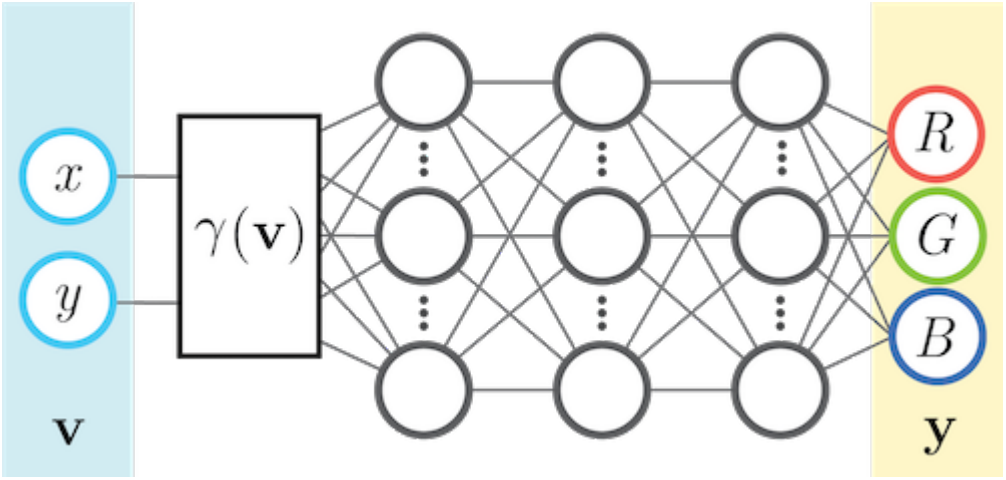
Assignment 2

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: $\gamma(\mathbf{v})=\left[\cos(2 \pi \mathbf{v}),\sin(2 \pi \mathbf{v}) \right]^\mathrm{T}$.
- Gaussian Fourier feature mapping: $\gamma(\mathbf{v})= \left[\cos(2 \pi \mathbf B \mathbf{v}), \sin(2 \pi \mathbf B \mathbf{v}) \right]^\mathrm{T}$, 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, like $m=256$.
- You should show results for σ value of 1.

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

In [26]:
```
# # 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")
```

In [27]:
```
# 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 = "./assignment2_starter_code"
# if not os.path.exists(datadir):
#   !ln -s "/content/drive/My Drive/path/to/your/assignment2/" $datadir
# os.chdir(datadir)
# !pwd
```

Imports

In [49]:
```
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

Image Data and Feature Mappings (Fill in TODOs)

In [50]:
```
# 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
```

In [51]:
```
# Create the mappings dictionary of matrix B -  you will implement this
def get_B_dict(size):
  mapping_size = size // 2 # you may tweak this hyperparameter
  B_dict = {}
  B_dict['none'] = None
  B_dict['basic'] = np.identity(2)
  B_dict['gauss_X'] = np.random.normal(0,1,(mapping_size,2))
  # add B matrix for basic, gauss_1.0
  # TODO implement this

  return B_dict
```

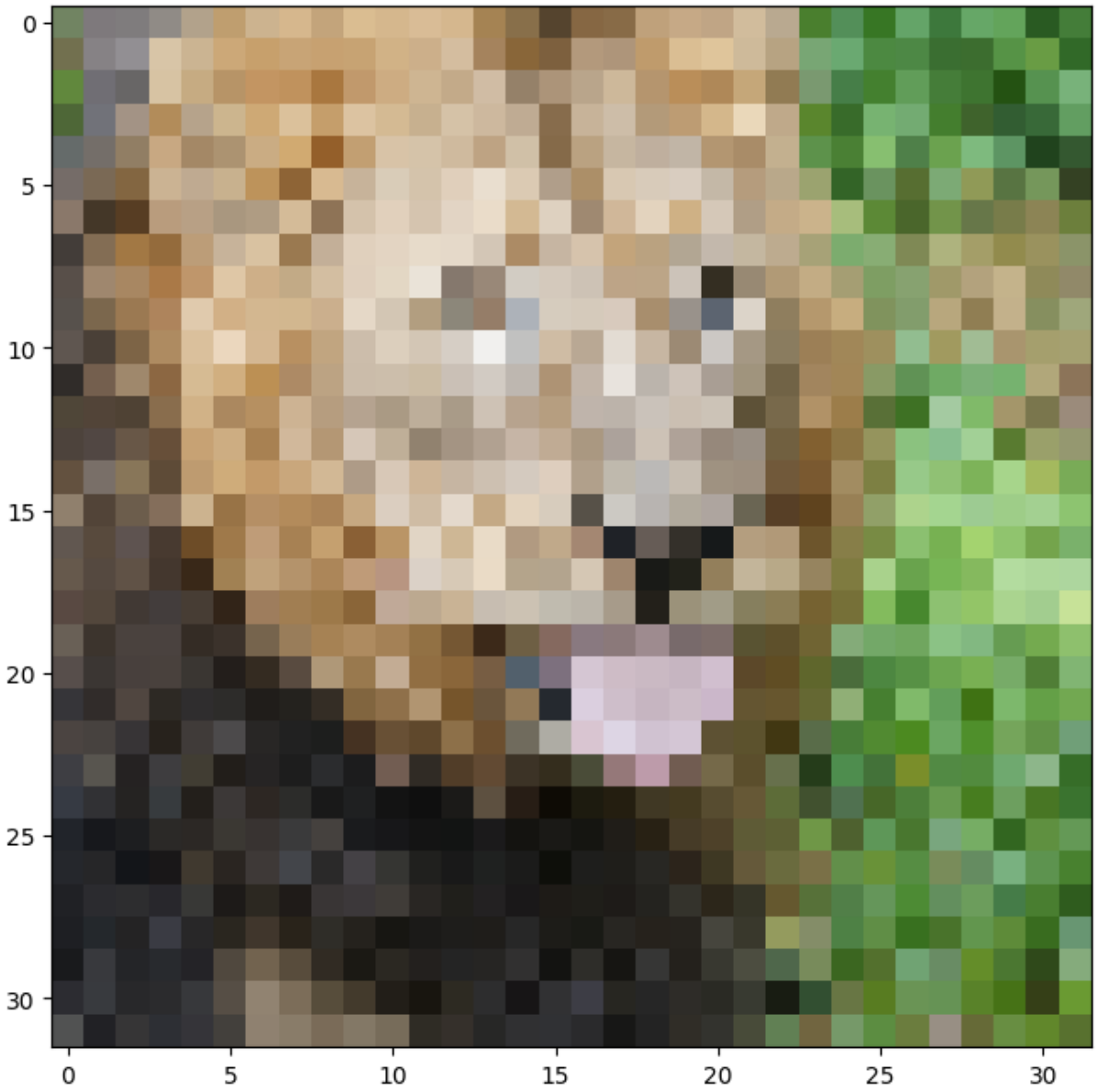
In [52]:
```
# Given tensor x of input coordinates, map it using B - you will implement
def input_mapping(x, B):
  if B is None:
    # "none" mapping - just returns the original input coordinates
    return x
  else:
    # m,d = x.shape
    # print( B.shape)
    x_map = B@x.T
    cos_BX = np.cos(2 * np.pi * x_map)  # Shape: (m, N)
    sin_BX = np.sin(2 * np.pi * x_map)
    x_map = np.hstack((cos_BX.T, sin_BX.T))
    # "basic" mapping and "gauss_X" mappings project input features using B
    # TODO implement this

