Experimento 3

[1]:
```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transform
import torch.nn.functional as F

import matplotlib.pyplot as plt
import numpy as np
import math

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
#device='cpu'
print(device)

batch_size = 32

img_transform = transform.Compose([transform.ToTensor(), transform.Normalize((0.
 ↪5,),(0.5,))])

data = torchvision.datasets.ImageFolder("C:/Users/matia/Desktop/AI_UTEC/
 ↪proyecto5/fotos",transform=img_transform)

print(len(data))


train_set,test_set=torch.utils.data.random_split(data,[14815,6350],␣
 ↪generator=torch.Generator().manual_seed(0))
val_set,test_set=torch.utils.data.random_split(test_set,[4233,2117],␣
 ↪generator=torch.Generator().manual_seed(0))

img, _ = train_set[0]
print(img.shape)

train_loader = torch.utils.data.DataLoader(dataset=train_set,␣
 ↪batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_set,␣
 ↪batch_size=batch_size, shuffle=False)
```

```
val_loader = torch.utils.data.DataLoader(dataset=val_set, batch_size=batch_size,␣
 ↪shuffle=False)
```

```
cuda:0
21165
torch.Size([3, 299, 299])
```

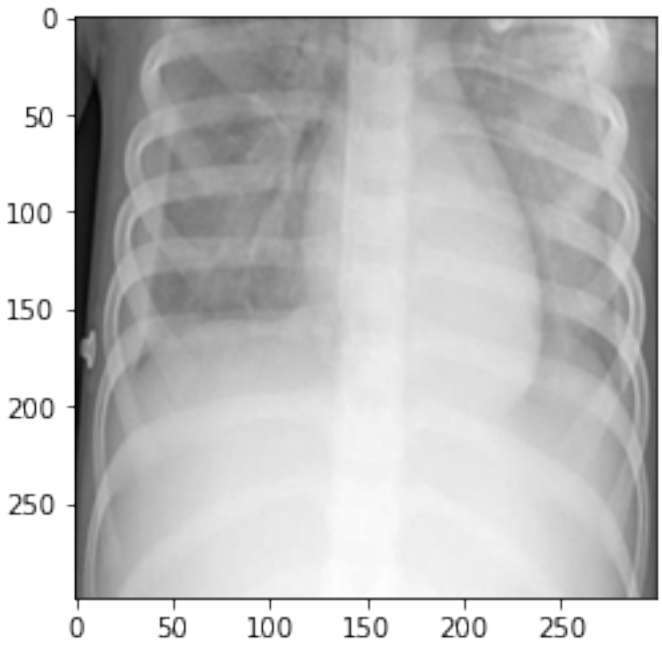
[2]:
```
def show_img(img):
    plt.imshow(img.numpy()[0], cmap='gray')
```

[3]:
```
print(len(train_set))
print(len(test_set))
print(len(val_set))
```

```
14815
2117
4233
```

[4]:
```
img, label = train_set[999]
print(label)
show_img(img)
```

```
3
```



[5]:
```
#hyperparametros
num_classes = 4
learning_rate =  0.01
```

```
num_epochs = 20

class CNN(nn.Module):
    def __init__(self, num_classes=4):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=10,␣
 ↪stride=1,padding=0),
            nn.ReLU(),nn.BatchNorm2d(16))
        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=16,out_channels=32, kernel_size=6, stride=1,␣
 ↪padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1,␣
 ↪padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer4 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1,␣
 ↪padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(37*37*128, num_classes)


    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out
```

[6]:
```
model         = CNN(num_classes).to(device)
loss_fn       = nn.CrossEntropyLoss()
optimizer     = torch.optim.Adam(model.parameters(), lr = learning_rate)
#loss_train    = train(model, optimizer, loss_fn, num_epochs)
#test(model)

print([ e.shape  for e in model.fc.parameters()])

model.fc.weight
```

```
[torch.Size([4, 175232]), torch.Size([4])]
```

[6]:
```
Parameter containing:
tensor([[-9.4709e-04,  2.1630e-03, -1.2115e-03,  ..., -1.5297e-03,
         -1.3633e-03,  1.4940e-03],
        [ 1.4682e-03,  2.1715e-03, -1.7284e-04,  ..., -2.1634e-03,
         -2.1812e-04,  1.9144e-03],
        [-1.7999e-03,  8.1053e-04, -9.3822e-05,  ..., -2.1619e-03,
         -1.8192e-03, -2.1854e-03],
        [ 1.0879e-03,  6.7307e-04, -1.9577e-03,  ...,  1.1216e-03,
          2.2792e-03, -1.4664e-03]], device='cuda:0', requires_grad=True)
```

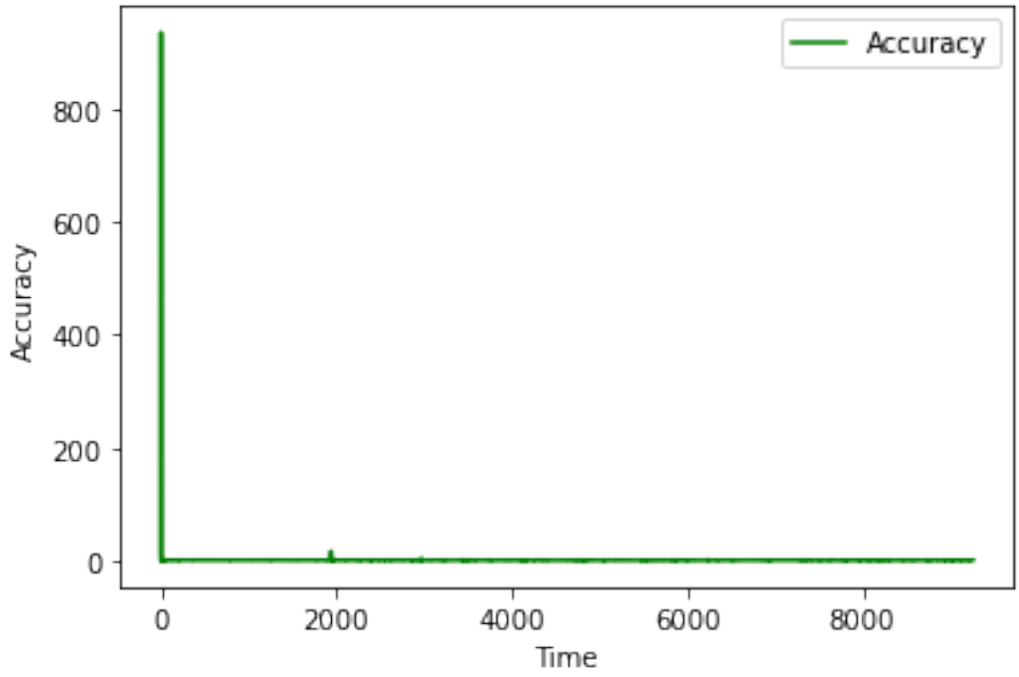
[7]:
```python
def train(model, optimizer, loos_fn, num_epochs):
    loss_vals = []
    running_loss =0.0
    # train the model
    total_step = len(train_loader)

    list_loss= []
    list_time = []
    j=0

    for epoch in range(num_epochs):
        for i, (images, labels) in enumerate(train_loader):
            images = images.to(device)
            labels = labels.to(device)
            # forward
            output = model(images)
            loss   = loss_fn(output, labels)
            # change the params
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            list_loss.append(loss.item())
            list_time.append(j)
            j+=1
            #print(i,end=",")
            if (i+1) % 100 == 0:
                #print()
                print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}' .
 ↪format(epoch+1, num_epochs, i+1, total_step, loss.item()))

    print('Finished Training Trainset')
    return list_loss,list_time
```

