Week8 – Satish Ramachandran

Problem#1

```
'''
Week 8 - Problem #1
Fibonacci - auto-encoder
'''

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn import metrics
from numpy.random import seed
from tensorflow import set_random_seed

def create_dataset():
    # 20 data points
    x = np.array([np.arange(0,20)])
    # Data is the fibonacci series
    fib = [1,1]
    num = 20
    for ctr in np.arange(2, num):
        fib.append(fib[ctr-1] + fib[ctr - 2])
    plt.title('generated data')
    plt.scatter(range(20), fib)
    plt.show()
    plt.waitforbuttonpress()
    plt.close()
    np_array_data = np.array([fib])
    np_array_data.astype(np.float32)
    return (x, np_array_data)

x, y = create_dataset()
print(x)
print(y)
seed(1)
set_random_seed(2)
```

```
model = Sequential()
# Hidden layer with 20 inputs
model.add(Dense(6, input_dim=x.shape[1], activation='relu'))
# Output layer is the same as input
model.add(Dense(x.shape[1]))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()

#Train the model. The output is the same as the input
model.fit(y, y, verbose = 0, epochs=500)
predict = model.predict(y)
print(predict)

#Calculate the RMSE
score = np.sqrt(metrics.mean_squared_error(predict, y))
print("RMSE score: {}".format(score))

plt.title('model output - 6 hidden nodes - 500 epochs')
plt.scatter(x, y, c='red', s=150)
plt.scatter(x, predict, c = 'blue')
plt.show()
plt.waitforbuttonpress()
```

Observations:

Tried the problem with different number of epochs and different number of hidden layers. When the number of epochs is less, the number of hidden layers makes a lot of difference. For example, when the number of epochs is 200, the number of nodes in the hidden layer makes a lot of difference.

Including plots of the different numbers of input nodes with 200 epochs, and also one plot with 6 input nodes and 500 epochs

The RED dots are the target Fibonacci series, and BLUE dots is the what the model predicts.

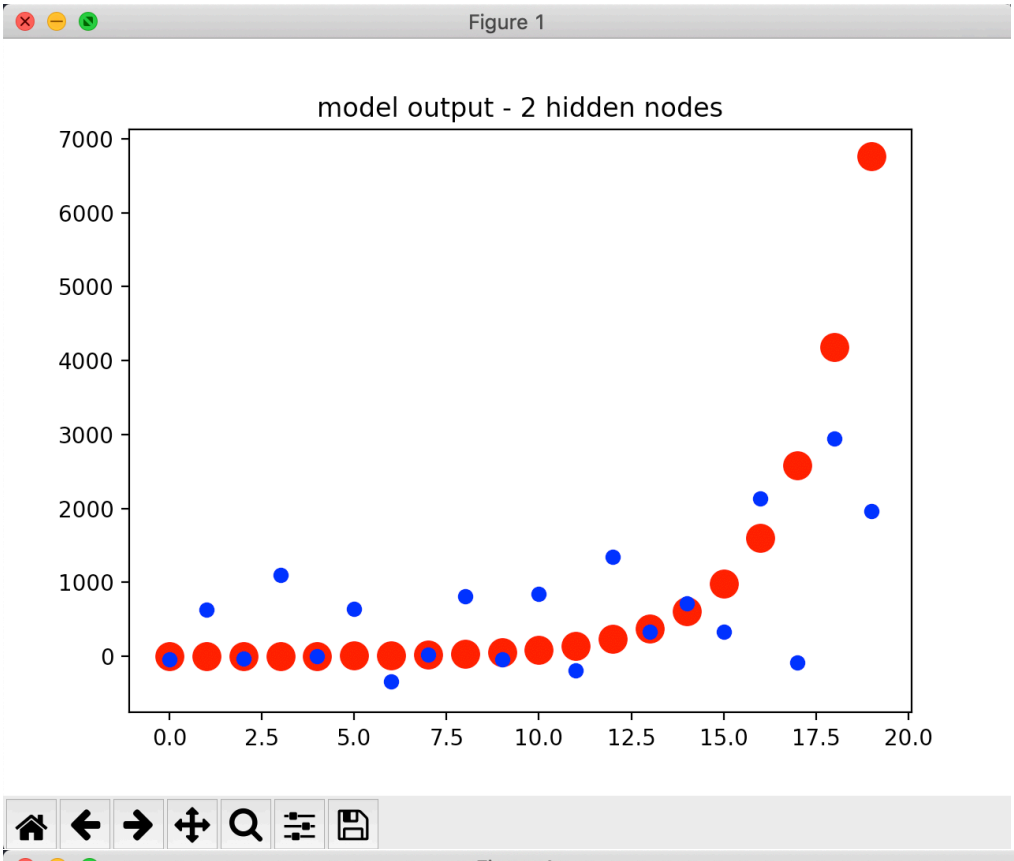


Figure 1
model output - 2 hidden nodes
7000
6000
5000
4000
3000
2000
1000
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

