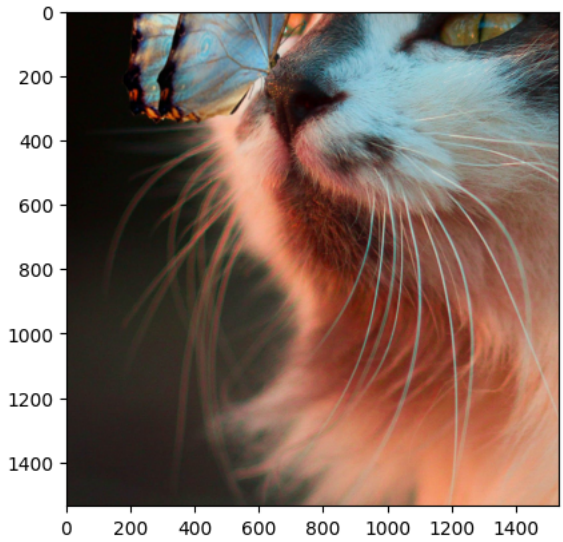

0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400

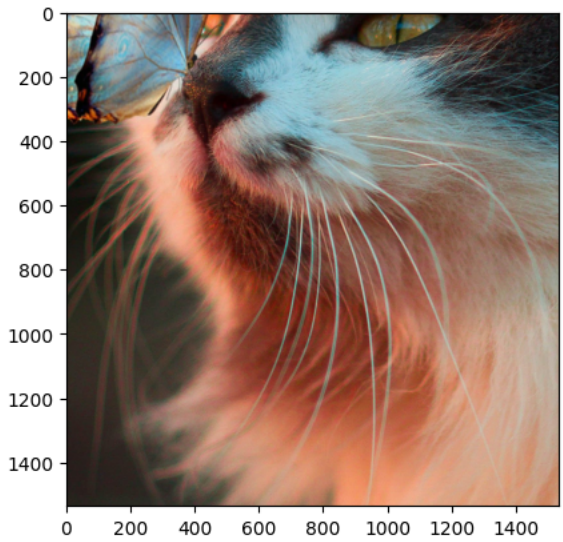

0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400

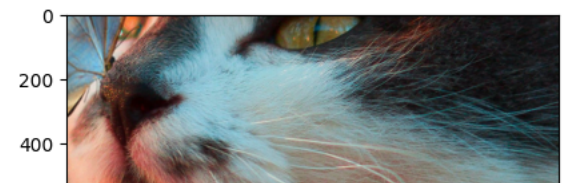

0 200 400 600 800
0 200 400 600 800

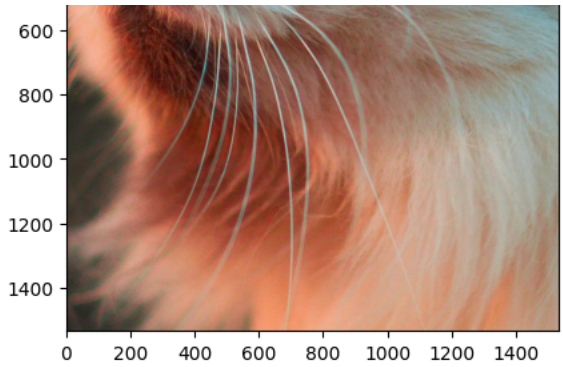

1000
1200
1400
0 200 400 600 800 1000 1200 1400

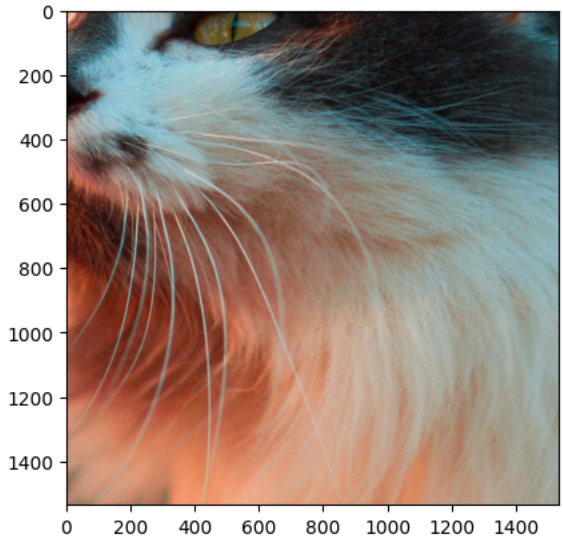

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

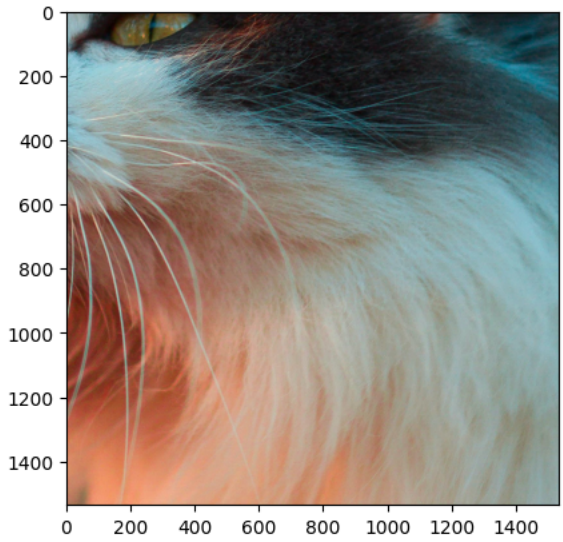

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

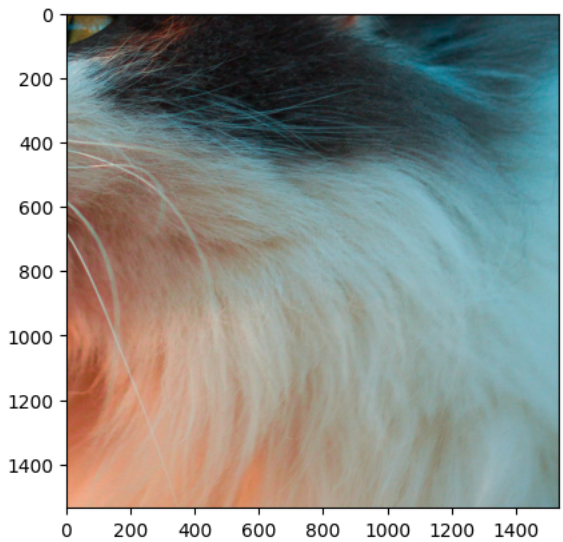

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

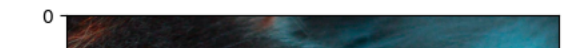

0
200
400

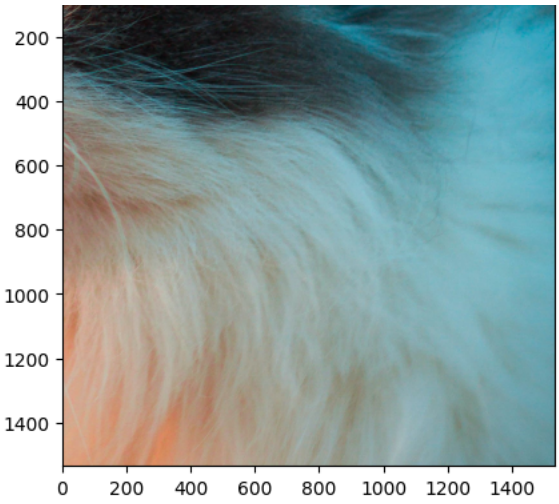