    return x_map
```

MSE Loss and PSNR Error (Fill in TODOs)

In [53]:
```
def mse(y, p):
  # TODO implement this
  # make sure it is consistent with your implementation in neural_net.py
  n = y.shape[0]*y.shape[1]
  norm = np.sum((p - y) ** 2)/n
  return norm

def psnr(y, p):
  return -10 * np.log10(2.*mse(y, p))
```

In [54]:
```
size = 32
train_data, test_data = get_image(size)
```

```
C:\Users\Carlos\AppData\Local\Temp\ipykernel_46996\2979632171.py:6: DeprecationWarni
ng: Starting with ImageIO v3 the behavior of this function will switch to that of ii
o.v3.imread. To keep the current behavior (and make this warning disappear) use `imp
ort imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  img = imageio.imread(image_url)[..., :3] / 255.
```



Some suggested hyperparameter choices to help you start

- hidden layer count: 4
- hidden layer size: 256
- number of epochs: 1000
- learning rate: 0.1

In [55]:
```
num_layers = 4  # you should not change this value

# TODO: Set the hyperparameters
hidden_size = 256
epochs = 2000
learning_rate = 0.2
output_size = 3
B_dict = get_B_dict(size)

print('B_dict items:')
for k,v in B_dict.items():
    print('\t',k,np.array(v).shape)
```

```
B_dict items:
	 none ()
	 basic (2, 2)
	 gauss_X (16, 2)
```

In [56]:
```
# 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
```

Plotting and video helper functions (you don't need to change anything here)

In [57]:
```
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):
  p_im = p.reshape(size,size,3)
  y_im = y.reshape(size,size,3)

  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, p))
  print("Final Test psnr",psnr(y, p))

def plot_reconstruction_progress(predicted_images, y, 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+1, 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.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='trainin
  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())))
```

Experiment Runner (Fill in TODOs)

In [58]:

```
def NN_experiment(X_train, y_train, X_test, y_test, input_size, num_layers,
                  hidden_size, output_size, epochs, learning_rate, opt='SGD'):

    # Initialize a new neural network model
    hidden_sizes = [hidden_size] * (num_layers - 1)
    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]))

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

        # Shuffle the dataset
        # TODO implement this
        # print(X_train.shape)
        batch_size = 32
        indices = np.random.permutation(X_train.shape[0])  # Shuffle along sample a
        X_train_shuffled = X_train[indices]
        y_train_shuffled = y_train[indices]
        for i in range(0, X_train_shuffled.shape[0], batch_size):
            X_batch = X_train_shuffled[i:i + batch_size]
            y_batch = y_train_shuffled[i:i + batch_size]

            predictions = net.forward(X_batch)
            # print(y_batch.shape,predictions.shape)
            loss = net.backward(y_batch)
            train_loss[epoch] += loss
            train_psnr[epoch] += psnr(y_batch, predictions)
        # Training
        # Run the forward pass of the model to get a prediction and record the psnr
        # TODO implement this
        # predictions = net.forward(X_train_shuffled)
        # loss = net.backward(y_train_shuffled)
        # train_loss[epoch] = loss
        # Run the backward pass of the model to compute the loss, record the loss,
        # TODO implement this

        test_predictions = net.forward(X_test)
        test_psnr[epoch] = psnr(y_test, test_predictions)
        # Testing
        # No need to run the backward pass here, just run the forward pass to compu
        # TODO implement this
        predicted_images[epoch] = test_predictions
        net.update(learning_rate)