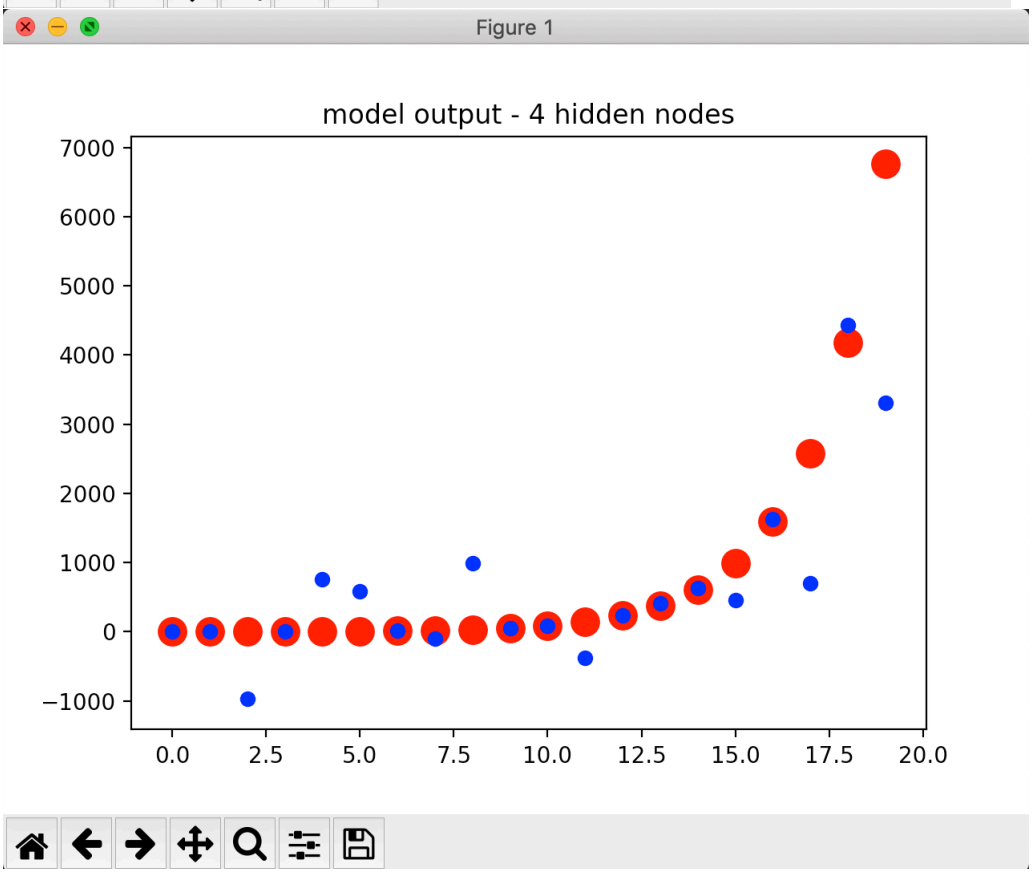
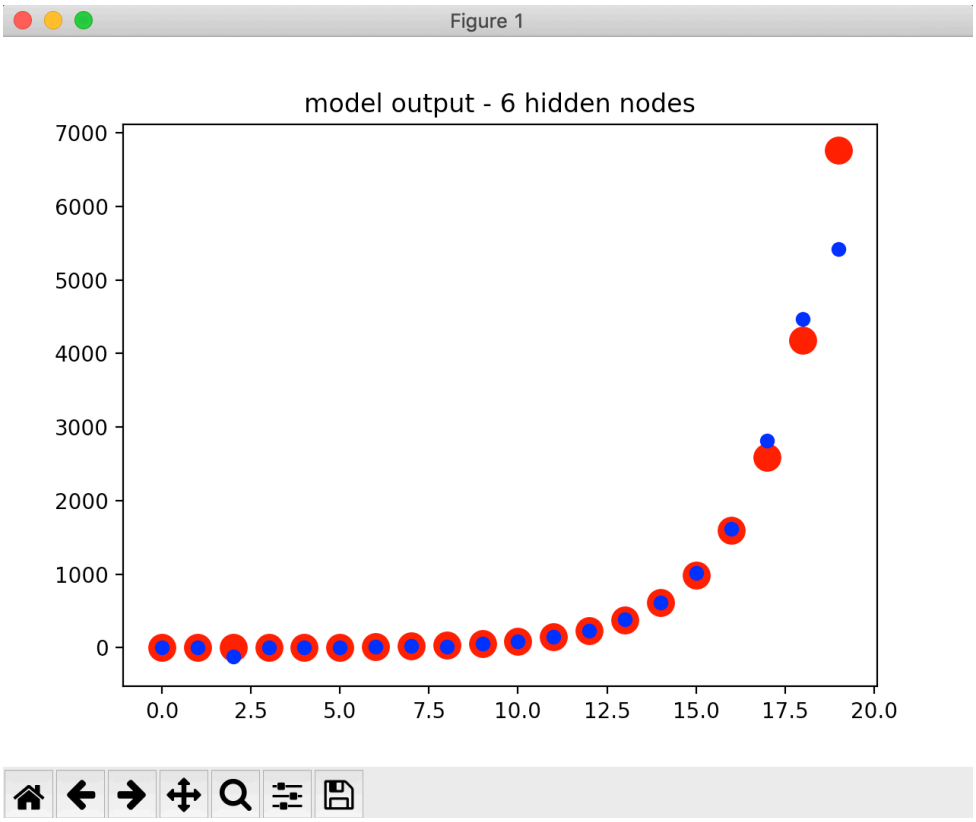
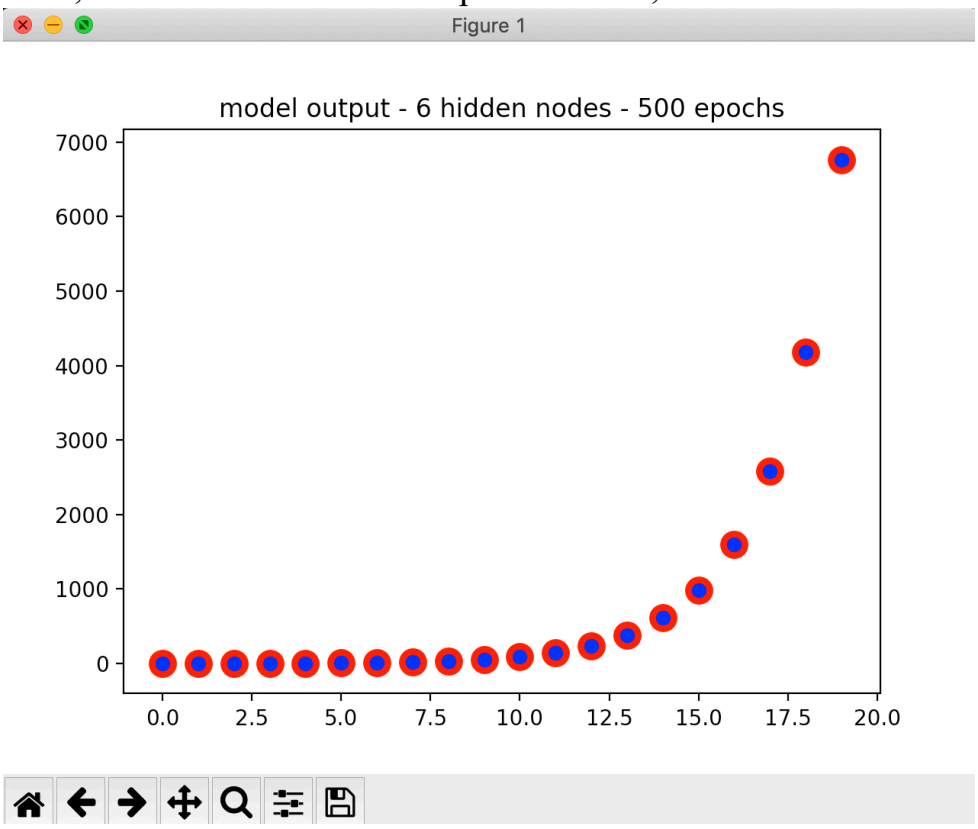


Figure 1
model output - 4 hidden nodes
7000
6000
5000
4000
3000
2000
1000
0
−1000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



Now, increase the number of epochs to 500, and the fit is much better.



The RMSE reflects exactly how the plots map

For 2 nodes:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2)                 42
_________________________________________________________________
dense_1 (Dense)              (None, 20)                60
=================================================================
Total params: 102
Trainable params: 102
Non-trainable params: 0
_________________________________________________________________
2020-05-24 12:04:49.027069: I tensorflow/core/platform/cpu_feature_guard.cc:145] Th
ollowing CPU instructions in performance critical operations:  SSE4.1 SSE4.2 AVX AV
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate c
2020-05-24 12:04:49.027402: I tensorflow/core/common_runtime/process_util.cc:115] C
 inter_op_parallelism_threads for best performance.
[[ -40.10772    634.4239     -26.331163  1102.5955       3.6699338
   639.4682    -342.24533     20.589197   815.69025    -42.78402
   847.8989    -194.78508   1344.2128     332.31268    720.0726
   326.94025   2138.307      -83.49088   2944.181     1964.4407   ]]
RMSE score: 1362.3010628975587
```

For 6 nodes:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 6)                 126
_________________________________________________________________
dense_1 (Dense)              (None, 20)                140
=================================================================
Total params: 266
Trainable params: 266
Non-trainable params: 0
_________________________________________________________________
2020-05-24 12:05:51.866963: I tensorflow/core/platform/cpu_feature_guard.cc:1
ollowing CPU instructions in performance critical operations:  SSE4.1 SSE4.2
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropr
2020-05-24 12:05:51.867340: I tensorflow/core/common_runtime/process_util.cc:
 inter_op_parallelism_threads for best performance.
[[ 1.2203109e+00  1.4814793e+00 -1.2047965e+02  2.5922453e+00
   6.2725830e-01  6.2649083e+00  1.3060483e+01  2.1814772e+01
   1.3668495e+01  5.5184620e+01  9.0193069e+01  1.4685010e+02
   2.3374716e+02  3.8057190e+02  6.1016754e+02  1.0128535e+03
   1.6175087e+03  2.8168018e+03  4.4655459e+03  5.4194458e+03]]
RMSE score: 313.2274689196692
```

As can be seen, the score gets better.

