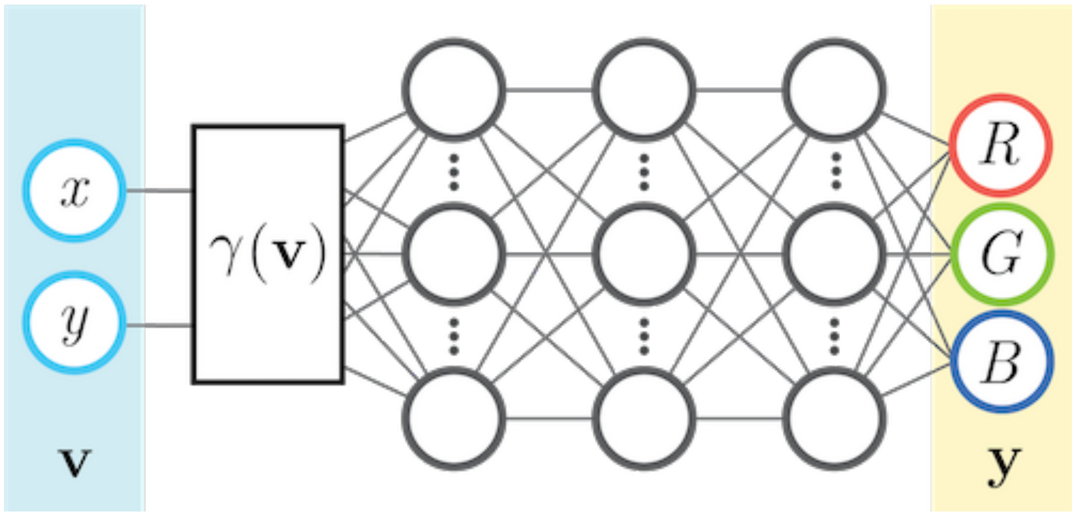
Assignment 2

In this assignment you will create a coordinate-based multilayer perceptron in numpy from scratch. For each input image coordinate (x, y), the model predicts the associated color (r, g, b).



You will then compare the following input feature mappings $\gamma(\mathbf{v})$.

- No mapping: $\gamma(\mathbf{v}) = \mathbf{v}$.
- Basic mapping: $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{v}), \sin(2\pi\mathbf{v})]^{\mathrm{T}}$.
- Gaussian Fourier feature mapping: $\gamma(\mathbf{v}) = [\cos(2\pi\mathbf{B}\mathbf{v}), \sin(2\pi\mathbf{B}\mathbf{v})]^{\mathrm{T}}$, where each entry in $\mathbf{B} \in \mathbb{R}^{m \times d}$ is sampled from $\mathcal{N}(0, \sigma^2)$.

Some notes to help you with that:

- You will implement the mappings in the helper functions `get_B_dict` and `input_mapping`.
- The basic mapping can be considered a case where $\mathbf{B} \in \mathbb{R}^{2 \times 2}$ is the indentity matrix.
- For this assignment, d is 2 because the input coordinates in two dimensions.
- You can experiment with m and σ values e.g. $m = 256$ and $\sigma \in \{1, 10, 100\}$.

Source: https://bmild.github.io/fourfeat/ This assignment is inspired by and built off of the authors' demo.

Setup

(Optional) Colab Setup

If you aren't using Colab, you can delete the following code cell. Replace the path below with the path in your Google Drive to the uploaded assignment folder. Mounting to Google Drive will allow you access the other .py files in the assignment folder and save outputs to this folder

```
# you will be prompted with a window asking to grant permissions
# click connect to google drive, choose your account, and click allow
from google.colab import drive
drive.mount("/content/drive")
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
# TODO: fill in the path in your Google Drive in the string below
# Note: do not escape slashes or spaces in the path string
import os
datadir = "/content/assignment2"
if not os.path.exists(datadir):
  !ln -s "/content/drive/My Drive/CS 444/assignment2/" $datadir
os.chdir(datadir)
!pwd
```

```
    /content/drive/My Drive/CS 444/assignment2
```

Imports

```
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
import os, imageio
import cv2
import numpy as np

# imports /content/assignment2/models/neural_net.py if you mounted correctly
from models.neural_net import NeuralNetwork

# makes sure your NeuralNetwork updates as you make changes to the .py file
%load_ext autoreload
%autoreload 2

# sets default size of plots
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0)
```

```
    The autoreload extension is already loaded. To reload it, use:
      %reload_ext autoreload
```

Helper Functions

Experiment Runner (Fill in TODOs)

```
def NN_experiment(X_train, y_train, X_test, y_test, input_size, num_layers,\
                  hidden_size, hidden_sizes, output_size, epochs,\
                  learning_rate, opt):

  # Initialize a new neural network model
  net = NeuralNetwork(input_size, hidden_sizes, output_size, num_layers, opt)

  # Variables to store performance for each epoch
  train_loss = np.zeros(epochs)
  train_psnr = np.zeros(epochs)
  test_psnr = np.zeros(epochs)
  predicted_images = np.zeros((epochs, y_test.shape[0], y_test.shape[1]))
  print("Predicted_images dim before storing:", predicted_images.shape)

  b1= 0.9
  b2 = 0.999
```

```
    eps = 1e-8

    idx = np.arange(X_train.shape[0])

    # For each epoch...
    for epoch in tqdm(range(epochs)):

        # Shuffle the dataset
        # TODO implement this
        #Using np.random.permutation because is useful when you need to shuffle order

        # perm = np.random.permutation(X_train.shape)
        # X, y = X_train[perm], y_train[perm]


        np.random.shuffle(idx)
        X = X_train[idx]
        y = y_train[idx]
        # Training
        # Run the forward pass of the model to get a prediction and record the psnr
        # TODO implement this
        pred = net.forward(X)
        train_psnr[epoch] = psnr(y,pred)

        # Run the backward pass of the model to compute the loss, record the loss, ar
        # TODO implement this
        train_loss[epoch] = net.backward(y)
        net.update(learning_rate,b1,b2, eps, opt)
        # Testing
        # No need to run the backward pass here, just run the forward pass to compute
        # TODO implement this
        predicted_images[epoch] = net.forward(X_test)
        test_psnr[epoch] = psnr(y_test,predicted_images[epoch])

    return net, train_psnr, test_psnr, train_loss, predicted_images
```

