Experimento 5

[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/USUARIO/Desktop/ESKERE/
 ↪MMM",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])
```

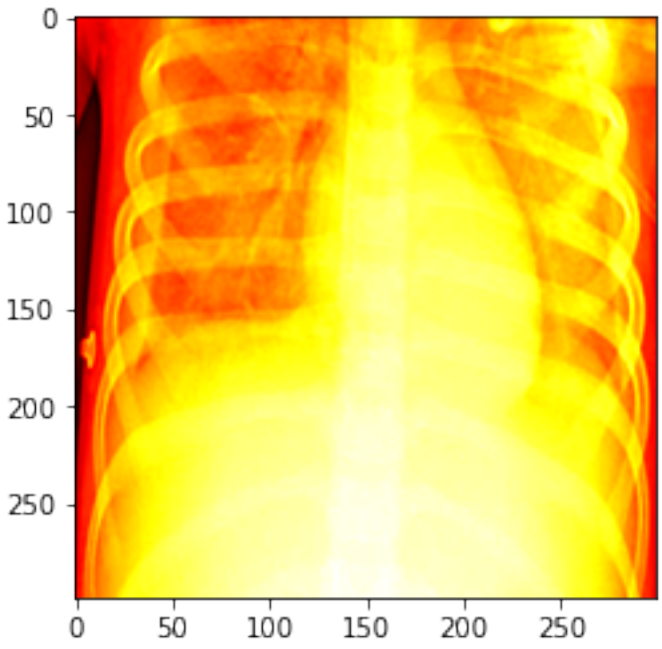
[2]:
```
def show_img(img):
    plt.imshow(img.numpy()[0], cmap='hot')
```

[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),nn.BatchNorm2d(32))

        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),nn.BatchNorm2d(64))

        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),nn.BatchNorm2d(128))

        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([[ 7.0322e-05, -2.1991e-03, -7.1058e-04,  ..., -2.3527e-03,
          -2.3879e-03,  7.2264e-04],
        [-1.1593e-03,  1.3432e-03, -6.7482e-04,  ...,  1.5090e-03,
          -1.2594e-03,  1.0038e-03],
        [ 1.6651e-03, -1.0660e-03,  2.6781e-04,  ..., -1.4467e-03,
          -6.3150e-04, -3.7005e-04],
        [ 1.5406e-03, -4.3205e-04,  9.6996e-04,  ..., -1.0950e-03,
          -1.1327e-03,  7.1710e-04]], device='cuda:0', requires_grad=True)
```

[7]:
```
with torch.no_grad():
    def validacion_acc():
        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()
        return correct/total
```

[8]:
```
def train(model, optimizer, loos_fn, num_epochs):
    loss_vals = []
    running_loss =0.0
    # train the model
    the_last_loss = 100
    patience = 2
    trigger_times = 0
    total_step = len(train_loader)
    list_loss_train= []
    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_train.append(loss.item())
            list_time.append(j)
            j+=1
            if (i+1) % 100 == 0:

                the_current_loss = validacion_acc()
                print('The current loss:', the_current_loss)


                if(the_current_loss>=0.84): return list_loss_train,list_time

                if the_current_loss < the_last_loss :
                    trigger_times += 1
                    print('trigger times:', trigger_times)
                    if trigger_times >= patience and the_current_loss>=0.83:
                        print('Early stopping!\nStart to test process.')
                        return list_loss_train,list_time
                else:
                    print('trigger times: 0')
                    trigger_times = 0

                the_last_loss = the_current_loss

                print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}' .
↪format(epoch+1, num_epochs, i+1, total_step, loss.item()))

        # Early stopping



    print('Finished Training Trainset')

    return list_loss_train,list_time
```