But, there is a significant improvement when the number of epochs is imcreased. Now, with 500 epochs.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 6)                 126
_________________________________________________________________
dense_1 (Dense)              (None, 20)                140
=================================================================
Total params: 266
Trainable params: 266
Non-trainable params: 0
_________________________________________________________________
2020-05-24 12:07:23.873385: I tensorflow/core/platform/cpu_feature_guard.cc:
ollowing CPU instructions in performance critical operations:  SSE4.1 SSE4.2
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the approp
2020-05-24 12:07:23.873673: I tensorflow/core/common_runtime/process_util.cc
 inter_op_parallelism_threads for best performance.
[[9.9892867e-01 1.0002300e+00 2.0010037e+00 3.0001681e+00 4.9994378e+00
  7.9996572e+00 1.2999986e+01 2.0999983e+01 3.3995445e+01 5.4999638e+01
  8.8999870e+01 1.4400002e+02 2.3300037e+02 3.7700027e+02 6.1000012e+02
  9.8700250e+02 1.5970018e+03 2.5840334e+03 4.1810430e+03 6.7648735e+03]]
RMSE score: 0.030815413180773778
```

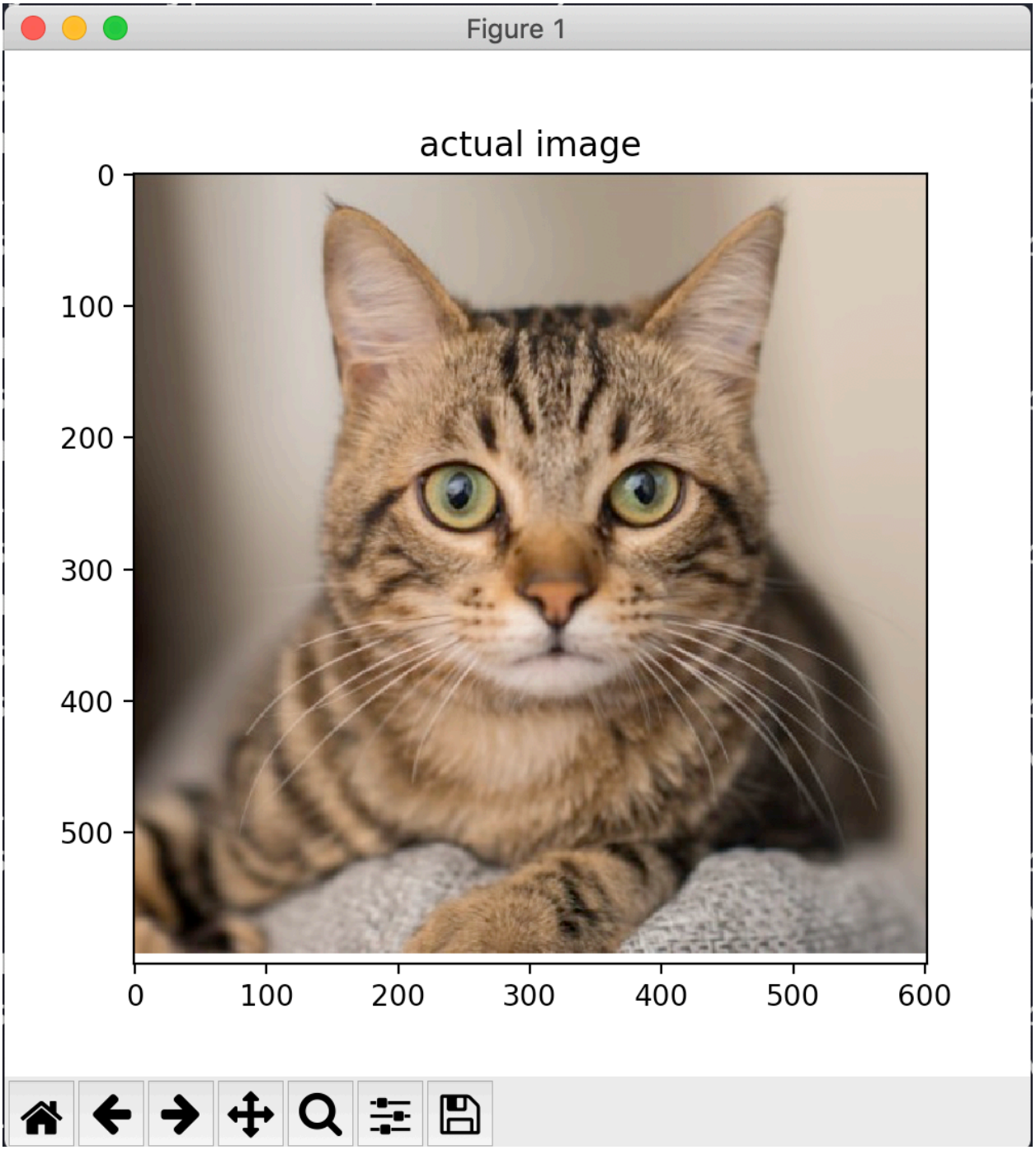
Problem #2

```
'''
Week 8 - Problem #2
Cat image - auto-encoder
'''

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn import metrics
from PIL import Image
from numpy.random import seed
from tensorflow import set_random_seed

def show_image(img, title):
    fig, aux = plt.subplots(figsize=(5,5))
    aux.imshow(img, cmap='gray')
    plt.title(title)
    plt.show()
    plt.waitforbuttonpress()

def read_image():
    img = Image.open('cat.jpg')
    show_image(img, 'actual image')

    # Resize the image
    img.load()
    img = img.resize((128, 128), Image.ANTIALIAS)
    img_array = np.asarray(img)
    img_array = img_array.flatten()
    img_array = np.array([img_array])
    img_array = img_array.astype(np.float32)
    print(img_array)
    return (img_array)
    #return img

def define_model(hidden_layers, image_data):
    # define the keras model
```

```
    seed(1)
    set_random_seed(2)
    model = Sequential()
    # Add a hidden layer
    model.add(Dense(hidden_layers, input_dim=image_data.shape[1],
activation='relu'))
    model.add(Dense(image_data.shape[1]))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.summary()
    return model

def train_and_predict_model(model, image_data):
    seed(1)
    set_random_seed(2)
    model.fit(image_data, image_data, verbose = 0, epochs=20)
    predict = model.predict(image_data)
    restored_img_data = predict[0].reshape(128, 128, 3)
    restored_img_data = restored_img_data.astype(np.uint8)
    restored_img = Image.fromarray(restored_img_data, 'RGB')
    return restored_img

# Get the image data
image_data = read_image()

# model with 4 input layers
model1 = define_model(4, image_data);
restored_img = train_and_predict_model(model1, image_data)
show_image(restored_img, 'restored image - 4 layers')

# model with 8 input layers
model2 = define_model(8, image_data);
restored_img = train_and_predict_model(model2, image_data)
show_image(restored_img, 'restored image - 8 layers')

# model with 12 input layers
model3 = define_model(12, image_data);
restored_img = train_and_predict_model(model3, image_data)
show_image(restored_img, 'restored image - 12 layers')

# model with 20 input layers
```

```
model4 = define_model(20, image_data);
restored_img = train_and_predict_model(model4, image_data)
show_image(restored_img, 'restored image - 20 layers')
```

These are the image plots with different number of layers with 20 epochs.



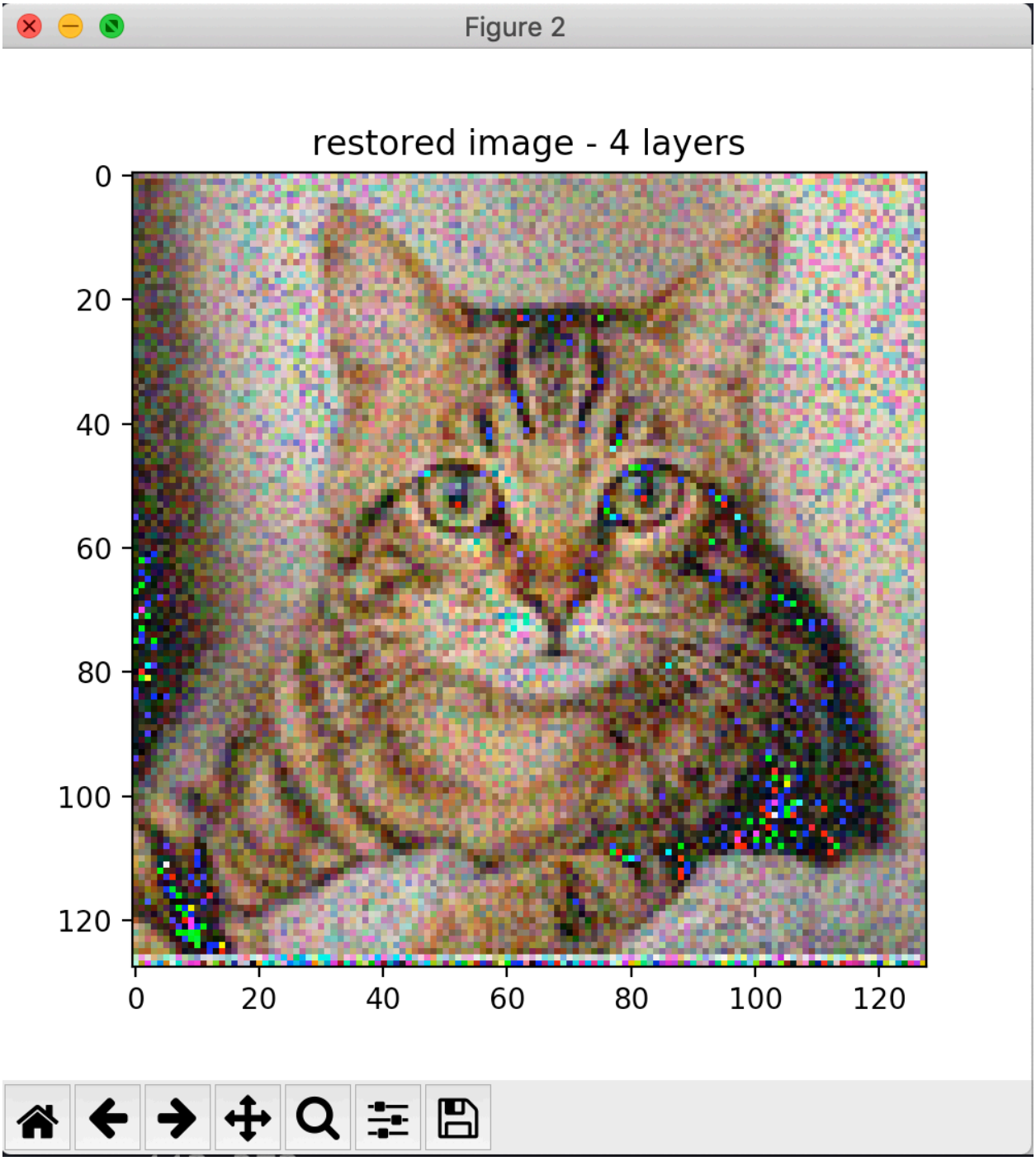


Figure 2
restored image - 4 layers
0
20
40
60
80
100
120
0
20
40
60
80
100
120

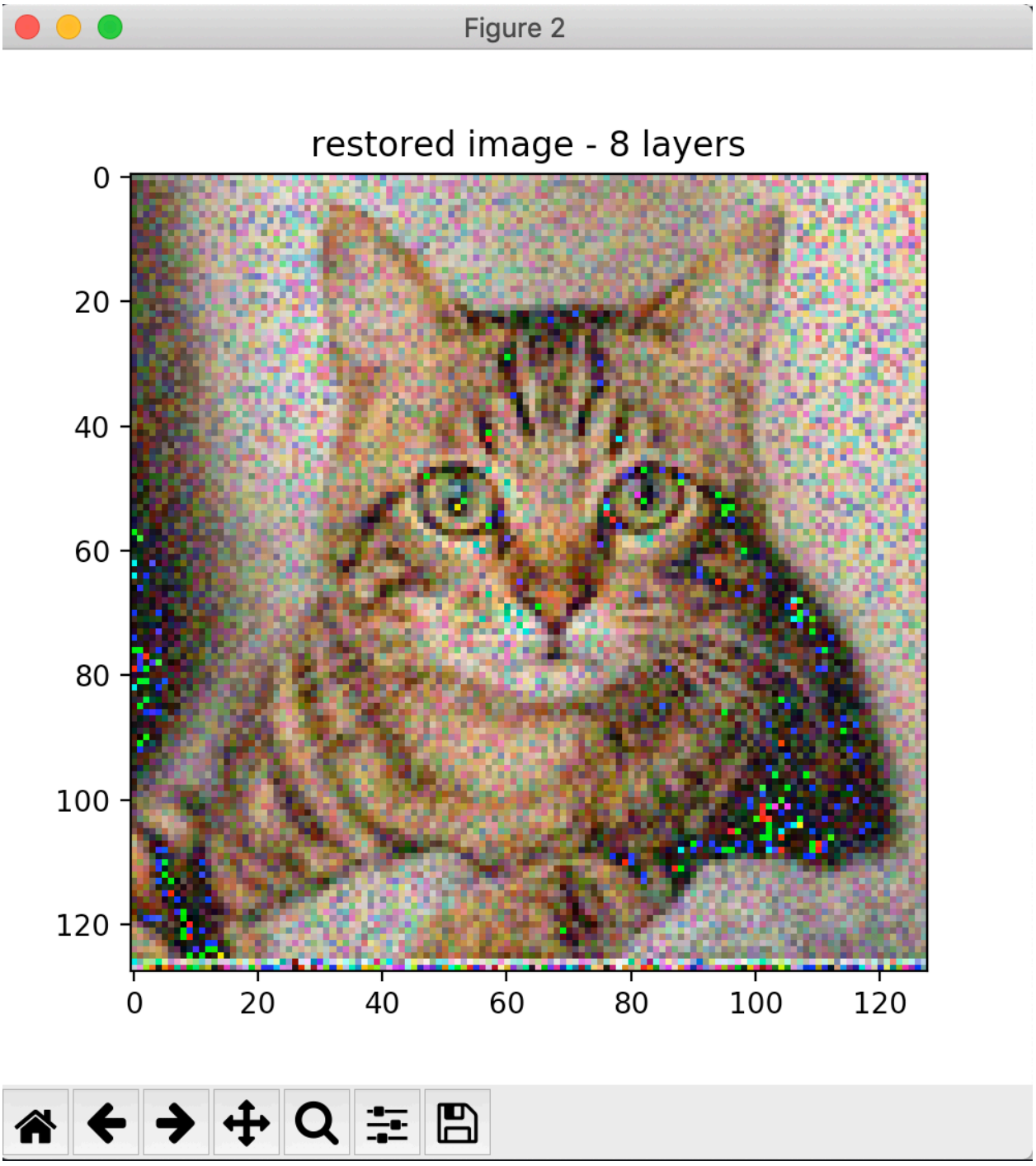


Figure 2
restored image - 8 layers
0
20
40
60
80
100
120
0
20
40
60
80
100
120

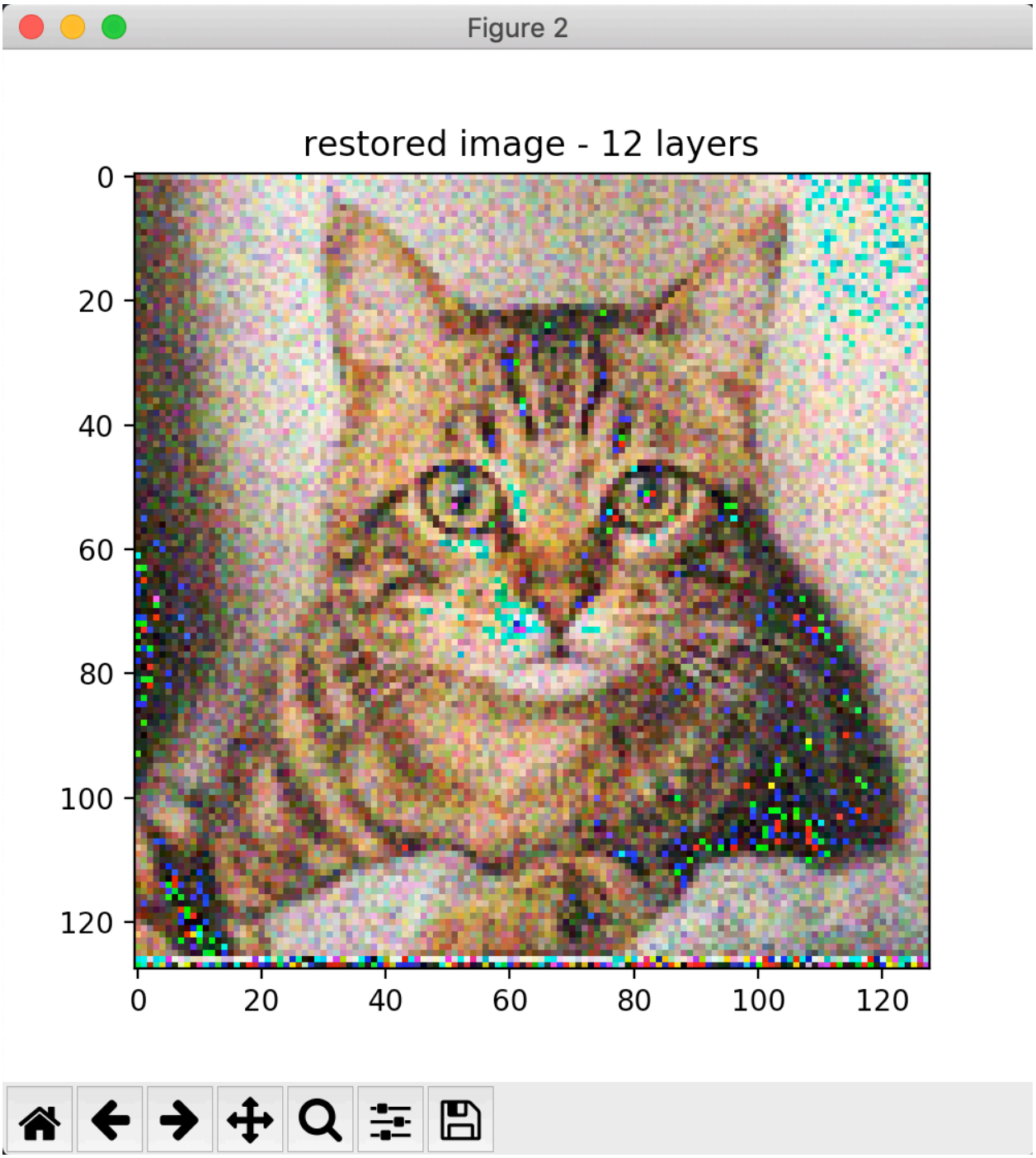


Figure 2
restored image - 12 layers