Image Data and Feature Mappings (Fill in TODOs)

```
# Data loader - already done for you
def get_image(size=512, \
              image_url='https://bmild.github.io/fourfeat/img/lion_orig.png'):

  # Download image, take a square crop from the center
  img = imageio.imread(image_url)[..., :3] / 255.
  c = [img.shape[0]//2, img.shape[1]//2]
  r = 256
  img = img[c[0]-r:c[0]+r, c[1]-r:c[1]+r]

  if size != 512:
    img = cv2.resize(img, (size, size))

  plt.imshow(img)
  plt.show()

  # Create input pixel coordinates in the unit square
  coords = np.linspace(0, 1, img.shape[0], endpoint=False)
  x_test = np.stack(np.meshgrid(coords, coords), -1)
  test_data = [x_test, img]
  train_data = [x_test[::2, ::2], img[::2, ::2]]

  return train_data, test_data


# Create the mappings dictionary of matrix B -  you will implement this
import jax.numpy as jnp
from jax import jit, grad, random
rand_key = random.PRNGKey(0)

def get_B_dict():
  B_dict = {}
  B_dict['none'] = None

  # add B matrix for basic, gauss_1.0, gauss_10.0, gauss_100.0
  # TODO implement this
  B_dict['basic'] = jnp.eye(2)

  B_gauss = np.random.normal(0,1, (256, 2))
  for scale in [1., 10., 100.]:
    B_dict[f'gauss_{scale}'] = B_gauss * scale

  return B_dict
```

```
# Given tensor x of input coordinates, map it using B - you will implement
def input_mapping(x, B):
  if B is None:
    # "none" mapping - just returns the original input coordinates
    return x
  else:
    # "basic" mapping and "gauss_X" mappings project input features using B
    # TODO implement this
    x_proj = (2*np.pi*x) @ B.T
    return np.concatenate([np.cos(x_proj), np.sin(x_proj)], axis=-1)


# Apply the input feature mapping to the train and test data - already done for you
def get_input_features(B_dict, mapping):
  # mapping is the key to the B_dict, which has the value of B
  # B is then used with the function `input_mapping` to map x
  y_train = train_data[1].reshape(-1, output_size)
  y_test = test_data[1].reshape(-1, output_size)
  X_train = input_mapping(train_data[0].reshape(-1, 2), B_dict[mapping])
  X_test = input_mapping(test_data[0].reshape(-1, 2), B_dict[mapping])
  return X_train, y_train, X_test, y_test
```

MSE Loss and PSNR Error (Fill in TODOs)

```
def mse(y, p):
  # TODO implement this
  # make sure it is consistent with your implementation in neural_net.py
  diff_array = np.subtract(y,p)
  squared_array = np.square(diff_array)
  mse_value  = squared_array.mean()
  return mse_value

def psnr(y, p):
  # TODO implement this
  mse_val = mse(y,p)
  max_pixel = 255
  psnr_value = 10 * np.log10(max_pixel / mse_val)
  return psnr_value
```

Plotting

```
def plot_training_curves(train_loss, train_psnr, test_psnr):
  # plot the training loss
  plt.subplot(2, 1, 1)
  plt.plot(train_loss)
  plt.title('MSE history')
  plt.xlabel('Iteration')
  plt.ylabel('MSE Loss')

  # plot the training and testing psnr
  plt.subplot(2, 1, 2)
  plt.plot(train_psnr, label='train')
  plt.plot(test_psnr, label='test')
  plt.title('PSNR history')
  plt.xlabel('Iteration')
  plt.ylabel('PSNR')
  plt.legend()

  plt.tight_layout()
  plt.show()


def plot_reconstruction(p, y_test):
  p_im = p.reshape(size,size,3)
  y_im = y_test.reshape(size,size,3) #Changded from y to y_test

  plt.figure(figsize=(12,6))

  # plot the reconstruction of the image
  plt.subplot(1,2,1), plt.imshow(p_im), plt.title("reconstruction")

  # plot the ground truth image
  plt.subplot(1,2,2), plt.imshow(y_im), plt.title("ground truth")

  print("Final Test MSE", mse(y_test, p)) #Changded from y to y_test
  print("Final Test psnr",psnr(y_test, p)) #Changded from y to y_test
```

```
def plot_reconstruction_progress(predicted_images, y_test, N=8):
  total = len(predicted_images)
  step = total // N
  plt.figure(figsize=(24, 4))

  # plot the progress of reconstructions
  for i, j in enumerate(range(0,total, step)):
      plt.subplot(1, N, i+1)
      plt.imshow(predicted_images[j].reshape(size,size,3))
      plt.axis("off")
      plt.title(f"iter {j}")

  # plot ground truth image
  plt.subplot(1, N+1, N+1)
  plt.imshow(y_test.reshape(size,size,3))
  plt.title('GT')
  plt.axis("off")
  plt.show()
```

```
def plot_feature_mapping_comparison(outputs, gt):
  # plot reconstruction images for each mapping
  plt.figure(figsize=(24, 4))
  N = len(outputs)
  for i, k in enumerate(outputs):
      plt.subplot(1, N+1, i+1)
      plt.imshow(outputs[k]['pred_imgs'][-1].reshape(size, size, -1))
      plt.title(k)
  plt.subplot(1, N+1, N+1)
  plt.imshow(gt)
  plt.title('GT')
  plt.show()

  # plot train/test error curves for each mapping
  iters = len(outputs[k]['train_psnrs'])
  plt.figure(figsize=(16, 6))
  plt.subplot(121)
  for i, k in enumerate(outputs):
      plt.plot(range(iters), outputs[k]['train_psnrs'], label=k)
  plt.title('Train error')
  plt.ylabel('PSNR')
  plt.xlabel('Training iter')
  plt.legend()
  plt.subplot(122)
  for i, k in enumerate(outputs):
      plt.plot(range(iters), outputs[k]['test_psnrs'], label=k)
  plt.title('Test error')
  plt.ylabel('PSNR')
  plt.xlabel('Training iter')
  plt.legend()
  plt.show()
```

```
# Save out video
def create_and_visualize_video(outputs, size=size, epochs=epochs, filename='trainir
  all_preds = np.concatenate([outputs[n]['pred_imgs'].reshape(epochs,size,size,3)[:
  data8 = (255*np.clip(all_preds, 0, 1)).astype(np.uint8)
  f = os.path.join(filename)
  imageio.mimwrite(f, data8, fps=20)

  # Display video inline
  from IPython.display import HTML
  from base64 import b64encode
  mp4 = open(f, 'rb').read()
  data_url = "data:video/mp4;base64," + b64encode(mp4).decode()

  N = len(outputs)
  if N == 1:
    return HTML(f'''
    <video width=256 controls autoplay loop>
          <source src="{data_url}" type="video/mp4">
    </video>
    ''')
  else:
    return HTML(f'''
    <video width=1000 controls autoplay loop>
          <source src="{data_url}" type="video/mp4">
    </video>
    <table width="1000" cellspacing="0" cellpadding="0">
      <tr>{''.join(N*[f'<td width="{1000//len(outputs)}"></td>'])}</tr>
      <tr>{''.join(N*['<td style="text-align:center">{}</td>'])}</tr>
    </table>
    '''.format(*list(outputs.keys())))
```

Low Resolution Reconstruction

Some suggested hyperparameter choices to help you start

- hidden layer count: 4
- hidden layer size: 256
- number of epochs: 1000
- learning reate: 1e-4

```
size = 32
train_data, test_data = get_image(size)
```

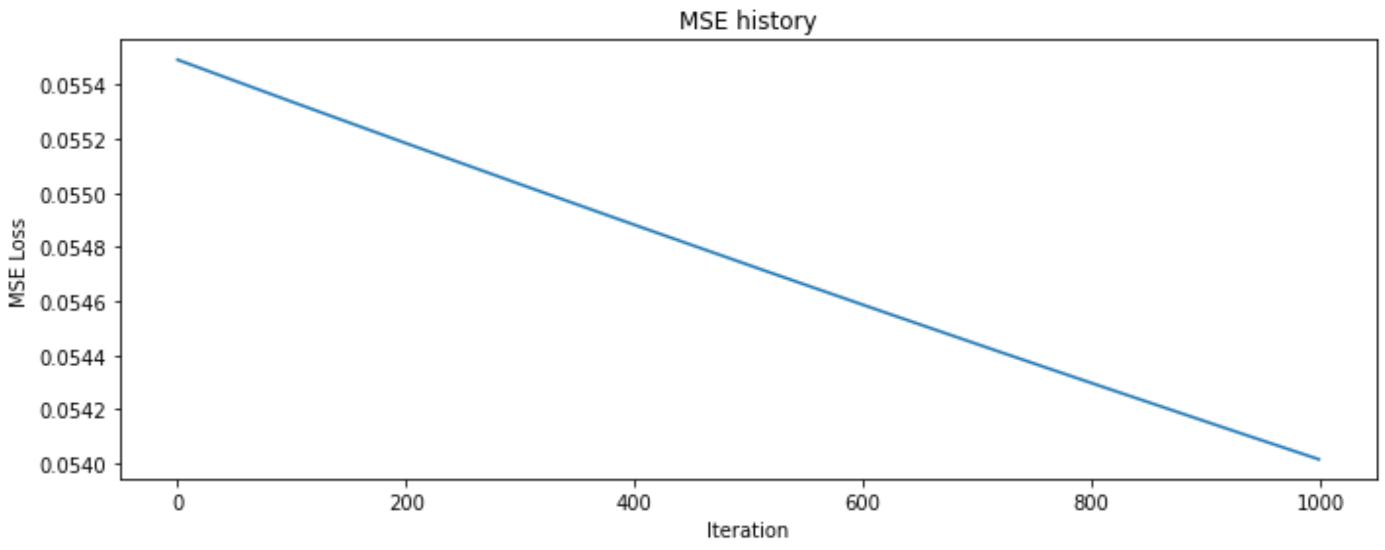
```
# hyper parameters
num_layers = 5
hidden_size = 256
hidden_sizes = [hidden_size] * (num_layers - 1)
learning_rate = 1e-4
output_size = 3
epochs = 1000
```

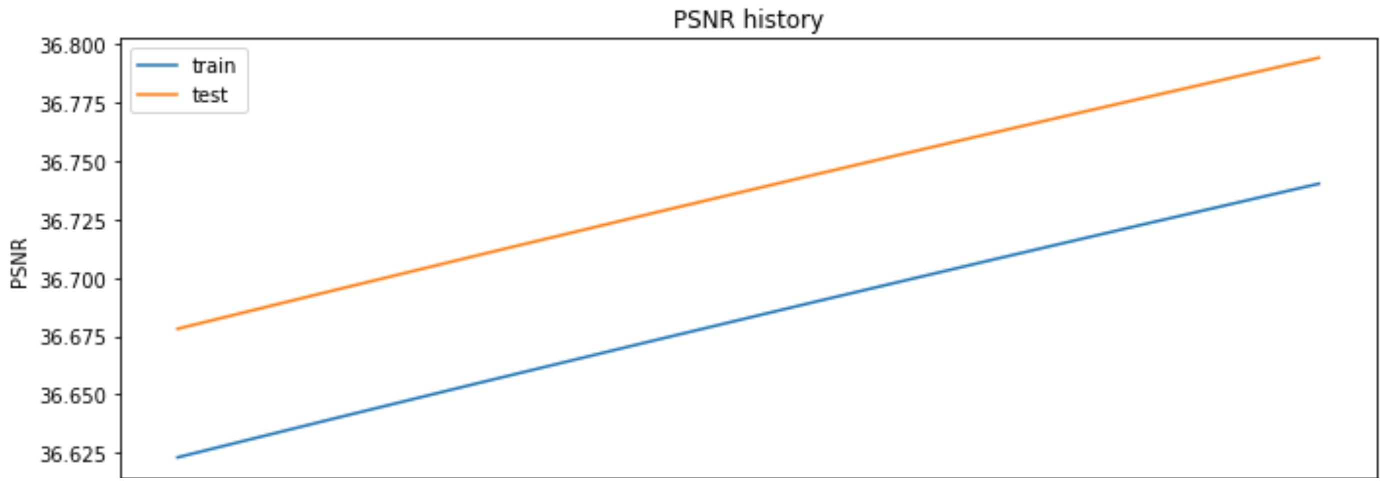
Low Resolution Reconstruction - SGD - None Mapping

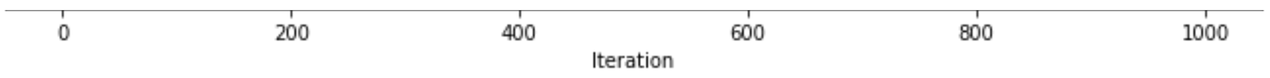
```
# get input features
opt = 'SGD'
```

```
# TODO implement this by using the get_B_dict() and get_input_features() helper fur
mapping = 'none'
B_dict = get_B_dict()
X_train, y_train, X_test, y_test = get_input_features(B_dict, mapping)
input_size = X_train.shape[1]
# run NN experiment on input features
# TODO implement by using the NN_experiment() helper function

net,train_psnr, test_psnr, train_loss, predicted_images = NN_experiment(X_train, y_
                                                                        hidden_size
                                                                        opt)


# plot results of experiment
plot_training_curves(train_loss, train_psnr, test_psnr)
plot_reconstruction(net.forward(X_test), y_test)
plot_reconstruction_progress(predicted_images, y_test)
```

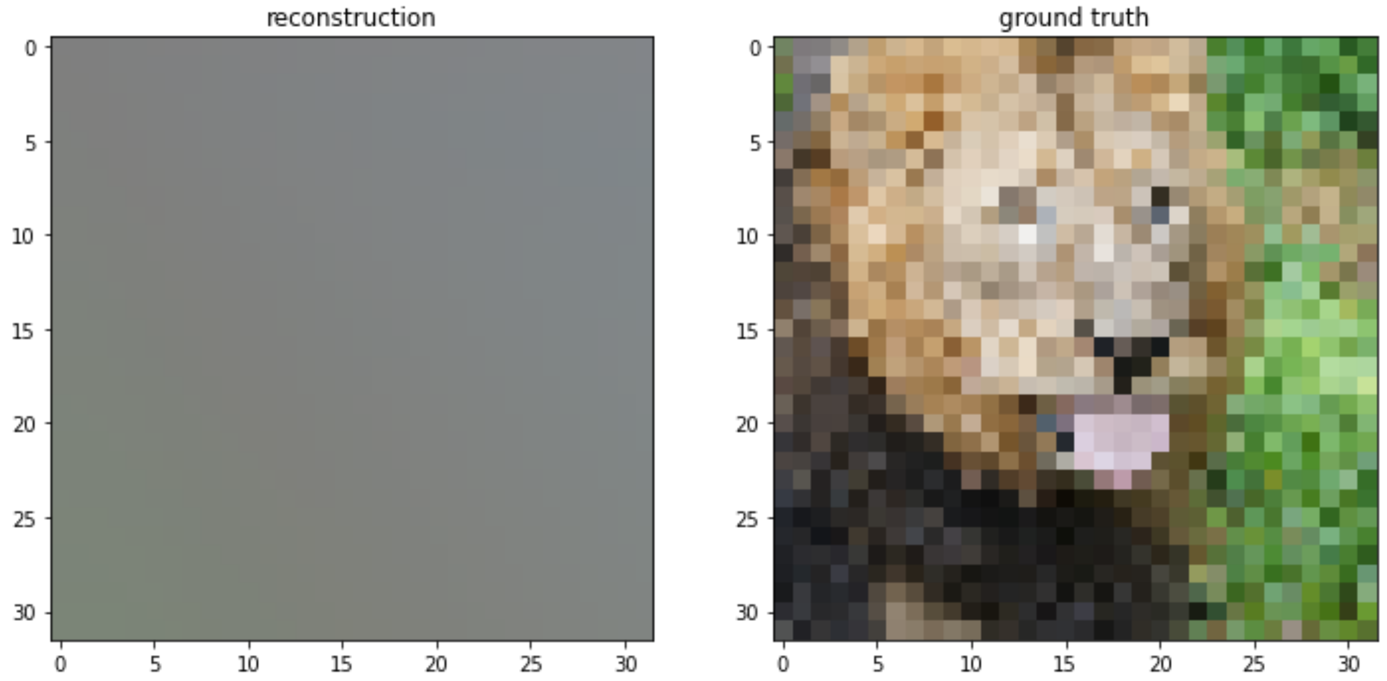
```
Predicted_images dim before storing: (1000, 1024, 3)
100%                                        1000/1000 [00:53<00:00, 22.85it/s]
```

