IMAGE SUPER RESOLUTION USING SRCNN

<Report of the STAT 6289 Final project>

Vikki Sui, Chengyi Yang

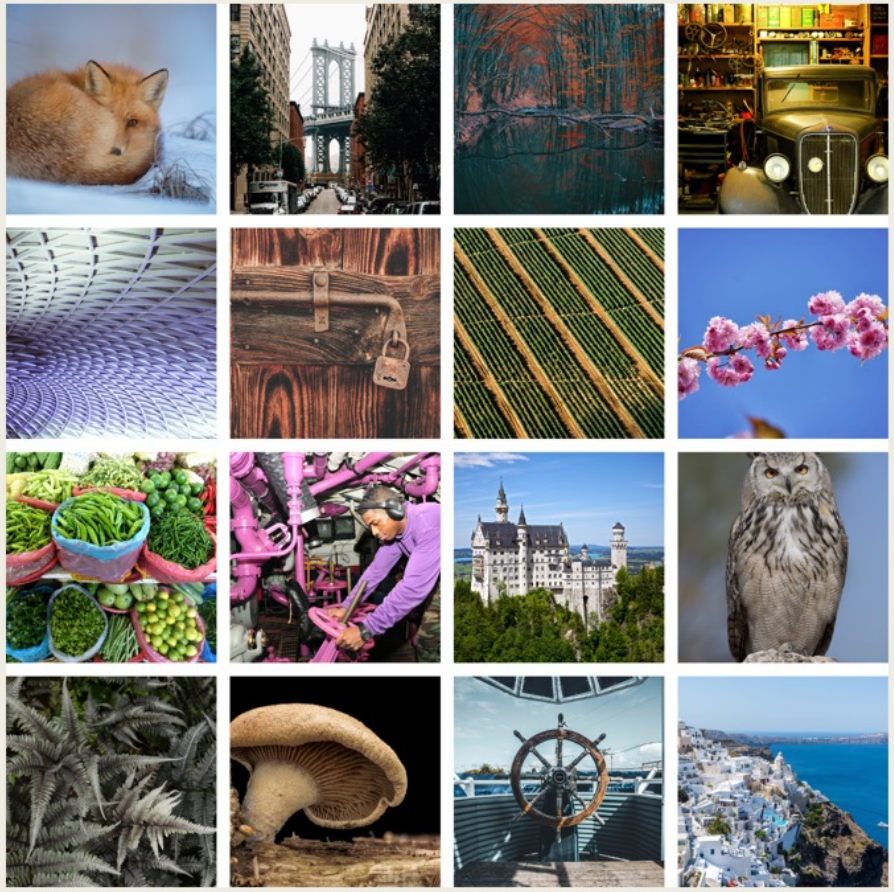
GitHub link: https://github.com/VikkiSui1129/6289HW/tree/main/final_proj

Motivation:

- As the technology improving in recent year, our phone, iPad, laptop can display higher and higher resolution of image, but for some image we may not have the high-resolution version for some reason.
- Sending, uploading, downloading the high-resolution images may take longer time. If there is a way to effectively increase the resolution of images, one can store low-resolution images and convert them to high-resolution when they are used.
- On the Internet, some early pictures and memes are of low resolution, but they are memorable and valuable. If there is a way to effectively increase the resolution of the image, we can convert it to a high-definition version.

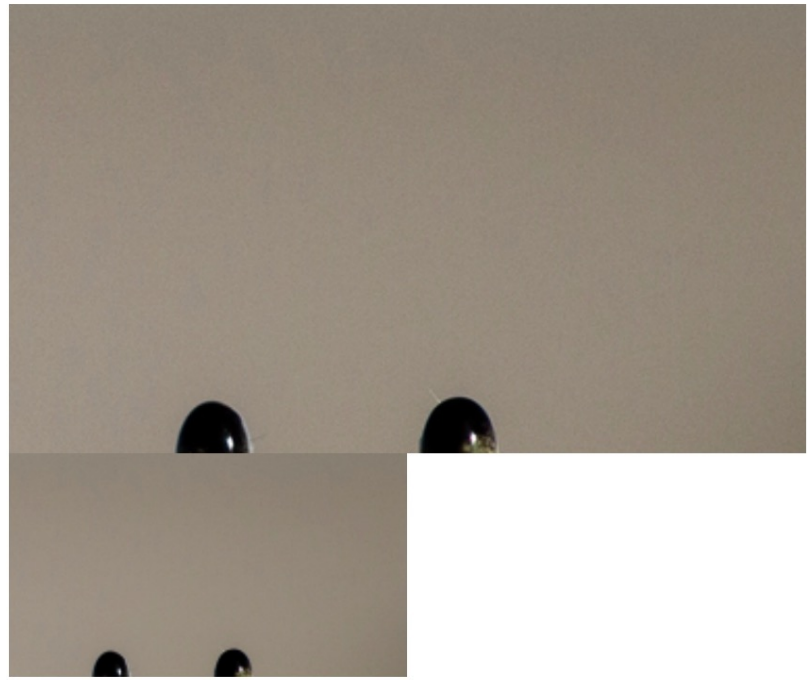
The dataset:

- The dataset is from Kaggle: https://www.kaggle.com/datasets/bansalyash/div2k-valid-hr
- It contains 400 train, 100 validation, but we only used 100 train and 20 validation



Data pre-processing

- Resize picture of various resolution into 1080*1920.
- Read image in “YUV” instead of “RGB”.
- Resize the picture with dimension 1080*1920(High resolution) into dimension 540*960 (Low Resolution).



Data loader

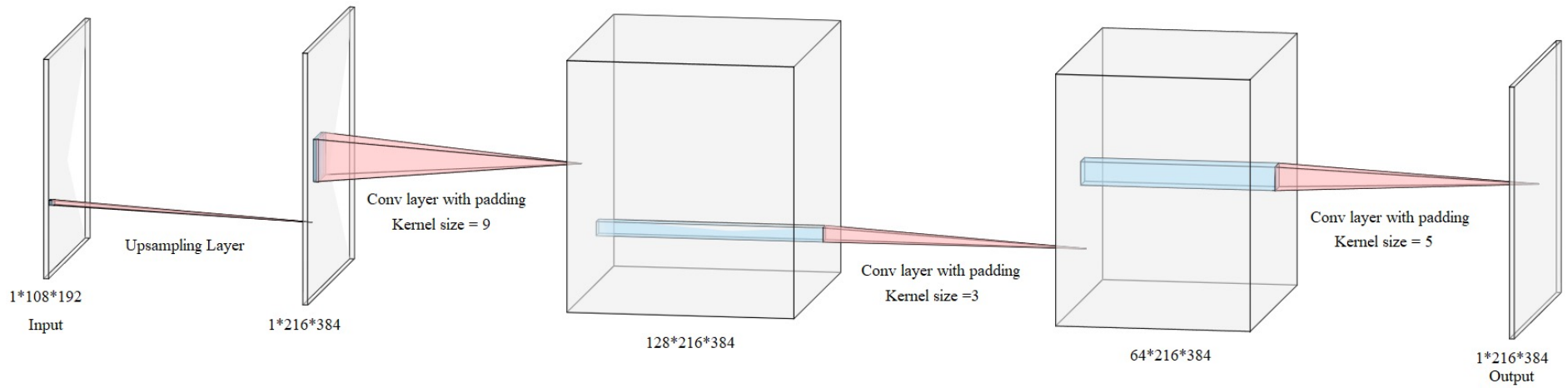
- Reference to the idea in pix2pix, in many cases, models trained by low resolution images perform better. So, we cut a small piece of image with a resolution of 108*192 from a 540*960 image and use it to generate an image with a resolution of 216*384.
- Added the augmentation with 50% probability of doing a horizontal flip and 50% probability of doing vertical flip.
- Because of the 2nd and 3rd channel of YUV is a linear combination of the 1st channel, we only used the 1st channel to train the model.