0
20
40
60
80
100
120
0
20
40
60
80
100
120

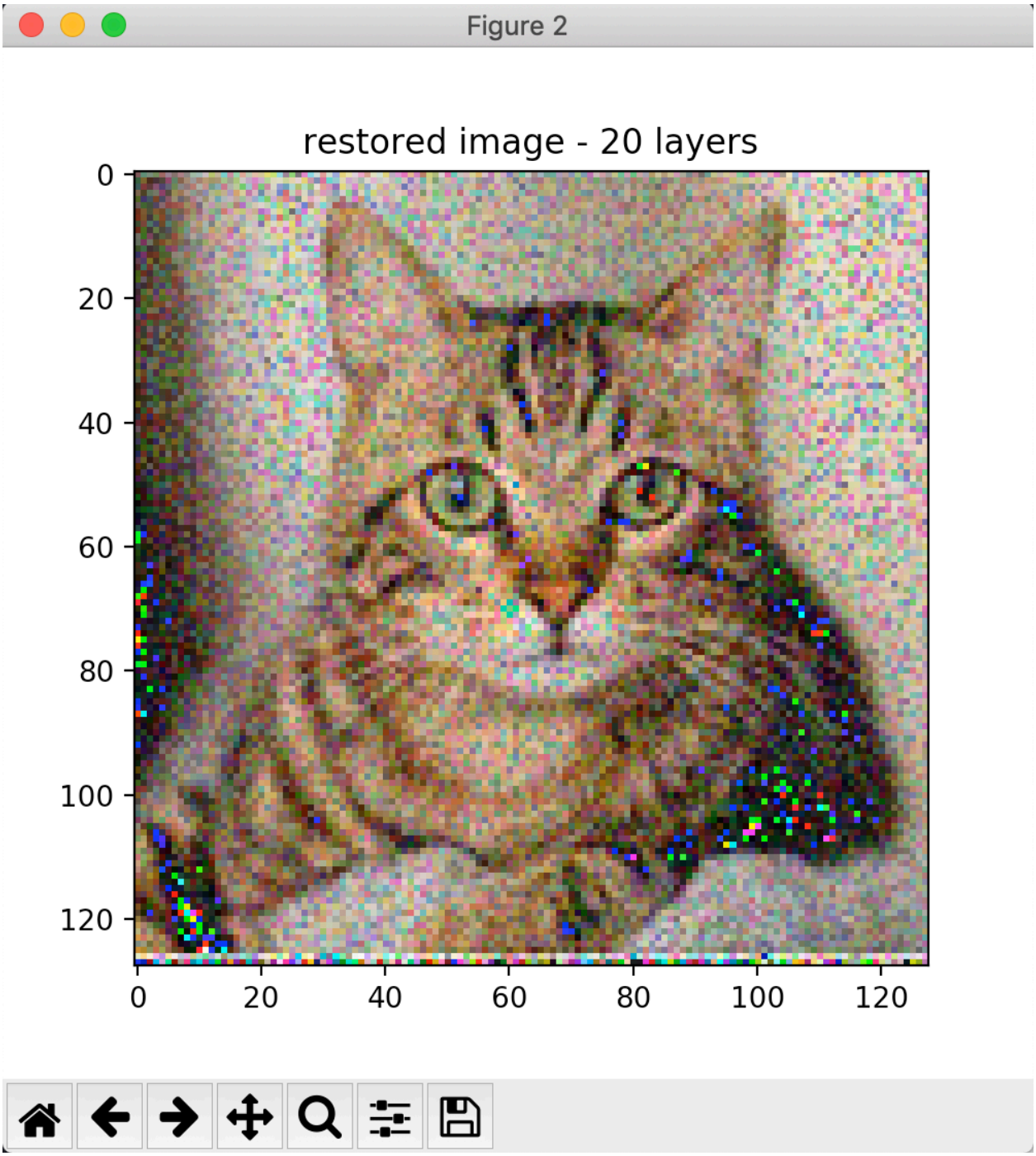
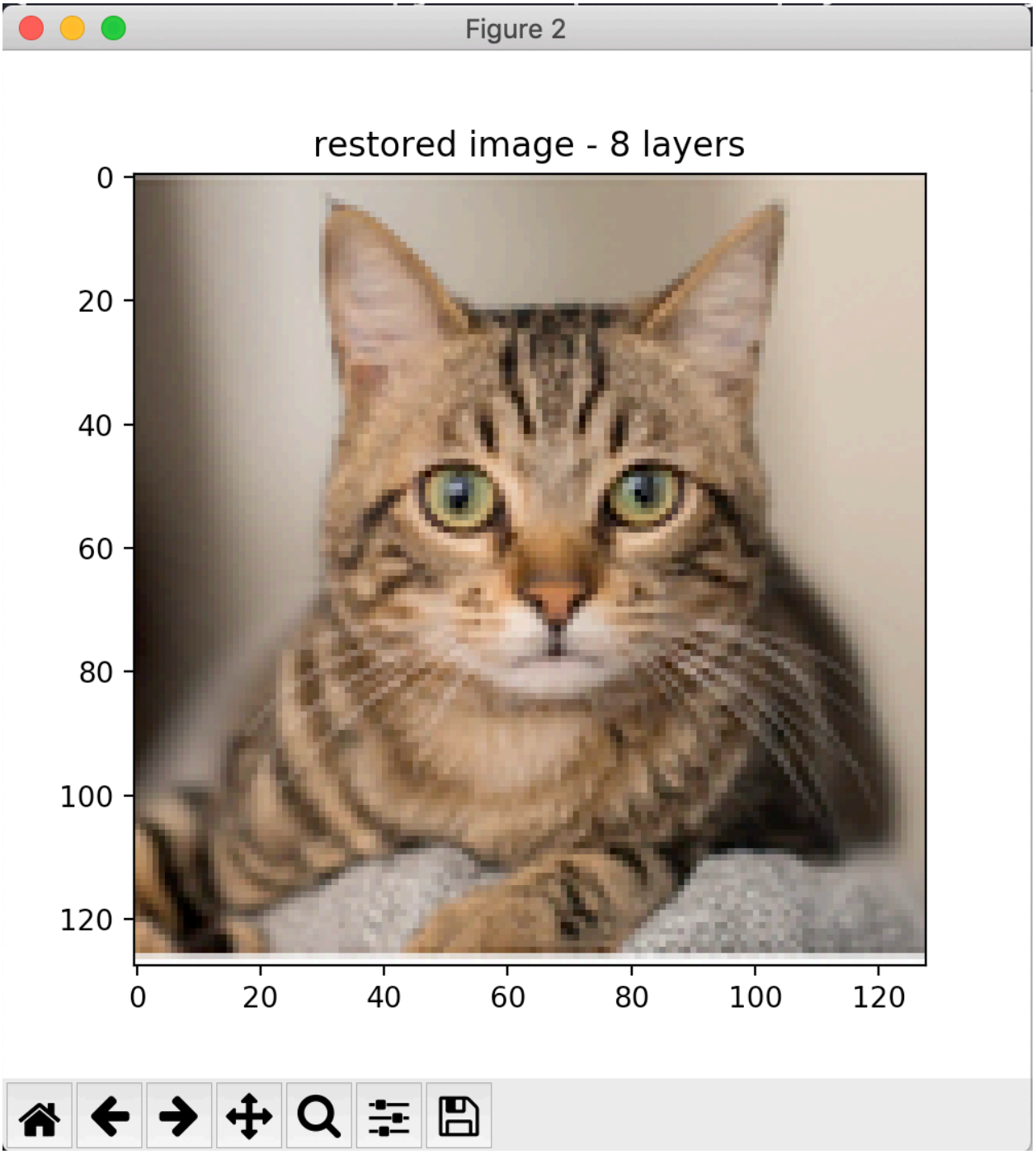


Figure 2
restored image - 20 layers
0
20
40
60
80
100
120
0
20
40
60
80
100
120

Now, there is a significant improvement when the number of epochs is increased to a few hundreds. An example here with 200 epochs and 8 layers.

As can be seen, there is a big difference with the same number of hidden layers with much fewer epochs.



Problem #3