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

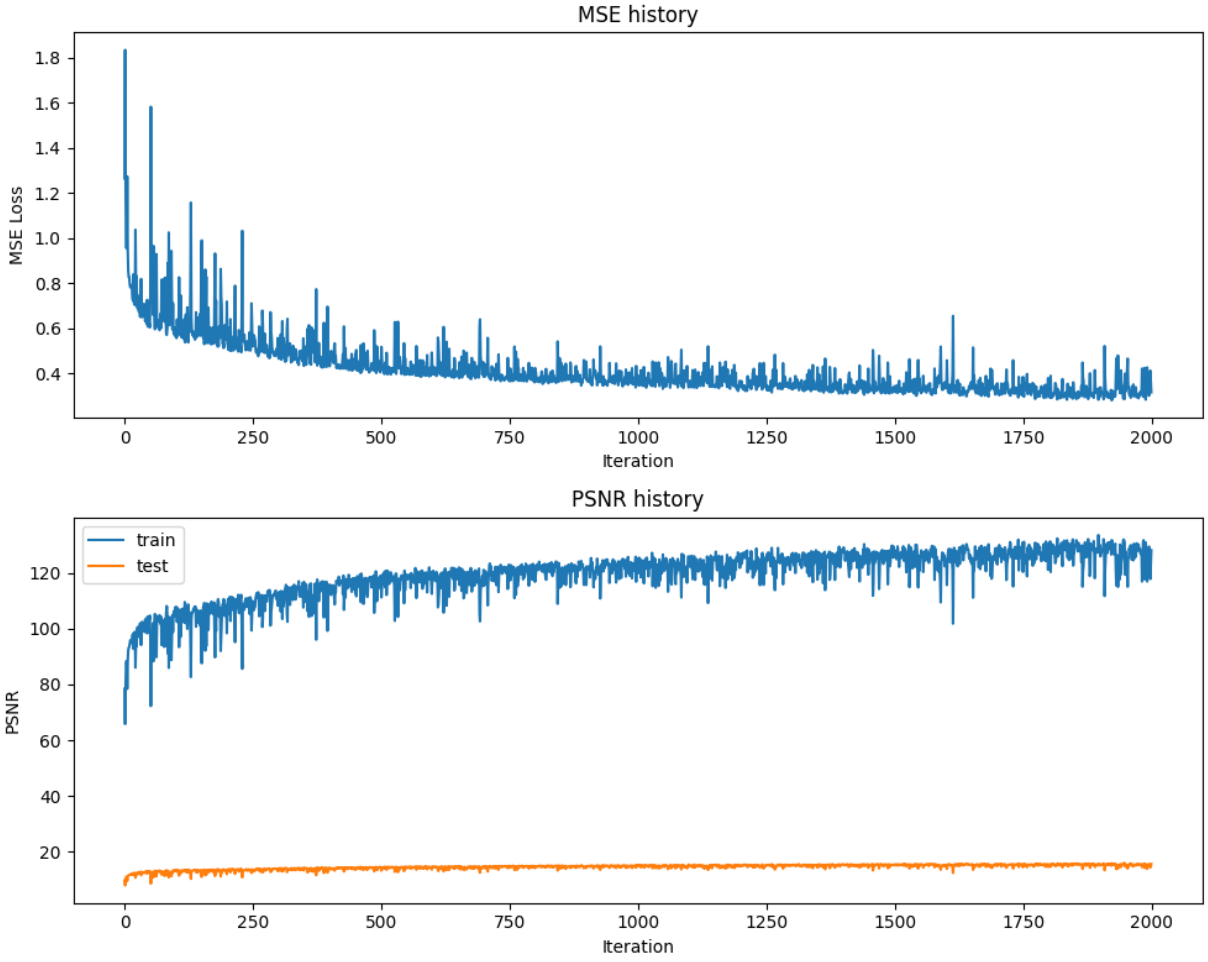
Low Resolution Reconstruction

Low Resolution Reconstruction - SGD - None Mapping

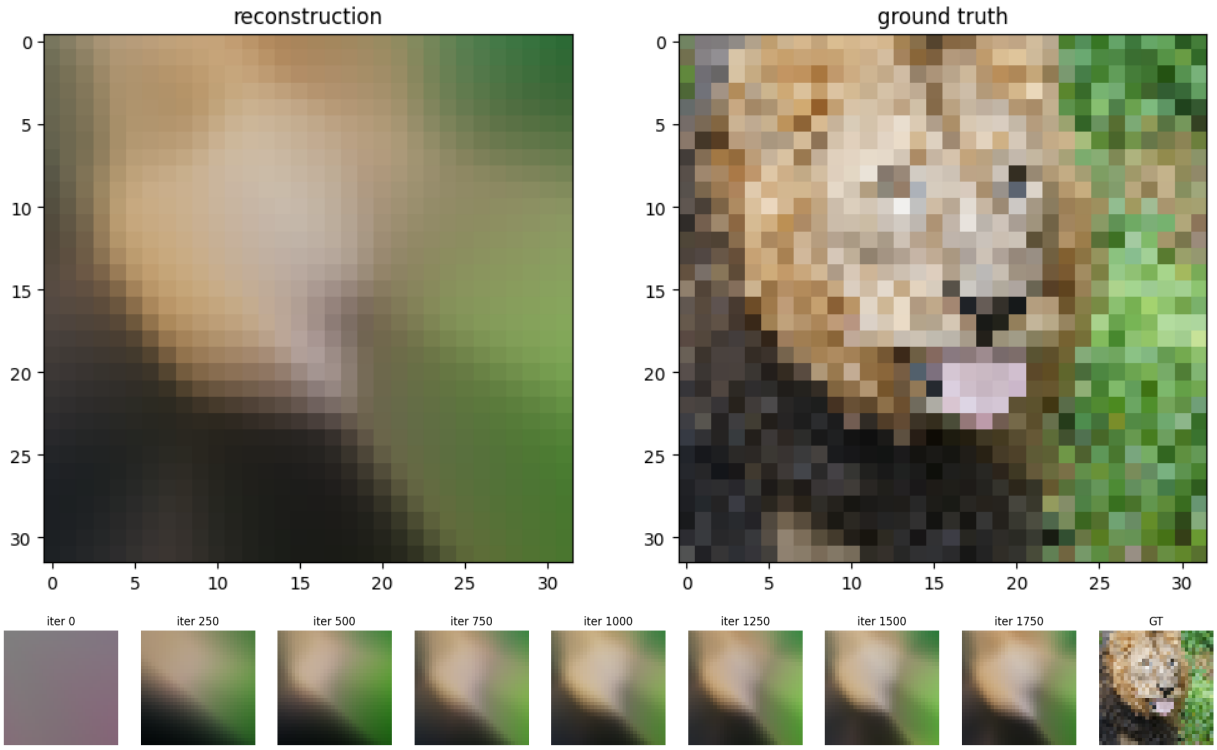
In [59]:
```
# get input features
# TODO implement this by using the get_B_dict() and get_input_features() helper fun
B_dict = get_B_dict(size)
X_train, y_train, X_test, y_test = get_input_features(B_dict, "none")
input_size = X_train.shape[1]
```

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

```
(256, 2)
  0%|          | 0/2000 [00:00<?, ?it/s]
```




```
Final Test MSE 0.014387615365863367
Final Test psnr 15.409811853310472
```



Low Resolution Reconstruction - SGD - Various Input Mapping Stategies

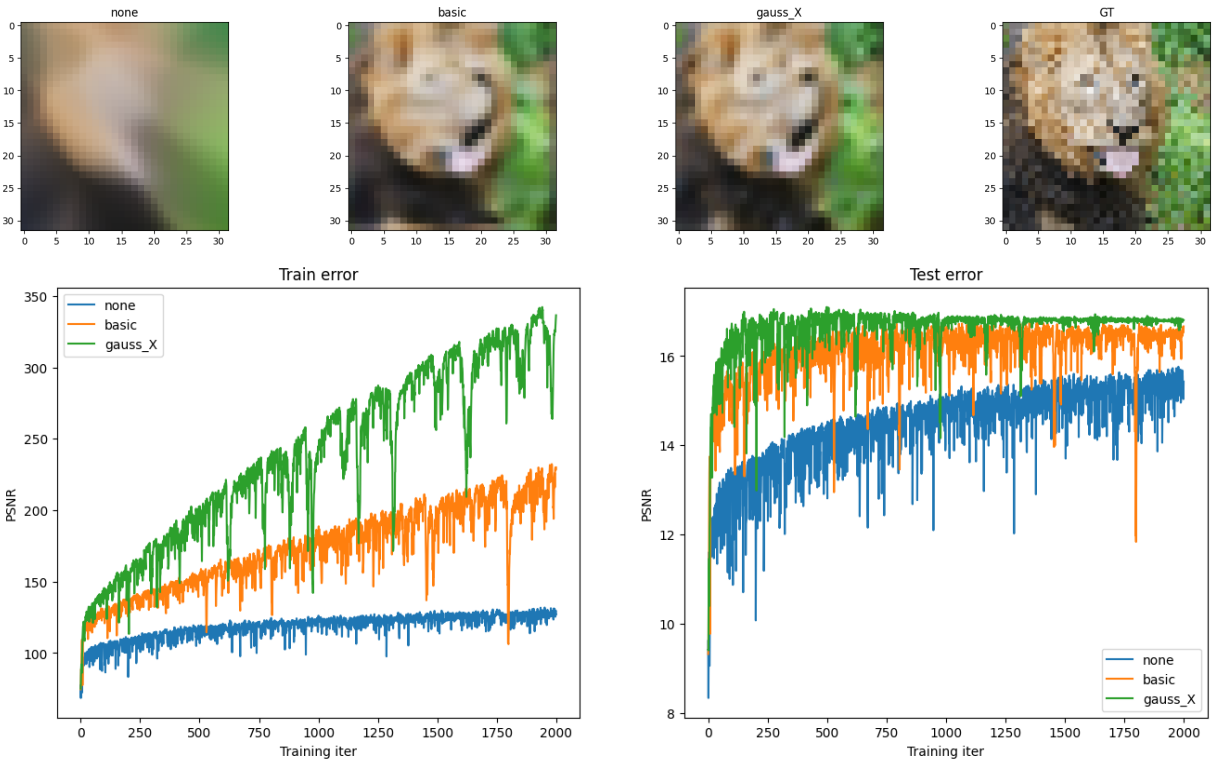
In [60]:
```
def train_wrapper(mapping, size, num_layers, hidden_size, output_size, epochs,  lea
    # TODO implement me
    # makes it easy to run all your mapping experiments in a for loop
    # this will similar to what you did previously in the last two sections
    B_dict = get_B_dict(size)
    X_train, y_train, X_test, y_test = get_input_features(B_dict, mapping)
    input_size = X_train.shape[1]
    print(X_train.shape)
    net, train_psnrs, test_psnrs, train_loss, predicted_images = NN_experiment(X_tr

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

In [61]:
```
outputs = {}
for k in tqdm(B_dict):
  print("training", k)
  outputs[k] = train_wrapper(k, size, num_layers, hidden_size, output_size, epochs,
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
training none
(256, 2)
  0%|          | 0/2000 [00:00<?, ?it/s]
training basic
(256, 4)
  0%|          | 0/2000 [00:00<?, ?it/s]
```