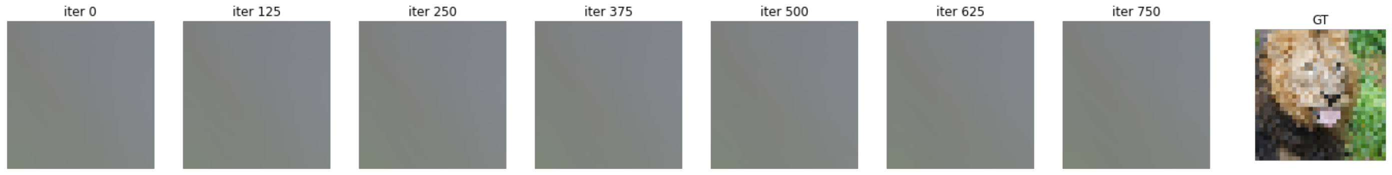






```
Final Test MSE 0.05334753233727459
Final Test psnr 36.79425845091287
```



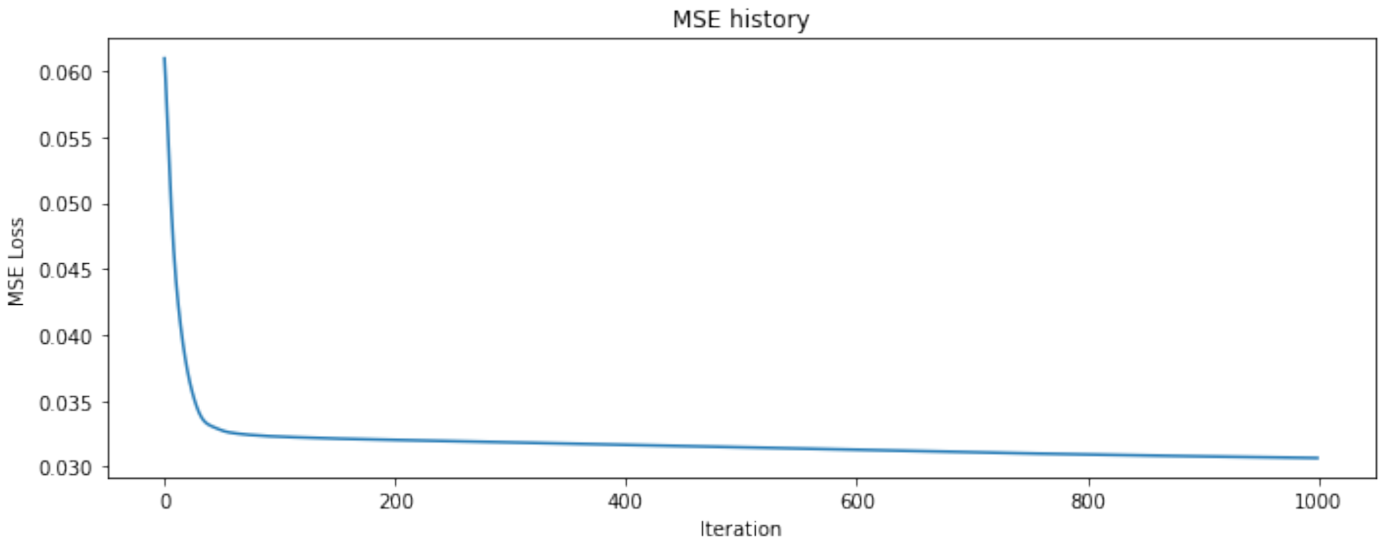


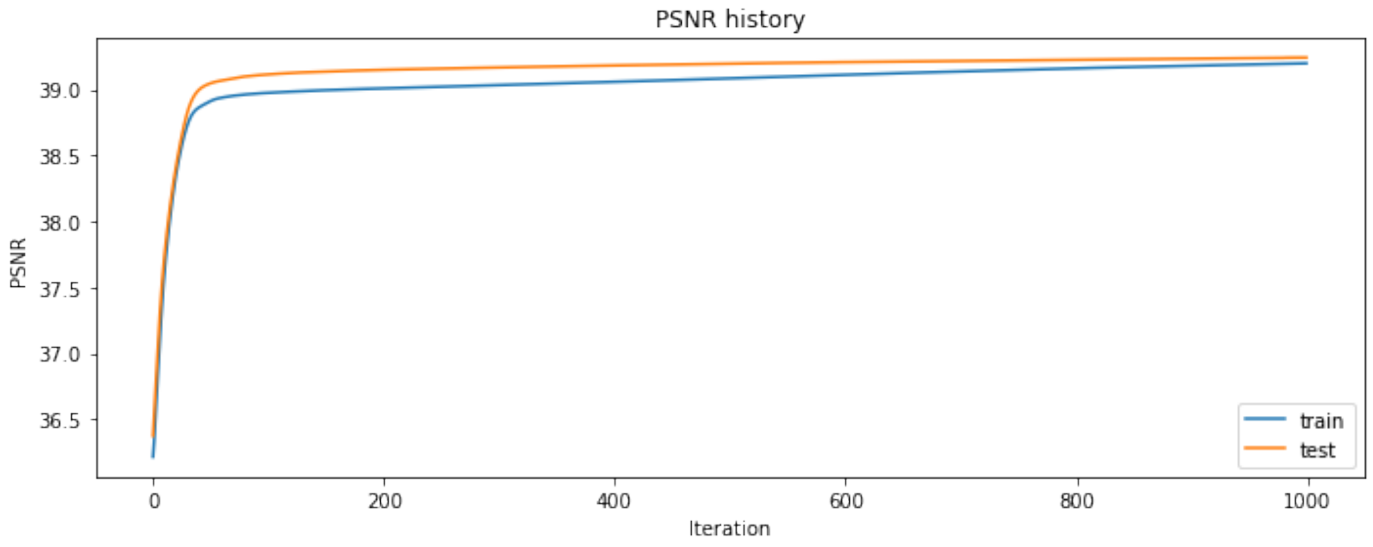
Low Resolution Reconstruction - Adam - None Mapping

```
# get input features
opt = 'ADAM'
# TODO implement this by using the get_B_dict() and get_input_features() helper fur
mapping = 'none'
B_dict = get_B_dict()
X_train, y_train, X_test, y_test = get_input_features(B_dict, mapping)
```

```
input_size = X_train.shape[1]
# run NN experiment on input features
# TODO implement by using the NN_experiment() helper function
net,train_psnr, test_psnr, train_loss, predicted_images = NN_experiment(X_train, y_
                                                                          hidden_size

# plot results of experiment
plot_training_curves(train_loss, train_psnr, test_psnr)
plot_reconstruction(net.forward(X_test), y_test)
plot_reconstruction_progress(predicted_images, y_test)
```