[8]:
```python
list_loss,list_time=train(model,optimizer,loss_fn,num_epochs)

plt.plot(list_time,list_loss,color="green", label="Accuracy")
```

```
plt.legend()
plt.xlabel("Time")
plt.ylabel("Accuracy")
plt.show()
```

```
Epoch [1/20], Step [100/463], Loss: 1.2155
Epoch [1/20], Step [200/463], Loss: 1.2081
Epoch [1/20], Step [300/463], Loss: 1.1235
Epoch [1/20], Step [400/463], Loss: 1.0604
Epoch [2/20], Step [100/463], Loss: 1.2279
Epoch [2/20], Step [200/463], Loss: 1.1592
Epoch [2/20], Step [300/463], Loss: 1.2901
Epoch [2/20], Step [400/463], Loss: 1.1010
Epoch [3/20], Step [100/463], Loss: 1.1965
Epoch [3/20], Step [200/463], Loss: 1.1855
Epoch [3/20], Step [300/463], Loss: 1.1324
Epoch [3/20], Step [400/463], Loss: 1.0923
Epoch [4/20], Step [100/463], Loss: 1.1264
Epoch [4/20], Step [200/463], Loss: 1.4292
Epoch [4/20], Step [300/463], Loss: 1.2406
Epoch [4/20], Step [400/463], Loss: 1.1045
Epoch [5/20], Step [100/463], Loss: 1.1081
Epoch [5/20], Step [200/463], Loss: 1.1022
Epoch [5/20], Step [300/463], Loss: 1.0910
Epoch [5/20], Step [400/463], Loss: 1.0902
Epoch [6/20], Step [100/463], Loss: 1.0136
Epoch [6/20], Step [200/463], Loss: 1.0852
Epoch [6/20], Step [300/463], Loss: 0.9771
Epoch [6/20], Step [400/463], Loss: 1.1041
Epoch [7/20], Step [100/463], Loss: 1.0270
Epoch [7/20], Step [200/463], Loss: 1.0492
Epoch [7/20], Step [300/463], Loss: 1.0466
Epoch [7/20], Step [400/463], Loss: 1.0625
Epoch [8/20], Step [100/463], Loss: 0.9880
Epoch [8/20], Step [200/463], Loss: 0.8552
Epoch [8/20], Step [300/463], Loss: 1.4600
Epoch [8/20], Step [400/463], Loss: 1.0644
Epoch [9/20], Step [100/463], Loss: 1.0430
Epoch [9/20], Step [200/463], Loss: 0.7856
Epoch [9/20], Step [300/463], Loss: 0.8781
Epoch [9/20], Step [400/463], Loss: 1.1859
Epoch [10/20], Step [100/463], Loss: 0.9191
Epoch [10/20], Step [200/463], Loss: 0.9157
Epoch [10/20], Step [300/463], Loss: 0.9650
Epoch [10/20], Step [400/463], Loss: 0.7295
Epoch [11/20], Step [100/463], Loss: 0.9834
Epoch [11/20], Step [200/463], Loss: 0.8449
Epoch [11/20], Step [300/463], Loss: 0.9846
```

```
Epoch [11/20], Step [400/463], Loss: 0.9690
Epoch [12/20], Step [100/463], Loss: 1.0576
Epoch [12/20], Step [200/463], Loss: 1.0967
Epoch [12/20], Step [300/463], Loss: 0.9086
Epoch [12/20], Step [400/463], Loss: 0.9322
Epoch [13/20], Step [100/463], Loss: 1.0330
Epoch [13/20], Step [200/463], Loss: 1.1576
Epoch [13/20], Step [300/463], Loss: 0.7888
Epoch [13/20], Step [400/463], Loss: 0.9810
Epoch [14/20], Step [100/463], Loss: 1.1704
Epoch [14/20], Step [200/463], Loss: 0.8800
Epoch [14/20], Step [300/463], Loss: 1.0530
Epoch [14/20], Step [400/463], Loss: 0.9399
Epoch [15/20], Step [100/463], Loss: 0.8664
Epoch [15/20], Step [200/463], Loss: 0.9916
Epoch [15/20], Step [300/463], Loss: 0.6895
Epoch [15/20], Step [400/463], Loss: 1.0541
Epoch [16/20], Step [100/463], Loss: 0.9471
Epoch [16/20], Step [200/463], Loss: 1.0457
Epoch [16/20], Step [300/463], Loss: 0.8010
Epoch [16/20], Step [400/463], Loss: 1.1985
Epoch [17/20], Step [100/463], Loss: 1.0387
Epoch [17/20], Step [200/463], Loss: 1.0713
Epoch [17/20], Step [300/463], Loss: 0.9556
Epoch [17/20], Step [400/463], Loss: 0.7234
Epoch [18/20], Step [100/463], Loss: 1.0660
Epoch [18/20], Step [200/463], Loss: 0.8599
Epoch [18/20], Step [300/463], Loss: 1.1548
Epoch [18/20], Step [400/463], Loss: 0.9046
Epoch [19/20], Step [100/463], Loss: 0.6636
Epoch [19/20], Step [200/463], Loss: 1.0473
Epoch [19/20], Step [300/463], Loss: 0.8127
Epoch [19/20], Step [400/463], Loss: 1.1208
Epoch [20/20], Step [100/463], Loss: 0.8305
Epoch [20/20], Step [200/463], Loss: 1.0279
Epoch [20/20], Step [300/463], Loss: 0.6081
Epoch [20/20], Step [400/463], Loss: 0.8848
Finished Training Trainset
```