Model 1: SRCNN

```
self.train_Upsampler = nn.Upsample((tHR_H, tHR_W), mode="bilinear", align_corners=False)
self.valid_Upsampler = nn.Upsample((HR_H, HR_W), mode="bilinear", align_corners=False)
m = []
m.append(ConvBlock(1, 128, 9, "reflect", bias=True, act=nn.ReLU(True)))
m.append(ConvBlock(128, 64, 3, "replicate", bias=True, act=nn.ReLU(True)))
m.append(ConvBlock(64, 1, 5, "reflect", bias=True, act=None))

self.body = nn.Sequential(*m)
```

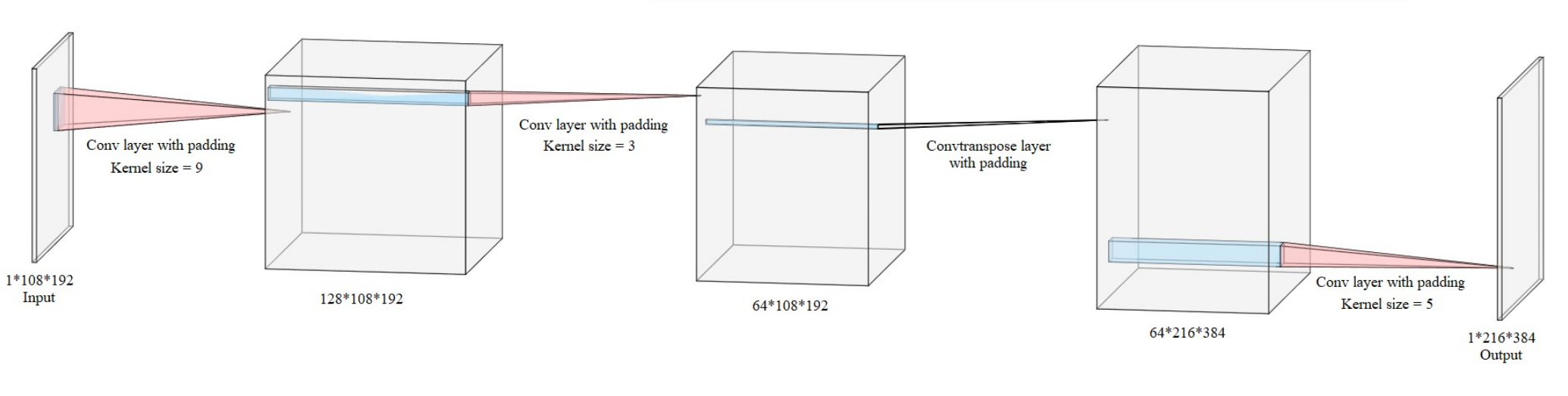
Plot of model structure:



Model 2: SRCNNPlus

```
m = []
m.append(ConvBlock(1, 128, 9, "reflect", bias=True, act=nn.ReLU(True)))
m.append(ConvBlock(128, 64, 3, "replicate", bias=True, act=nn.ReLU(True)))
m.append(ConvTransBlock(64, 64, act=nn.ReLU(True)))
m.append(ConvBlock(64, 1, 5, "reflect", bias=True, act=None))
```

Plot of model structure:

