Experimento 4

[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]:

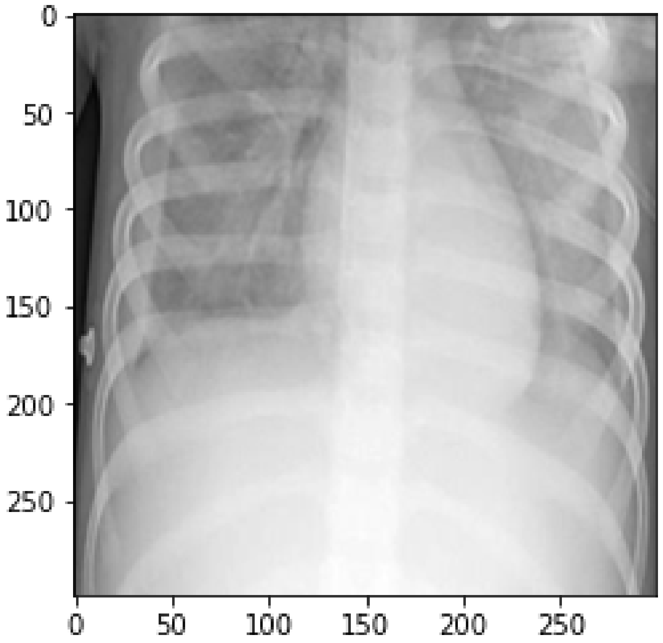
```
print(len(train_set))
print(len(test_set))
print(len(val_set))
```

```
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),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
```

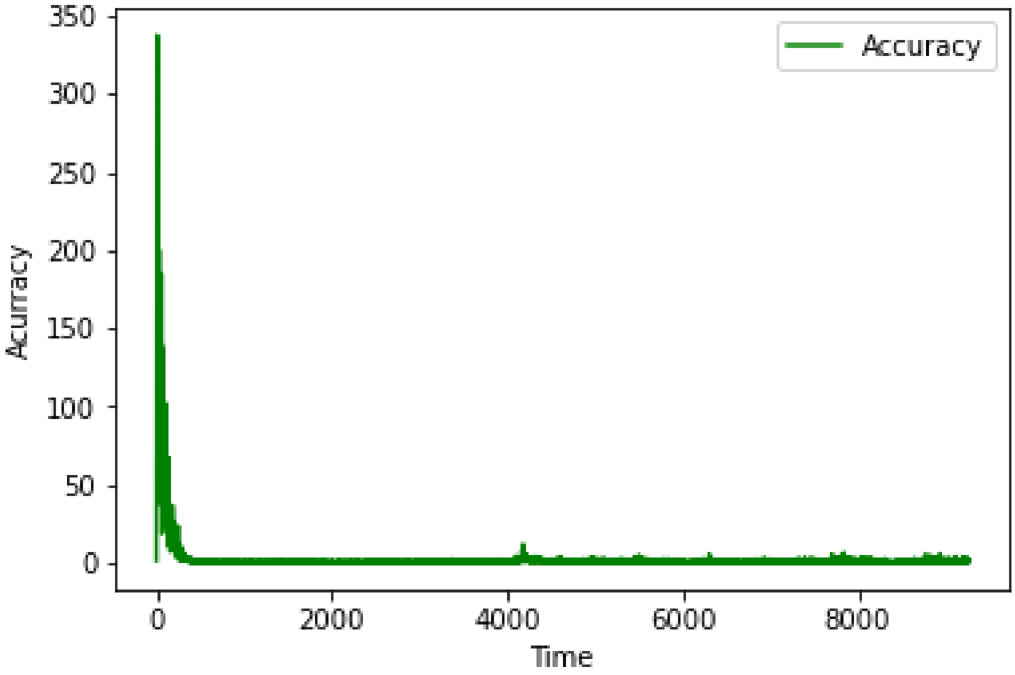
```
[torch.Size([4, 175232]), torch.Size([4])]
```

[6]:
```
Parameter containing:
tensor([[-0.0010,  0.0014,  0.0012,  ..., -0.0020, -0.0016, -0.0003],
        [ 0.0003, -0.0015,  0.0016,  ..., -0.0008, -0.0013, -0.0018],
        [-0.0012, -0.0020,  0.0020,  ..., -0.0013, -0.0021,  0.0021],
        [-0.0023, -0.0016, -0.0015,  ..., -0.0010,  0.0003,  0.0005]],
       device='cuda:0', requires_grad=True)
```

[7]:
```
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]:
```
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: 26.5680
Epoch [1/20], Step [200/463], Loss: 17.9783
Epoch [1/20], Step [300/463], Loss: 4.2819
Epoch [1/20], Step [400/463], Loss: 0.7125
Epoch [2/20], Step [100/463], Loss: 0.4105
Epoch [2/20], Step [200/463], Loss: 0.6548
Epoch [2/20], Step [300/463], Loss: 0.3909
Epoch [2/20], Step [400/463], Loss: 0.8094
Epoch [3/20], Step [100/463], