main

September 11, 2024

[41]:
```python
from itertools import product
import torch
from torchvision.transforms import ToTensor
from torchvision import datasets
from pysr import PySRRegressor
from model import CNN
import pandas as pd
import numpy as np
from scipy.fftpack import dct, idct
import matplotlib.pyplot as plt
import seaborn as sb
```

0.0.1 Load Model and get Kernels

[2]:
```python
cnn = CNN()
cnn.load_state_dict(torch.load('cnn.pt'))
print(cnn)

for name, param in cnn.named_parameters():
    if name == 'conv1.weight':
        print(f"amount of kernels of Conv1: {param.shape}")
        kernels1 = param
    if name == 'conv2.weight':
        print(f"amount of kernels of Conv2: {param.shape}")
print(f"kernels of first layer:\n{kernels1}")
```

```
/tmp/ipykernel_9886/3040839533.py:2: FutureWarning: You are using `torch.load`
with `weights_only=False` (the current default value), which uses the default
pickle module implicitly. It is possible to construct malicious pickle data
which will execute arbitrary code during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
```

```
experimental feature.
  cnn.load_state_dict(torch.load('cnn.pt'))
```

```
CNN(
  (conv1): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (relu1): ReLU()
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (relu2): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (out): Linear(in_features=1568, out_features=10, bias=True)
)
amount of kernels of Conv1: torch.Size([16, 1, 5, 5])
amount of kernels of Conv2: torch.Size([32, 16, 5, 5])
kernels of first layer:
Parameter containing:
tensor([[[[-0.2335, -0.2440, -0.9097, -1.0703, -0.8168],
          [-0.0917,  0.0114, -0.2991, -0.3230,  0.6103],
          [ 0.0865,  0.0287,  0.0781,  0.3111,  0.9103],
          [ 0.0553, -0.7562, -0.8900, -0.9722, -0.9644],
          [ 0.0107, -0.6425, -0.8105, -0.5706, -1.1133]]],


        [[[-0.0280, -0.6446, -1.0783, -0.0510, -0.1909],
          [-1.2921, -1.5182, -0.0516,  0.5699, -0.5167],
          [-0.9574, -0.2885,  0.0691,  0.1755, -0.4727],
          [-1.2423, -0.3863,  0.5111,  0.0746, -0.8479],
          [-1.3129,  0.2300,  0.8279, -0.4460, -0.5626]]],


        [[[ 0.5291,  0.1248,  0.2948,  0.1704, -0.1348],
          [-1.0246,  0.1614,  0.4258,  0.0904,  0.0019],
          [-0.5680,  0.4559,  0.0938,  0.1415,  0.0510],
          [ 0.1021,  0.2176, -0.0204,  0.1813,  0.2024],
          [ 0.4457,  0.3654,  0.0859, -0.0848, -0.0403]]],


        [[[ 0.7099, -0.0518, -0.3999, -1.3199, -0.2031],
          [ 0.3740,  0.1910, -0.3168, -0.6762,  0.4298],
          [ 0.5073, -0.0837, -1.3188, -0.0196,  0.5805],
          [ 0.0269, -1.0321, -0.0367,  0.0597,  0.1610],
          [-0.8166, -0.5479,  0.0032,  0.4293, -0.1114]]],


        [[[ 0.2957,  0.1430,  0.0766, -0.2650,  0.5334],
          [ 0.1057, -0.6504, -0.8972, -0.5839, -0.4445],
  [ 0.2597,  0.0572,  0.0540,  0.1157, -0.3733],
  [ 0.3074,  0.4609,  0.4423, -0.2986, -0.2100],
  [-0.4084, -0.1961,  0.4710,  0.2451,  0.4294]]],


[[[-0.2075, -0.1561,  0.2745,  0.2814, -0.6771],
  [-1.2981, -0.1775,  0.1404,  0.5747, -0.1818],
  [-0.5521, -0.6337, -0.1747,  0.3559,  0.3817],
  [-0.6198, -0.8836, -0.2142,  0.1903,  0.2794],
  [-0.1596, -0.7851,  0.1739,  0.0362,  0.1972]]],


[[[ 0.1360, -0.0918,  0.1104, -0.0646,  0.2103],
  [ 0.1062,  0.3018,  0.4091,  0.5279,  0.1627],
  [ 0.0302, -0.3303, -0.0802,  0.2925,  0.2513],
  [-0.6473, -1.4883, -0.8111, -0.0514, -0.3394],
  [ 0.5227, -0.2562, -0.3381, -0.7960, -0.1770]]],


[[[-0.3574,  0.1063,  0.2514,  0.1884, -0.4400],
  [-0.6735, -0.4578, -0.8785, -1.0236, -1.5146],
  [-1.4320, -1.0363, -0.8359, -0.6239, -0.6141],
  [-0.9241, -0.3779, -0.0595,  0.0761,  0.2091],
  [ 0.0723,  0.3981,  0.4368,  0.3846,  0.2235]]],


[[[-1.0633, -0.1753,  0.4940,  0.4017,  0.7569],
  [-0.0771, -1.0514, -1.7260, -1.4314, -0.5460],
  [-0.0308, -0.0643,  0.0095, -0.2382, -0.7718],
  [-0.1070,  0.5910,  0.3556, -0.4146, -0.3264],
  [-0.2768,  0.2324,  0.4141, -0.0533, -0.1740]]],


[[[-0.3569, -0.0474, -0.0295, -0.3177,  0.0279],
  [-0.4427, -0.2376,  0.1075,  0.0865,  0.0894],
  [-0.5682, -0.0398,  0.0419, -0.1239, -0.2208],
  [-0.2371, -0.1306, -0.2446, -0.3164, -0.2015],
  [ 0.0484, -0.4290, -0.4255, -0.3033, -0.3477]]],


[[[ 0.2488,  0.4134,  0.3528,  0.0210, -1.0239],
  [-0.0268,  0.0338,  0.1708, -0.0692, -1.6336],
  [-0.1032,  0.0310,  0.3139, -0.8555, -0.1318],
  [ 0.0948, -0.3197, -1.3580, -0.0030, -0.1405],
  [-1.3413, -1.7017, -0.4596, -0.2455, -0.7042]]],


[[[-0.2504, -0.3598, -0.1199,  0.2551, -0.5003],
         [ 0.1365, -0.3004, -0.3045,  0.1119, -0.1684],
         [ 0.1980, -0.0510, -0.2116,  0.0321, -0.1766],
         [-0.1648, -0.6461, -0.5829, -0.1265,  0.0871],
         [ 0.1124, -0.4109, -0.9819, -0.1864,  0.4388]]],


        [[[-0.5147,  0.1186,  0.5365, -0.5049, -0.9201],
          [-0.0189, -0.0284, -0.9864, -0.2854, -0.1827],
          [-1.3057, -0.6926, -0.6505,  0.2068,  0.3300],
          [-1.3735,  0.2037,  0.2468,  0.3166,  0.0710],
          [-0.5940, -0.6309, -0.4049,  0.7009,  0.1724]]],


        [[[ 0.3788,  0.0794, -0.4265, -0.1576,  0.0312],
          [-0.1865, -0.9333, -0.6402,  0.2178,  0.3531],
          [-0.3694, -0.1459,  0.6929,  0.6214,  0.2413],
          [ 0.2096,  0.3032,  0.4789, -0.2633, -0.4940],
          [ 0.1812, -0.2943, -0.5340, -1.2313,  0.1955]]],


        [[[ 0.1814, -0.2550, -0.3302, -0.2672, -0.0566],
          [-0.5218, -0.0665, -0.1889, -0.1543,  0.1820],
          [-0.4794, -0.0881, -0.5455,  0.1927, -0.2243],
          [-0.1688, -0.1002, -0.1748,  0.1656, -0.0872],
          [-0.4696, -0.3445, -0.0150,  0.0150,  0.0429]]],


        [[[-0.0110,  0.0712,  0.1751,  0.1512, -0.7895],
          [-0.1340,  0.0589,  0.2575, -0.0446, -1.0411],
          [-0.2241, -0.0130,  0.5968,  0.0430, -0.3906],
          [-0.9658, -0.3102,  0.5802,  0.0912, -0.0806],
          [-0.6764, -0.1164, -0.1150,  0.1443, -0.1137]]]], requires_grad=True)
```

0.0.2 Load Dataset and get results

[3]:
```python
test_data = datasets.MNIST(root='data', train=False, transform=ToTensor(),)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=20,␣
 ↪shuffle=True, num_workers=1)
samples, labels = next(iter(test_loader))

cnn.eval()
with torch.no_grad():
    results = cnn(samples)
```

0.0.3 Prepare Data for PySR and extract features

Extract the 5x5 submatrices (incl. padding) from images to use them as input

```
[4]: # Take every image and split it into 5x5 submatrices => np.array.shape = (7840,␣
     ↪5, 5)
     # 7840 <- 28 * 28 patches per image * 10 images (batch_size)
     kernel_size = 5
     X = None
     for x in samples:
         x = torch.nn.functional.pad(input=x[0], pad=(2, 2, 2, 2), mode="constant",␣
      ↪value=0)
         for i, j in np.ndindex((x.size()[0] - kernel_size + 1, x.size()[1] -␣
      ↪kernel_size + 1)):
             slice = x[i:i + kernel_size, j:j + kernel_size]
             if X is None:
                 # X = np.array([slice.numpy().flatten()])
                 X = np.array([slice.numpy()])
             else:
                 # X = np.concatenate((X, [slice.numpy().flatten()]))
                 X = np.concatenate((X, [slice.numpy()]))

     print(X.shape)

     # Get the result for every 5x5 submatrix for each kernel => np.array.shape =␣
      ↪(16, 7840)
     # 16 <- amount of kernels in the first layer
     y = results['relu1'].numpy().transpose(1, 0, 2, 3).reshape(16, X.shape[0])
     print(y.shape)
```

```
(15680, 5, 5)
(16, 15680)
```

Discrete Cosine Transform (DCT, Fourier Transform) Get DCT features of image, only keep low frequencies.

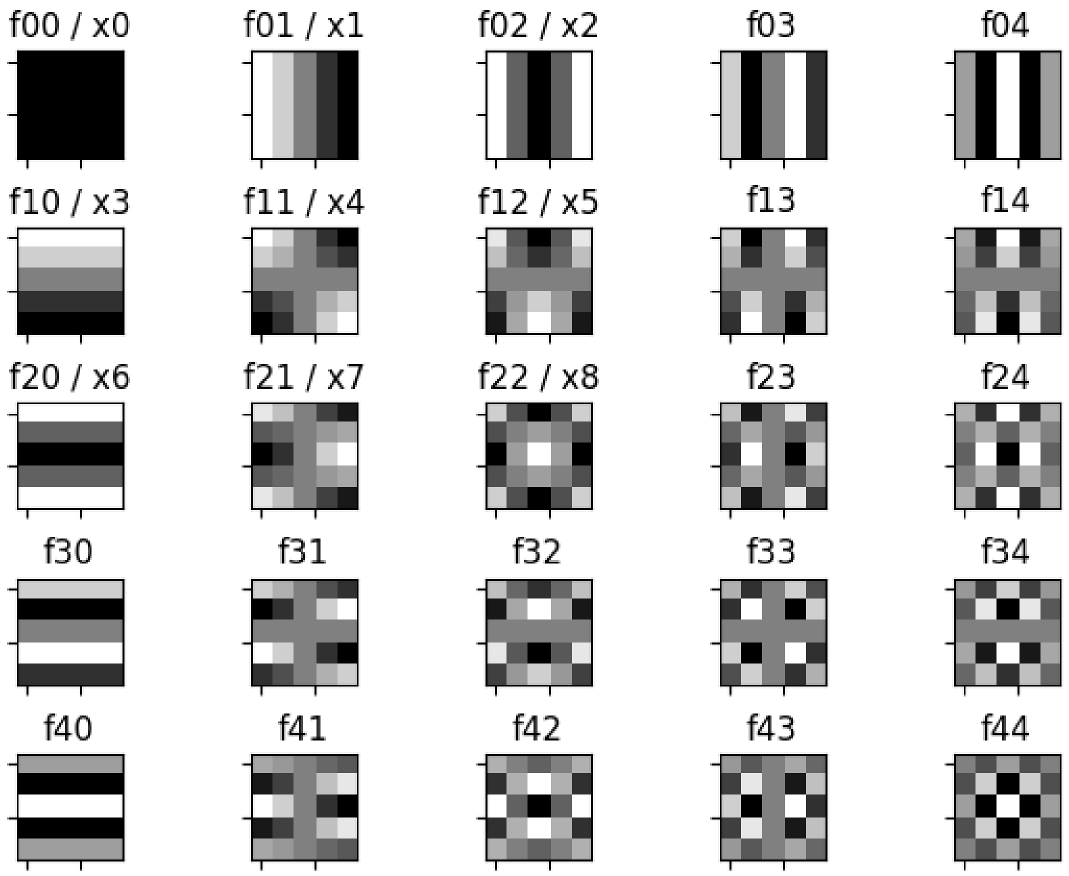
```
[5]: def dct2(m):
         return dct(dct(m.T, norm='ortho').T, norm='ortho')

     Xdct = np.array(list(map(dct2, X)))
     dct_filter_param = 3
     Xdct_filtered = np.array(list(map(lambda m: m[:dct_filter_param, :
      ↪dct_filter_param], Xdct)))
     print(Xdct_filtered.shape)
```

```
(15680, 3, 3)
```

Figures of corresponding frequencies

```
[134]: fig, axs = plt.subplots(5,5)
       fig.tight_layout(h_pad=0.5, w_pad=0)
       for i, j in list(product(range(5), range(5))):
```

```
    img = np.zeros((5, 5))
    img[i, j] = 1
    axs[i, j].imshow(idct(idct(img.T, norm='ortho').T, norm='ortho'),␣
↪cmap='gray')
    axs[i, j].set_yticklabels([])
    axs[i, j].set_xticklabels([])
    if i < 3 and j < 3:
        axs[i, j].set_title(f"f{i}{j} / x{3*i + j}")
    else:
        axs[i, j].set_title(f"f{i}{j}")
```



0.0.4 Symbolic Regression

[ ]:
```
def newPySRRegressor():
    return PySRRegressor(
        niterations=40,
        binary_operators=["+", "*", "-", "/"],
        unary_operators=[
            "cos",
            "exp",
            "sin",
            "square",
            "cube",
            "inv(x) = 1/x",  # Julia syntax
        ],
        extra_sympy_mappings={"inv": lambda x: 1 / x},  # Sympy syntax
        elementwise_loss="loss(prediction, target) = (prediction - target)^2", ␣
↪# Julia syntax
        warm_start=False,
        verbosity=0,
        temp_equation_file=True,
    )
```

Over the first kernel

```
[ ]: regr_first_kernel = newPySRRegressor()
     regr_first_kernel.fit(Xdct_filtered.reshape(X.shape[0], dct_filter_param**2),␣
      ↪y[0])  # Input data coded for position and summed
```

```
[ ]: print(regr_first_kernel.equations_['lambda_format'])
     f = regr_first_kernel.equations_['lambda_format'][4]
     print(f(np.array(range(9))))
```

Over all 16 Kernels

```
[ ]: regr_all_kernels = pd.DataFrame()
     regr_all_kernels.index.names = ['complexity']
     for i in range(16):
         regr = newPySRRegressor()
         regr.fit(Xdct_filtered.reshape((X.shape[0], dct_filter_param**2)), y[i])  #␣
      ↪Input data as is
         # print(regr.equations_)
         regr_all_kernels.insert(loc=i, column=f'Kernel {i}', value=regr.
      ↪equations_['equation'])
         print(f"Done with Kernel {i} | {(i+1)/16 * 100}%")

     print(regr_all_kernels)
```

```
[ ]: regr_all_kernels.to_csv('results_dct_filter3_allkernels.csv')
```

Over the first kernel multiple times (Stability check)

```
[ ]: regr_stability = pd.DataFrame()
     regr_stability.index.names = ['complexity']
     for i in range(10):
         regr = newPySRRegressor()
         regr.fit(Xdct_filtered.reshape((X.shape[0], dct_filter_param**2)), y[0])
         # print(regr.equations_)
```

```
    regr_stability.insert(loc=i, column=f'Iteration {i}', value=regr.
↪equations_['equation'])
    # print(regr_stability)
    print(f"Done with Iteration {i} | {(i+1)/10 * 100}%")
```

[ ]:
```
regr_stability.to_csv('results_dct_filter3_stability.csv')
```

0.0.5 Analysis of the results

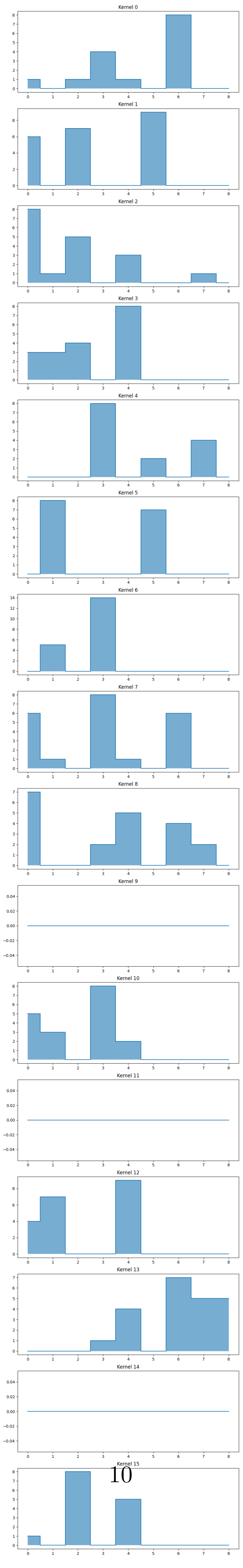
Distribution of variables in functions First, see how often each variable occurs in the functions for all complexity levels in each kernel.
Then look at the overall distribution of variable usage.

[72]:
```
# Count variable occurance for each kernel
df_res_all_kernels = pd.read_csv('results_dct_filter3_allkernels.csv',␣
↪index_col='complexity').fillna("")
variables = dct_filter_param**2 + 0

df_variable_occurences_all = pd.DataFrame()
df_variable_occurences_all.index.names = ['Kernel']
summed_all = df_res_all_kernels.sum()
for i in range(variables):
    df_variable_occurences_all.insert(loc=i, column=f"x{i}", value=[s.
↪count(f"x{i}") for s in summed_all])
display(df_variable_occurences_all)
```

```
        x0  x1  x2  x3  x4  x5  x6  x7  x8
Kernel
0        1   0   1   4   1   0   8   0   0
1        6   0   7   0   0   9   0   0   0
2        8   1   5   0   3   0   0   1   0
3        3   3   4   0   8   0   0   0   0
4        0   0   0   8   0   2   0   4   0
5        0   8   0   0   0   7   0   0   0
6        0   5   0  14   0   0   0   0   0
7        6   1   0   8   1   0   6   0   0
8        7   0   0   2   5   0   4   2   0
9        0   0   0   0   0   0   0   0   0
10       5   3   0   8   2   0   0   0   0
11       0   0   0   0   0   0   0   0   0
12       4   7   0   0   9   0   0   0   0
13       0   0   0   1   4   0   7   5   5
14       0   0   0   0   0   0   0   0   0
15       1   0   8   0   5   0   0   0   0
```

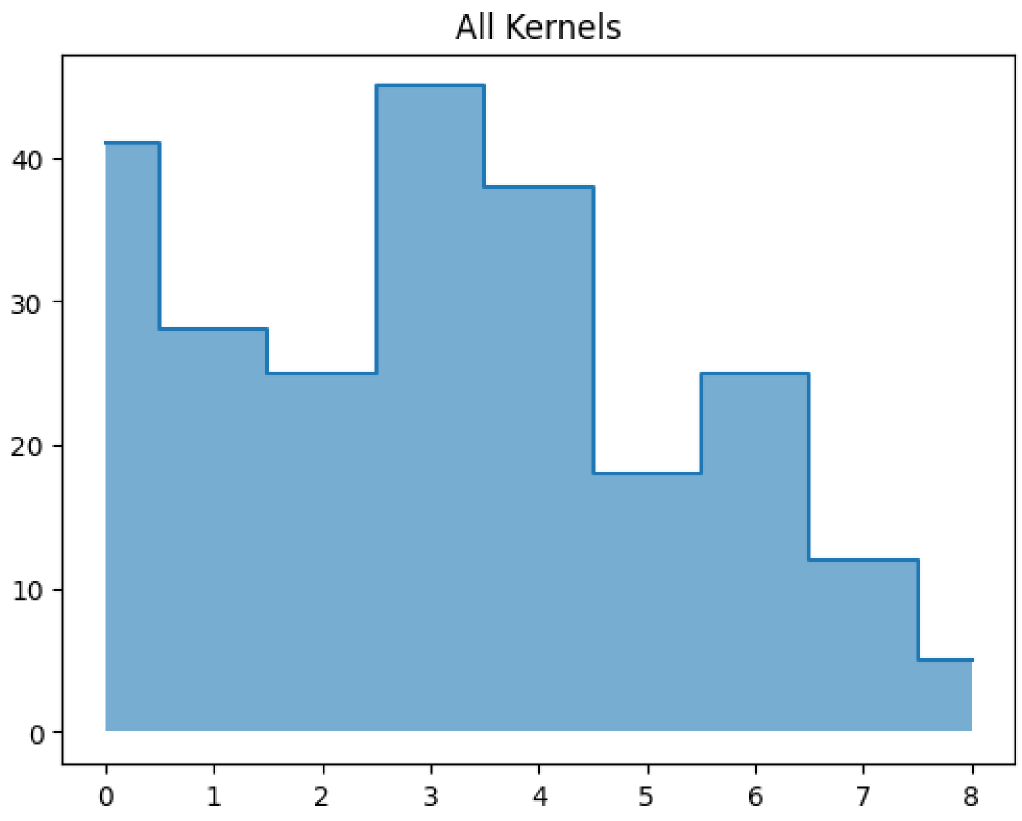
[119]:
```
# Plot the occurances for each kernel
fig_all, axs_all = plt.subplots(df_variable_occurences_all.shape[0],␣
↪figsize=(10, 70))
```

```python
for i, ax in enumerate(axs_all):
    ax.step(range(variables), df_variable_occurences_all.loc[i], where='mid')
    ax.fill_between(range(variables), df_variable_occurences_all.loc[i],␣
 ↪step="mid", alpha=0.6)
    ax.set(title=f"Kernel {i}")#, xlim=[0,variables-1], ylim=[0,20])
```


Kernel 0
Kernel 1
Kernel 2
Kernel 3
Kernel 4
Kernel 5
Kernel 6
Kernel 7
Kernel 8
Kernel 9
Kernel 10
Kernel 11
Kernel 12
Kernel 13
Kernel 14
Kernel 15

```
[117]: # Plot the occurances over all kernels
       plt.step(range(variables), df_variable_occurences_all.sum(), where='mid')
       plt.fill_between(range(variables), df_variable_occurences_all.sum(),␣
        ↪step='mid', alpha=0.6)
       plt.title("All Kernels")
```

```
[117]: Text(0.5, 1.0, 'All Kernels')
```



Stability on first Kernel See if throughout different iterations the same variable is important for Kernel 1.

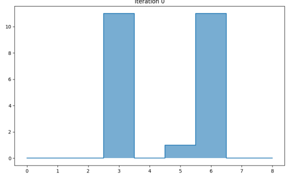
```
[120]: # Count variable occurance for each iteration
       df_res_stability = pd.read_csv('results_dct_filter3_stability.csv',␣
        ↪index_col='complexity').fillna("")
       variables = dct_filter_param**2 + 0

       df_variable_occurences_stab = pd.DataFrame()
       df_variable_occurences_stab.index.names = ['Iteration']
```

```
summed_stab = df_res_stability.sum()
for i in range(variables):
    df_variable_occurences_stab.insert(loc=i, column=f"x{i}", value=[s.
 ↪count(f"x{i}") for s in summed_stab])
display(df_variable_occurences_stab)
```

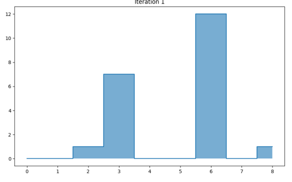
```
           x0  x1  x2  x3  x4  x5  x6  x7  x8
Iteration
0           0   0   0  11   0   1  11   0   0
1           0   0   1   7   0   0  12   0   1
2           3   0   0   5   3   2  12   0   0
3           0   0   0   3   1   0   8   0   0
4           0   0   0   6   0   0  10   0   0
5           0   0   0  18   0   0  11   0   1
6           0   0   0   8   0   0  10   0   0
7           0   1   0   5   0   0  10   0   0
8           0   0   0   8   0   0  12   0   0
9           0   2   0   3   0   2   7   0   0
```

```
[121]: # Plot the occurances for each kernel
fig_stab, axs_stab = plt.subplots(df_variable_occurences_stab.shape[0],␣
 ↪figsize=(10, 70))
for i, ax in enumerate(axs_stab):
    ax.step(range(variables), df_variable_occurences_stab.loc[i], where='mid')
    ax.fill_between(range(variables), df_variable_occurences_stab.loc[i],␣
 ↪step="mid", alpha=0.6)
    ax.set(title=f"Iteration {i}")#, xlim=[0,variables-1], ylim=[0,20])
```

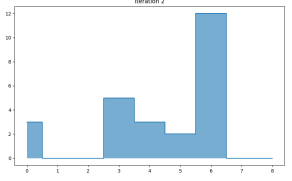
Iteration 0



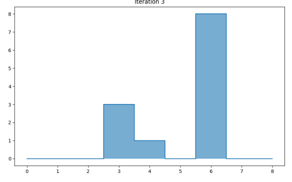
Iteration 1



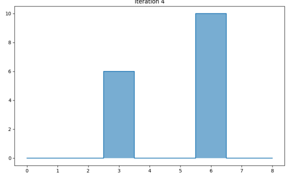
Iteration 2



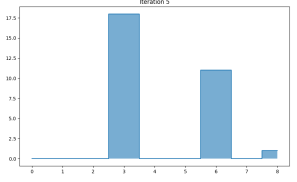
Iteration 3



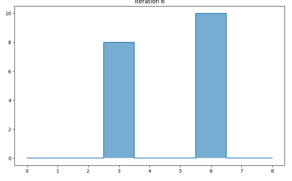
Iteration 4



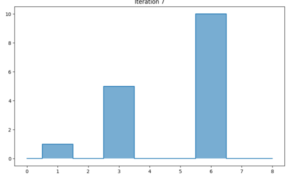
Iteration 5



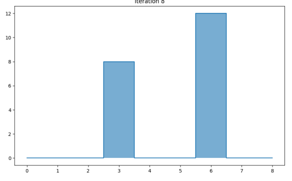
Iteration 6



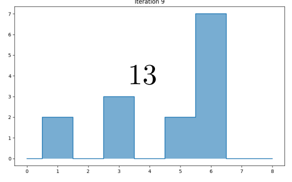
Iteration 7



Iteration 8

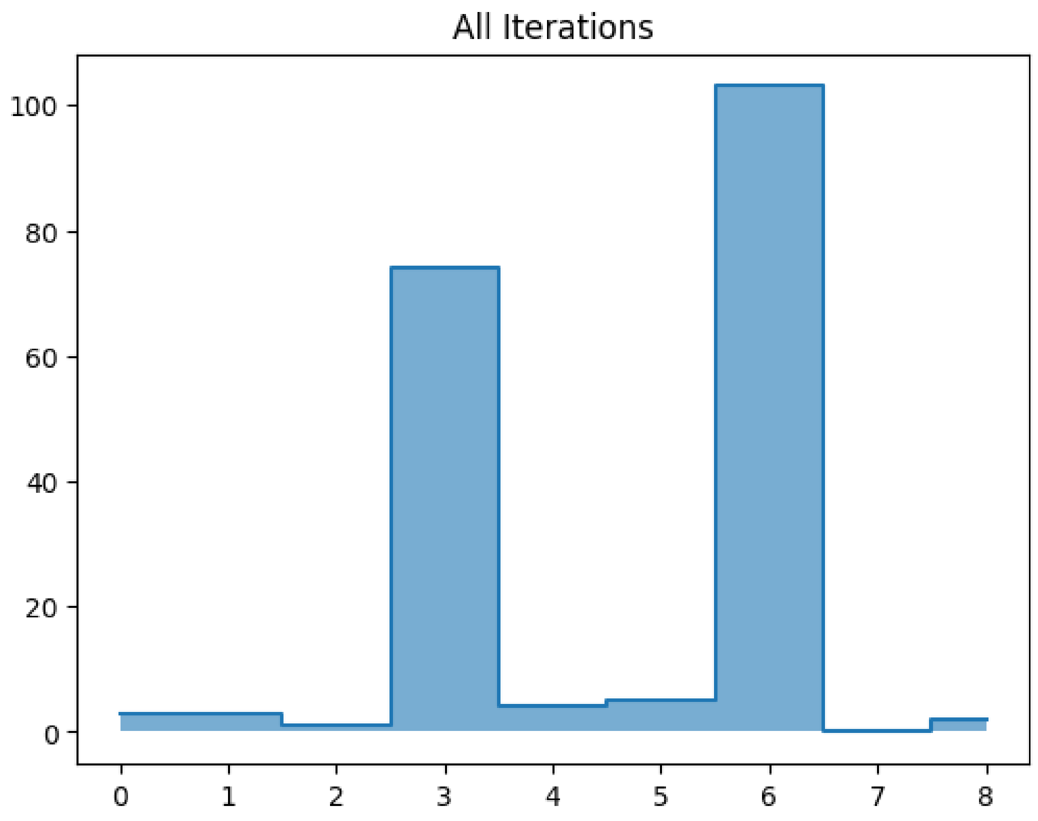


Iteration 9



[122]:
```
# Plot the occurances over all kernels
plt.step(range(variables), df_variable_occurences_stab.sum(), where='mid')
plt.fill_between(range(variables), df_variable_occurences_stab.sum(),␣
 ↪step='mid', alpha=0.6)
plt.title("All Iterations")
```

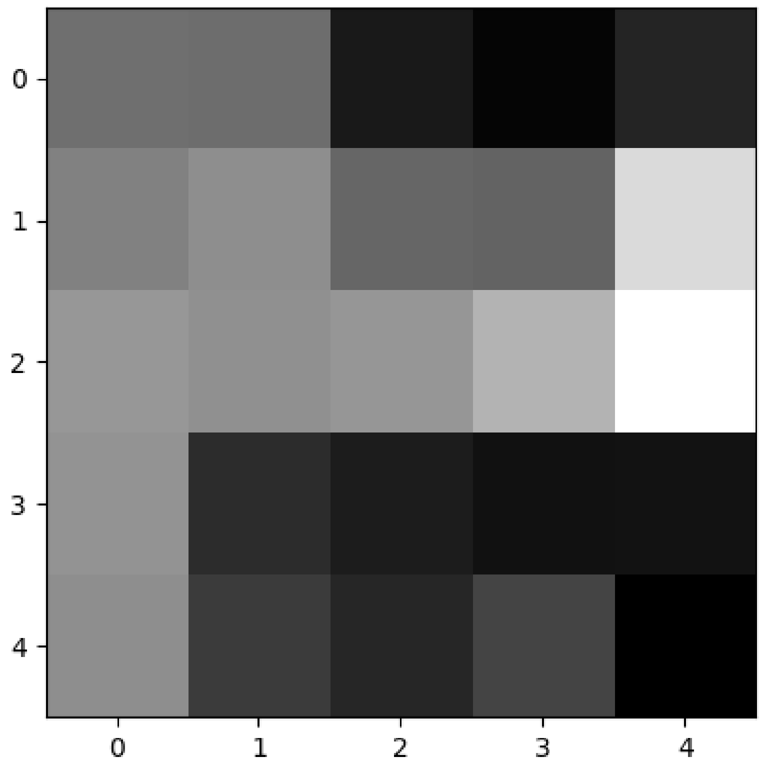
[122]:
```
Text(0.5, 1.0, 'All Iterations')
```

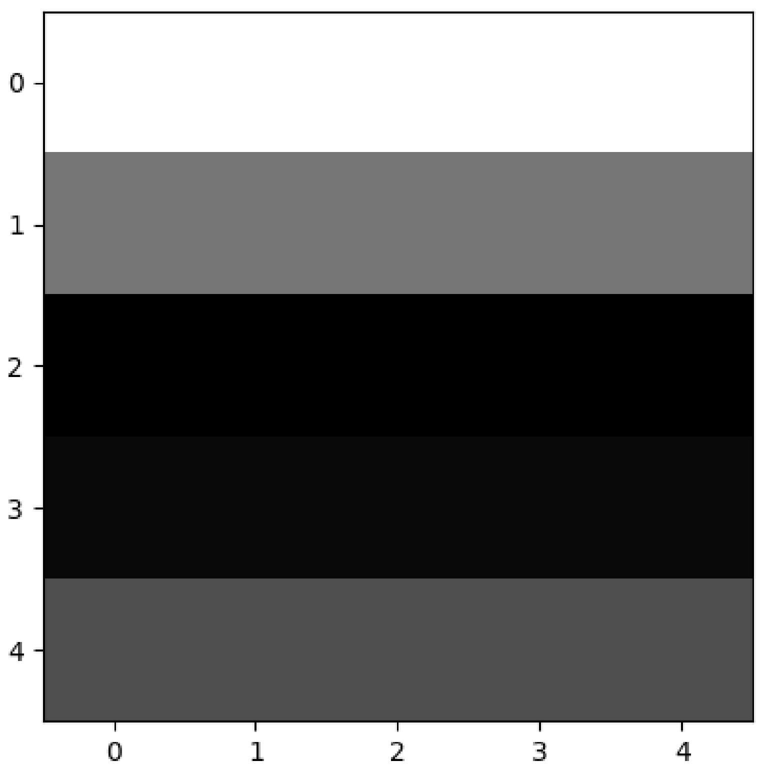


Kernel 1 seems to pay special attention to x3 and x6, which are the horizontal frequencies. Let's see if something like this can be recognized in the kernel…

[141]:
```
plt.figure(0)
plt.imshow(kernels1[0].detach().numpy()[0], cmap='gray')

x36 = np.zeros((5, 5))
x36[1, 0] = 1
x36[2, 0] = 1
plt.figure(1)
plt.imshow(idct(idct(x36.T, norm='ortho').T, norm='ortho'), cmap='gray')
```

```
[141]: <matplotlib.image.AxesImage at 0x7d92d19ff0d0>
```




0
1
2
3
4
0
1
2
3
4