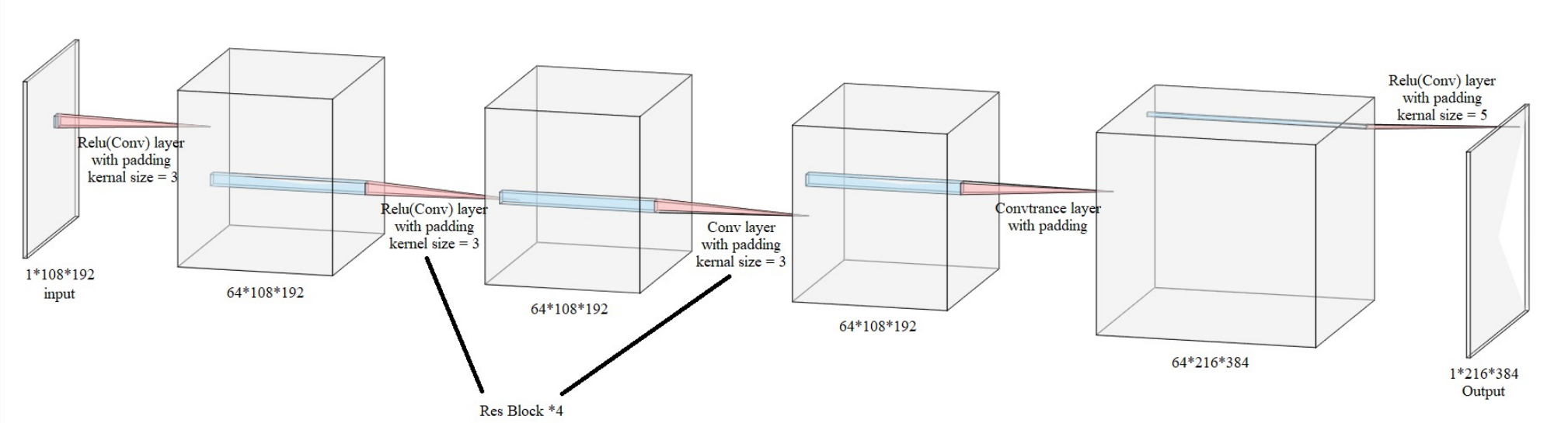


Model 3: ResCNN

```
m = []
m.append(Encoder_in(1, 64, 3, 4, act=nn.ReLU(True)))
m.append(ConvTransBlock(64, 64, act=nn.ReLU(True)))
m.append(ConvBlock(64, 1, 5, "reflect", bias=True, act=None))

self.body = nn.Sequential(*m)
```

Plot of model structure:



Model training

```
def scheduler_step(self):
  self.scheduler.step()
  if self.scheduler._step_count in self.args_optimizer["milestones"]:
    print("Change Learning Rate to {}".format(self.scheduler._last_lr[0]))
```

```
3 args_optimizer["lr"] = 0.001
4 args_optimizer["beta1"] = 0.9
5 args_optimizer["beta2"] = 0.999
6 args_optimizer["milestones"] = [100]
7 args_optimizer["gamma"] = 0.5
```

- Trained each model for 200 epochs, save the images generate by training every 20 epoch.
- The beginning learning rate is 0.001 and will decrease 50% when the epoch reaches 100.

Introduction of PSNR

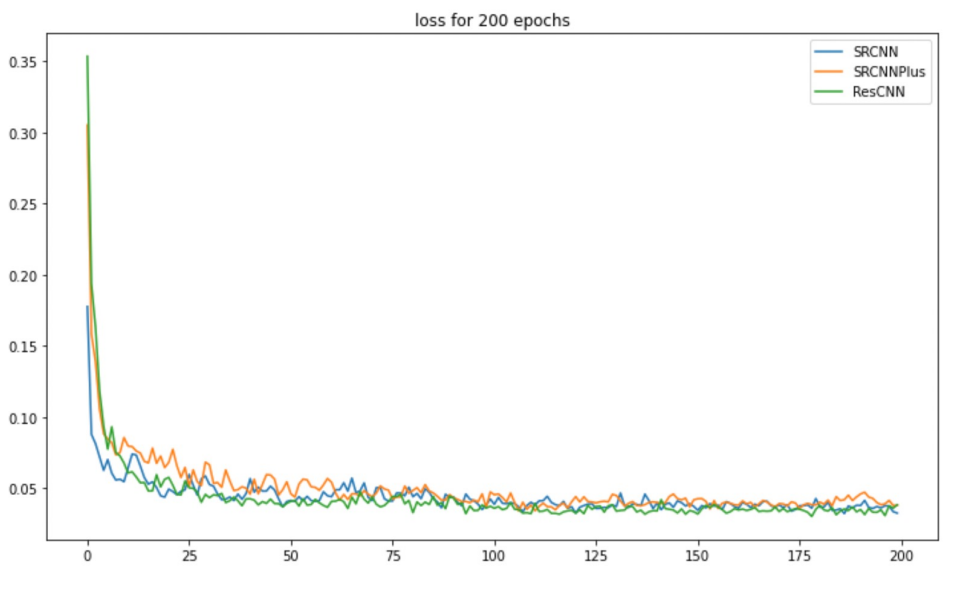
Peak signal-to-noise ratio (PSNR) is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Because many signals have a very wide dynamic range, PSNR is usually expressed as a logarithmic quantity using the decibel scale.

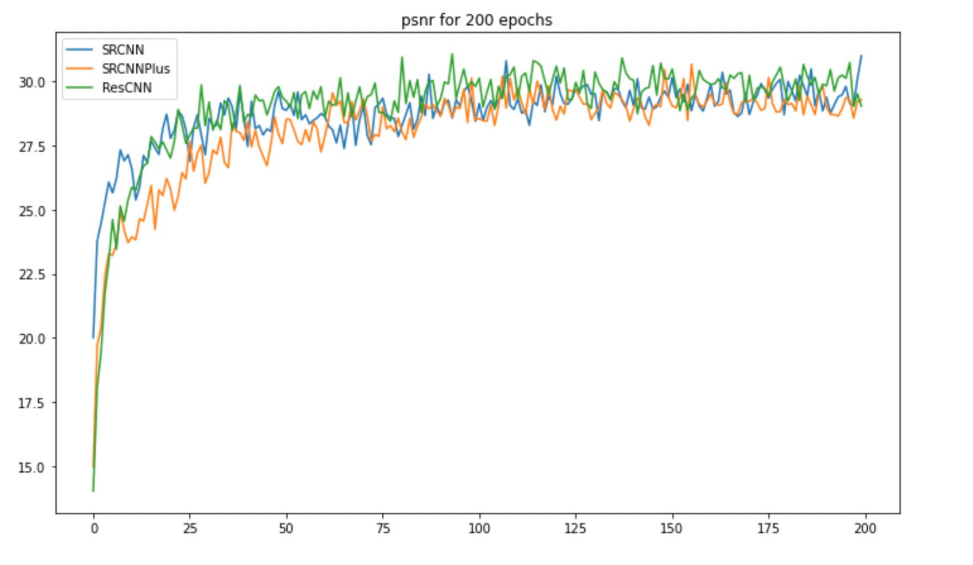
PSNR is commonly used to quantify reconstruction quality for images and video subject to lossy compression.

PSNR

< 10dB
Human can't tell if two pictures are the same.
10dB – 20dB
Human can intuitively judge that there is no big difference between the two images.
20dB – 30dB
Human can detect the difference between the two images.
30dB – 50dB
Human can’t detect the difference between the two images.
>50dB
Two images have only few differences.

Model Result:





Summary

1. All of three model convergence very fast. When the number of epochs is close to 30, all three models are nearly convergent.
2. Among these models, ResCNN performs best. The final psnr can be larger than 30dB. This means that there is no visible difference between the increased resolution image and the original image.

Conclusion:

All three models performed well, with Model 3 being the best.

So, we can think that the ResCNN model can be used to do Image Super Resolution

Image display

SRCNN




SRCNNPlus




ResCNN




Epoch 19 Epoch 79 Epoch 139 Epoch 199

The Novelty of Our Projects

1. Our project can customize the size and magnification of the input image, which is very practical in practical use.
2. Our model is written in pytorch. Compared to TensorFlow in the original text, pytorch is now more widely used.
3. Our model can choose to use RGB or YUV as input color space for training. Only a few parameters need to be changed.

```
args_data_norm = {}
args_data_norm["color_field"] = "YUV"
args_data_norm["color_mean"] = [0.5, 0.5, 0.5]
args_data_norm["color_std"] = [0.5, 0.5, 0.5]
```

```
if self.color_field == "RGB":
  HR_train = HR
  LR_train = LR
elif self.color_field == "YUV":
  HR_train = HR[:,:1]
  LR_train = LR[:,:1]
else:
  print("Color Field Error")
```

Imperfections

1. The final PSNR is around 30dB. Although this is a good result, it would be even better if it could be raised to around 50dB.
2. The training set of the model has only 100 images, which is too few. If you increase the number of images in the training set, the results may be better.
3. The number of layers in these models are few. If these models have more layers, maybe the results will be better.

What we learned from projects.

1. Although the CNN model was developed earlier among various deep learning models, it is also a relatively simple one. But the situations of its applications are still worth exploring.
2. The CNN model is not only able to handle the situation of image to label, but it can also handle image to image as well.
3. Sometimes a simple model can get a quite good result. But if you want to improve further on this result, it may take a lot of time to readjust.
4. When working on a project, it’s not just special needs to be considered. Sometimes a little more work can make a project suitable for a variety of situations.

Acknowledgement

We are greatly appreciating Prof. Fang Jin. Through her course, we learned a lot about deep learning.

At the same time, we would also like to thank the TA (Zhou Yang) of this class, he is a very competent and kind TA, who answered a lot of questions to the students during the Office Hour.

References

1. http://mmlab.ie.cuhk.edu.hk/projects/SRCNN.html
2. https://github.com/kunal-visoulia/Image-Restoration-using-SRCNN#:~:text=The%20SRCNN%20is%20a%20deep,quality%20of%20low%20resolution%20images