[9]:
```
list_loss_train,list_time=train(model,optimizer,loss_fn,num_epochs)
```

```
The current loss: 0.6272147413182141
trigger times: 1
Epoch [1/20], Step [100/463], Loss: 63.7795
The current loss: 0.6227261989133003
trigger times: 2
Epoch [1/20], Step [200/463], Loss: 24.7076
The current loss: 0.6695015355539806
trigger times: 0
Epoch [1/20], Step [300/463], Loss: 11.3158
```

```
The current loss: 0.7306874557051737
trigger times: 0
Epoch [1/20], Step [400/463], Loss: 5.7439
The current loss: 0.6924167257264352
trigger times: 1
Epoch [2/20], Step [100/463], Loss: 1.5298
The current loss: 0.7238365225608315
trigger times: 0
Epoch [2/20], Step [200/463], Loss: 0.3095
The current loss: 0.7507677769903142
trigger times: 0
Epoch [2/20], Step [300/463], Loss: 0.8905
The current loss: 0.7543113630994567
trigger times: 0
Epoch [2/20], Step [400/463], Loss: 0.8905
The current loss: 0.7410819749586581
trigger times: 1
Epoch [3/20], Step [100/463], Loss: 0.7959
The current loss: 0.7406094968107725
trigger times: 2
Epoch [3/20], Step [200/463], Loss: 0.6168
The current loss: 0.7793527049373966
trigger times: 0
Epoch [3/20], Step [300/463], Loss: 0.6108
The current loss: 0.7710843373493976
trigger times: 1
Epoch [3/20], Step [400/463], Loss: 1.1089
The current loss: 0.7791164658634538
trigger times: 0
Epoch [4/20], Step [100/463], Loss: 0.4085
The current loss: 0.731632412000945
trigger times: 1
Epoch [4/20], Step [200/463], Loss: 0.5038
The current loss: 0.788566028821167
trigger times: 0
Epoch [4/20], Step [300/463], Loss: 0.9827
The current loss: 0.7564375147649421
trigger times: 1
Epoch [4/20], Step [400/463], Loss: 0.8509
The current loss: 0.7748641625324829
trigger times: 0
Epoch [5/20], Step [100/463], Loss: 0.5115
The current loss: 0.7637609260571698
trigger times: 1
Epoch [5/20], Step [200/463], Loss: 0.5399
The current loss: 0.7725017717930546
trigger times: 0
Epoch [5/20], Step [300/463], Loss: 0.5351
```

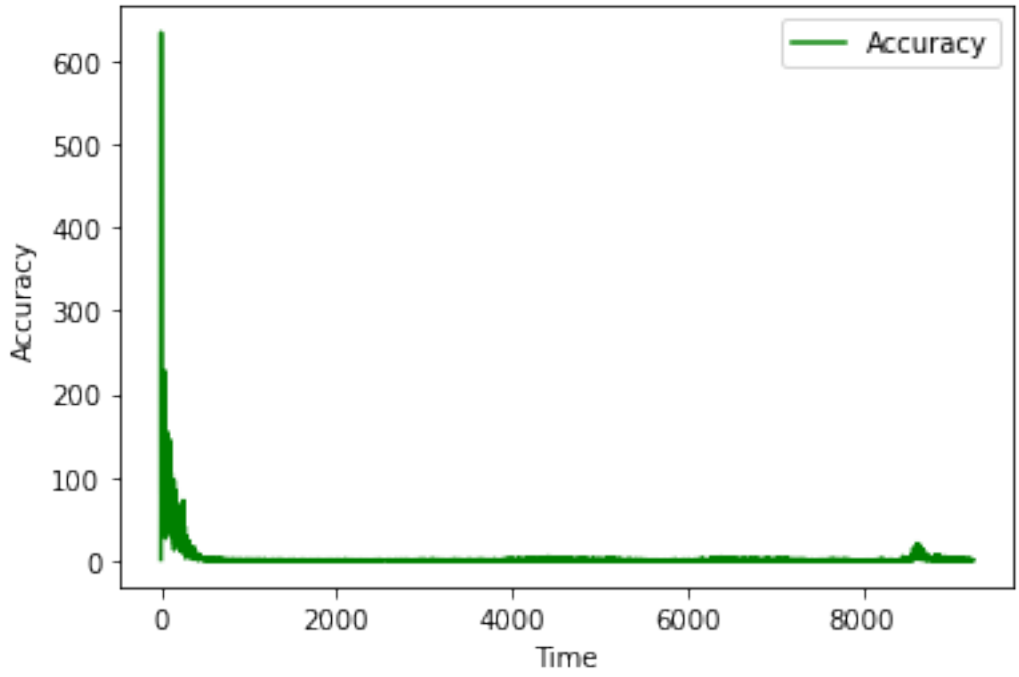
```
The current loss: 0.7838412473423104
trigger times: 0
Epoch [5/20], Step [400/463], Loss: 0.6299
The current loss: 0.7911646586345381
trigger times: 0
Epoch [6/20], Step [100/463], Loss: 0.3947
The current loss: 0.738483345145287
trigger times: 1
Epoch [6/20], Step [200/463], Loss: 0.3915
The current loss: 0.7663595558705409
trigger times: 0
Epoch [6/20], Step [300/463], Loss: 0.5252
The current loss: 0.7781715095676824
trigger times: 0
Epoch [6/20], Step [400/463], Loss: 0.5033
The current loss: 0.7906921804866525
trigger times: 0
Epoch [7/20], Step [100/463], Loss: 0.4838
The current loss: 0.7703756201275691
trigger times: 1
Epoch [7/20], Step [200/463], Loss: 0.2575
The current loss: 0.7342310418143161
trigger times: 2
Epoch [7/20], Step [300/463], Loss: 0.3792
The current loss: 0.7854949208599102
trigger times: 0
Epoch [7/20], Step [400/463], Loss: 0.5783
The current loss: 0.7571462319867706
trigger times: 1
Epoch [8/20], Step [100/463], Loss: 0.8836
The current loss: 0.7895109851169383
trigger times: 0
Epoch [8/20], Step [200/463], Loss: 0.4432
The current loss: 0.7665957949444838
trigger times: 1
Epoch [8/20], Step [300/463], Loss: 0.5925
The current loss: 0.7836050082683675
trigger times: 0
Epoch [8/20], Step [400/463], Loss: 0.8366
The current loss: 0.7628159697613985
trigger times: 1
Epoch [9/20], Step [100/463], Loss: 0.4728
The current loss: 0.7807701393810537
trigger times: 0
Epoch [9/20], Step [200/463], Loss: 0.3414
The current loss: 0.7795889440113395
trigger times: 1
Epoch [9/20], Step [300/463], Loss: 0.7951
```

```
The current loss: 0.7748641625324829
trigger times: 2
Epoch [9/20], Step [400/463], Loss: 0.5272
The current loss: 0.7550200803212851
trigger times: 3
Epoch [10/20], Step [100/463], Loss: 1.3007
The current loss: 0.7396645405150012
trigger times: 4
Epoch [10/20], Step [200/463], Loss: 2.8506
The current loss: 0.7694306638317978
trigger times: 0
Epoch [10/20], Step [300/463], Loss: 1.3658
The current loss: 0.7233640444129459
trigger times: 1
Epoch [10/20], Step [400/463], Loss: 1.2189
The current loss: 0.771793054571226
trigger times: 0
Epoch [11/20], Step [100/463], Loss: 0.7144
The current loss: 0.7124970470115757
trigger times: 1
Epoch [11/20], Step [200/463], Loss: 0.4074
The current loss: 0.7500590597684857
trigger times: 0
Epoch [11/20], Step [300/463], Loss: 2.5725
The current loss: 0.7424994094023152
trigger times: 1
Epoch [11/20], Step [400/463], Loss: 0.7622
The current loss: 0.7335223245924876
trigger times: 2
Epoch [12/20], Step [100/463], Loss: 1.5992
The current loss: 0.7330498464446019
trigger times: 3
Epoch [12/20], Step [200/463], Loss: 0.6334
The current loss: 0.7814788566028821
trigger times: 0
Epoch [12/20], Step [300/463], Loss: 0.3945
The current loss: 0.7843137254901961
trigger times: 0
Epoch [12/20], Step [400/463], Loss: 0.3893
The current loss: 0.7895109851169383
trigger times: 0
Epoch [13/20], Step [100/463], Loss: 0.6722
The current loss: 0.789983463264824
trigger times: 0