600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

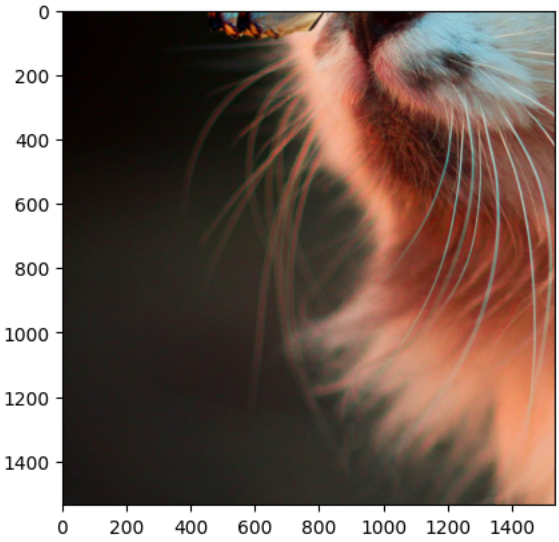

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

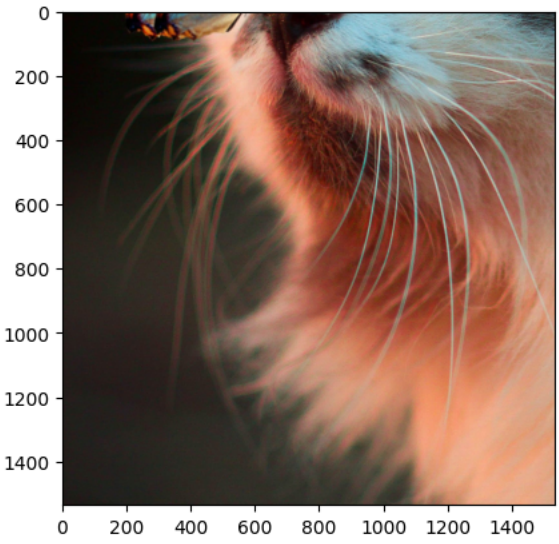

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

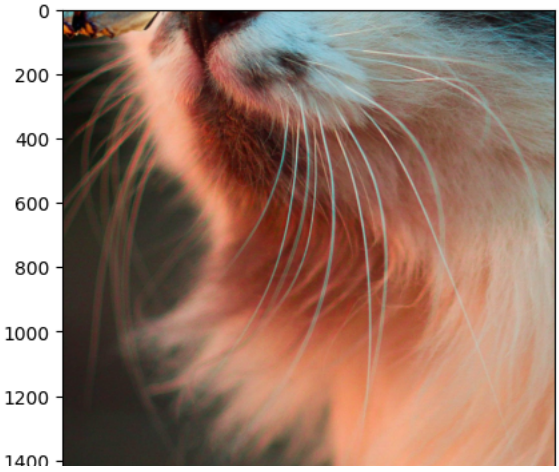

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

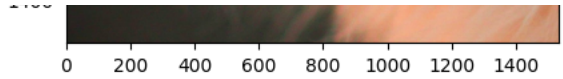

0

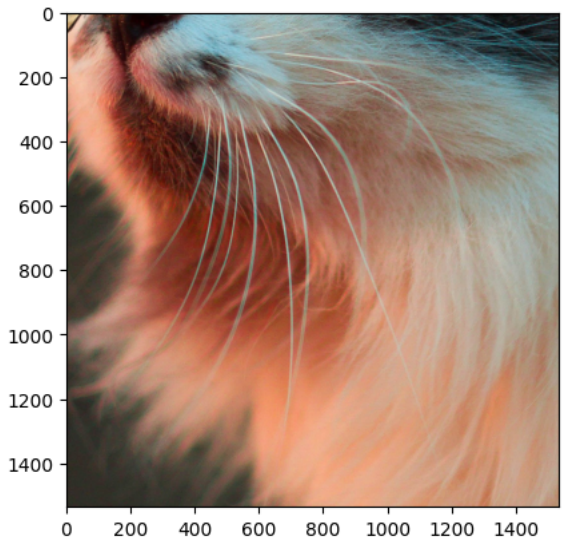

200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

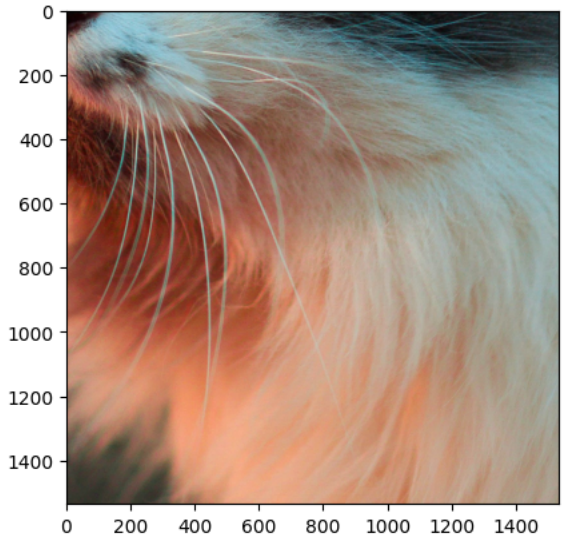

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

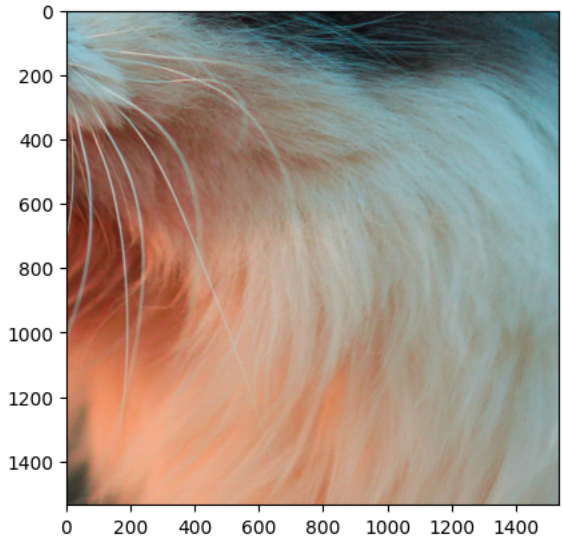

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200
400
600
800
1000
1200
1400

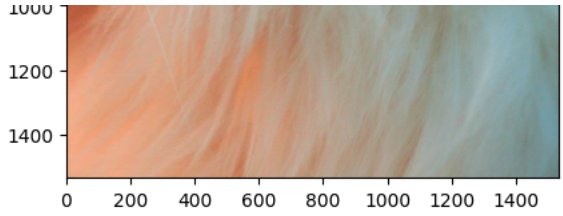

0
200
400
600
800
1000
1200
1400

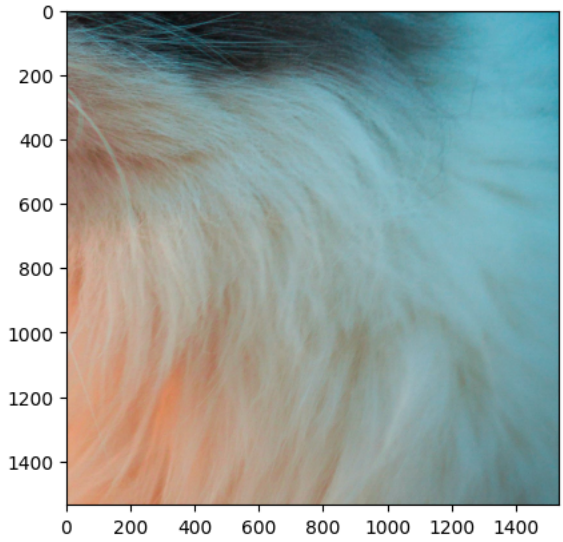

0
200
400
600
800
1000
1200
1400

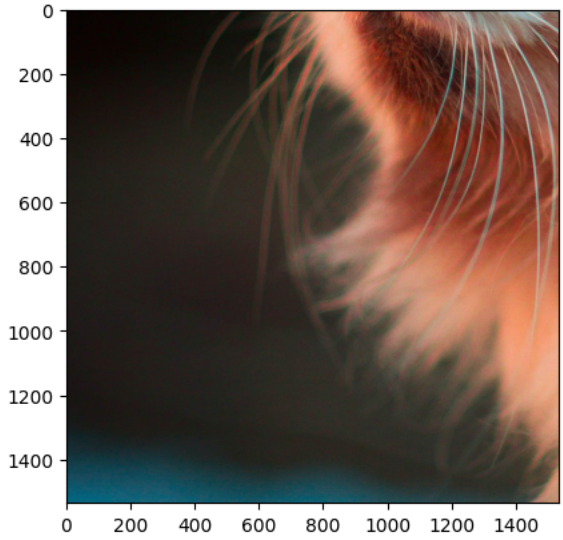

0
200
400
600
800
1000
1200
1400

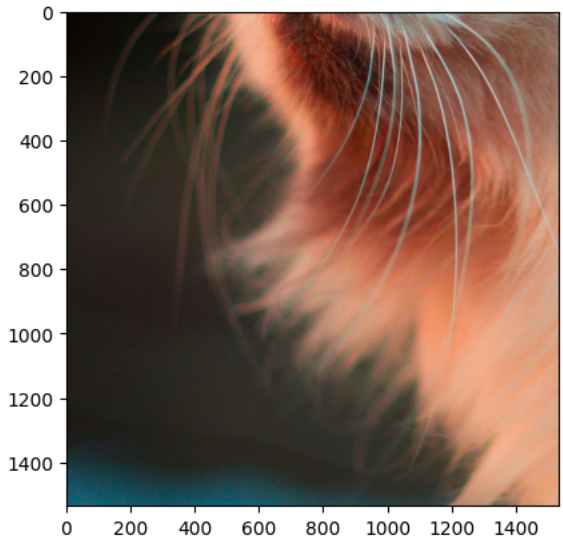

0
200
400
600
800
1000
1200
1400

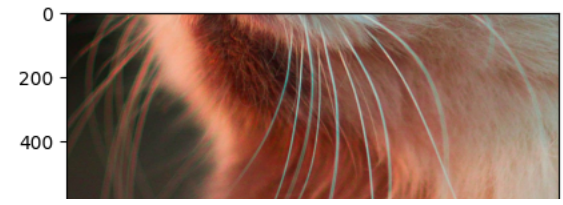

0
200
400
600
800

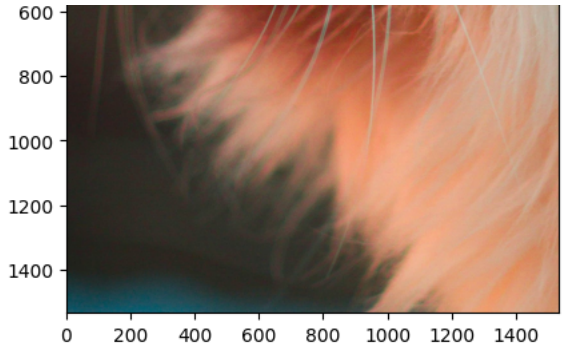

1000
1200
1400
0 200 400 600 800 1000 1200 1400

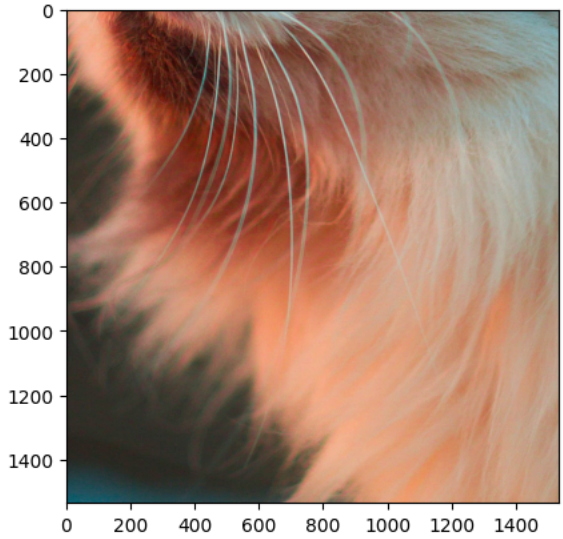

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

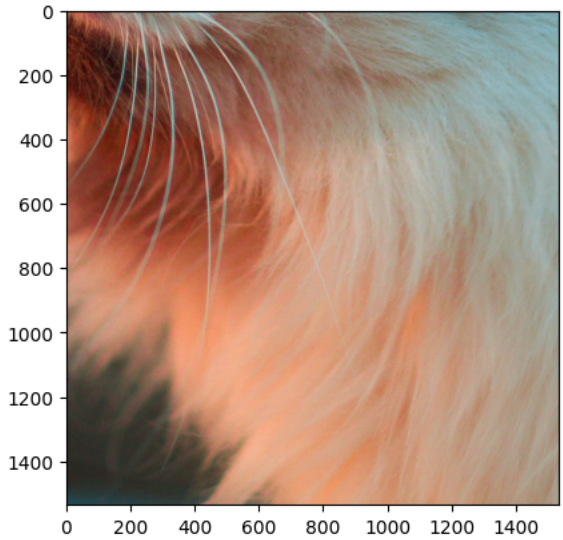

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

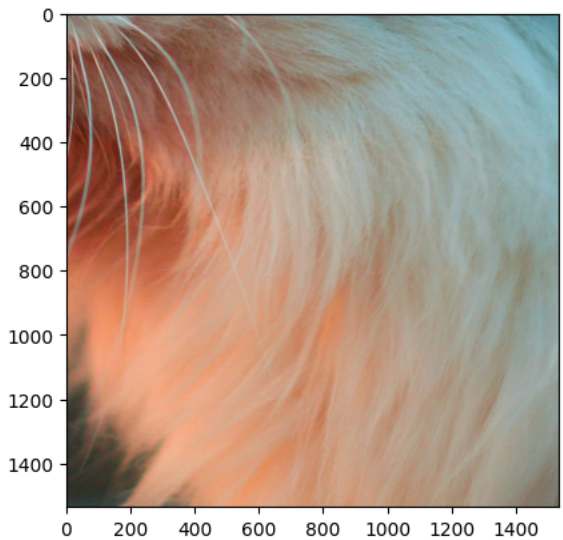

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

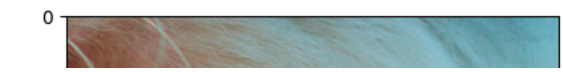

0
200
400

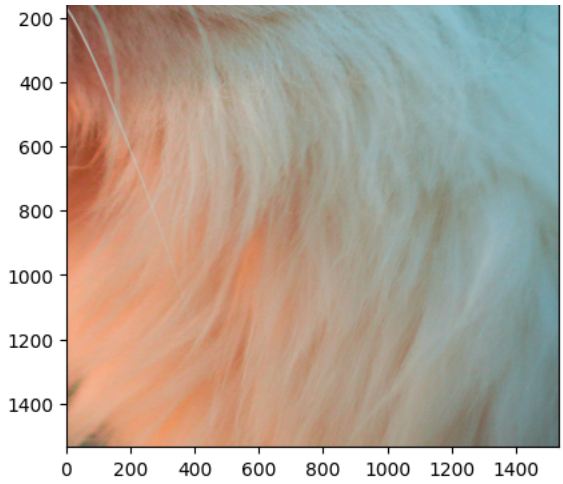