```
training gauss_X
(256, 32)
  0%|          | 0/2000 [00:00<?, ?it/s]
```

In [62]:
```
# 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))
```



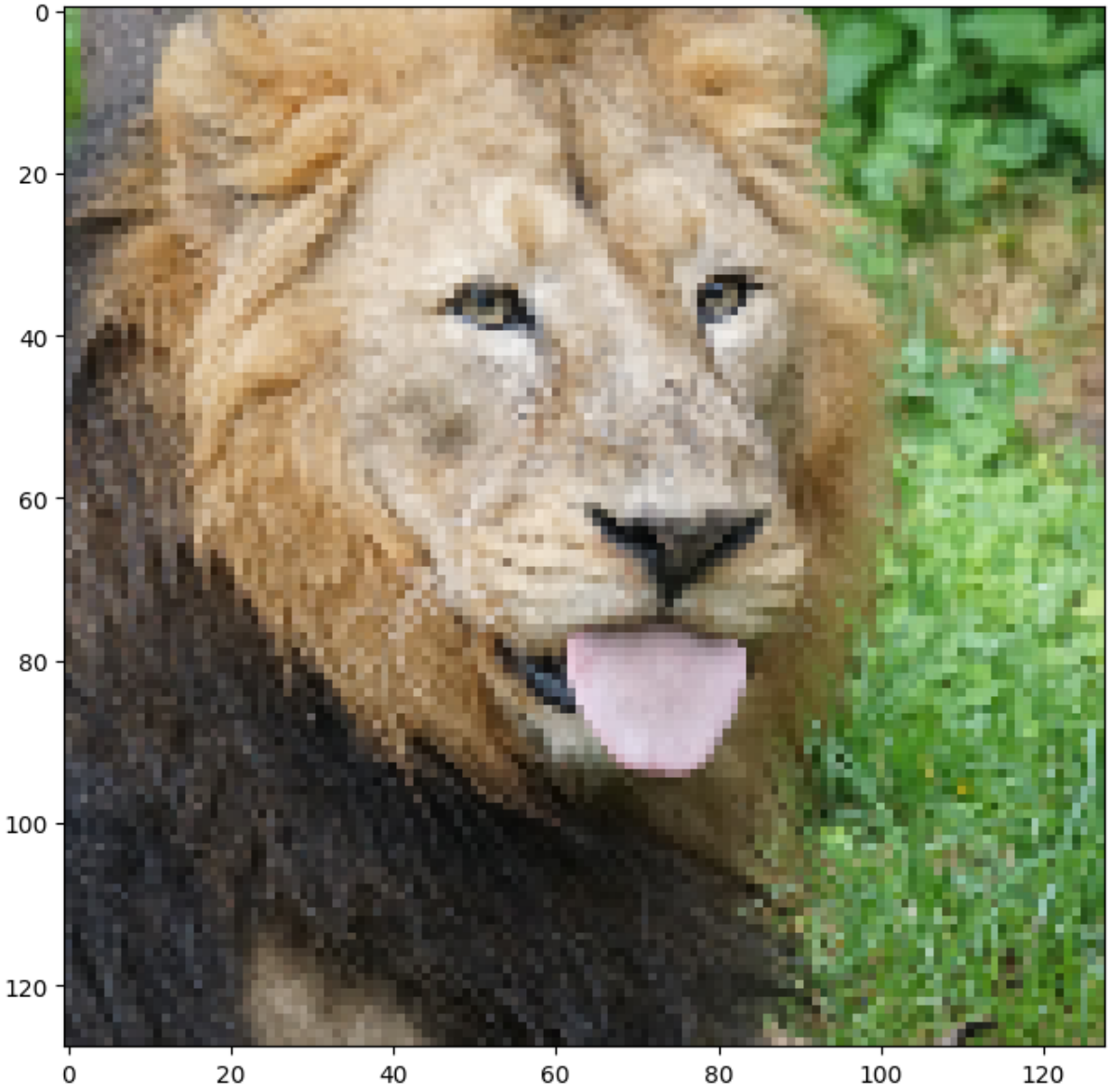
High Resolution Reconstruction

High Resolution Reconstruction - SGD - 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.

In [63]:
```
# load hi-res image
size = 128
train_data, test_data = get_image(size)
```

```
C:\Users\Carlos\AppData\Local\Temp\ipykernel_46996\2979632171.py:6: DeprecationWarni
ng: Starting with ImageIO v3 the behavior of this function will switch to that of ii
o.v3.imread. To keep the current behavior (and make this warning disappear) use `imp
ort imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  img = imageio.imread(image_url)[..., :3] / 255.
```

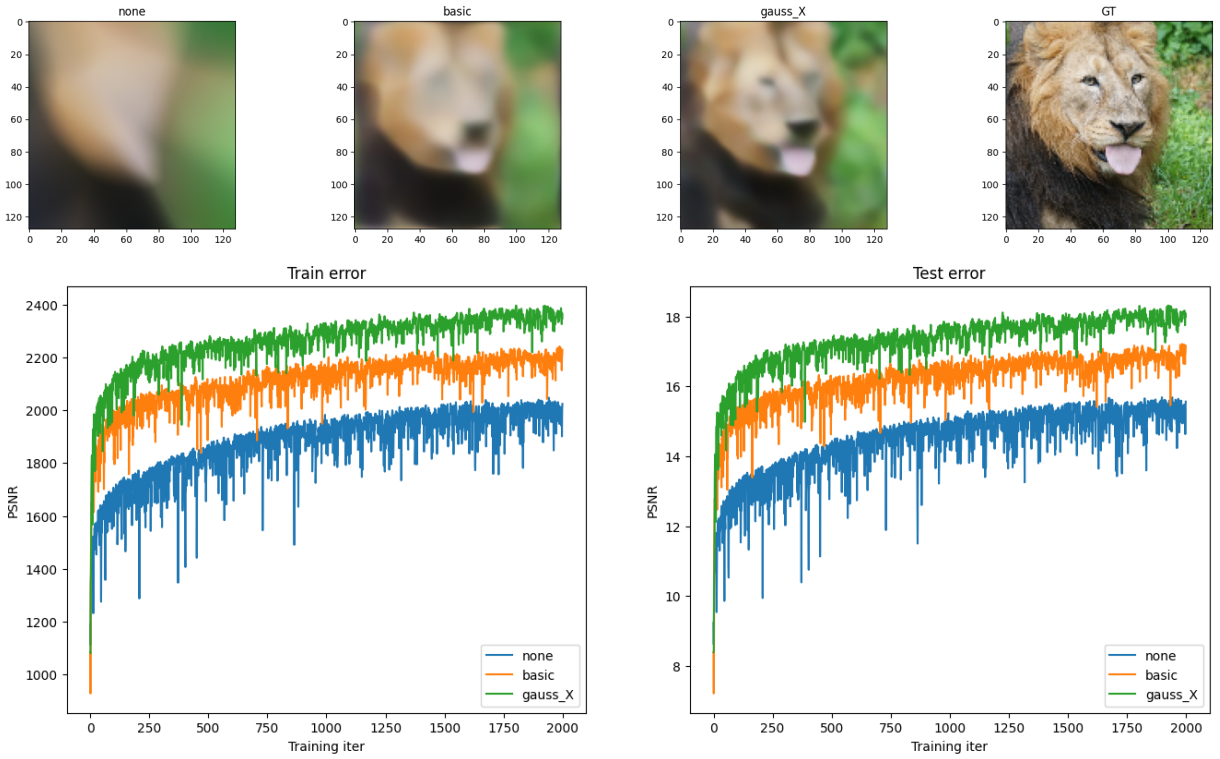


In [64]:
```
outputs = {}
# hidden_size = 256
for k in tqdm(B_dict):
  print("training", k)
  outputs[k] = train_wrapper(k,size, num_layers, hidden_size, output_size, epochs,
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
training none
(4096, 2)
  0%|          | 0/2000 [00:00<?, ?it/s]
training basic
(4096, 4)
  0%|          | 0/2000 [00:00<?, ?it/s]
training gauss_X
(4096, 128)
  0%|          | 0/2000 [00:00<?, ?it/s]
```

In [65]:
```
X_train, y_train, X_test, y_test = get_input_features(get_B_dict(size), "none")  #
```

```
# 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))
```

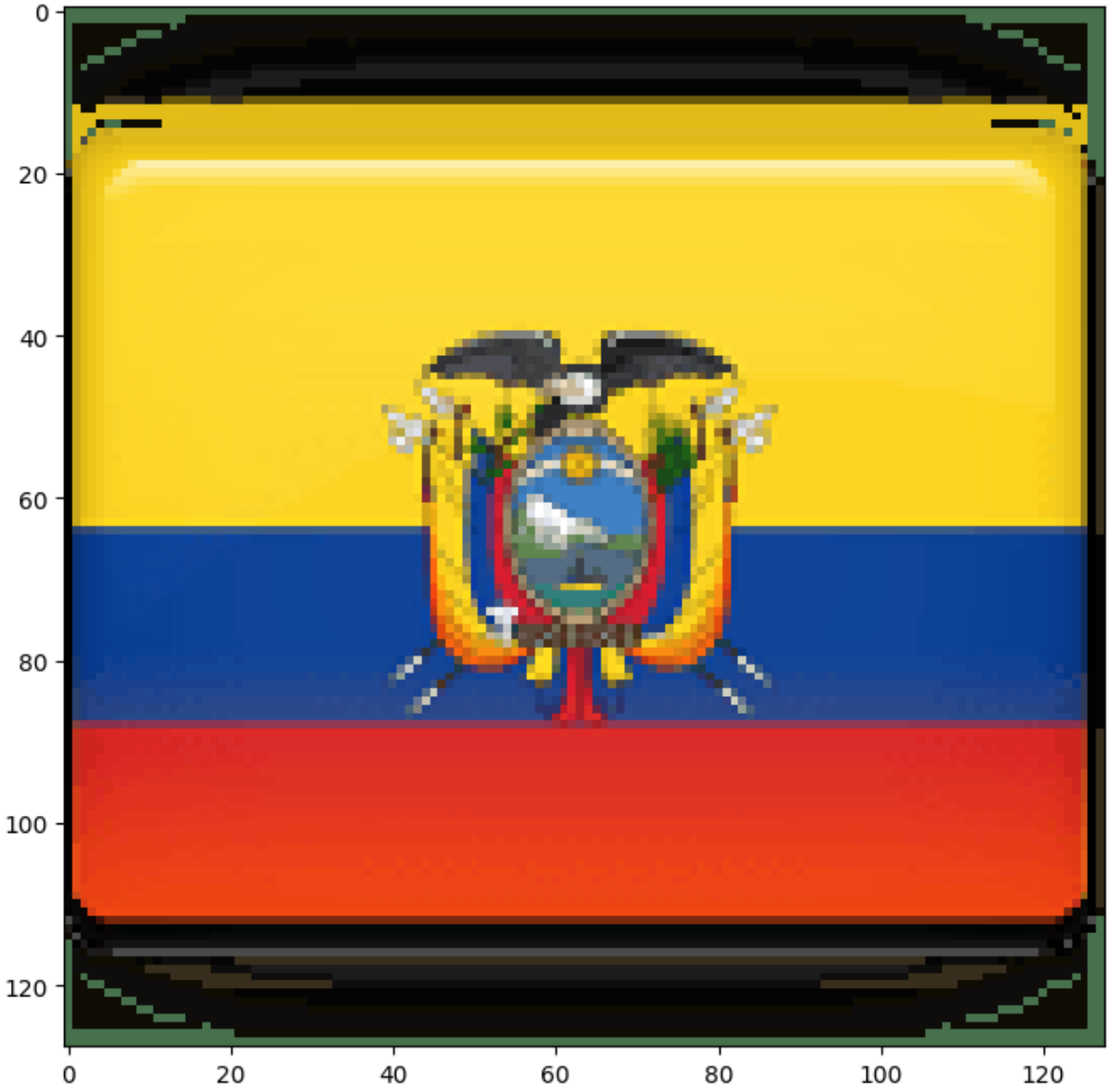


High Resolution Reconstruction - Image of your Choice

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.

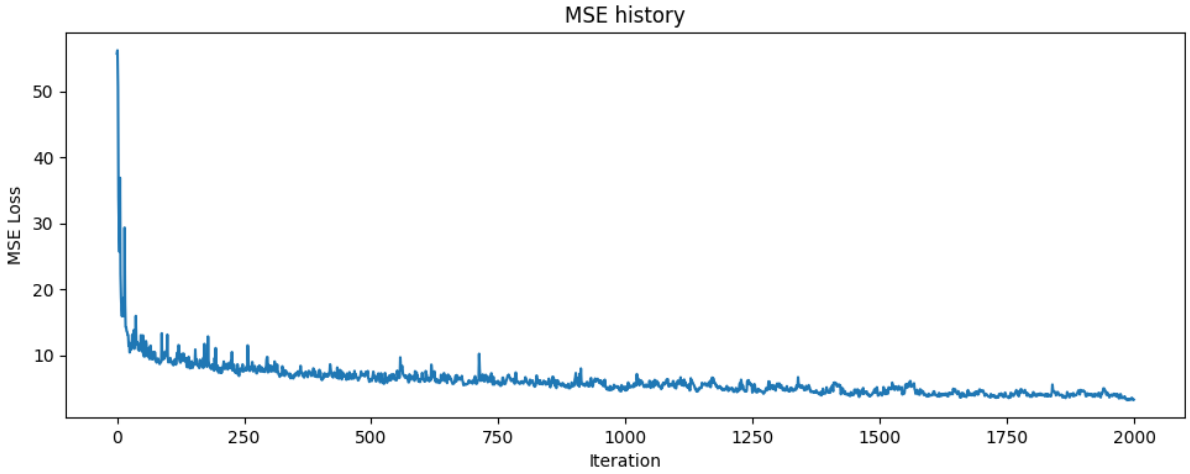
In [66]:
```
size = 128
# TODO pick an image and replace the url string
train_data, test_data = get_image(size, image_url="https://icons.iconarchive.com/ic
```

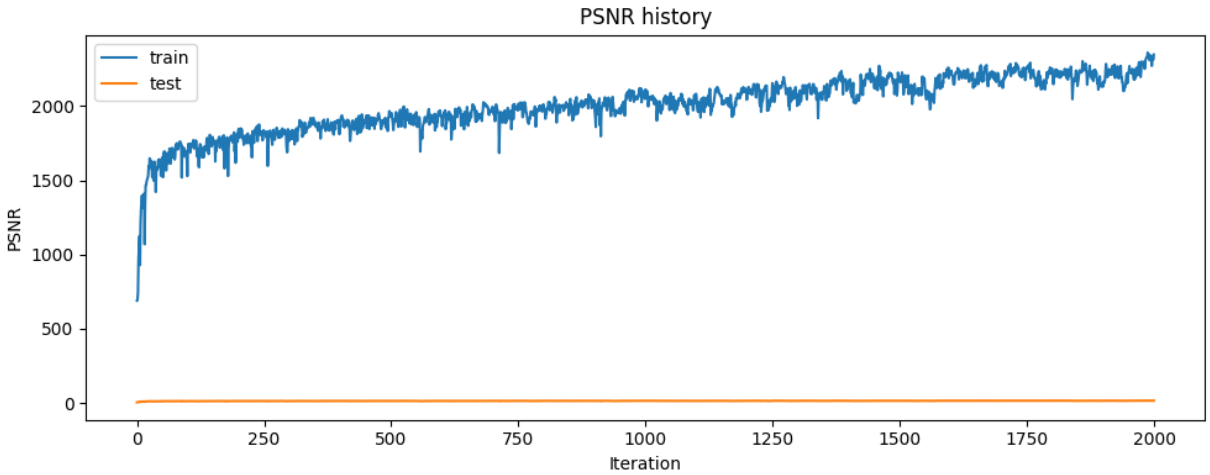
```
C:\Users\Carlos\AppData\Local\Temp\ipykernel_46996\2979632171.py:6: DeprecationWarni
ng: Starting with ImageIO v3 the behavior of this function will switch to that of ii
o.v3.imread. To keep the current behavior (and make this warning disappear) use `imp
ort imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  img = imageio.imread(image_url)[..., :3] / 255.
```



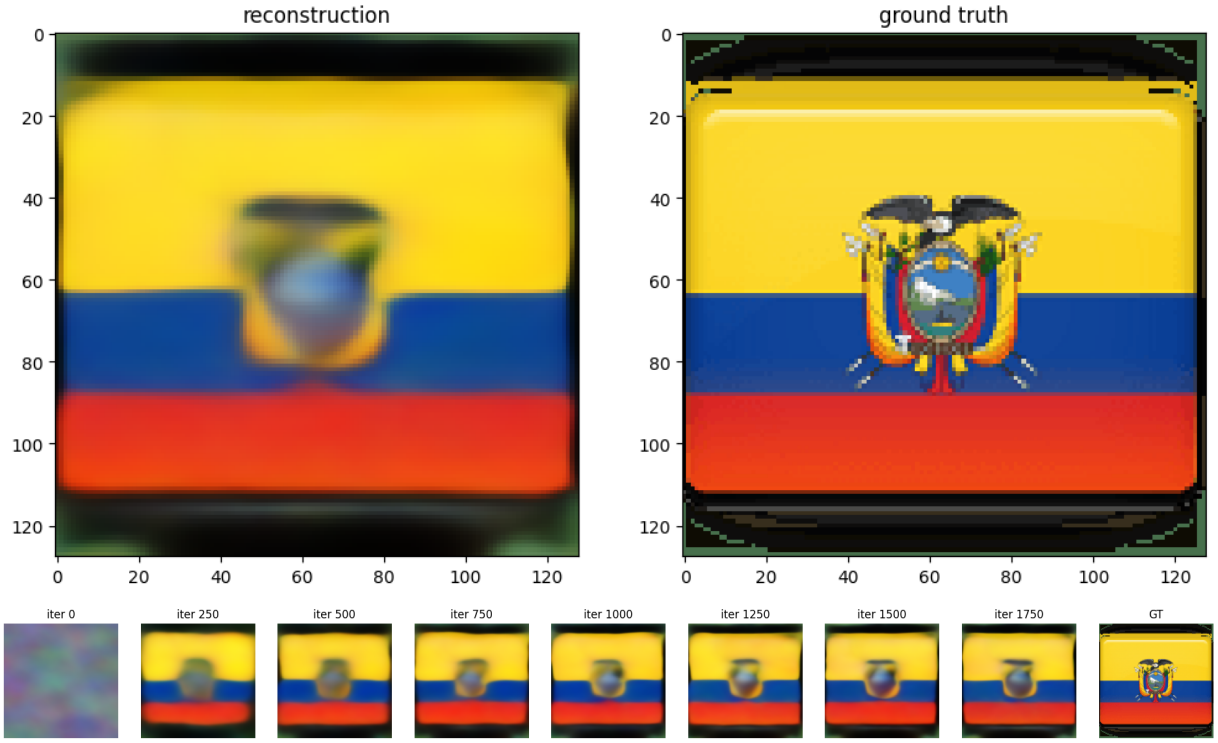
In [67]:
```
# get input features
# TODO implement this by using the get_B_dict() and get_input_features() helper fun

# run NN experiment on input features
# TODO implement by using the NN_experiment() helper function
B_dict = get_B_dict(size)
X_train, y_train, X_test, y_test = get_input_features(B_dict, "gauss_X")
input_size = X_train.shape[1]
# run NN experiment on input features
# TODO implement by using the NN_experiment() helper function
print(X_train.shape)
net, train_psnr, test_psnr, train_loss, predicted_images = NN_experiment(X_train,y_
# 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)
```

```
(4096, 128)
  0%|          | 0/2000 [00:00<?, ?it/s]
```






```
Final Test MSE 0.011076428217657898
Final Test psnr 16.545702669978596
```

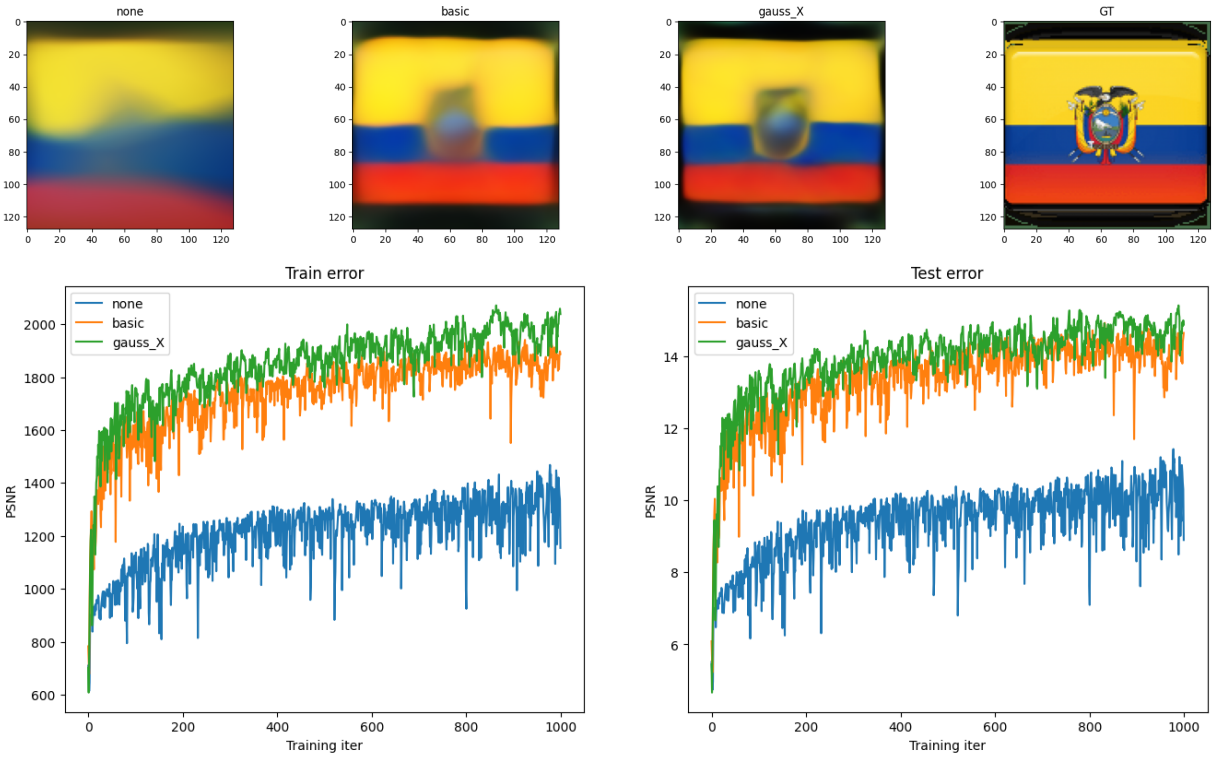


In [47]:
```
for k in tqdm(B_dict):
  print("training", k)
```

```
    outputs[k] = train_wrapper(k, size, num_layers, hidden_size, output_size, epochs,
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
training none
(4096, 2)
  0%|          | 0/1000 [00:00<?, ?it/s]
training basic
(4096, 4)
  0%|          | 0/1000 [00:00<?, ?it/s]
training gauss_X
(4096, 128)
  0%|          | 0/1000 [00:00<?, ?it/s]
```

In [48]:
```
X_train, y_train, X_test, y_test = get_input_features(get_B_dict(size), "none")
plot_feature_mapping_comparison(outputs, y_test.reshape(size,size,3))
```



Reconstruction Process Video (Optional)

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

In [68]:
```
# requires installing this additional dependency
!pip install imageio-ffmpeg
```

```
Requirement already satisfied: imageio-ffmpeg in c:\users\carlos\appdata\local\progr
ams\python\python312\lib\site-packages (0.6.0)
```

In [69]:
```
# single video example
create_and_visualize_video({"gauss": {"pred_imgs": predicted_images}}, filename="tr
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[69], line 2
      1 # single video example
----> 2 create_and_visualize_video({"gauss": {"pred_imgs": predicted_images}}, filen
ame="training_high_res_gauss.mp4")

Cell In[57], line 89, in create_and_visualize_video(outputs, size, epochs, filename)
     88 def create_and_visualize_video(outputs, size=size, epochs=epochs, filename
='training_convergence.mp4'):
---> 89   all_preds = np.concatenate([outputs[n]['pred_imgs'].reshape(epochs,size,si
ze,3)[::25] for n in outputs], axis=-2)
     90   data8 = (255*np.clip(all_preds, 0, 1)).astype(np.uint8)
     91   f = os.path.join(filename)

ValueError: cannot reshape array of size 98304000 into shape (2000,32,32,3)
```

In [ ]:
```
# multi video example
create_and_visualize_video(outputs, epochs=1000, size=32)
```

Extra Credit - Adam Optimizer

Low Resolution Reconstruction - Adam - None Mapping

In [ ]:
```
# load low-res image
size = 32
train_data, test_data = get_image(size)
```

In [ ]:
```
# get input features
# TODO implement this by using the get_B_dict() and get_input_features() helper fun

# run NN experiment on input features
# TODO implement by using the NN_experiment() helper function

# 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)
```

Low Resolution Reconstruction - Adam - Various Input Mapping Stategies

In [ ]:
```
# start training
outputs = {}
for k in tqdm(B_dict):
  print("training", k)
  outputs[k] = train_wrapper(k, size, num_layers, hidden_size, output_size, epochs,
```

High Resolution Reconstruction - Adam - 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.

In [ ]:
```
# load image
size = 128
train_data, test_data = get_image(size)

# start training
outputs = {}
for k in tqdm(B_dict):
  print("training", k)
  outputs[k] = train_wrapper(k, size, num_layers, hidden_size, output_size, epochs,
```

In [ ]:
```
X_train, y_train, X_test, y_test = get_input_features(get_B_dict(size), "none")  #

# 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))
```