```
'''
Week8 - Problem 3
Entropy, Cross entropy and KL divergence
'''

import numpy as np

def entropy(data):
    return np.sum(-data * np.log(data))

def x_entropy(data1, data2):
    return np.sum(-data1 * np.log(data2))

def kl_divergence(data1, data2):
    return np.sum(-data1 * np.log(data2/data1))

img1_dist = np.array([35, 36, 45, 95, 24, 15, 6]) / 256
img2_dist = np.array([20, 56, 85, 52, 22, 20, 1]) / 256

print('Entropy of img1 distribution: {}'.format(entropy(img1_dist)))
print('Entropy of img2 distribution: {}'.format(entropy(img2_dist)))
print('Cross Entropy of img1 w.r.t img2 distribution:
{}'.format(x_entropy(img1_dist, img2_dist)))
print('Cross Entropy of img2 w.r.t img1 distribution:
{}'.format(x_entropy(img2_dist, img1_dist)))
print('KL divergence of img1 w.r.t img2 distribution:
{}'.format(kl_divergence(img1_dist, img2_dist)))
print('KL divergence of img2 w.r.t img1 distribution:
{}'.format(kl_divergence(img2_dist, img1_dist)))
```

```
Entropy of img1 distribution: 1.697495481829104
Entropy of img2 distribution: 1.6532167032095546
Cross Entropy of img1 w.r.t img2 distribution: 1.857006533416811
Cross Entropy of img2 w.r.t img1 distribution: 1.8029044775422078
KL divergence of img1 w.r.t img2 distribution: 0.15951105158770723
KL divergence of img2 w.r.t img1 distribution: 0.1496877743326534
```

The output from the code matches the expected output in the problem statement.