[9]:
```python
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print("Test Accuracy",correct / total)

    correct = 0
    total = 0

    for images, labels in val_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
```

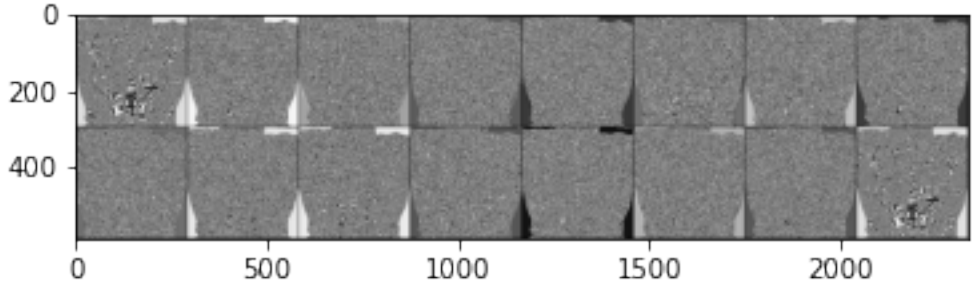
```
    print("Validation Accuracy",correct / total)

    correct = 0
    total = 0

    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print("Train Accuracy",correct / total)
```

```
Test Accuracy 0.596126594237128
Validation Accuracy 0.5931963146704465
Train Accuracy 0.5801552480593992
```

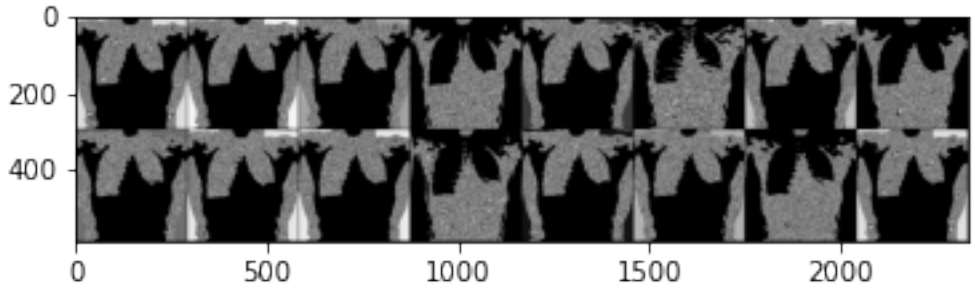
[10]:
```
def Show(out, title = ''):
    print(title)
    out = out.permute(1,0,2,3)
    grilla = torchvision.utils.make_grid(out)
    plt.imshow(transform.ToPILImage()(grilla), 'jet')
    plt.show()

def Show_Weight(out):
    grilla = torchvision.utils.make_grid(out)
    plt.imshow(transform.ToPILImage()(grilla), 'jet')
    plt.show()


with torch.no_grad():
    model.to('cpu')
    img, label = test_set[456]
    img = img.unsqueeze(0)
    out = model(img)
    print(out)
    print ((out == out.max()).nonzero())

    out = model.layer1[0](img)
    Show(out, 'layer 1: Convolution output')
    out = model.layer1[1](out)
    Show(out, 'layer 1: Activation function output')
```

```
    out = model.layer2[0](out)
    Show(out, 'layer 2: Convolution output')
    out = model.layer2[1](out)
    Show(out, 'layer 2: Activation function output')
    out = model.layer2[2](out)
    Show(out, 'layer 2: Max-Pooling')

    out = model.layer3[0](out)
    Show(out, 'layer 3: Convolution output')
    out = model.layer3[1](out)
    Show(out, 'layer 3: Activation function output')
    out = model.layer3[2](out)
    Show(out, 'layer 3: Max-Pooling')

    out = model.layer4[0](out)
    Show(out, 'layer 4: Convolution output')
    out = model.layer4[1](out)
    Show(out, 'layer 4: Activation function output')
    out = model.layer4[2](out)
    Show(out, 'layer 4: Max-Pooling')
```

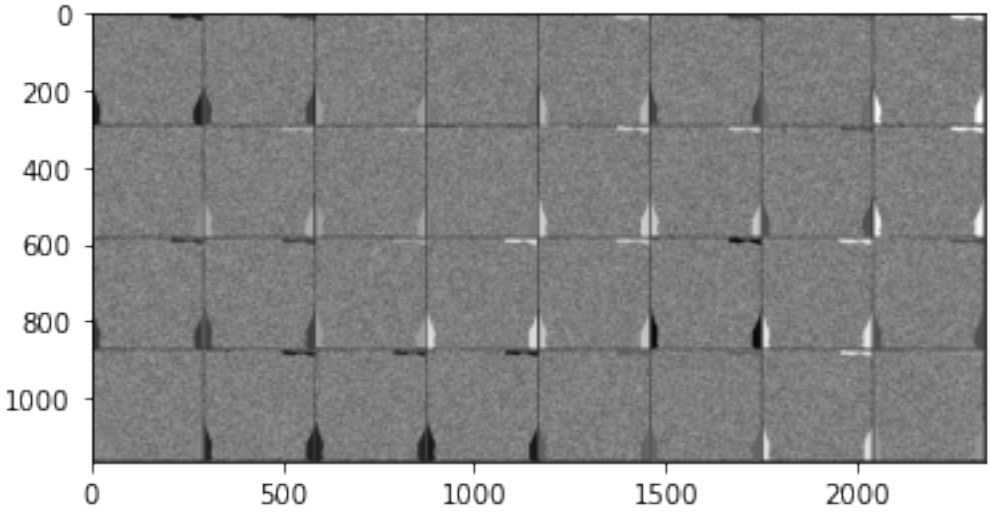
```
tensor([[-0.3140, -1.7436,  0.6980, -3.1916]])
tensor([[0, 2]])
layer 1: Convolution output
```



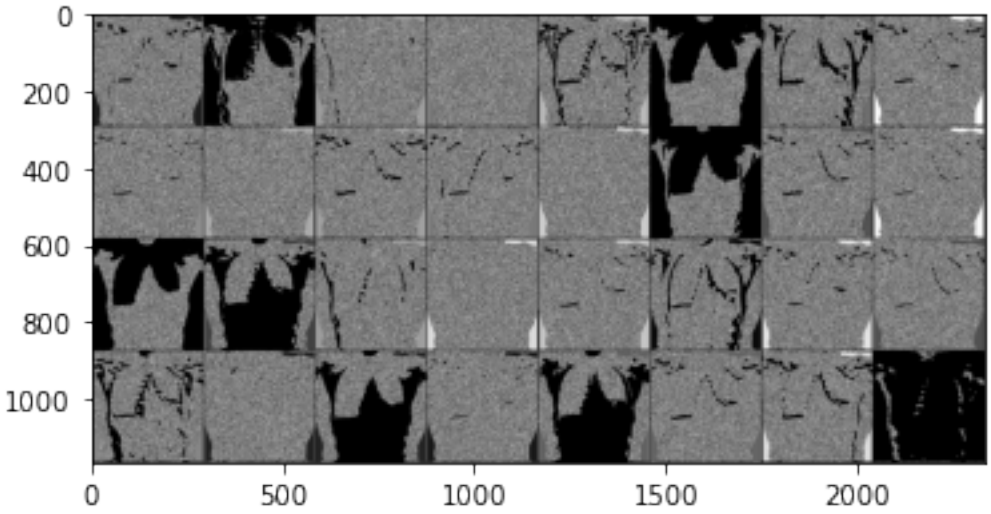

```
layer 1: Activation function output
```



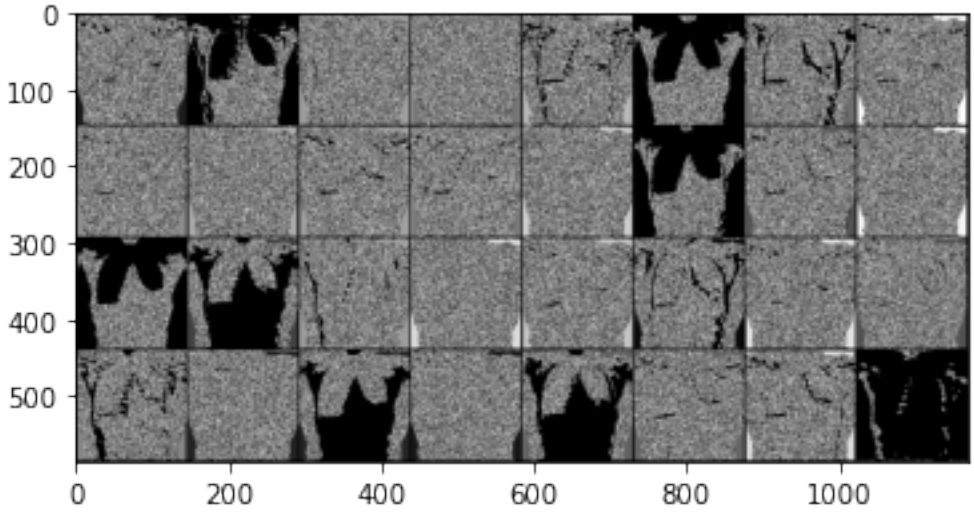
layer 2: Convolution output



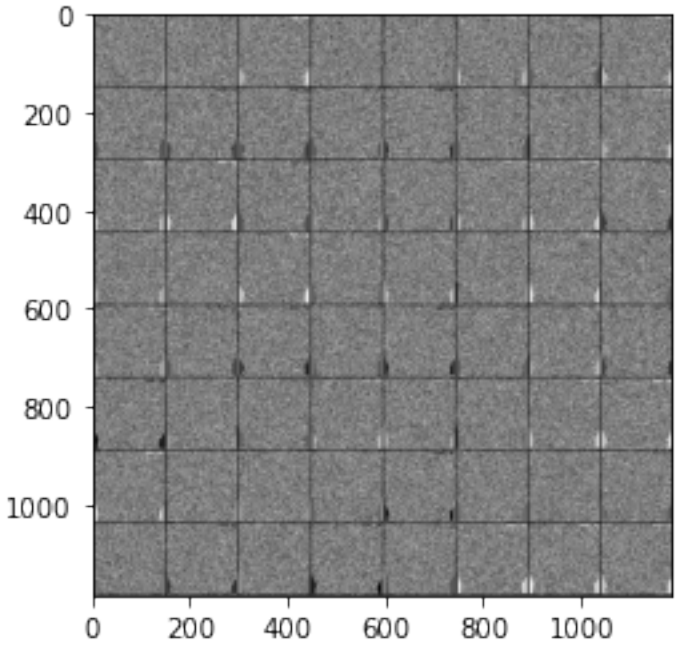
layer 2: Activation function output



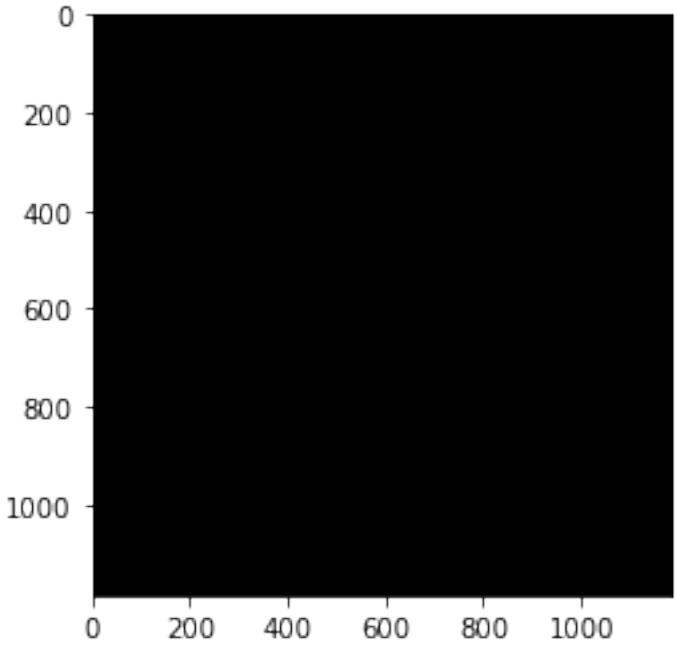
layer 2: Max-Pooling



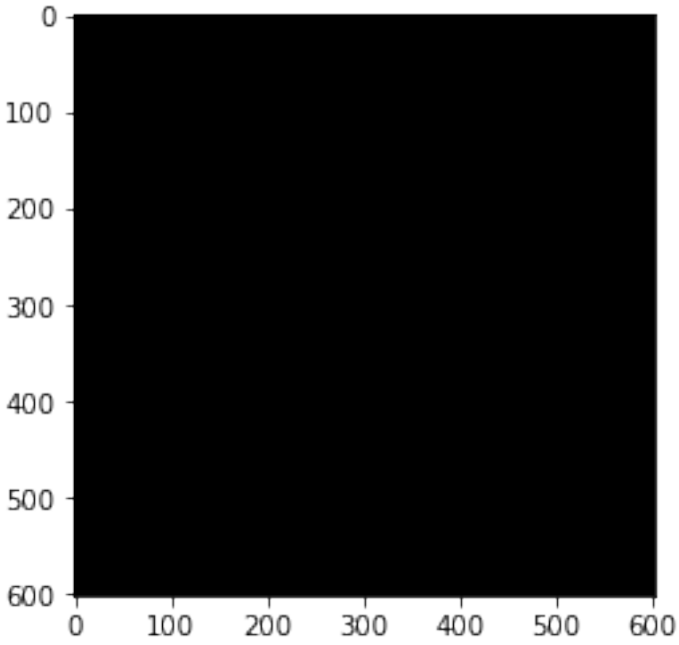
layer 3: Convolution output



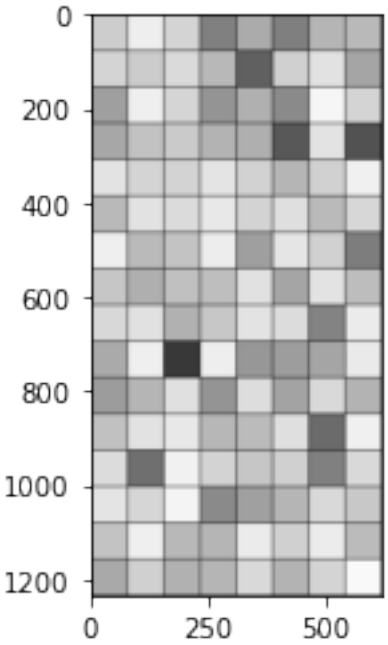
layer 3: Activation function output



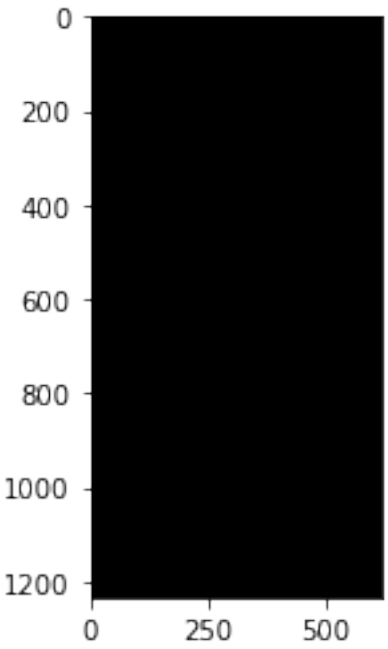
layer 3: Max-Pooling



layer 4: Convolution output



layer 4: Activation function output



layer 4: Max-Pooling

[ ]: