



Debugging training PyTorch code

Dmytro Mishkin, FEE, CTU in Prague

**Everything from the
“Debugging handcrafted PyTorch code”
applies here as well**

Checklist

Recap from handcrafted code

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?
3. Did I printed shape, data types, and values?

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?
3. Did I printed shape, data types, and values?
4. Did I checked for a stupid mistakes? Like typos in variable names, naming variables as function

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?
3. Did I printed shape, data types, and values?
4. Did I checked for a stupid mistakes? Like typos in variable names, naming variables as function
5. Did I checked library versions and updates?

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?
3. Did I printed shape, data types, and values?
4. Did I checked for a stupid mistakes? Like typos in variable names, naming variables as function
5. Did I checked library versions and updates?
 - E.g. old `torch.solve(B, A)`, but `torch.linalg.solve(A, B)`

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?
3. Did I printed shape, data types, and values?
4. Did I checked for a stupid mistakes? Like typos in variable names, naming variables as function
5. Did I checked library versions and updates?
 - E.g. old `torch.solve(B, A)`, but `torch.linalg.solve(A, B)`
6. Do I have NaN-prone operations?

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?
3. Did I printed shape, data types, and values?
4. Did I checked for a stupid mistakes? Like typos in variable names, naming variables as function
5. Did I checked library versions and updates?
 - E.g. old `torch.solve(B, A)`, but `torch.linalg.solve(A, B)`
6. Do I have NaN-prone operations?
 - e.g. `log`, `sqrt`, `division`, etc. Use `eps` there or some kind of guards

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?
3. Did I printed shape, data types, and values?
4. Did I checked for a stupid mistakes? Like typos in variable names, naming variables as function
5. Did I checked library versions and updates?
 - E.g. old `torch.solve(B, A)`, but `torch.linalg.solve(A, B)`
6. Do I have NaN-prone operations?
 - e.g. `log`, `sqrt`, `division`, etc. Use `eps` there or some kind of guards
7. Do I have some memory sharing?

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?
3. Did I printed shape, data types, and values?
4. Did I checked for a stupid mistakes? Like typos in variable names, naming variables as function
5. Did I checked library versions and updates?
 - E.g. old `torch.solve(B, A)`, but `torch.linalg.solve(A, B)`
6. Do I have NaN-prone operations?
 - e.g. `log`, `sqrt`, `division`, etc. Use `eps` there or some kind of guards
7. Do I have some memory sharing?
8. Is there anything hardcoded?

Checklist

Recap from handcrafted code

1. Did I prepared minimal input and expected output? Math-based, or reliable library based
2. Did I visualize everything?
3. Did I printed shape, data types, and values?
4. Did I checked for a stupid mistakes? Like typos in variable names, naming variables as function
5. Did I checked library versions and updates?
 - E.g. old `torch.solve(B, A)`, but `torch.linalg.solve(A, B)`
6. Do I have NaN-prone operations?
 - e.g. `log`, `sqrt`, `division`, etc. Use `eps` there or some kind of guards
7. Do I have some memory sharing?
8. Is there anything hardcoded?
9. Can the bug in one function be compensated by other bug in other function?

Neural networks fail silently

And that is the problem

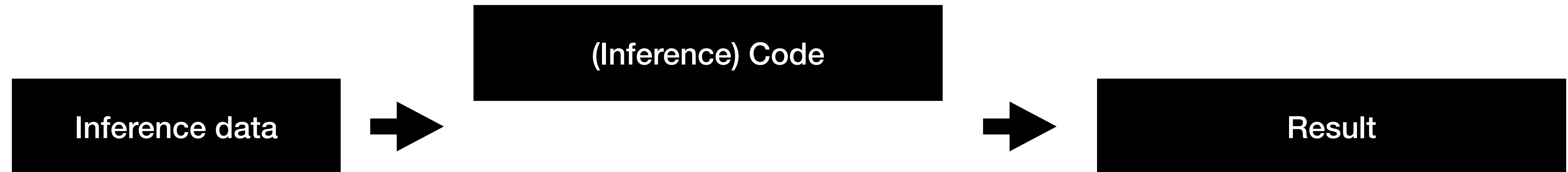
Handcrafted code flow



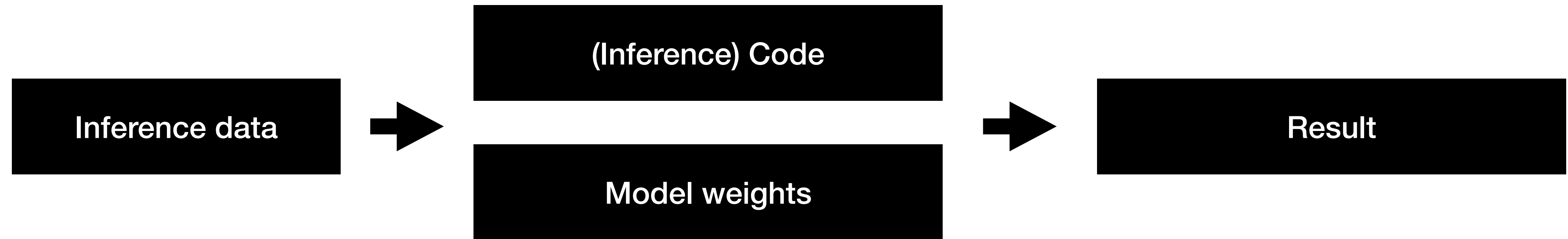
Deep Learning model code flow



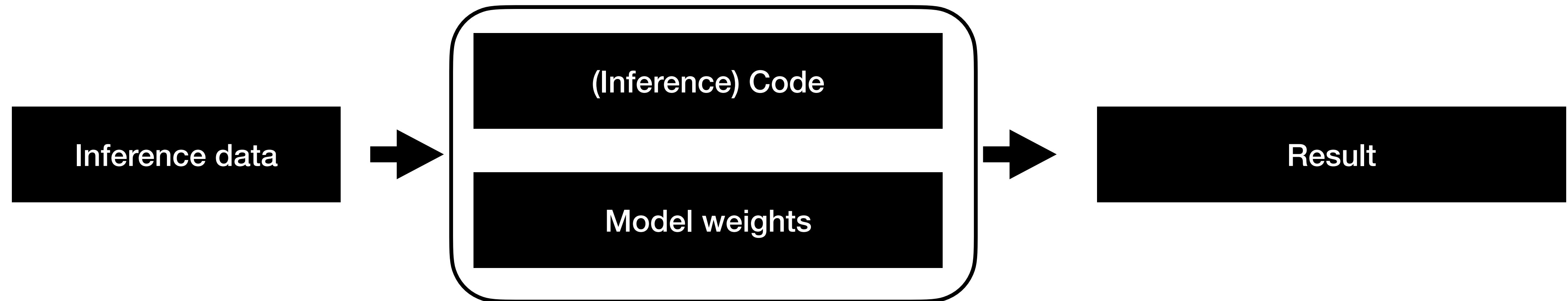
Deep Learning model code flow



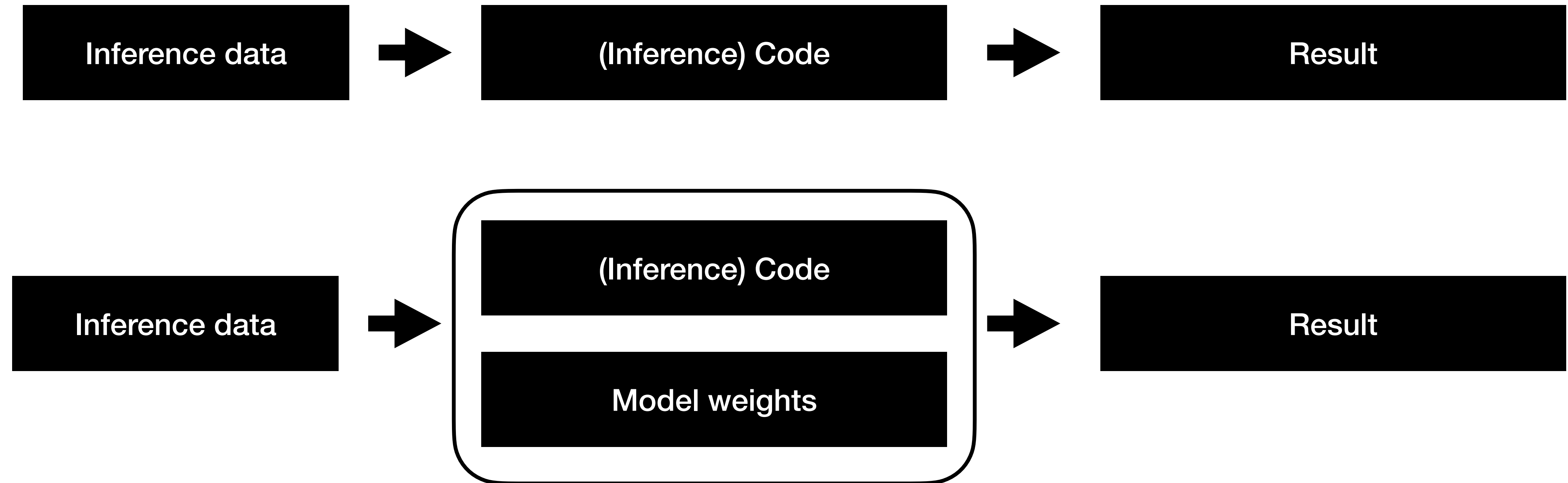
Deep Learning model code flow



Deep Learning model code flow



Inference code flow



Issues without training

Issues without training

- Not loading the weights

Issues without training

- Not loading the weights
- Not moving to `.eval()` mode

Issues without training

- Not loading the weights
- Not moving to `.eval()` mode
- Applying wrong preprocessing

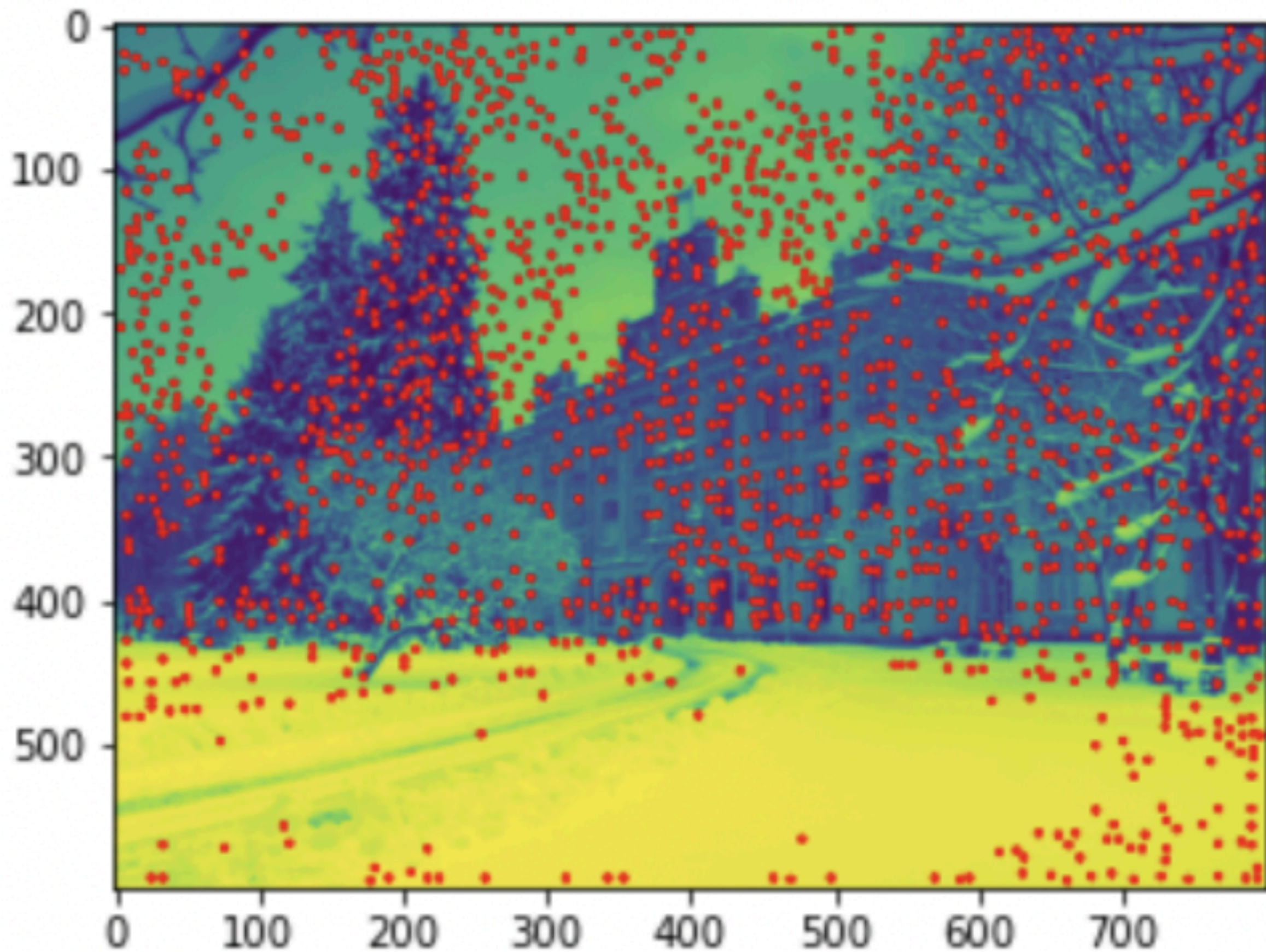
Not loading the weights

Happened to me, playing with SuperPoint

Not loading the weights

Happened to me, playing with SuperPoint

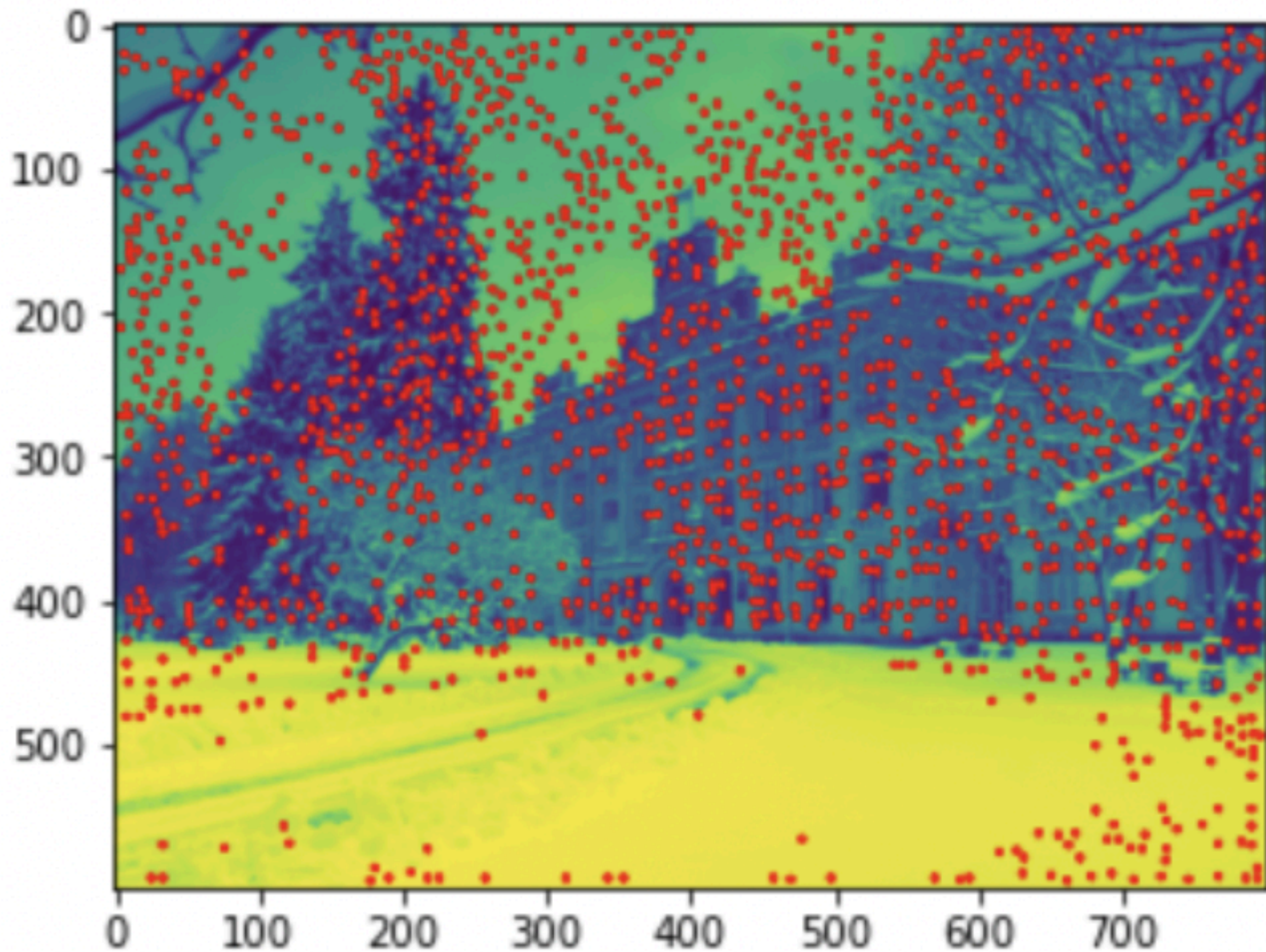
Model 1



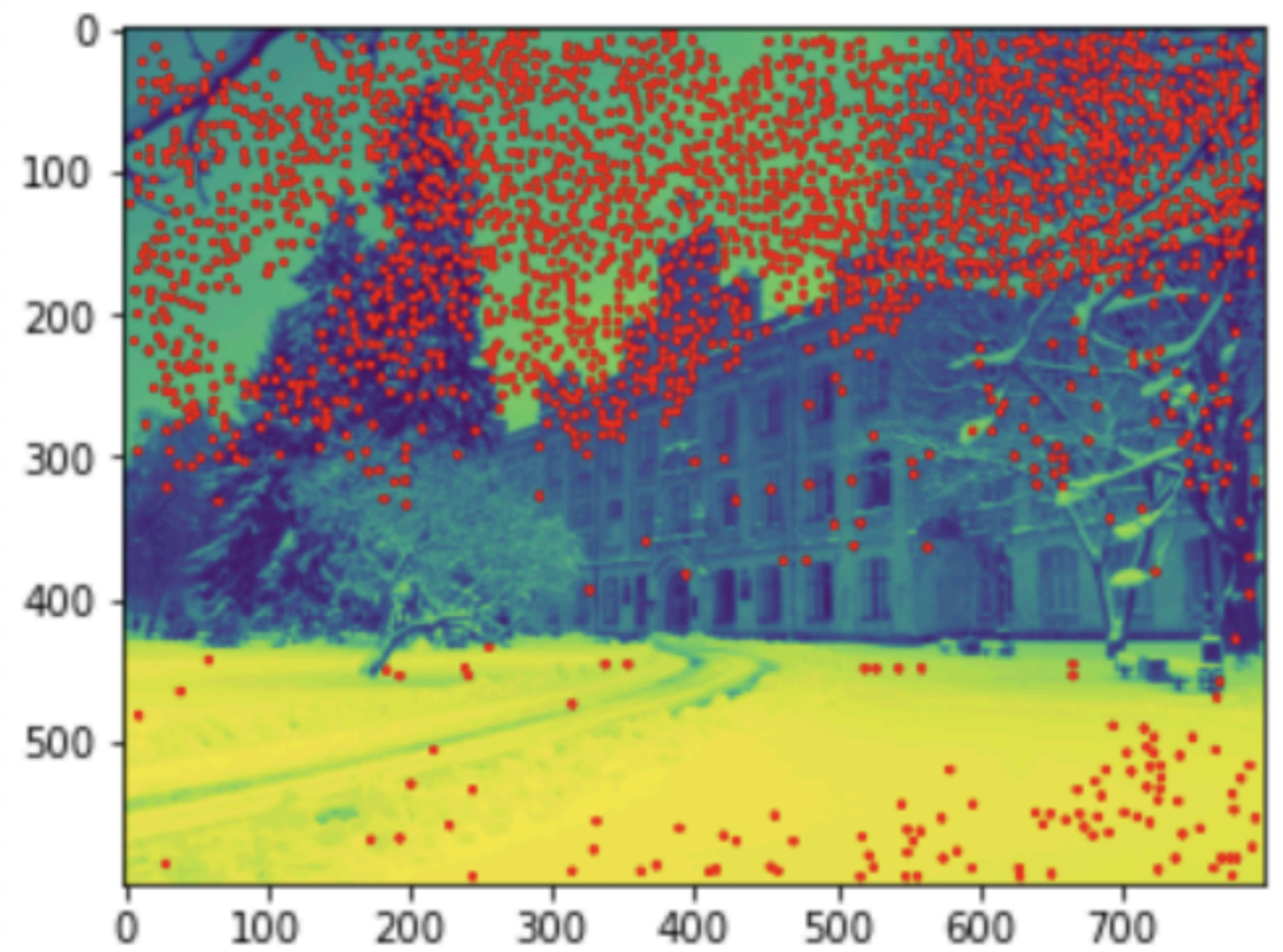
Not loading the weights

Happened to me, playing with SuperPoint

Model 1



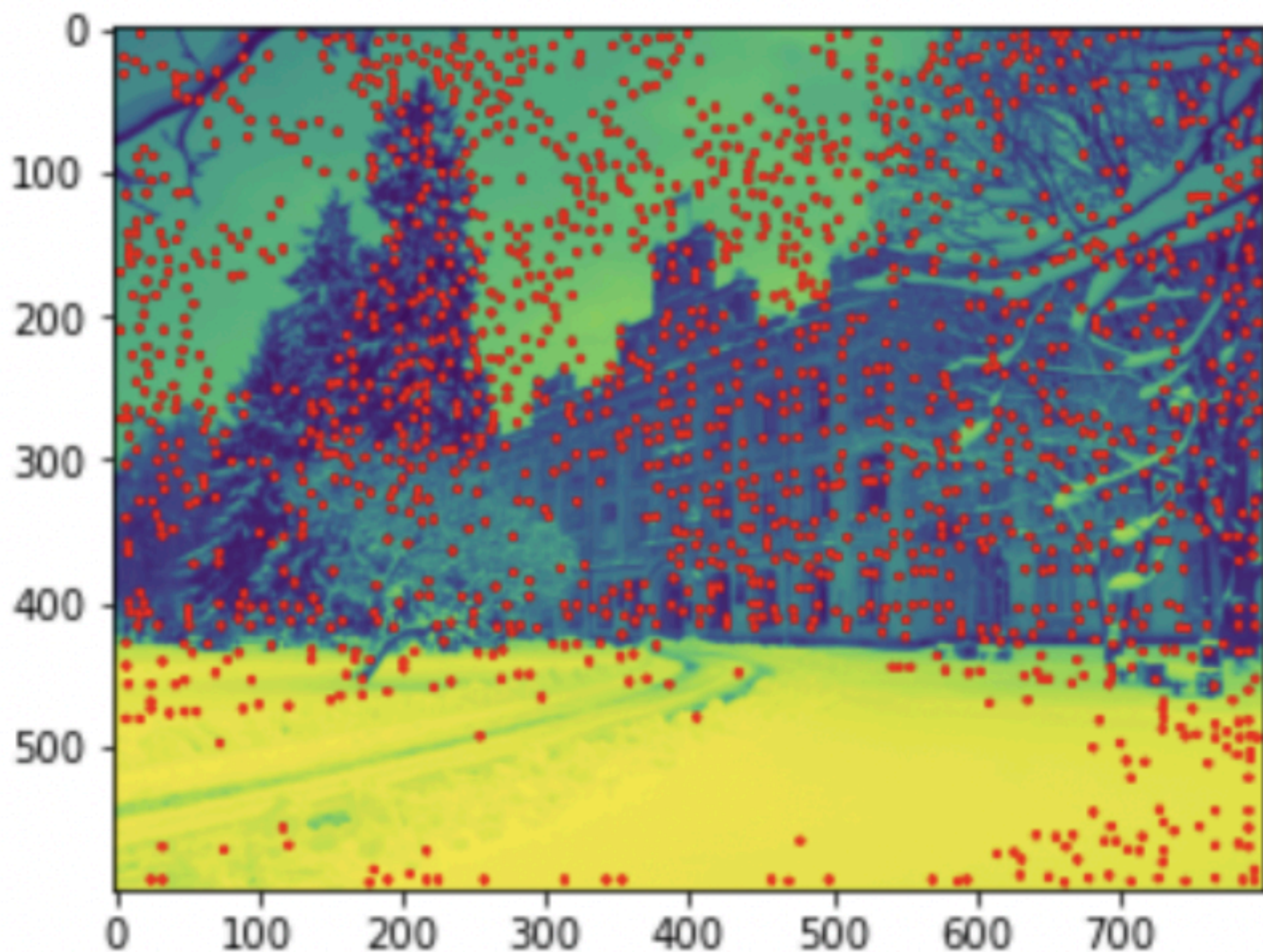
Model 2



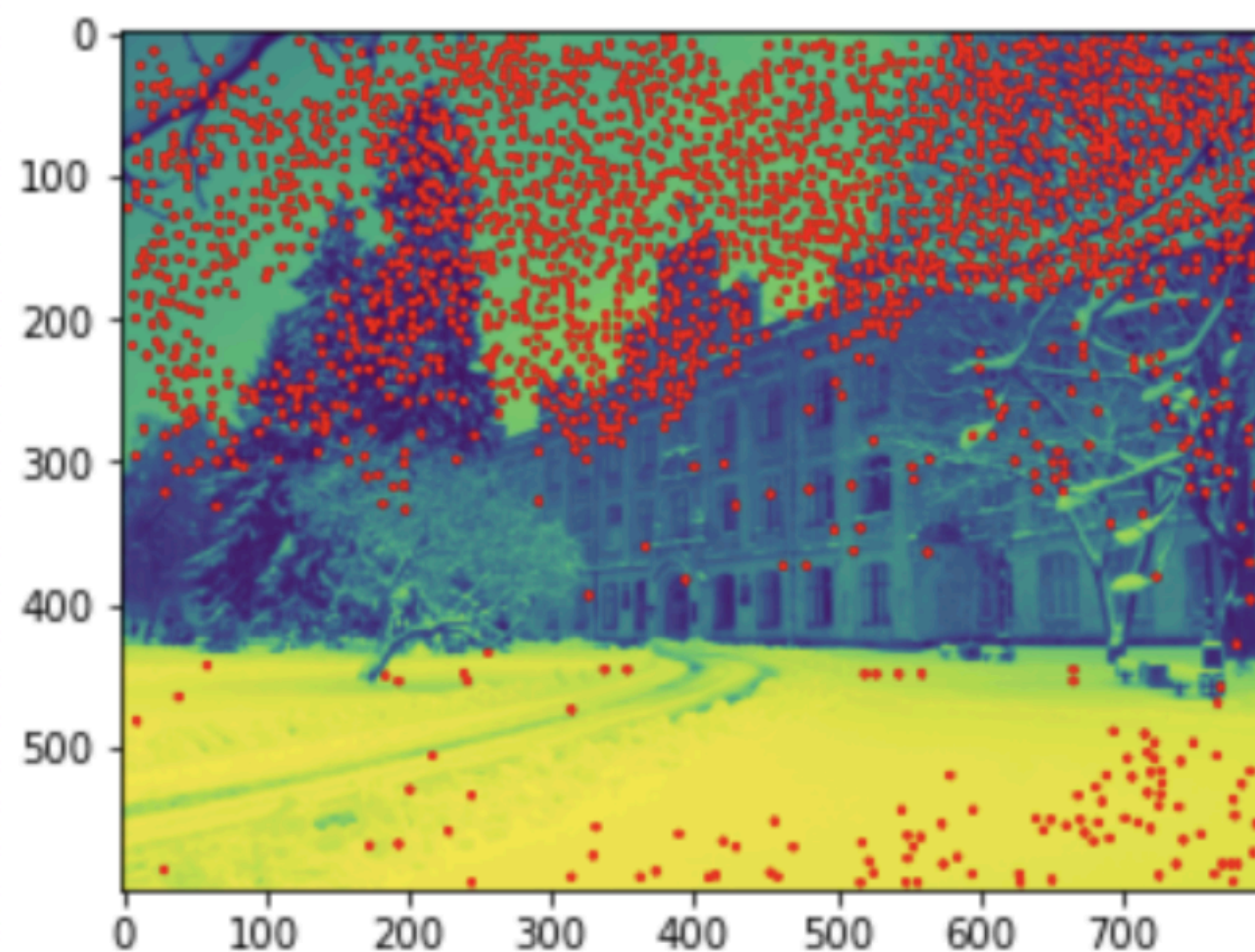
Not loading the weights

Happened to me, playing with SuperPoint

Model 1



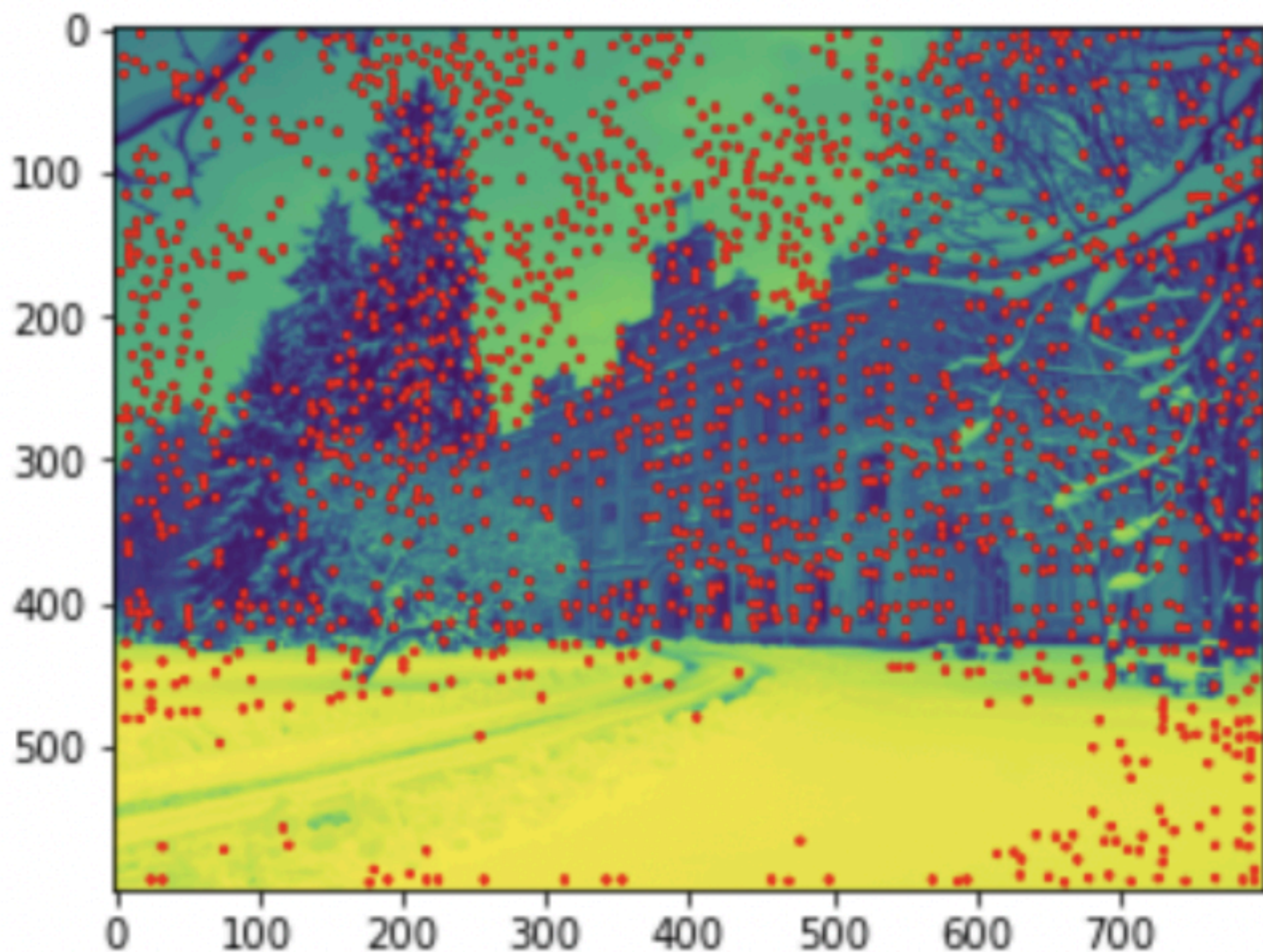
Model 2



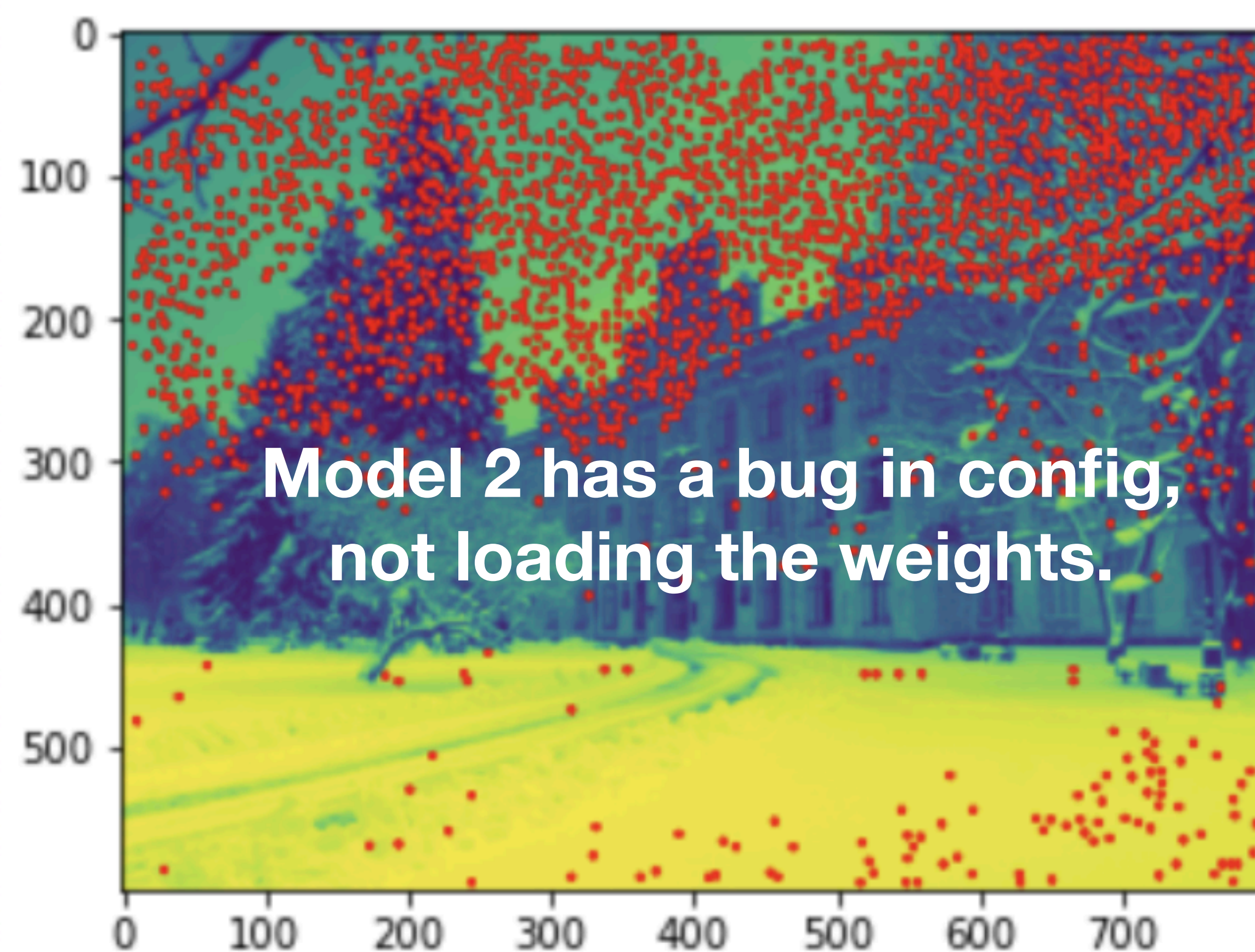
Not loading the weights

Happened to me, playing with SuperPoint

Model 1



Model 2

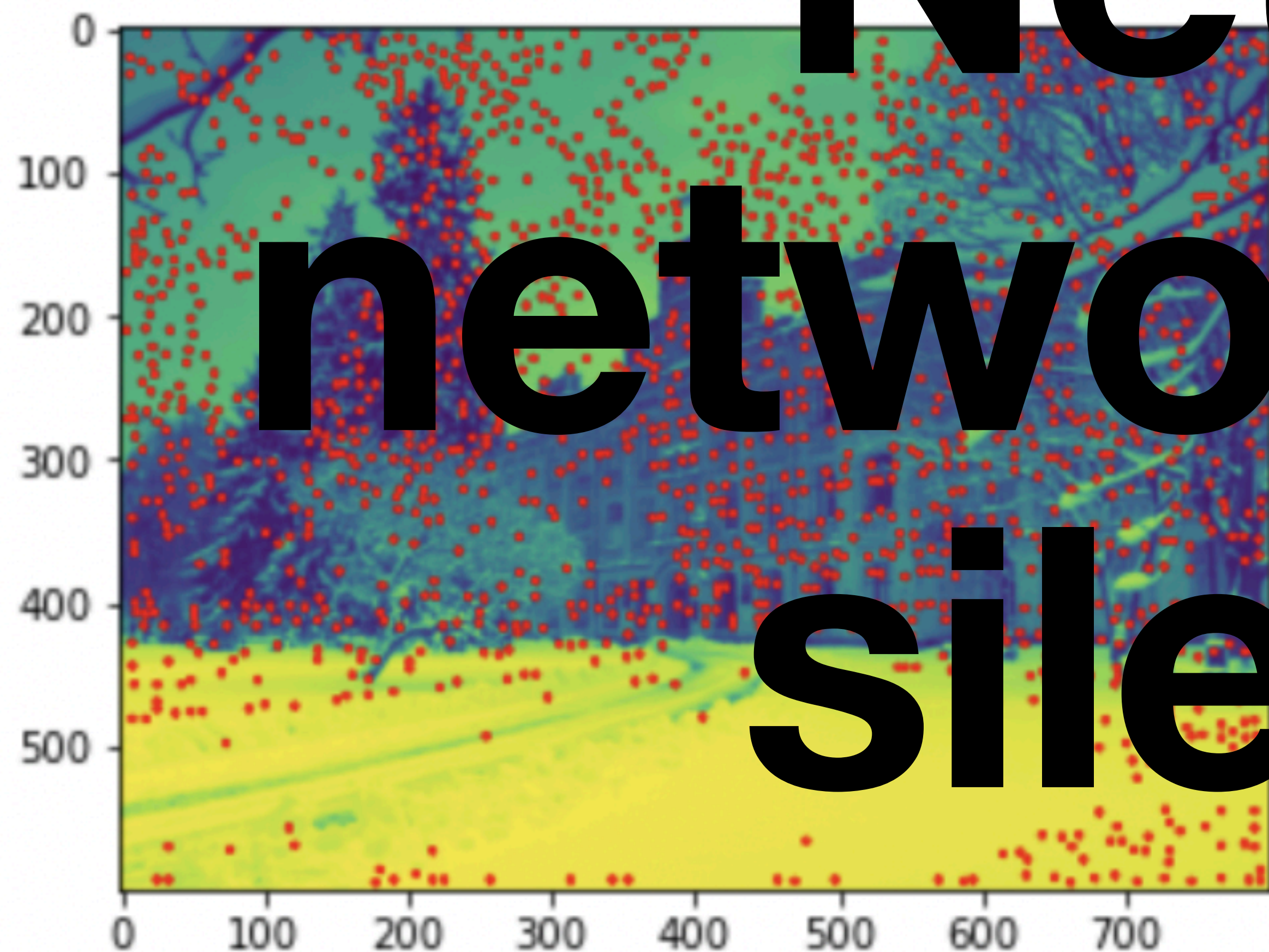


Not loading the weights

Happened to me, playing with SuperPoint

Model 1

Model 2



Model 2 has a bug in config,
not loading the weights.

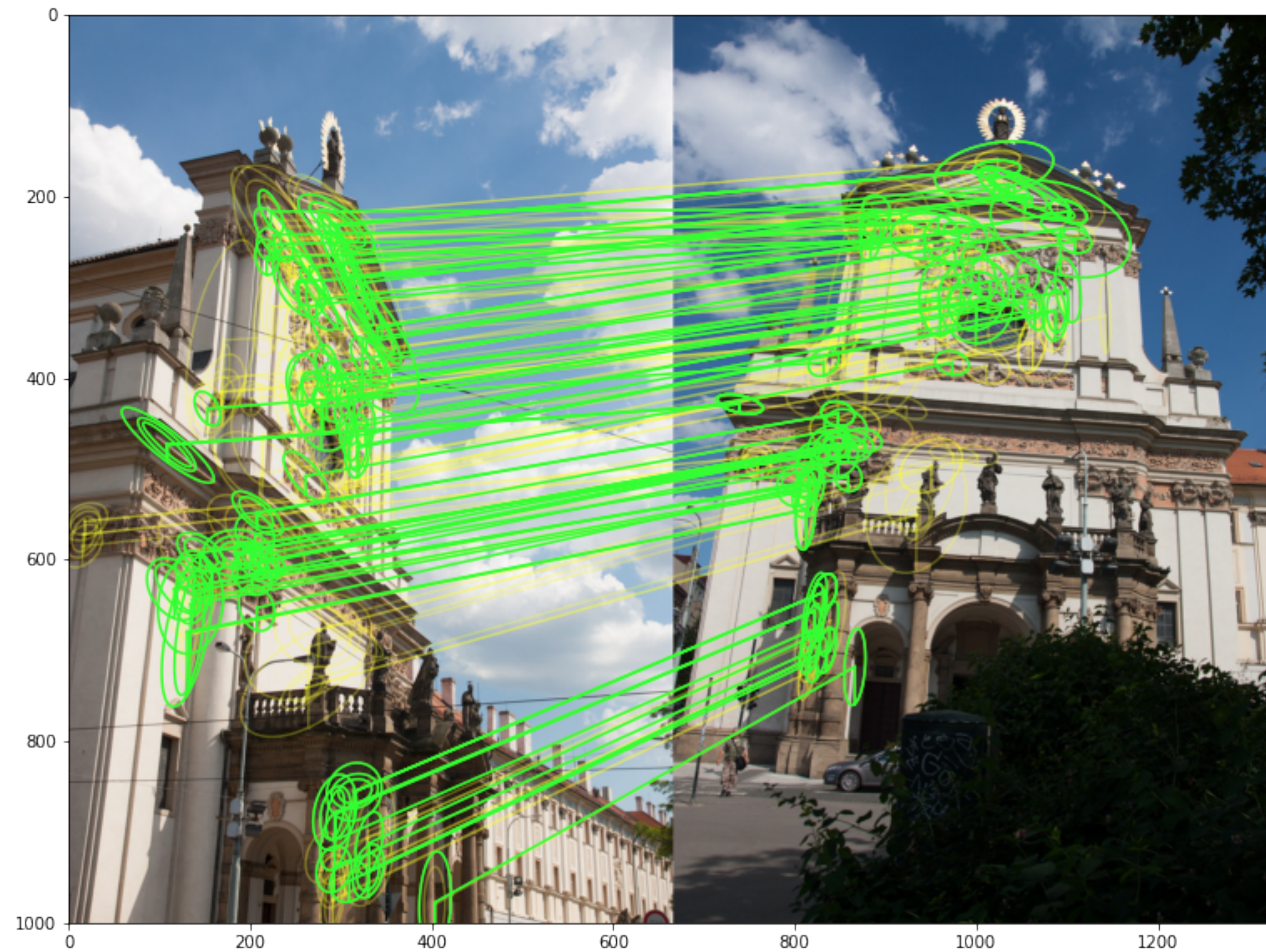
Neural
networks fail
silently

Not moving to eval mode

Model often works, just worse

Not moving to eval mode

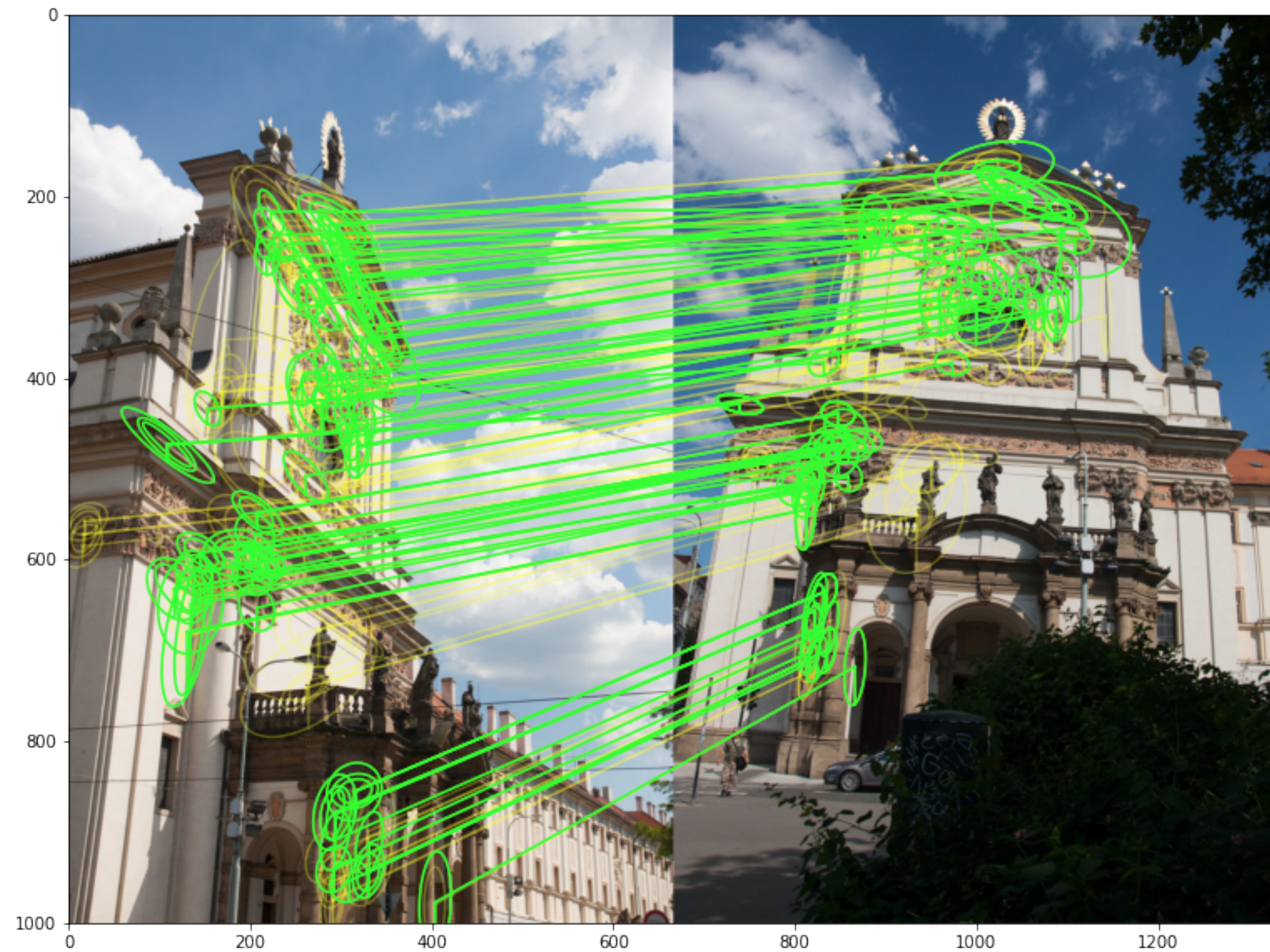
Model often works, just worse



Not moving to eval mode

Model often works, just worse

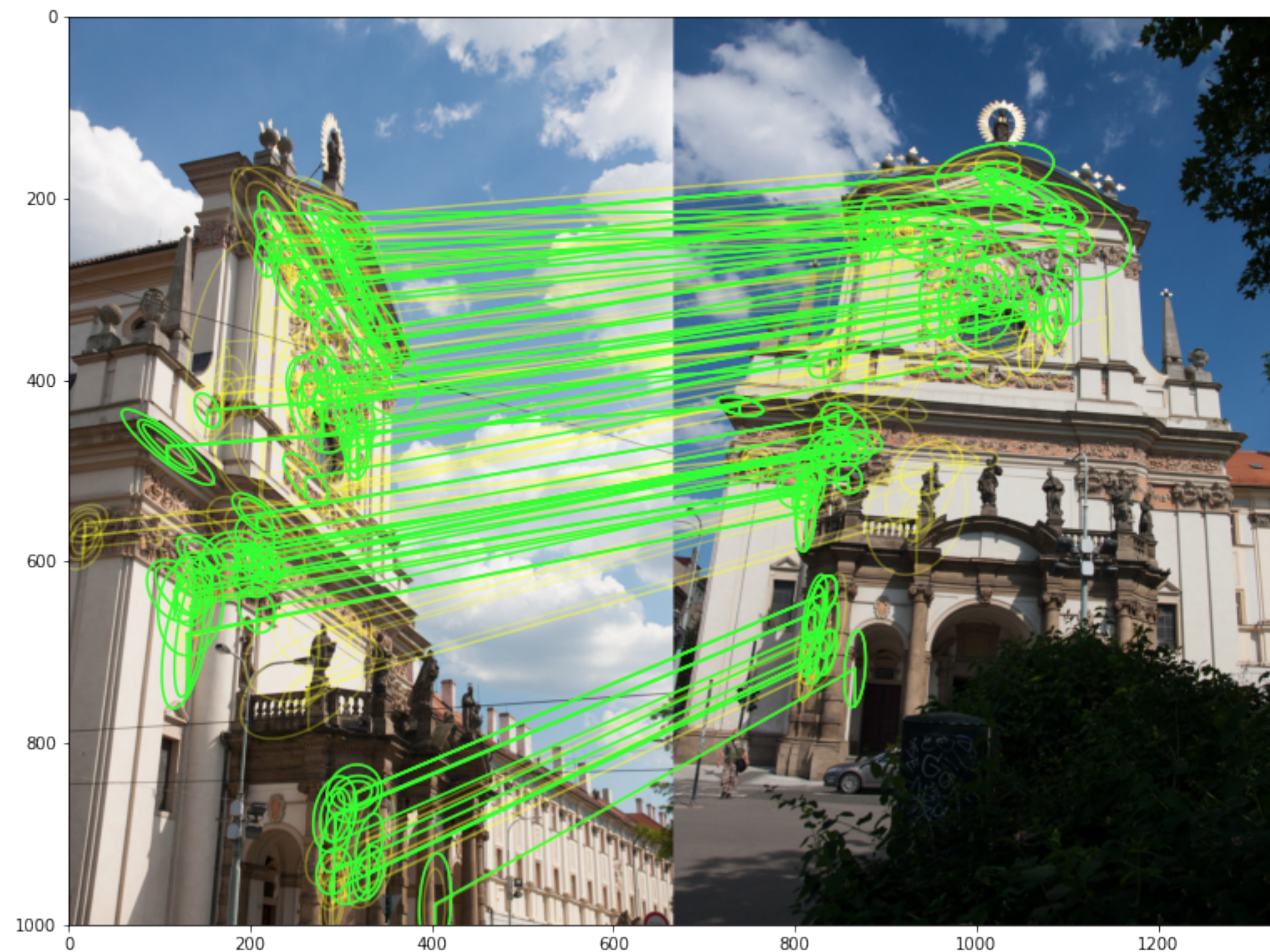
`model.train()`, 80 inliers



Not moving to eval mode

Model often works, just worse

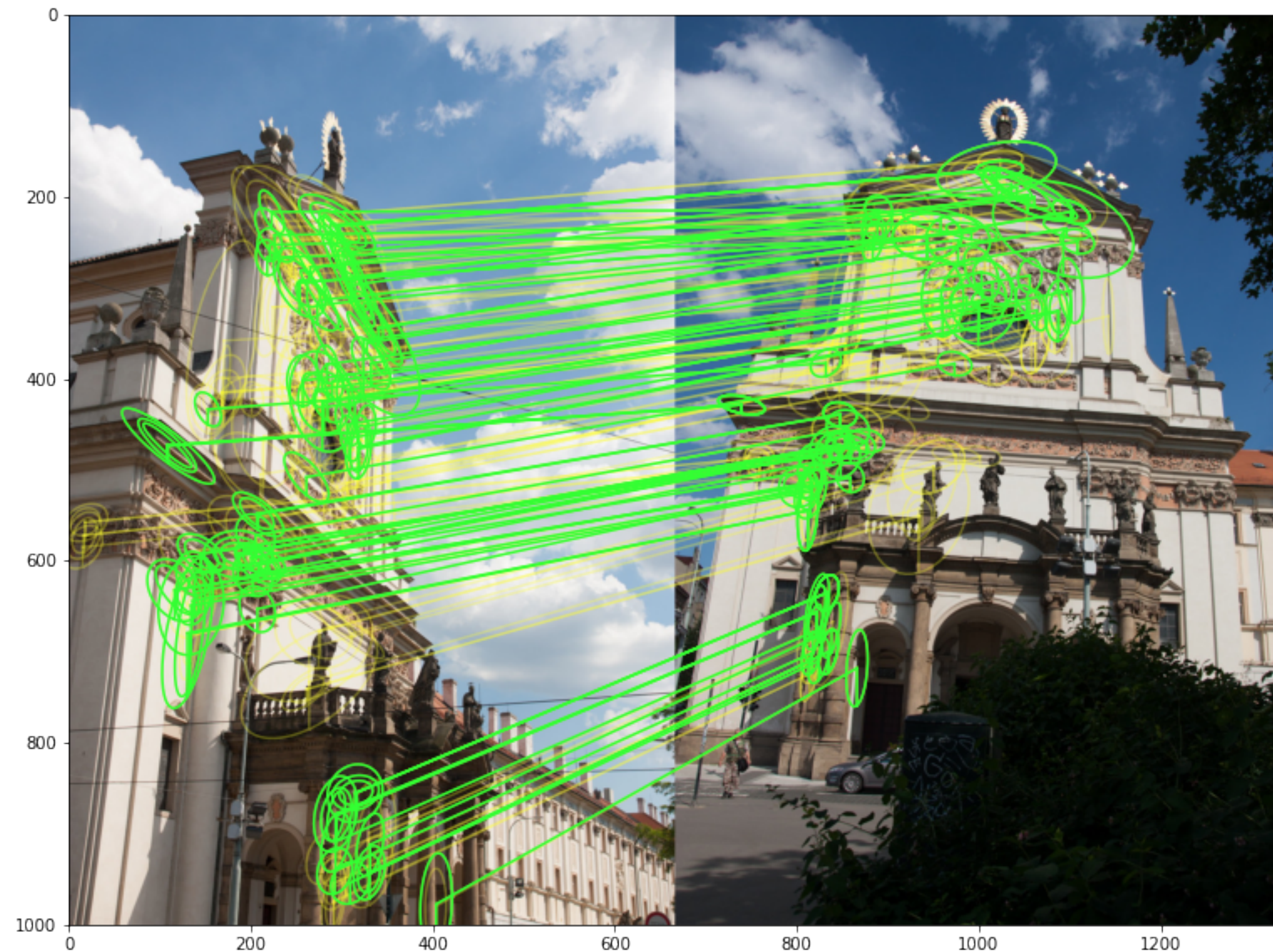
`model.train()`, 80 inliers



Not moving to eval mode

Model often works, just worse

`model.train()`, 80 inliers



`model.eval()`, 179 inliers



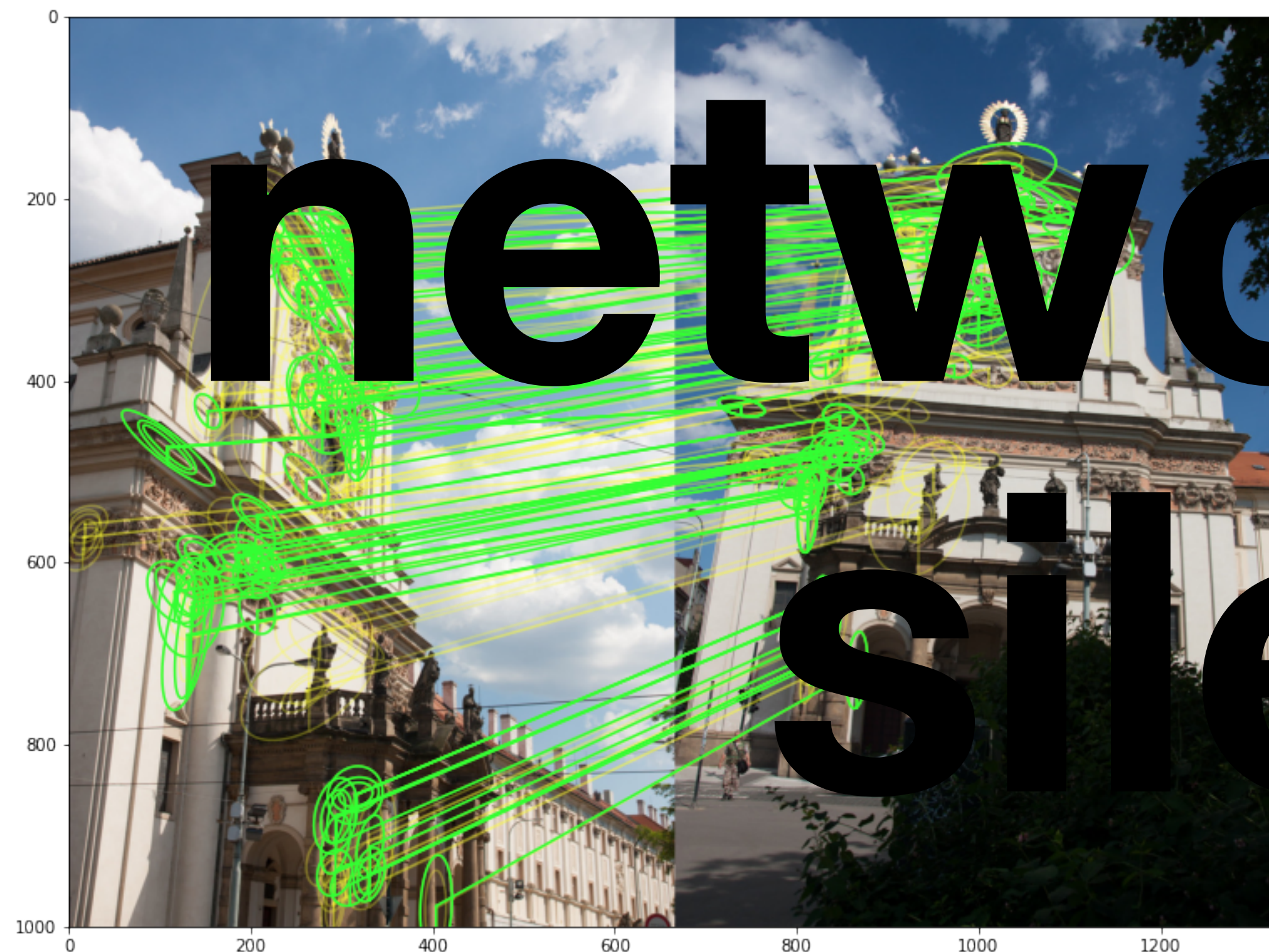
Not moving to eval mode

Model often works just worse

Neural

model.train(), 80 inliers

model.eval(), 179 inliers



Wrong preprocessing

RGB/BGR, mean, std...

```
1 import timm
2
3 import timm
4 from timm.data import resolve_data_config
5 from timm.data.transforms_factory import create_transform
6
7
8 model1 = timm.create_model('vgg16_bn')
9 config = resolve_data_config({}, model=model1)
10 transform = create_transform(**config)
11 print(transform)
12
13 model1 = timm.create_model('vit_large_patch14_xp_224')
14 config = resolve_data_config({}, model=model1)
15 transform = create_transform(**config)
16 print(transform)
17
18
```

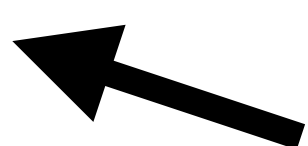
```
Compose(
  Resize(size=256, interpolation=bilinear, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.4850, 0.4560, 0.4060]), std=tensor([0.2290, 0.2240, 0.2250]))
)
Compose(
  Resize(size=248, interpolation=bicubic, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.5000, 0.5000, 0.5000]), std=tensor([0.5000, 0.5000, 0.5000]))
)
```

Wrong preprocessing

RGB/BGR, mean, std...

```
1 import timm
2
3 import timm
4 from timm.data import resolve_data_config
5 from timm.data.transforms_factory import create_transform
6
7
8 model1 = timm.create_model('vgg16_bn')
9 config = resolve_data_config({}, model=model1)
10 transform = create_transform(**config)
11 print(transform)
12
13 model1 = timm.create_model('vit_large_patch14_xp_224')
14 config = resolve_data_config({}, model=model1)
15 transform = create_transform(**config)
16 print(transform)
17
18
```

```
Compose(
  Resize(size=256, interpolation=bilinear, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.4850, 0.4560, 0.4060]), std=tensor([0.2290, 0.2240, 0.2250]))
)
Compose(
  Resize(size=248, interpolation=bicubic, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.5000, 0.5000, 0.5000]), std=tensor([0.5000, 0.5000, 0.5000]))
)
```

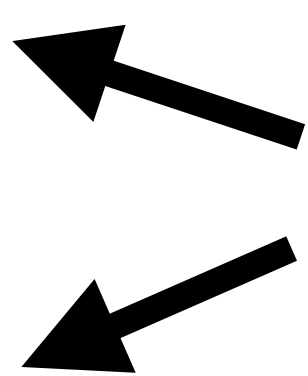


Wrong preprocessing

RGB/BGR, mean, std...

```
1 import timm
2
3 import timm
4 from timm.data import resolve_data_config
5 from timm.data.transforms_factory import create_transform
6
7
8 model1 = timm.create_model('vgg16_bn')
9 config = resolve_data_config({}, model=model1)
10 transform = create_transform(**config)
11 print(transform)
12
13 model1 = timm.create_model('vit_large_patch14_xp_224')
14 config = resolve_data_config({}, model=model1)
15 transform = create_transform(**config)
16 print(transform)
17
18
```

```
Compose(
  Resize(size=256, interpolation=bilinear, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.4850, 0.4560, 0.4060]), std=tensor([0.2290, 0.2240, 0.2250]))
)
Compose(
  Resize(size=248, interpolation=bicubic, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.5000, 0.5000, 0.5000]), std=tensor([0.5000, 0.5000, 0.5000]))
)
```



Wrong preprocessing

RGB/BGR, mean, std...

```
1 import timm
2
3 import timm
4 from timm.data import resolve_data_config
5 from timm.data.transforms_factory import create_transform
6
7
8 model1 = timm.create_model('vgg16_bn')
9 config = resolve_data_config({}, model=model1)
10 transform = create_transform(**config)
11 print(transform)
12
13 model1 = timm.create_model('vit_large_patch14_xp_224')
14 config = resolve_data_config({}, model=model1)
15 transform = create_transform(**config)
16 print(transform)
17
18
```

```
Compose(
  Resize(size=256, interpolation=bilinear, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.4850, 0.4560, 0.4060]), std=tensor([0.2290, 0.2240, 0.2250]))
)
Compose(
  Resize(size=248, interpolation=bicubic, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.5000, 0.5000, 0.5000]), std=tensor([0.5000, 0.5000, 0.5000]))
)
```

2x difference in std



Wrong preprocessing

RGB/BGR, mean, std...

```
1 import timm
2
3 import timm
4 from timm.data import resolve_data_config
5 from timm.data.transforms_factory import create_transform
6
7
8 model1 = timm.create_model('vgg16_bn')
9 config = resolve_data_config({}, model=model1)
10 transform = create_transform(**config)
11 print(transform)
12
13 model1 = timm.create_model('vit_large_patch14_xp_224')
14 config = resolve_data_config({}, model=model1)
15 transform = create_transform(**config)
16 print(transform)
17
18
```

Good practice:
store the preprocessing code
together with your model

```
Compose(
  Resize(size=256, interpolation=bilinear, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.4850, 0.4560, 0.4060]), std=tensor([0.2290, 0.2240, 0.2250]))
)
Compose(
  Resize(size=248, interpolation=bicubic, max_size=None, antialias=None)
  CenterCrop(size=(224, 224))
  ToTensor()
  Normalize(mean=tensor([0.5000, 0.5000, 0.5000]), std=tensor([0.5000, 0.5000, 0.5000]))
)
```

2x difference in std

Wrong preprocessing

Another realworld example...

LoFTR pretrained='indoor' does not work #2257

✓ Closed

schwarzwalder93 opened this issue on Mar 5 · 7 comments



schwarzwalder93 commented on Mar 5 · edited ▼

...

Describe the bug

LoFTR for image matching does not produce expected results when choosing the model pretrained on indoor datasets (`KF.LoFTR(pretrained='indoor')`)

Wrong preprocessing

Another realworld example...

LoFTR pretrained='indoor' does not work #2257

Wrong preprocessing

Another realworld example...

LoFTR pretrained='indoor' does not work #2257



Wrong preprocessing

Another realworld example...

LoFTR pretrained='indoor' does not work #2257

Indeed ...



Wrong preprocessing

Another realworld example...

LoFTR pretrained='indoor' does not work #2257

Wrong preprocessing

Another realworld example...

LoFTR pretrained='indoor' does not work #2257

However...LoFTR indoor model was trained on 640 x 480 images and we are providing 1296 x 968

Wrong preprocessing

Another realworld example...

LoFTR pretrained='indoor' does not work #2257

However...LoFTR indoor model was trained on 640 x 480 images and we are providing 1296 x 968

What if we resize?

Wrong preprocessing

Another realworld example...

LoFTR pretrained='indoor' does not work #2257

However...LoFTR indoor model was trained on 640 x 480 images and we are providing 1296 x 968

What if we resize?

```
img1 = load_torch_image(fname1)
img2 = load_torch_image(fname2)
img1 = K.geometry.resize(img1, (480, 640), antialias=True)
img2 = K.geometry.resize(img2, (480, 640), antialias=True)
```

Wrong preprocessing

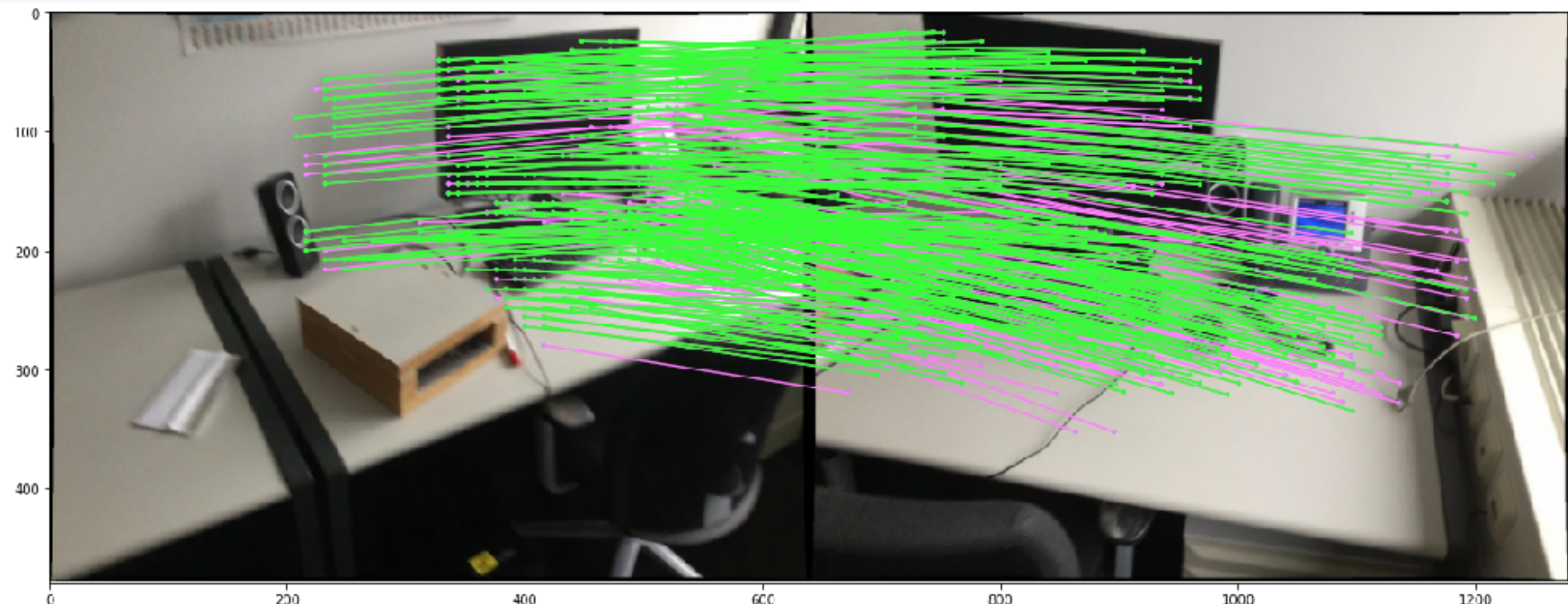
Another realworld example...

LoFTR pretrained='indoor' does not work #2257

However...LoFTR indoor model was trained on 640 x 480 images and we are providing 1296 x 968

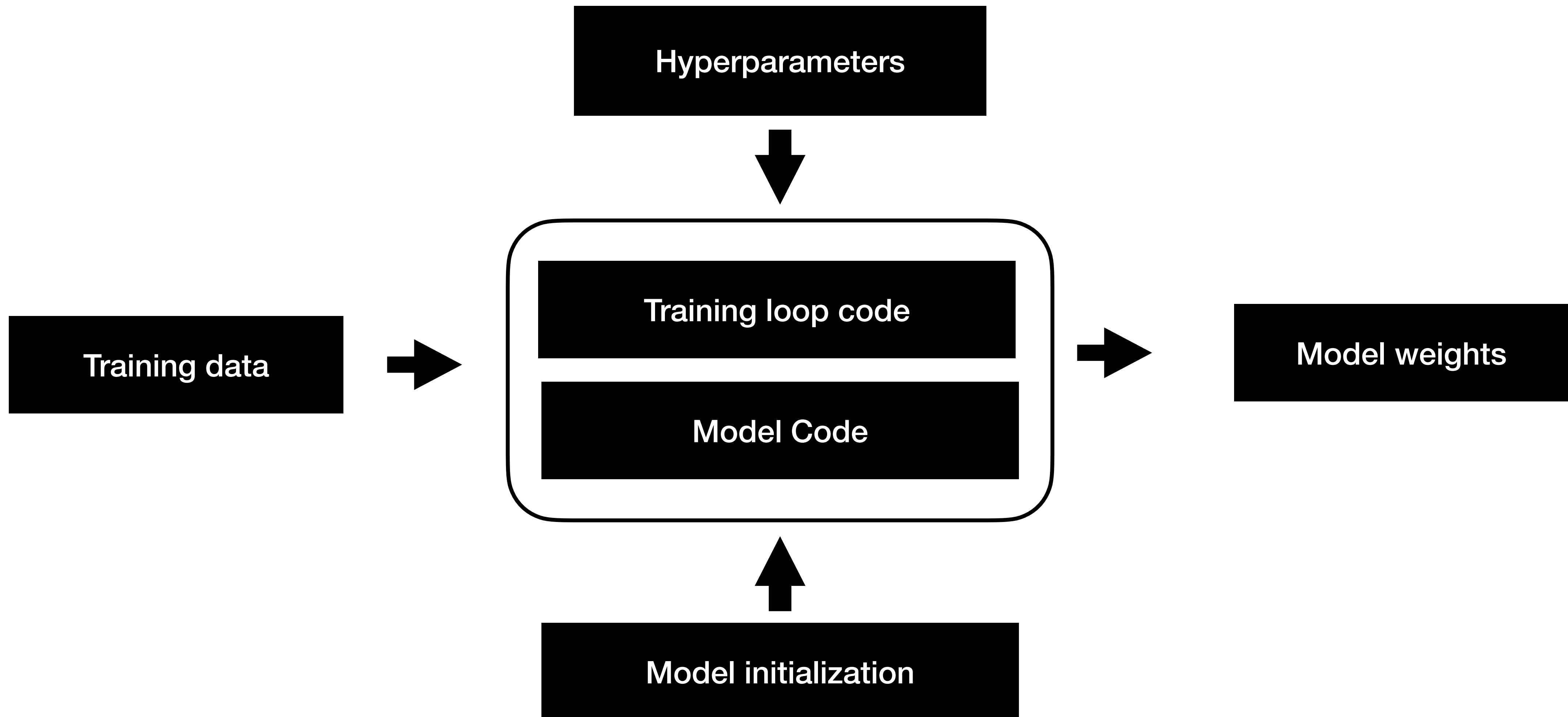
What if we resize?

```
img1 = load_torch_image(fname1)
img2 = load_torch_image(fname2)
img1 = K.geometry.resize(img1, (480, 640), antialias=True)
img2 = K.geometry.resize(img2, (480, 640), antialias=True)
```



Finally, training

Training code flow



The process is very complex

The process is very complex

That's why you should keep everything else simple

Andrej Karpathy pipeline

Andrej Karpathy pipeline

1. Become one with the data

Andrej Karpathy pipeline

1. Become one with the data
2. Set up the end-to-end training/evaluation skeleton + get dumb baselines

Andrej Karpathy pipeline

1. Become one with the data
2. Set up the end-to-end training/evaluation skeleton + get dumb baselines
3. Overfit

Andrej Karpathy pipeline

1. Become one with the data
2. Set up the end-to-end training/evaluation skeleton + get dumb baselines
3. Overfit
 - 3.1. to a single batch for the beginning

Andrej Karpathy pipeline

1. Become one with the data
2. Set up the end-to-end training/evaluation skeleton + get dumb baselines
3. Overfit
 - 3.1. to a single batch for the beginning
4. Regularize

Andrej Karpathy pipeline

First overfit

Andrej Karpathy pipeline

First overfit

1. fix random seed (Jeremy Howard doesn't agree)

Andrej Karpathy pipeline

First overfit

1. fix random seed (Jeremy Howard doesn't agree)
2. simplify

Andrej Karpathy pipeline

First overfit

1. fix random seed (Jeremy Howard doesn't agree)
2. simplify
3. verify loss @ init

Andrej Karpathy pipeline

First overfit

1. fix random seed (Jeremy Howard doesn't agree)
2. simplify
3. verify loss @ init
4. init well

Andrej Karpathy pipeline

First overfit

1. fix random seed (Jeremy Howard doesn't agree)
2. simplify
3. verify loss @ init
4. init well
5. input-indepent baseline

Andrej Karpathy pipeline

First overfit

1. fix random seed (Jeremy Howard doesn't agree)
2. simplify
3. verify loss @ init
4. init well
5. input-indepent baseline
6. verify decreasing training loss

Andrej Karpathy pipeline

First overfit

1. fix random seed (Jeremy Howard doesn't agree)
2. simplify
3. verify loss @ init
4. init well
5. input-indepent baseline
6. verify decreasing training loss
7. visualize just before the net

Andrej Karpathy pipeline

First overfit

1. fix random seed (Jeremy Howard doesn't agree)
2. simplify
3. verify loss @ init
4. init well
5. input-indepent baseline
6. verify decreasing training loss
7. visualize just before the net
8. visualize prediction dynamics

Andrej Karpathy pipeline

First overfit

1. fix random seed (Jeremy Howard doesn't agree)
2. simplify
3. verify loss @ init
4. init well
5. input-indepent baseline
6. verify decreasing training loss
7. visualize just before the net
8. visualize prediction dynamics
9. adam is safe

Andrej Karpathy pipeline

Then regularize

Andrej Karpathy pipeline

Then regularize

1. get more data

Andrej Karpathy pipeline

Then regularize

1. get more data
2. data augment.

Andrej Karpathy pipeline

Then regularize

1. get more data
2. data augment.
3. pretrain

Andrej Karpathy pipeline

Then regularize

1. get more data
2. data augment.
3. pretrain
4. Try smaller model size

Andrej Karpathy pipeline

Then regularize

1. get more data
2. data augment.
3. pretrain
4. Try smaller model size
 1. try a larger model

Andrej Karpathy pipeline

Then regularize

1. get more data
2. data augment.
3. pretrain
4. Try smaller model size
 1. try a larger model
5. Dropout, weight decay

Common bug with training

Common bug with training

- Not moving to `.train()` mode

Common bug with training

- Not moving to `.train()` mode
- Not applying proper data preprocessing

Common bug with training

- Not moving to `.train()` mode
- Not applying proper data preprocessing
- Not shuffling the data (esp. for BatchNorm)

Common bug with training

- Not moving to `.train()` mode
- Not applying proper data preprocessing
- Not shuffling the data (esp. for BatchNorm)
- Wrong learning rate/hyper parameters (lr_find to save)

Common bug with training

- Not moving to `.train()` mode
- Not applying proper data preprocessing
- Not shuffling the data (esp. for BatchNorm)
- Wrong learning rate/hyper parameters (lr_find to save)
- Bad initialization

Move from pure PyTorch to higher-level frameworks

They have tuned training loops and less bugs

- Fastai
- ignite
- pytorch-lightning
- catalyst
- etc