Epoch [13/20], Step [200/463], Loss: 0.3701
The current loss: 0.7937632884479093
trigger times: 0
Epoch [13/20], Step [300/463], Loss: 0.7388
```

```
The current loss: 0.7618710134656272
trigger times: 1
Epoch [13/20], Step [400/463], Loss: 0.2398
The current loss: 0.785731159933853
trigger times: 0
Epoch [14/20], Step [100/463], Loss: 0.6410
The current loss: 0.788566028821167
trigger times: 0
Epoch [14/20], Step [200/463], Loss: 0.2373
The current loss: 0.7805339003071108
trigger times: 1
Epoch [14/20], Step [300/463], Loss: 0.8419
The current loss: 0.7656508386487125
trigger times: 2
Epoch [14/20], Step [400/463], Loss: 1.6693
The current loss: 0.7665957949444838
trigger times: 0
Epoch [15/20], Step [100/463], Loss: 0.7357
The current loss: 0.7658870777226553
trigger times: 1
Epoch [15/20], Step [200/463], Loss: 0.7044
The current loss: 0.7715568154972833
trigger times: 0
Epoch [15/20], Step [300/463], Loss: 0.3885
The current loss: 0.7864398771556815
trigger times: 0
Epoch [15/20], Step [400/463], Loss: 0.6201
The current loss: 0.7103708953460902
trigger times: 1
Epoch [16/20], Step [100/463], Loss: 0.7026
The current loss: 0.7167493503425466
trigger times: 0
Epoch [16/20], Step [200/463], Loss: 0.3410
The current loss: 0.7987243090007087
trigger times: 0
Epoch [16/20], Step [300/463], Loss: 0.3518
The current loss: 0.7954169619655092
trigger times: 1
Epoch [16/20], Step [400/463], Loss: 0.2270
The current loss: 0.8065201984408221
trigger times: 0
Epoch [17/20], Step [100/463], Loss: 1.0581
The current loss: 0.7888022678951099
trigger times: 1
Epoch [17/20], Step [200/463], Loss: 0.5957
The current loss: 0.7852586817859674
trigger times: 2
Epoch [17/20], Step [300/463], Loss: 0.1639
```

```
The current loss: 0.7729742499409402
trigger times: 3
Epoch [17/20], Step [400/463], Loss: 0.7755
The current loss: 0.7741554453106544
trigger times: 0
Epoch [18/20], Step [100/463], Loss: 0.7224
The current loss: 0.8103000236239074
trigger times: 0
Epoch [18/20], Step [200/463], Loss: 0.8620
The current loss: 0.7854949208599102
trigger times: 1
Epoch [18/20], Step [300/463], Loss: 0.1304
The current loss: 0.7921096149303095
trigger times: 0
Epoch [18/20], Step [400/463], Loss: 0.4481
The current loss: 0.7826600519725962
trigger times: 1
Epoch [19/20], Step [100/463], Loss: 0.2061
The current loss: 0.7595086227261989
trigger times: 2
Epoch [19/20], Step [200/463], Loss: 1.7885
The current loss: 0.6806047720292936
trigger times: 3
Epoch [19/20], Step [300/463], Loss: 5.5783
The current loss: 0.6695015355539806
trigger times: 4
Epoch [19/20], Step [400/463], Loss: 5.5122
The current loss: 0.7566737538388849
trigger times: 0
Epoch [20/20], Step [100/463], Loss: 1.4138
The current loss: 0.7930545712260808
trigger times: 0
Epoch [20/20], Step [200/463], Loss: 0.0128
The current loss: 0.7847862036380817
trigger times: 1
Epoch [20/20], Step [300/463], Loss: 0.7643
The current loss: 0.7762815969761399
trigger times: 2
Epoch [20/20], Step [400/463], Loss: 0.9886
Finished Training Trainset
```