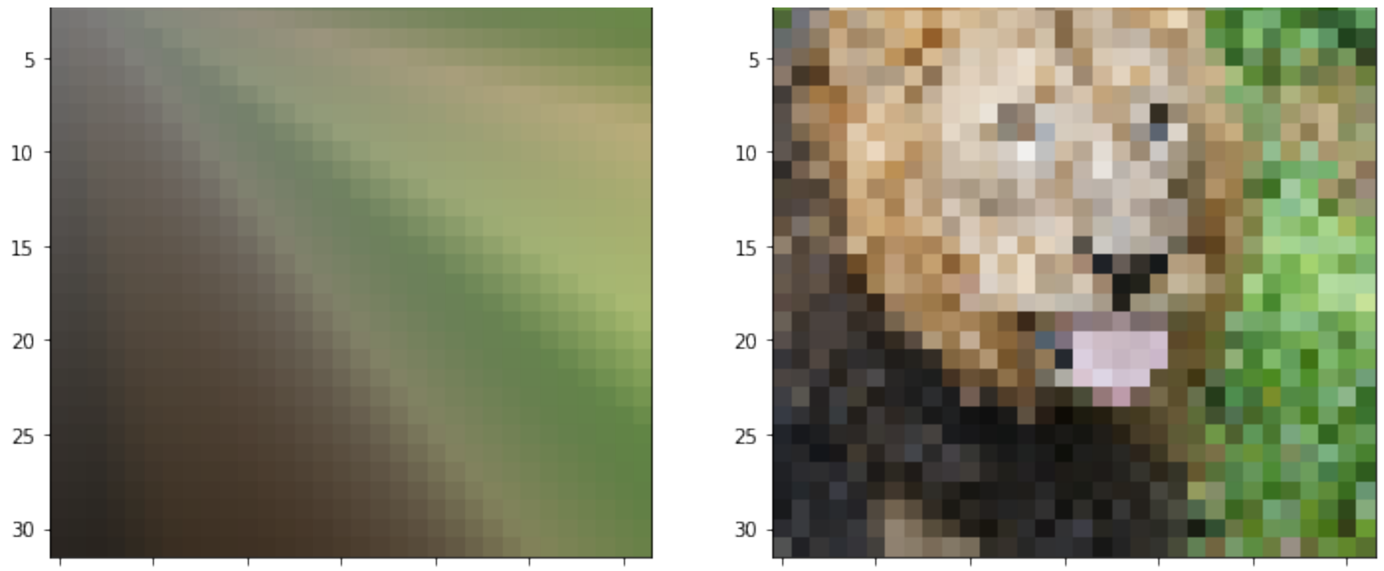
```
Predicted_images dim before storing: (1000, 1024, 3)
100%                                         1000/1000 [01:06<00:00, 19.59it/s]
```

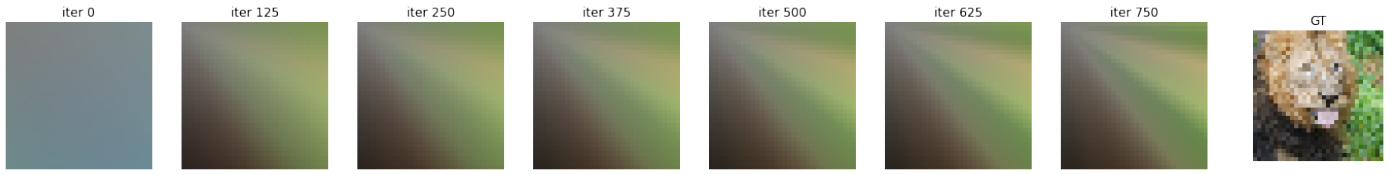




```
Final Test MSE 0.030359158575303496
Final Test psnr 39.24250449810203
```

reconstruction
ground truth
0
0





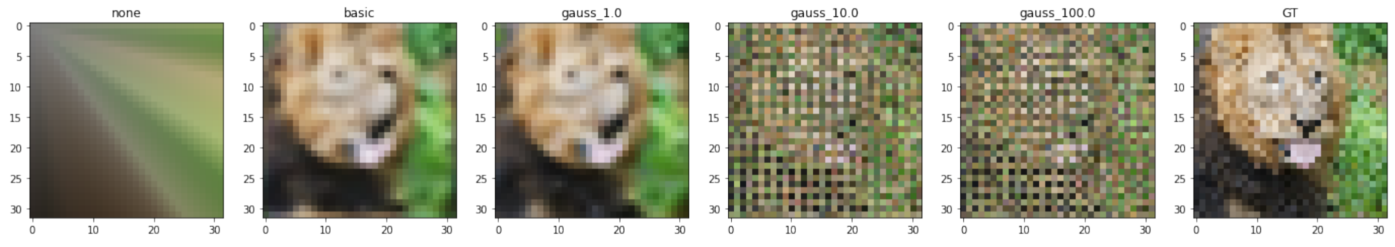
Low Resolution Reconstruction - Optimizer of your Choice - Various Input Mapping Stategies

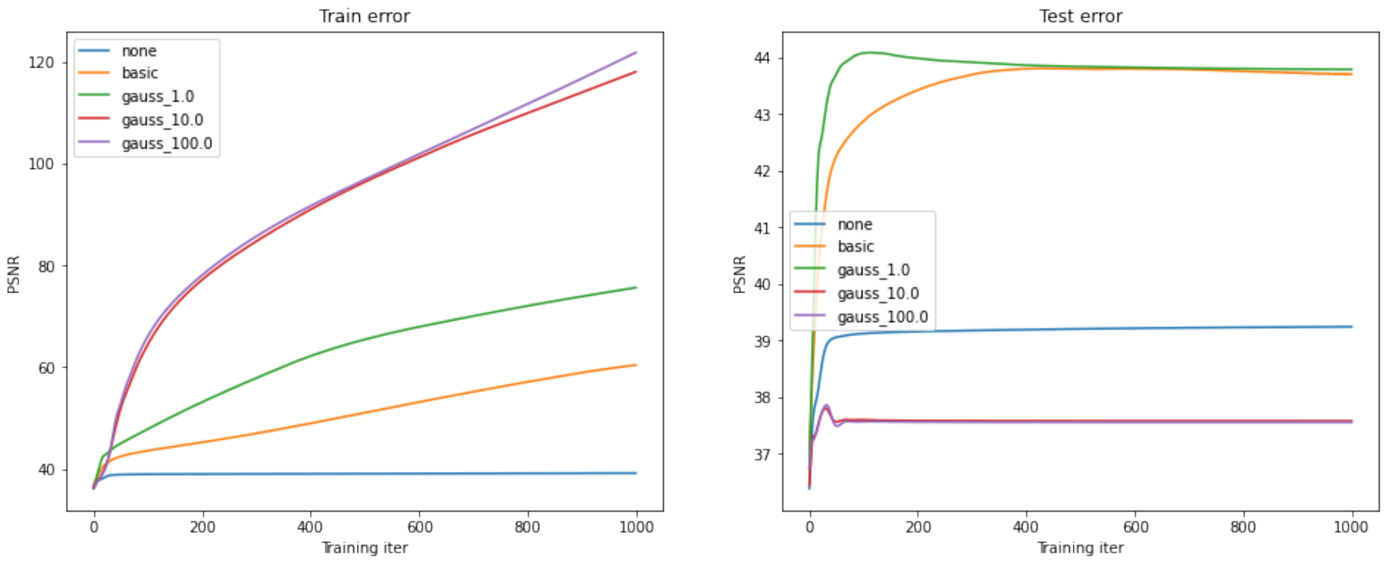
```
def train_wrapper(mapping, size, opt):
  # TODO implement
  # makes it easy to run all your mapping experiments in a for loop
  # this will similar to what you did previously in the last two sections
  B_dict = get_B_dict()
  X_train, y_train, X_test, y_test = get_input_features(B_dict, mapping)
  input_size = X_train.shape[1]
  # run NN experiment on input features
  # TODO implement by using the NN_experiment() helper function
  net,train_psnrs, test_psnrs, train_loss, predicted_images = NN_experiment(X_trair
                                                                            hidden_size

  return {
      'net': net,
      'train_psnrs': train_psnrs,
      'test_psnrs': test_psnrs,
      'train_loss': train_loss,
      'pred_imgs': predicted_images
  }
```

```
outputs = {}
for k in tqdm(B_dict):
  print("training", k)
  outputs[k] = train_wrapper(k, size, opt)
```

```
100%                                         5/5 [06:53<00:00, 89.01s/it]
training none
Predicted_images dim before storing: (1000, 1024, 3)
100%                                         1000/1000 [00:59<00:00, 19.75it/s]
training basic
Predicted_images dim before storing: (1000, 1024, 3)
100%                                         1000/1000 [01:06<00:00, 10.15it/s]
training gauss_1.0
Predicted_images dim before storing: (1000, 1024, 3)
100%                                         1000/1000 [01:35<00:00, 12.67it/s]
training gauss_10.0
Predicted_images dim before storing: (1000, 1024, 3)
100%                                         1000/1000 [01:34<00:00, 12.85it/s]
training gauss_100.0
Predicted_images dim before storing: (1000, 1024, 3)
100%                                         1000/1000 [01:36<00:00, 12.71it/s]
```

```
# if you did everything correctly so far, this should output a nice figure you can
plot_feature_mapping_comparison(outputs, y_test.reshape(size,size,3))
```





High Resolution Reconstruction

High Resolution Reconstruction - Optimizer of your Choice - Various Input Mapping Stategies

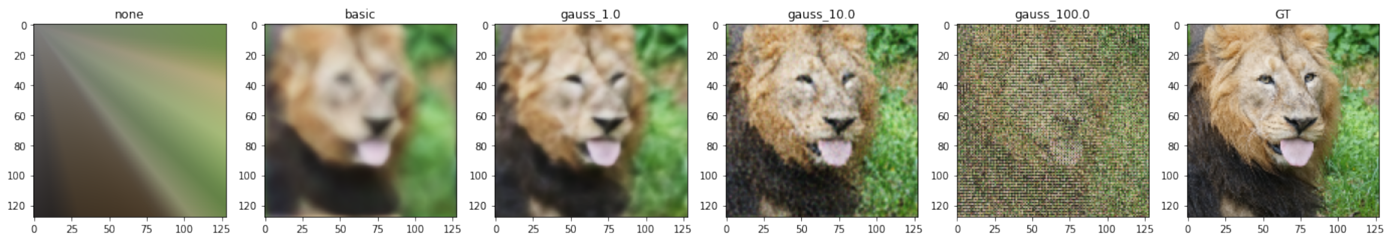
Repeat the previous experiment, but at the higher resolution. The reason why we have you first experiment with the lower resolution since it is faster to train and debug. Additionally, you will see how the mapping strategies perform better or worse at the two different input resolutions.

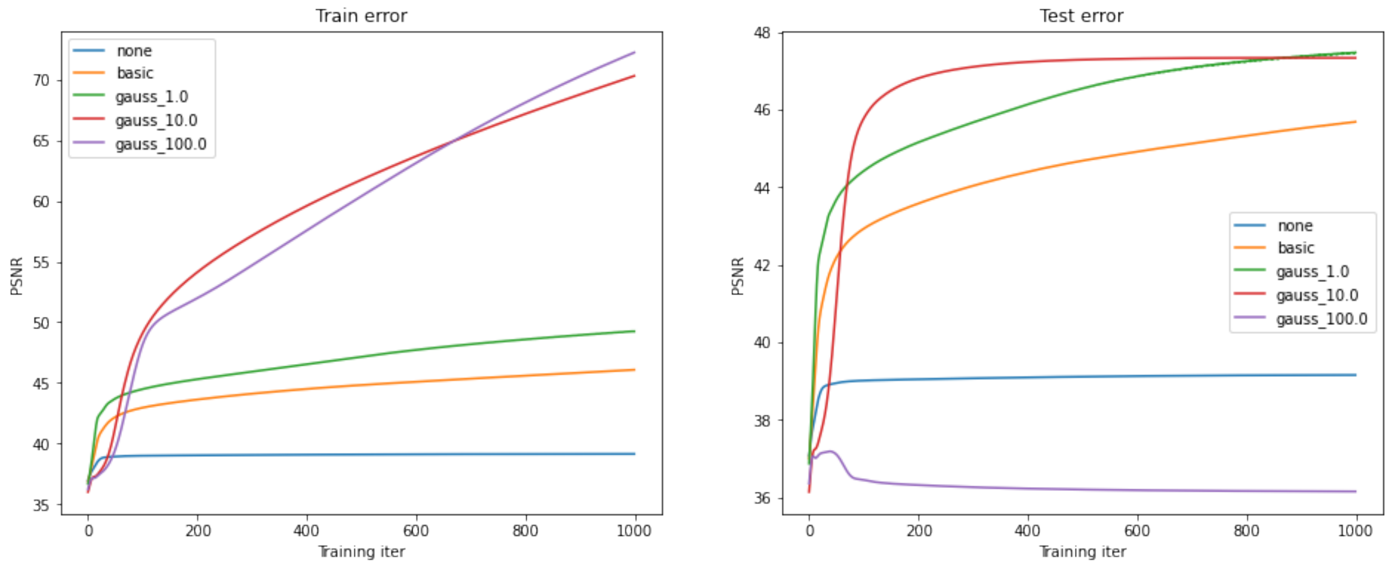
```
size = 128
train_data, test_data = get_image(size)
```

```
outputs = {}
for k in tqdm(B_dict):
  print("training", k)
  outputs[k] = train_wrapper(k, size,opt)
```

```
100%                                    5/5 [1:29:33<00:00, 1152.05s/it]
training none
Predicted_images dim before storing: (1000, 16384, 3)
100%                                    1000/1000 [12:38<00:00, 1.51it/s]
training basic
Predicted_images dim before storing: (1000, 16384, 3)
100%                                    1000/1000 [16:09<00:00, 1.06s/it]
training gauss_1.0
Predicted_images dim before storing: (1000, 16384, 3)
100%                                    1000/1000 [19:30<00:00, 1.02s/it]
training gauss_10.0
Predicted_images dim before storing: (1000, 16384, 3)
100%                                    1000/1000 [20:27<00:00, 1.05s/it]
training gauss_100.0
Predicted_images dim before storing: (1000, 16384, 3)
100%                                    1000/1000 [20:46<00:00, 1.47s/it]
```

```
#plots
X_train, y_train, X_test, y_test = get_input_features(B_dict, mapping)
plot_feature_mapping_comparison(outputs, y_test.reshape(size,size,3))
```

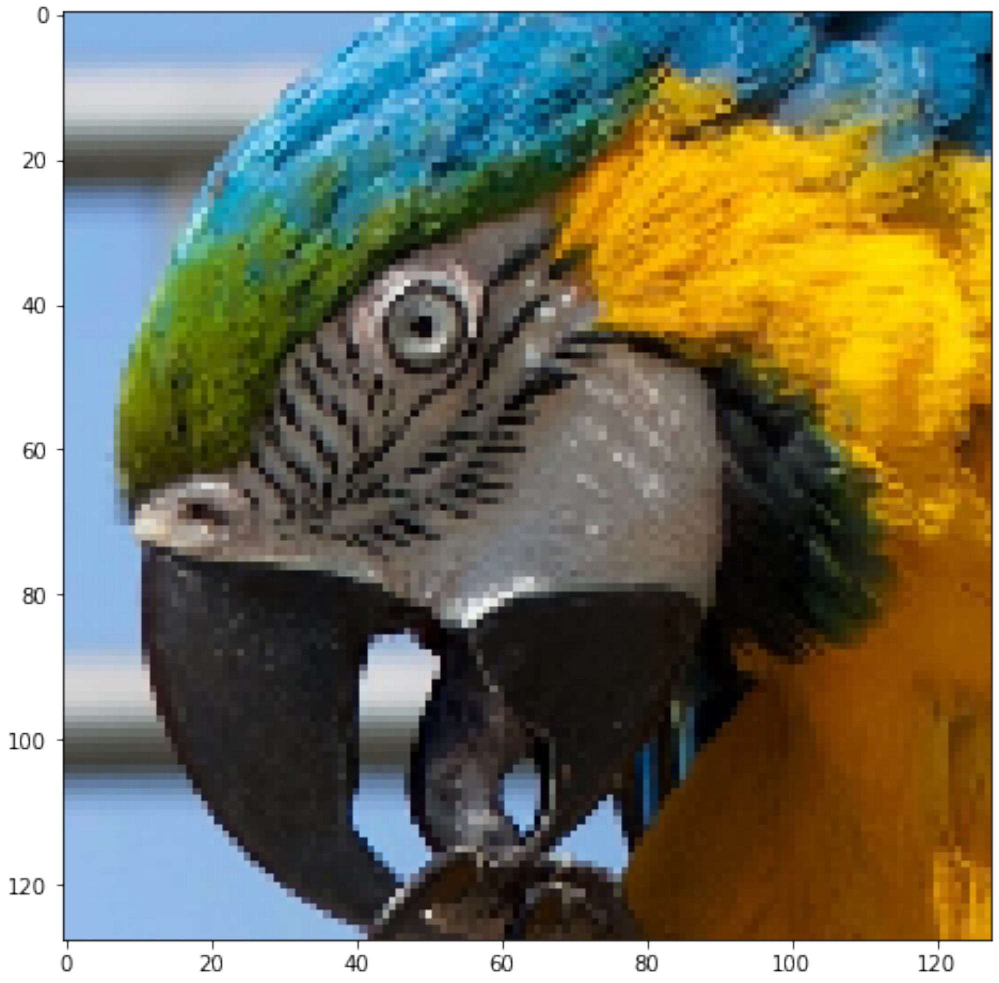




High Resolution Reconstruction - Image of your Choice

When choosing an image select one that you think will give you interesting results or a better insight into the performance of different feature mappings and explain why in your report template.

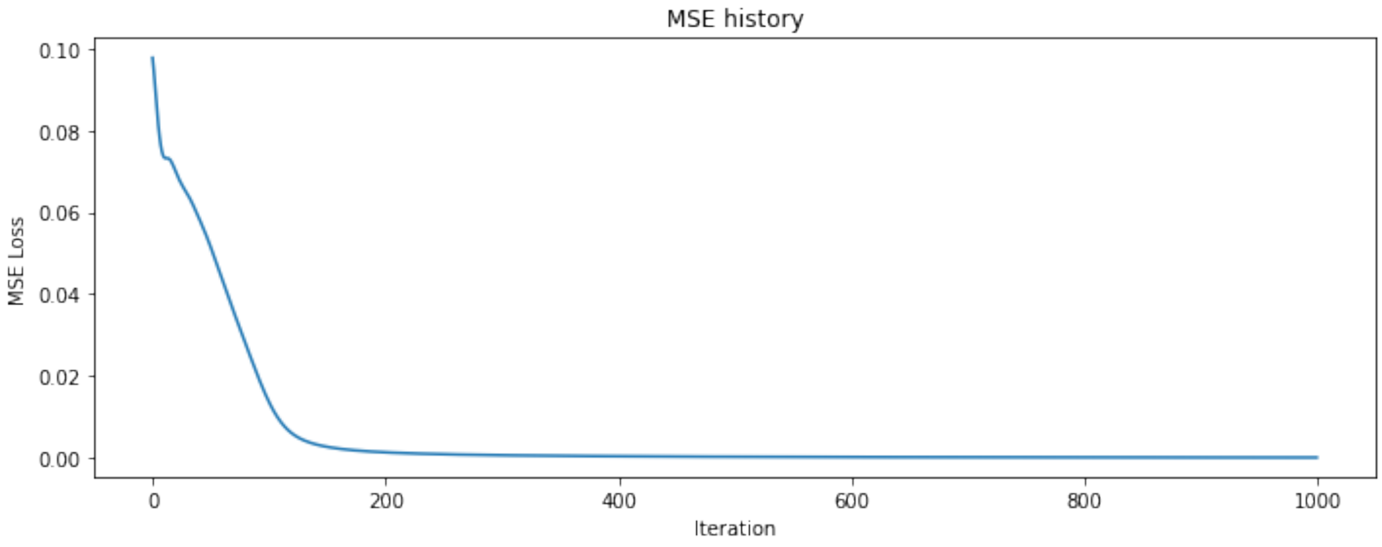
```
size = 128
# TODO pick an image and replace the url string
train_data, test_data = get_image(size, image_url="/content/drive/My Drive/CS 444/a
```

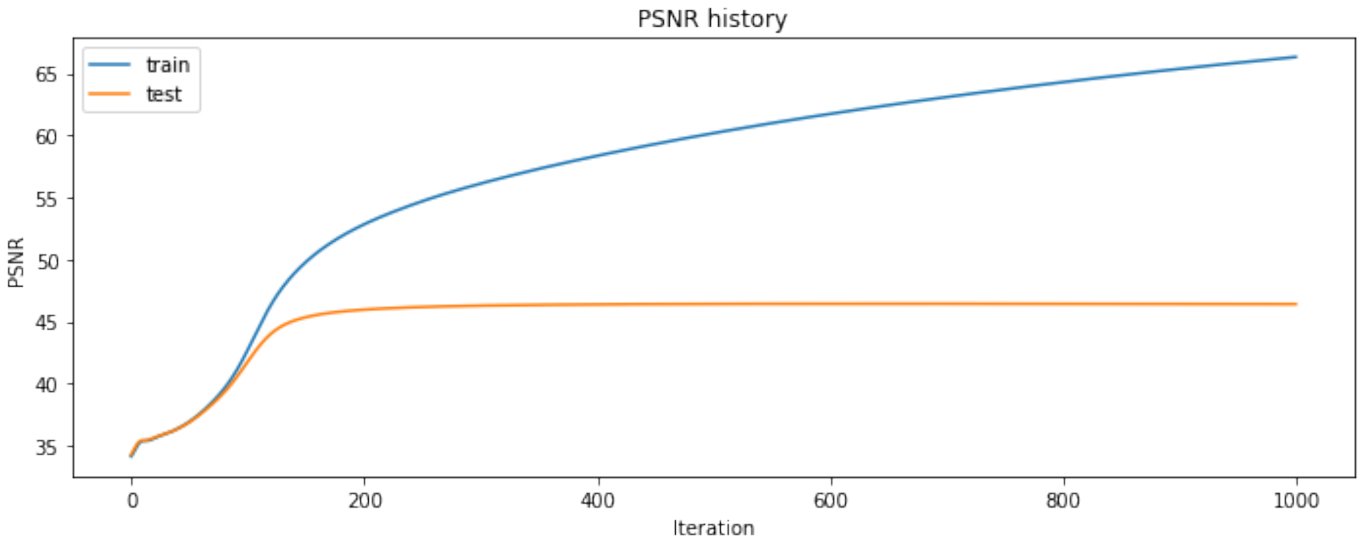


```
# get input features
# TODO implement this by using the get_B_dict() and get_input_features() helper fur
mapping = 'gauss_10.0'
B_dict = get_B_dict()
X_train, y_train, X_test, y_test = get_input_features(B_dict, mapping)
input_size = X_train.shape[1]
  # run NN experiment on input features
  # TODO implement by using the NN_experiment() helper function
```

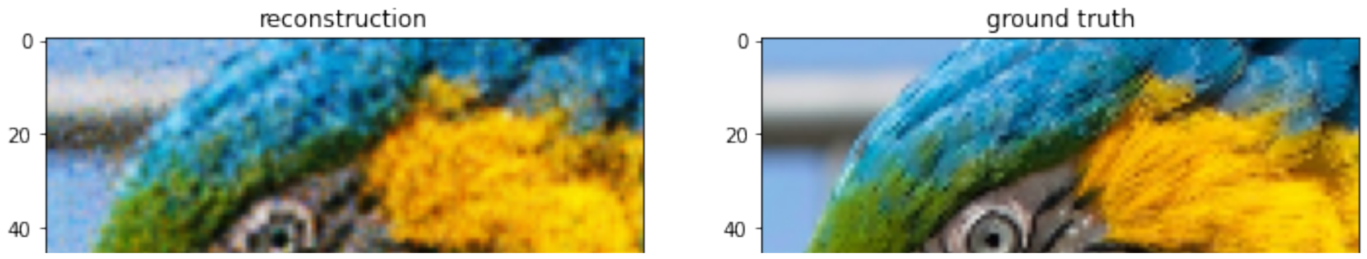
```
net,train_psnr, test_psnr, train_loss, predicted_images = NN_experiment(X_train, y_
                                                                        hidden_size

plot_training_curves(train_loss, train_psnr, test_psnr)
plot_reconstruction(net.forward(X_test), y_test)
plot_reconstruction_progress(predicted_images, y_test)
```

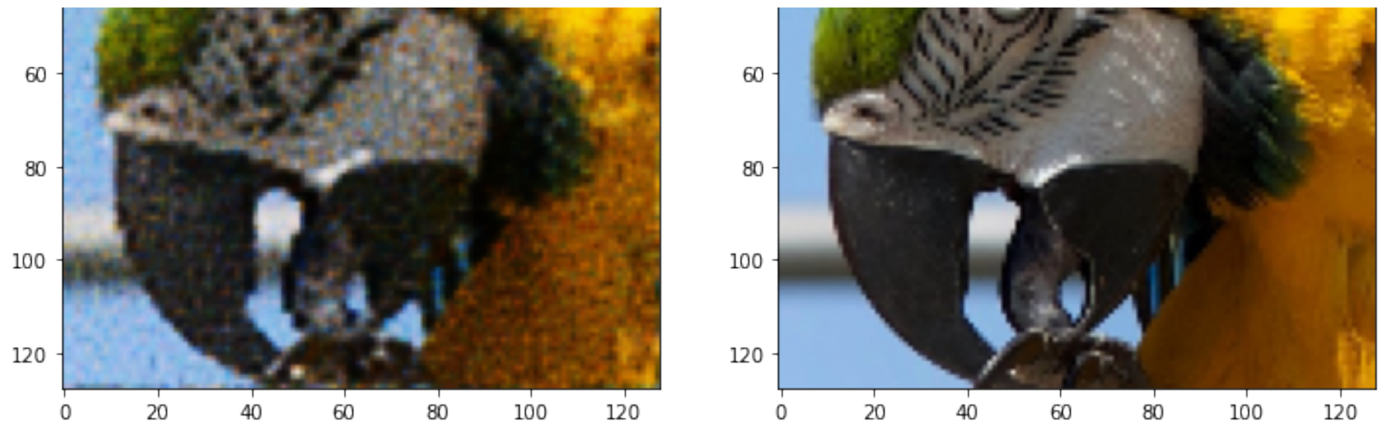
```
Predicted_images dim before storing: (1000, 16384, 3)
100%                                    1000/1000 [20:54<00:00, 1.47s/it]
```

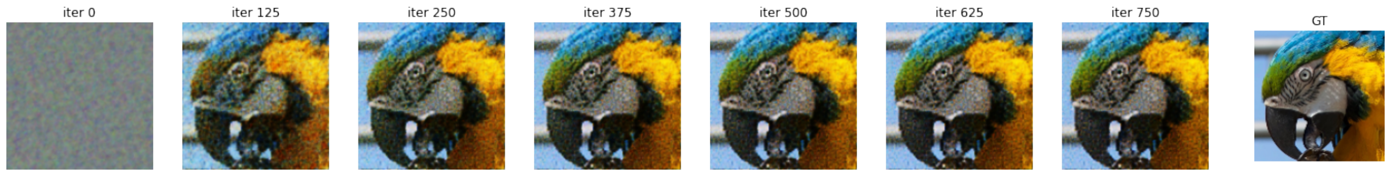




```
Final Test MSE 0.005817254624047251
Final Test psnr 46.41822106940119
```







Reconstruction Process Video (Optional)

(For Fun!) Visualize the progress of training in a video

```
# requires installing this additional dependency
!pip install imageio-ffmpeg

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
    Collecting imageio-ffmpeg
      Downloading imageio_ffmpeg-0.4.8-py3-none-manylinux2010_x86_64.whl (26.9 MB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.9/26.9 MB 21.2 MB/s eta 0:00
    Installing collected packages: imageio-ffmpeg
    Successfully installed imageio-ffmpeg-0.4.8
```

```
# single video example
create_and_visualize_video({"gauss": {"pred_imgs": predicted_images}}, filename="tr
```

```
# multi video example
create_and_visualize_video(outputs, epochs=1000, size=32)
```