600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

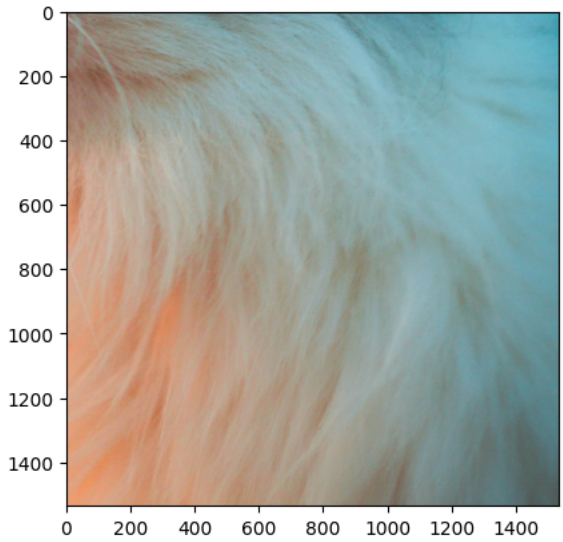

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

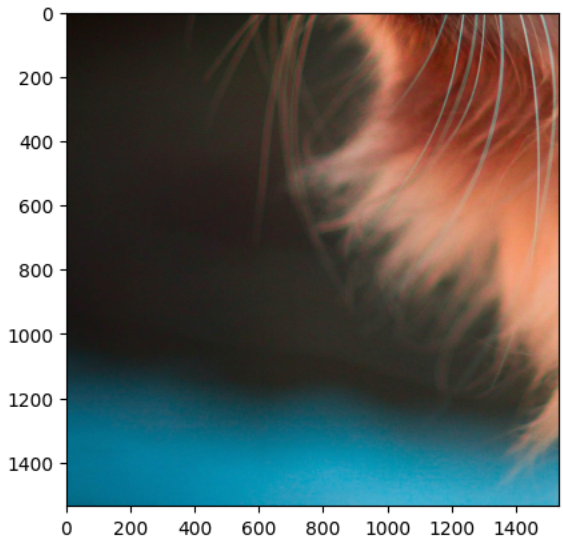

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

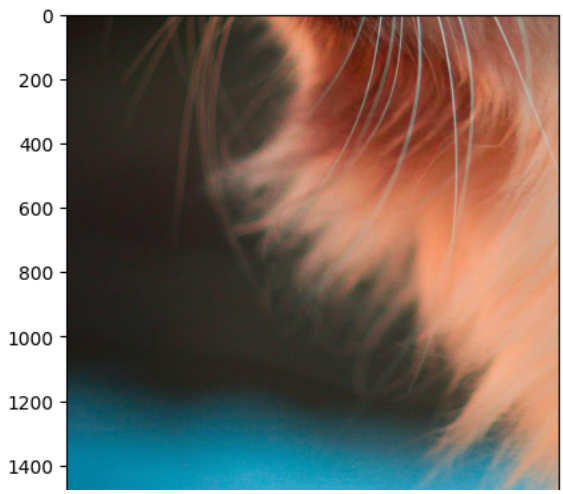

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

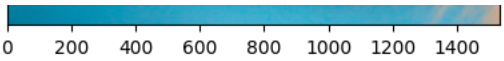

0

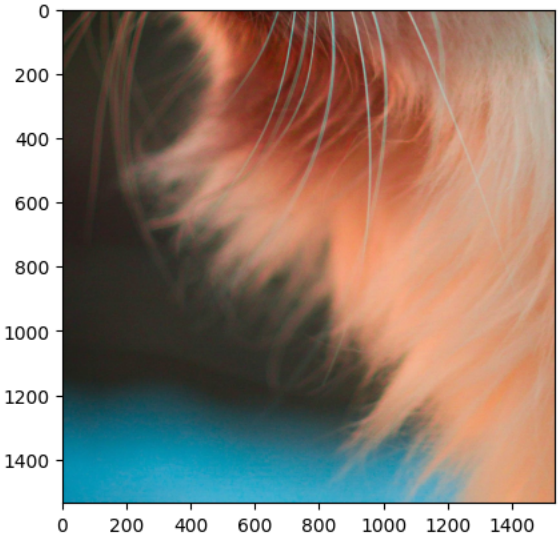

200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

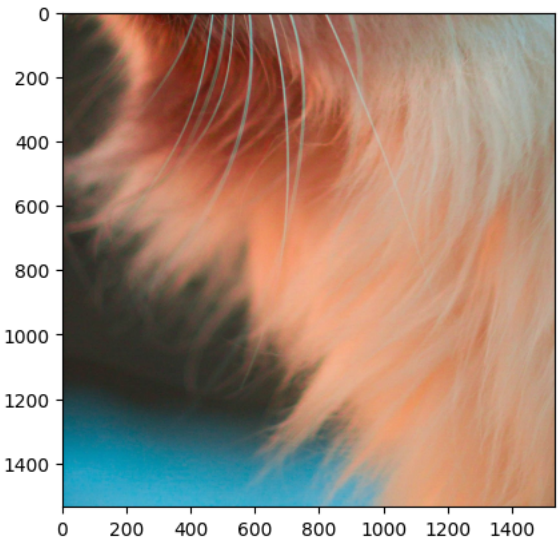

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

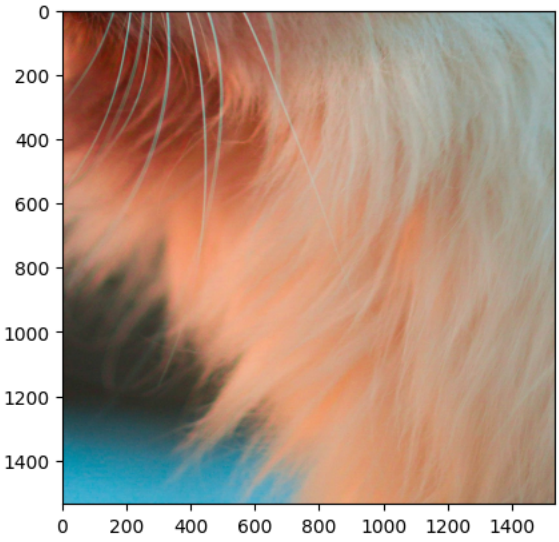

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

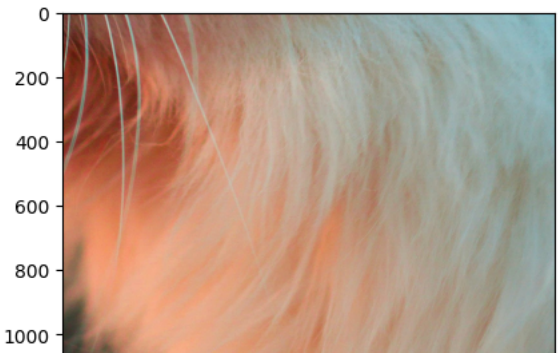

0
200
400
600
800
1000
1200
1400

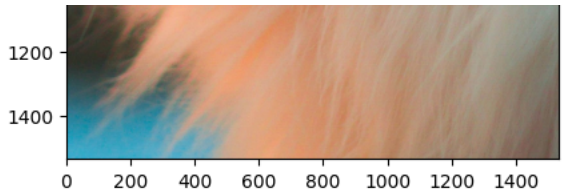

0
200
400
600
800
1000
1200
1400

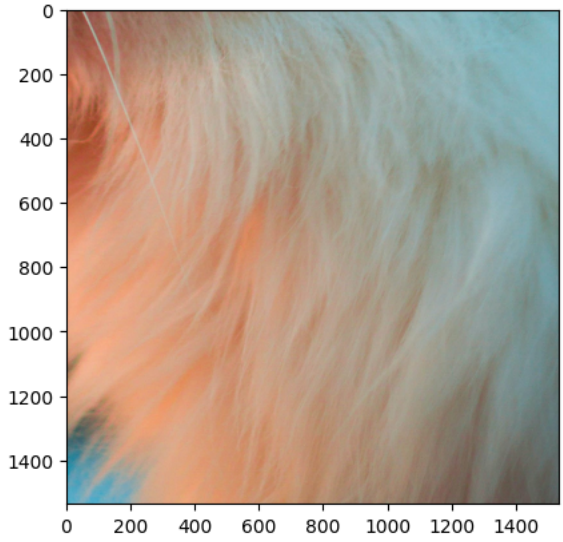

0
200
400
600
800
1000
1200
1400

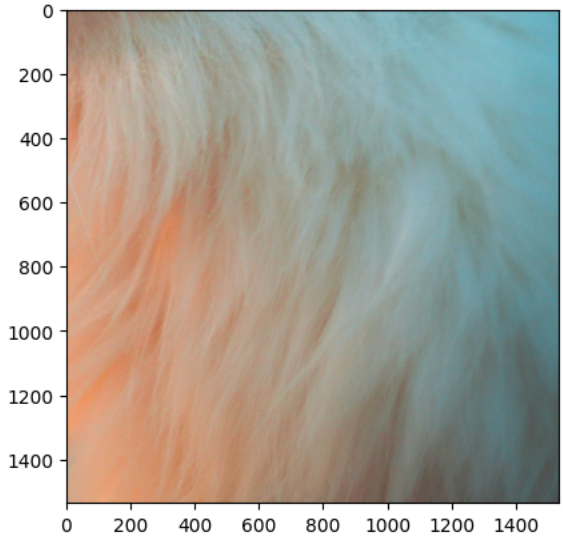

0
200
400
600
800
1000
1200
1400

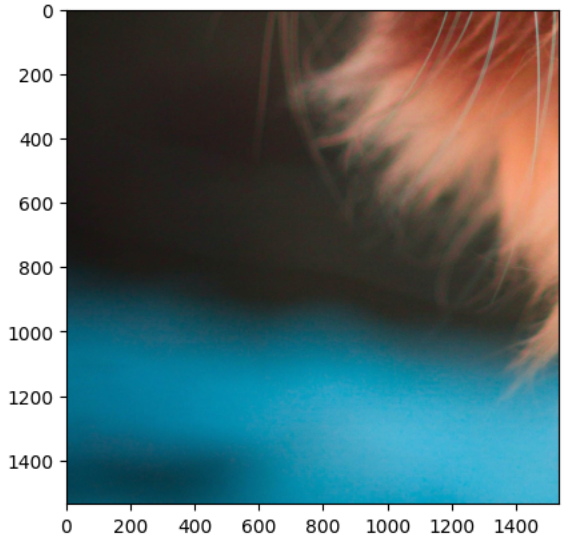

0
200
400
600
800
1000
1200
1400

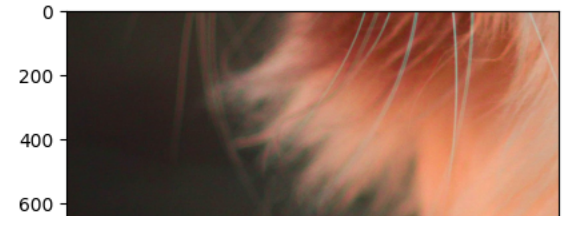

0
200
400
600
800
1000

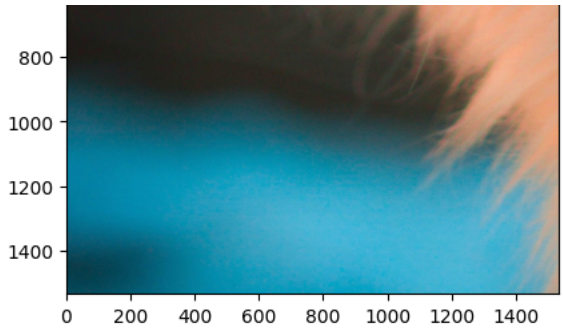

1200
1400
0 200 400 600 800 1000 1200 1400

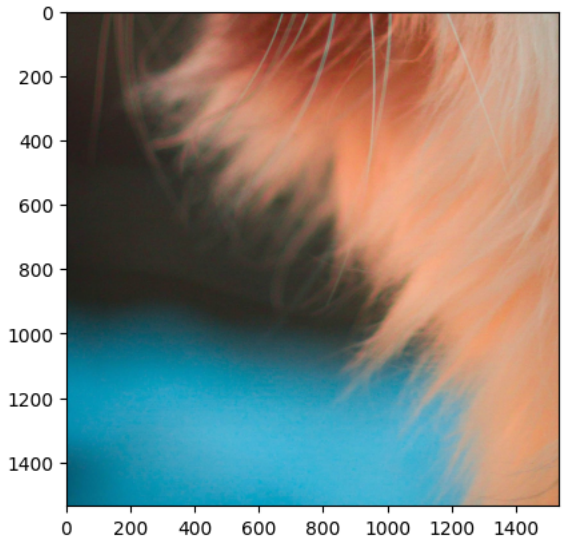

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

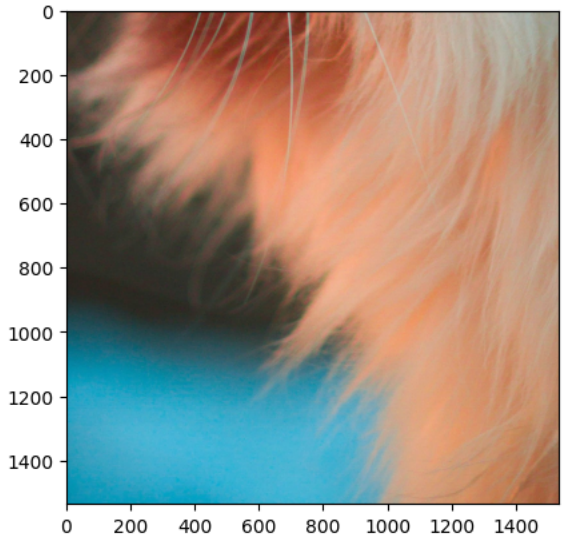

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

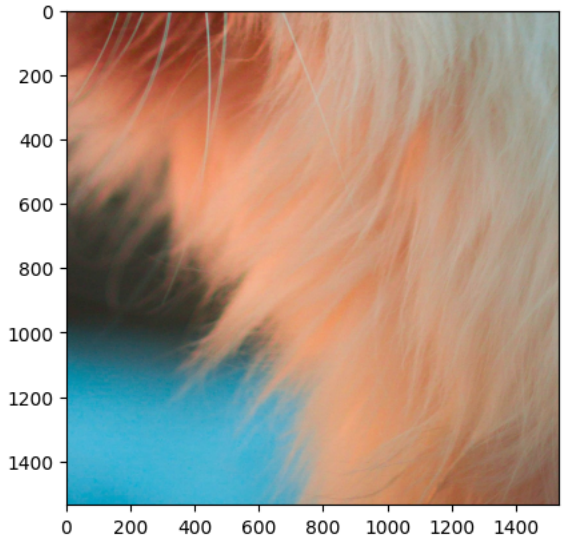

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

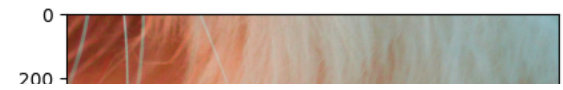

0
200
400
600

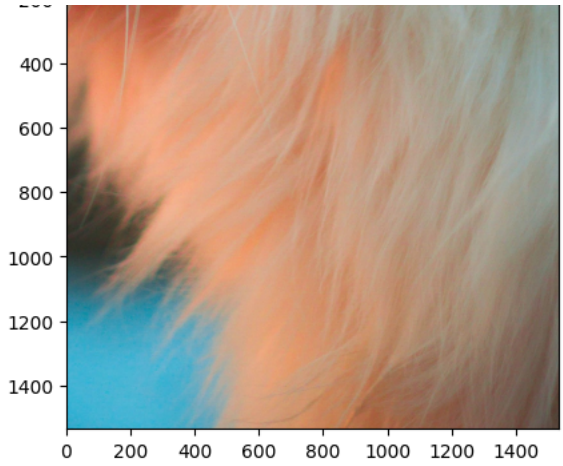

800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

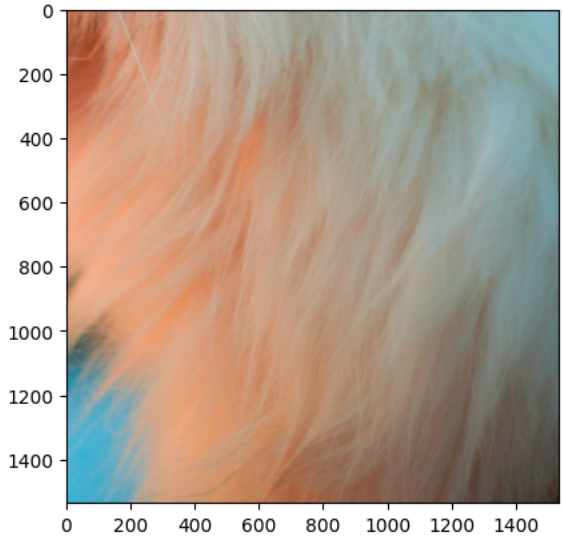

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

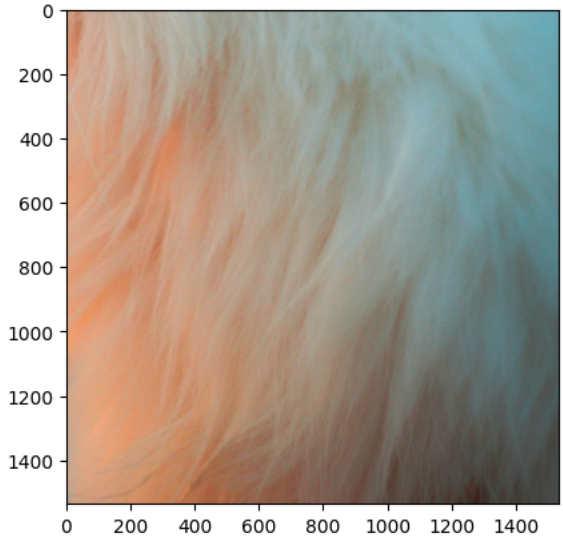

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

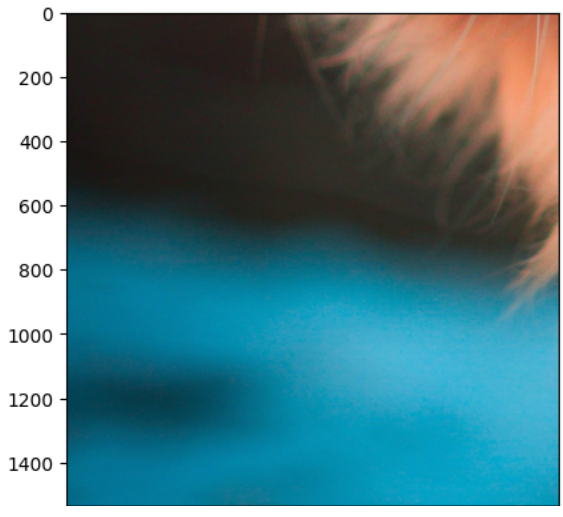

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

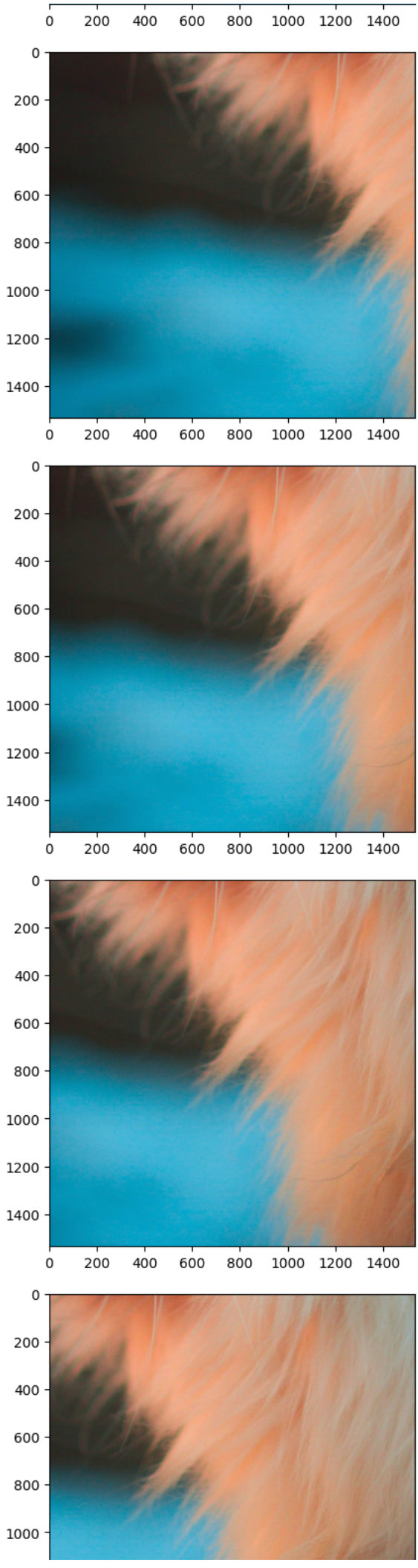

0
200

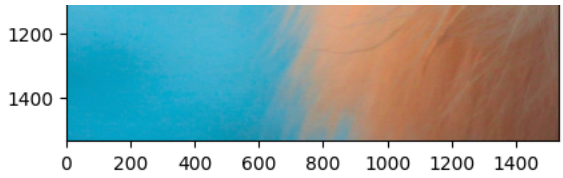

400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

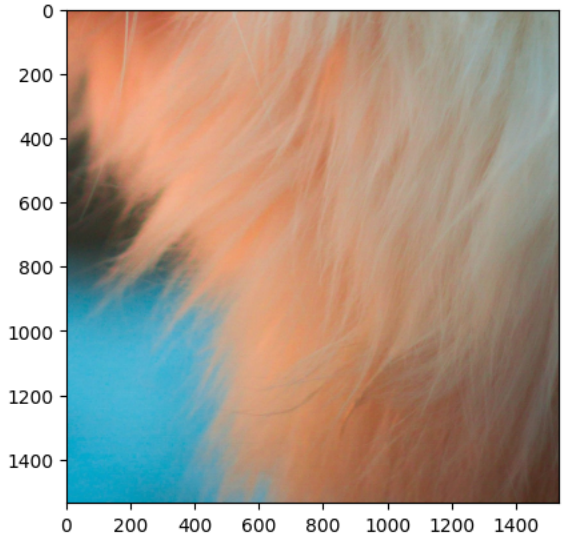

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

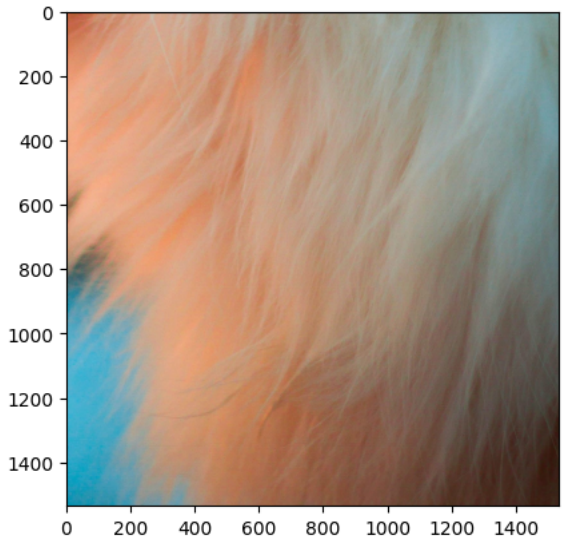

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

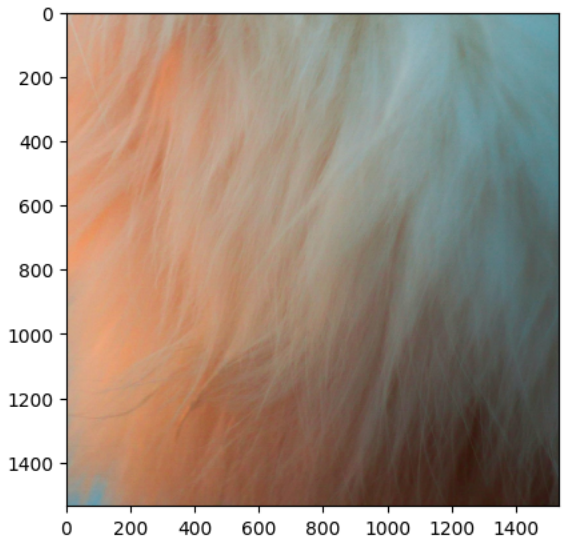

0
200
400
600
800
1000
1200
1400

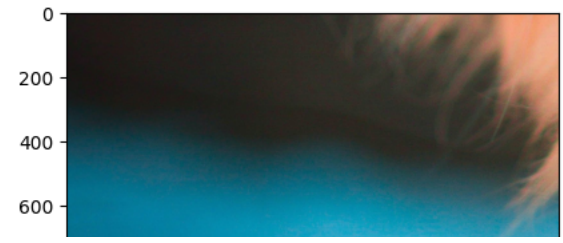

0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000 1200 1400
0 200 400 600 800 1000

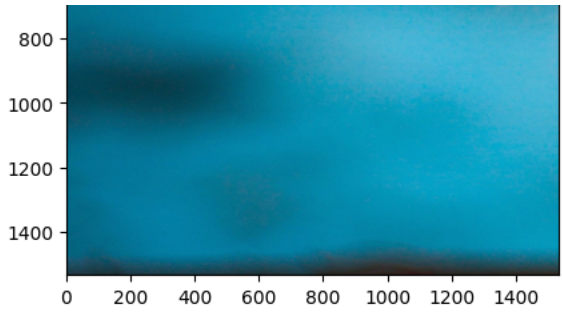

1200
1400
0 200 400 600 800 1000 1200 1400

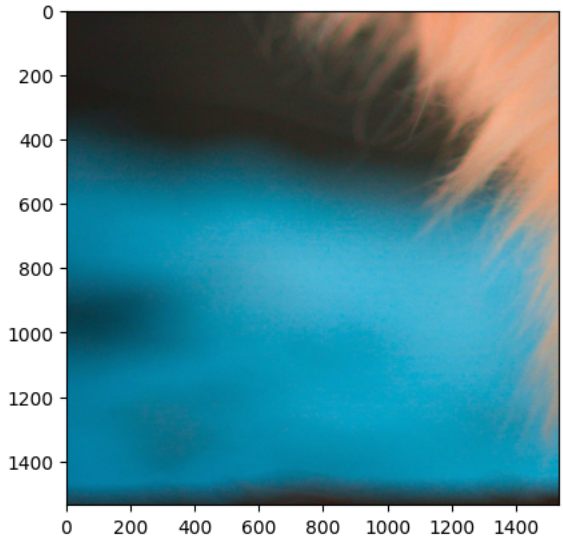

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

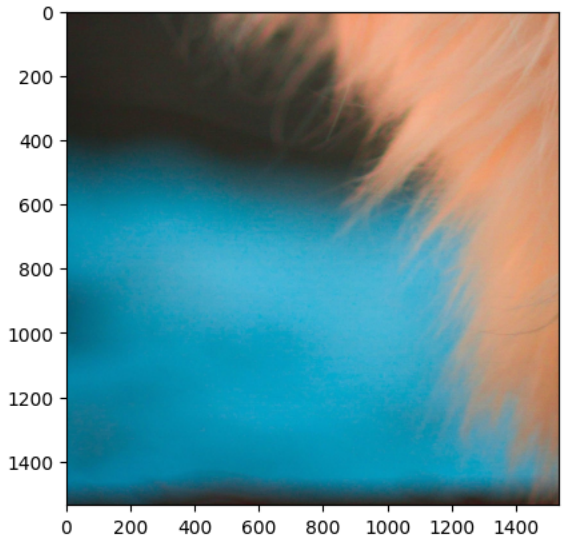

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400

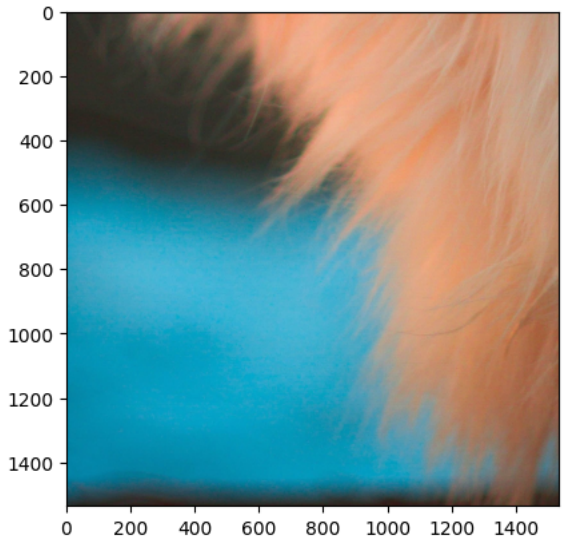

0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200
400
600

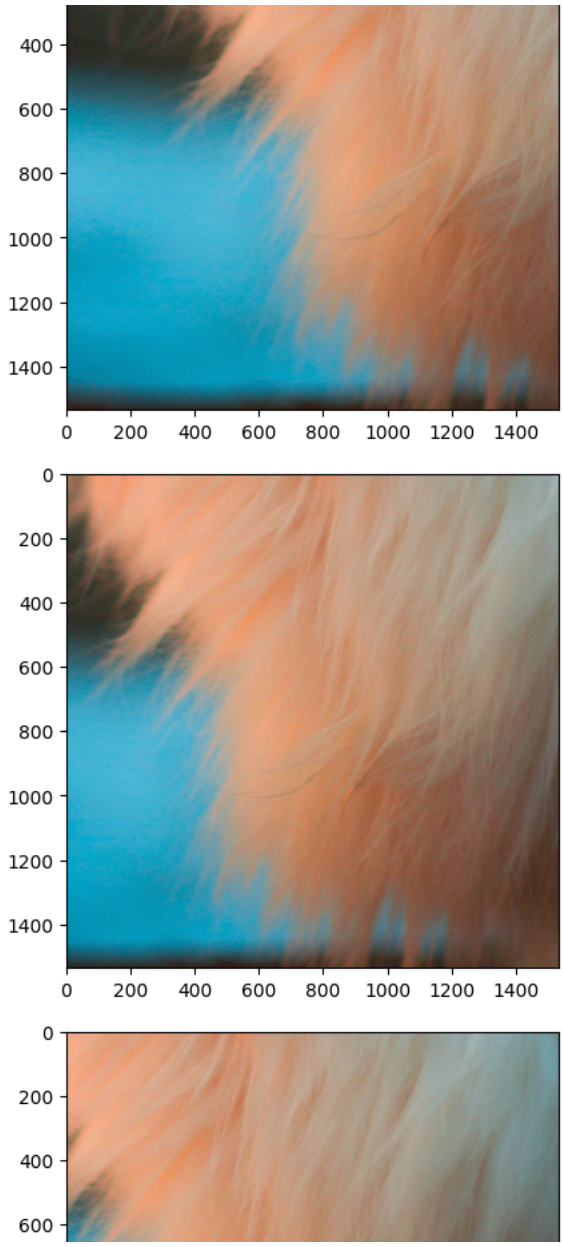

800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400


0
200


400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400
0
200
400
600
800
1000
1200
1400
0 200 400 600 800 1000 1200 1400
0
200
400
600

```
!pip install transformers
```

```
Collecting transformers
```

```
from transformers import CLIPProcessor, CLIPModel
import torch

# define processor and model
model_id = "openai/clip-vit-base-patch32"

processor = CLIPProcessor.from_pretrained(model_id)
model = CLIPModel.from_pretrained(model_id)

# move model to device if possible
device = 'cuda' if torch.cuda.is_available() else 'cpu'

model.to(device)
```

```
(…)32/resolve/main/preprocessor_config.json: 316/316 [00:00<00:00,
100% 7.75kB/s]
(…)tch32/resolve/main/tokenizer_config.json: 568/568 [00:00<00:00,
100% 13.8kB/s]
(…)vit-base- 862k/862k [00:00<00:00,
patch32/resolve/main/vocab.json: 100% 3.30MB/s]
(…)vit-base- 525k/525k [00:00<00:00,
patch32/resolve/main/merges.txt: 100% 11.9MB/s]
(…)base- 2.22M/2.22M [00:00<00:00,
patch32/resolve/main/tokenizer.json: 17.1MB/s]
100%
(…)h32/resolve/main/special_tokens_map.json: 389/389 [00:00<00:00,
100% 8.85kB/s]
(…)it-base- 4.19k/4.19k [00:00<00:00,
patch32/resolve/main/config.json: 100% 98.4kB/s]
pytorch_model.bin: 605M/605M [00:06<00:00,
100% 125MB/s]
CLIPModel(
  (text_model): CLIPTextTransformer(
    (embeddings): CLIPTextEmbeddings(
      (token_embedding): Embedding(49408, 512)
      (position_embedding): Embedding(77, 512)
    )
    (encoder): CLIPEncoder(
      (layers): ModuleList(
        (0-11): 12 x CLIPEncoderLayer(
          (self_attn): CLIPAttention(
            (k_proj): Linear(in_features=512, out_features=512, bias=True)
            (v_proj): Linear(in_features=512, out_features=512, bias=True)
            (q_proj): Linear(in_features=512, out_features=512, bias=True)
            (out_proj): Linear(in_features=512, out_features=512, bias=True)
          )
          (layer_norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (mlp): CLIPMLP(
            (activation_fn): QuickGELUActivation()
            (fc1): Linear(in_features=512, out_features=2048, bias=True)
            (fc2): Linear(in_features=2048, out_features=512, bias=True)
          )
          (layer_norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        )
      )
    )
    (final_layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
  )
  (vision_model): CLIPVisionTransformer(
    (embeddings): CLIPVisionEmbeddings(
      (patch_embedding): Conv2d(3, 768, kernel_size=(32, 32), stride=(32, 32),
bias=False)
      (position_embedding): Embedding(50, 768)
    )
    (pre_layrnorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (encoder): CLIPEncoder(
      (layers): ModuleList(
        (0-11): 12 x CLIPEncoderLayer(
          (self_attn): CLIPAttention(
            (k_proj): Linear(in_features=768, out_features=768, bias=True)
            (v_proj): Linear(in_features=768, out_features=768, bias=True)
            (q_proj): Linear(in_features=768, out_features=768, bias=True)
            (out_proj): Linear(in_features=768, out_features=768, bias=True)
          )
```

```
window = 6
stride = 1

scores = torch.zeros(patches.shape[1], patches.shape[2])
runs = torch.ones(patches.shape[1], patches.shape[2])

for Y in range(0, patches.shape[1]-window+1, stride):
    for X in range(0, patches.shape[2]-window+1, stride):
        big_patch = torch.zeros(patch*window, patch*window, 3)
        patch_batch = patches[0, Y:Y+window, X:X+window]
        for y in range(window):
            for x in range(window):
                big_patch[
                    y*patch:(y+1)*patch, x*patch:(x+1)*patch, :
                ] = patch_batch[y, x].permute(1, 2, 0)
        # we preprocess the image and class label with the CLIP processor
        inputs = processor(
            images=big_patch,  # big patch image sent to CLIP
            return_tensors="pt",  # tell CLIP to return pytorch tensor
            text="a fluffy cat",  # class label sent to CLIP
            padding=True
        ).to(device) # move to device if possible

        # calculate and retrieve similarity score
        score = model(**inputs).logits_per_image.item()
        # sum up similarity scores from current and previous big patches
        # that were calculated for patches within the current window
        scores[Y:Y+window, X:X+window] += score
        # calculate the number of runs on each patch within the current window
        runs[Y:Y+window, X:X+window] += 1
```

```
It looks like you are trying to rescale already rescaled images. If the input images have pixel values between 0 and 1, set `do_resc
```

```
scores /= runs


import numpy as np


# clip the scores
scores = np.clip(scores-scores.mean(), 0, np.inf)

# normalize scores
scores = (
    scores - scores.min()) / (scores.max() - scores.min()
)


scores.shape, patches.shape
```

```
(torch.Size([20, 13]), torch.Size([1, 20, 13, 3, 256, 256]))
```

```
# transform the patches tensor
adj_patches = patches.squeeze(0).permute(3, 4, 2, 0, 1)
adj_patches.shape
```

```
torch.Size([256, 256, 3, 20, 13])
```

```
# multiply patches by scores
adj_patches = adj_patches * scores

# rotate patches to visualize
adj_patches = adj_patches.permute(3, 4, 2, 0, 1)
adj_patches.shape
```

```
torch.Size([20, 13, 3, 256, 256])
```

Visualize Localization

```
Y = adj_patches.shape[0]
X = adj_patches.shape[1]

fig, ax = plt.subplots(Y, X, figsize=(X*.5, Y*.5))
for y in range(Y):
    for x in range(X):
        ax[y, x].imshow(adj_patches[y, x].permute(1, 2, 0))
        ax[y, x].axis("off")
        ax[y, x].set_aspect('equal')
plt.subplots_adjust(wspace=0, hspace=0)
plt.show()
```

```
# scores higher than 0.5 are positive
detection = scores > 0.5


# non-zero positions
np.nonzero(detection)
```

```
[13,  3],
[13,  4],
[13,  5],
[13,  6],
[13,  7],
[13,  8],
[13,  9],
[13, 10],
[14,  2],
[14,  3],
[14,  4],
[14,  5],
[14,  6],
[14,  7],
[14,  8],
[14,  9],
[14, 10],
[15,  2],
[15,  3],
[15,  4],
[15,  5],
[15,  6],
[15,  7],
[15,  8],
[15,  9],
[16,  3],
[16,  4],
[16,  5],
[16,  6],
[16,  7],
[16,  8],
[16,  9],
[17,  4],
[17,  5],
[17,  6],
[17,  7]])
```

```
y_min, y_max = (
    np.nonzero(detection)[:,0].min().item(),
    np.nonzero(detection)[:,0].max().item()+1
)
y_min, y_max
```

```
(2, 18)
```

```
x_min, x_max = (
    np.nonzero(detection)[:,1].min().item(),
    np.nonzero(detection)[:,1].max().item()+1
)
x_min, x_max
```

```
(2, 11)
```

```
y_min *= patch
y_max *= patch
x_min *= patch
x_max *= patch
x_min, y_min
```

```
(512, 512)
```

```
height = y_max - y_min
width = x_max - x_min

height, width
```

```
(4096, 2304)
```

```
# image shape
img.data.numpy().shape
```

```
(3, 5184, 3456)
```

```
# move color channel to final dim
image = np.moveaxis(img.data.numpy(), 0, -1)
image.shape
```

```
(5184, 3456, 3)
```