[10]:
```
plt.plot(list_time,list_loss_train,color="green", label="Accuracy")
plt.legend()
plt.xlabel("Time")
plt.ylabel("Accuracy")
plt.show()
```

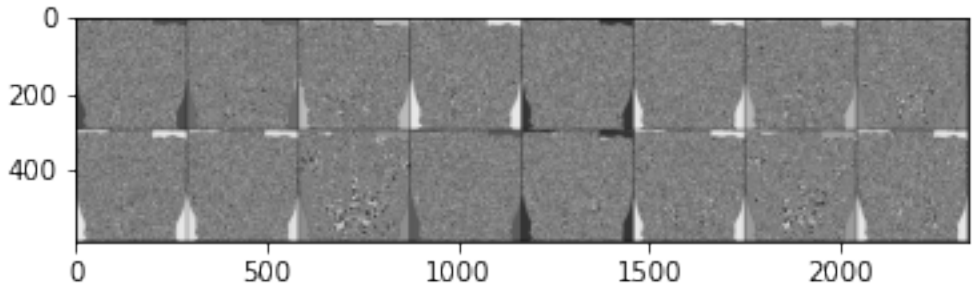


[11]:
```
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    print("Train Accuracy",correct / total)

    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)
```

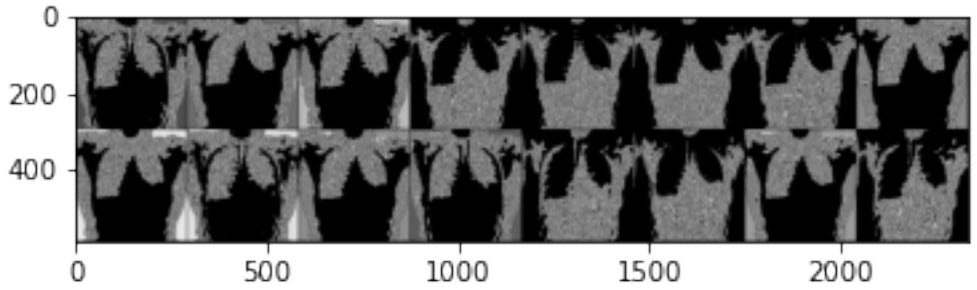
```
Train Accuracy 0.95543304586
Test Accuracy 0.8275059045
```

[12]:
```
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')
```

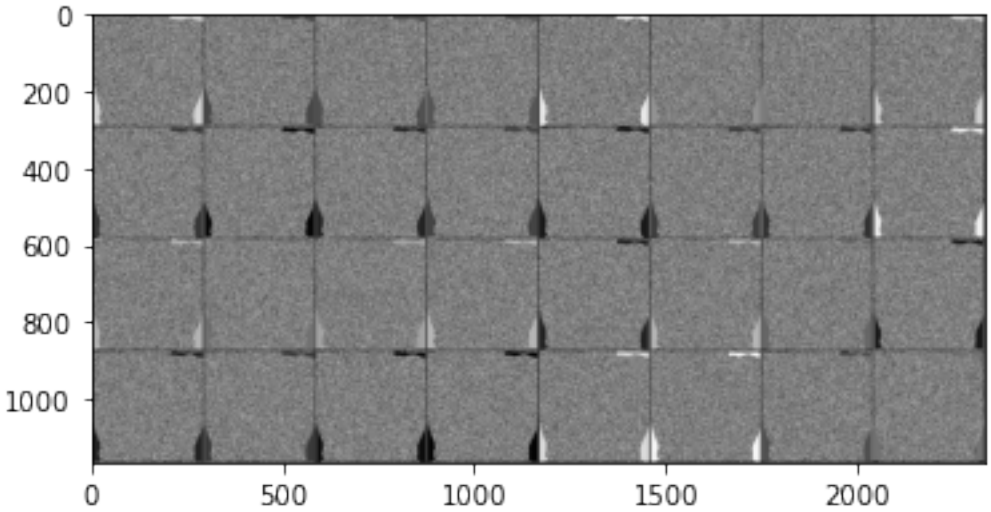
```
tensor([[-13.1560,  11.0222,   7.4993, -51.3124]])
tensor([[0, 1]])
layer 1: Convolution output
```



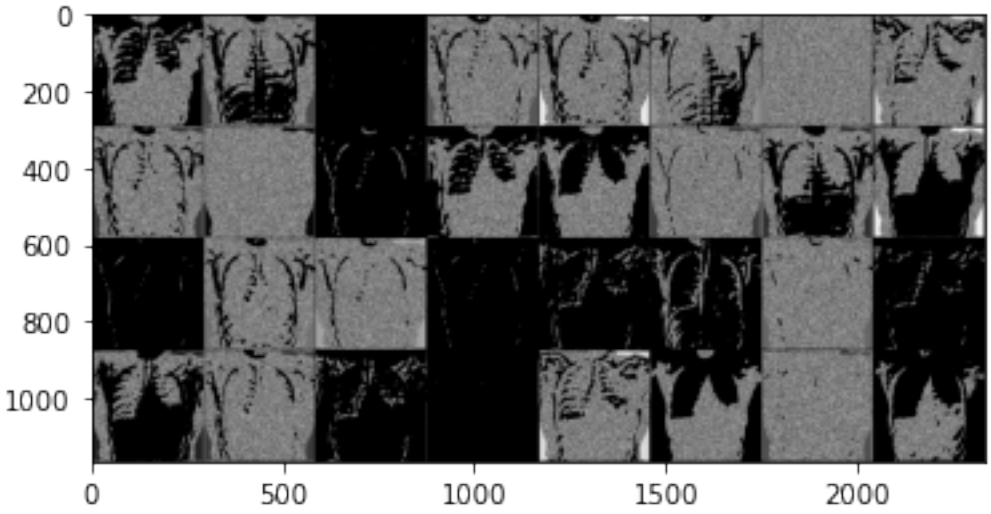

```
layer 1: Activation function output
```



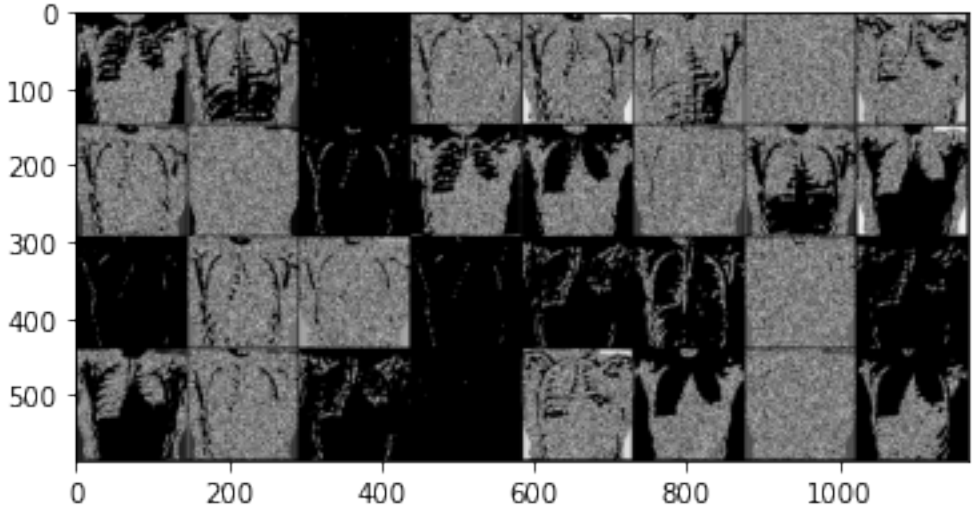

```
layer 2: Convolution output
```



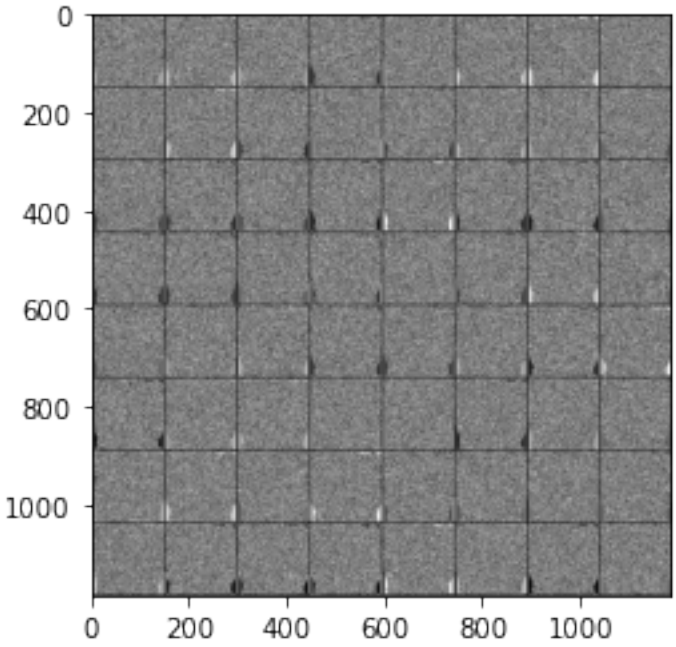
layer 2: Activation function output



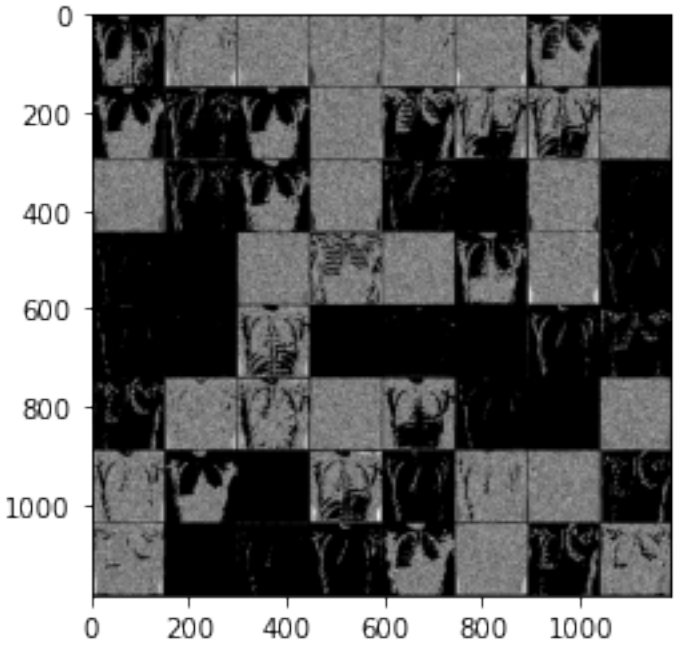
layer 2: Max-Pooling



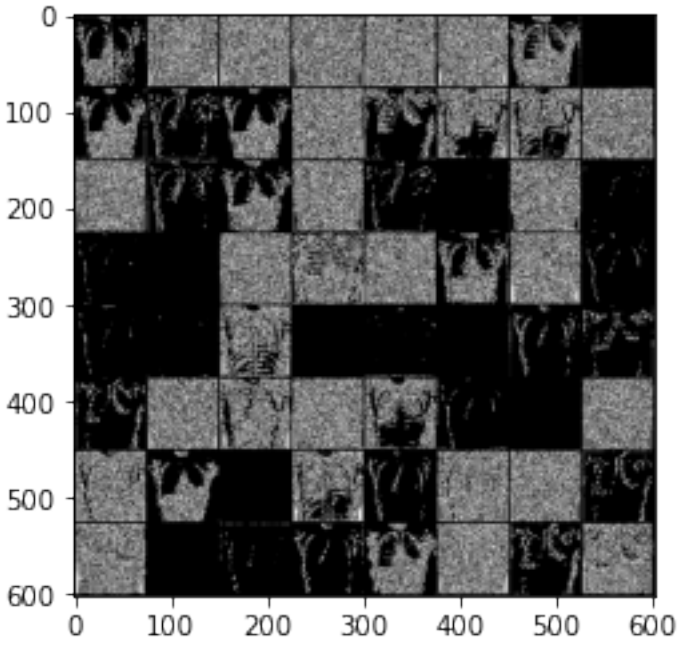
layer 3: Convolution output



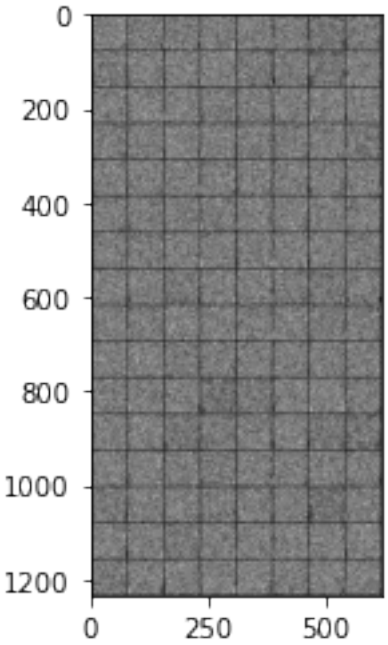
layer 3: Activation function output



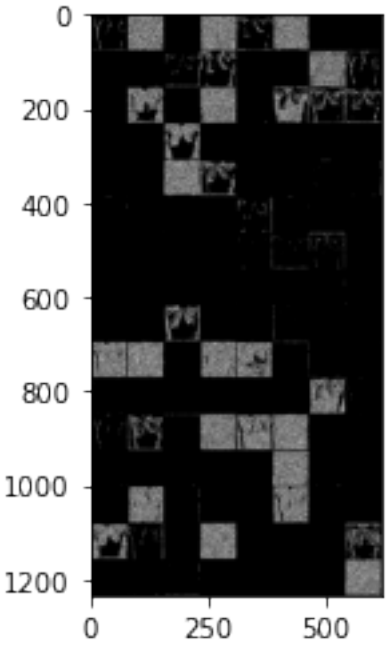
layer 3: Max-Pooling



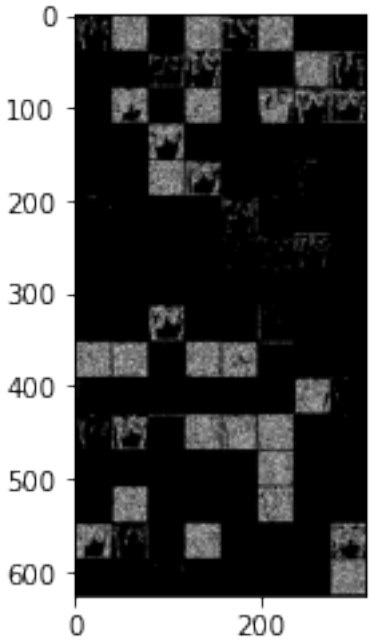
layer 4: Convolution output



layer 4: Activation function output



layer 4: Max-Pooling



[ ]: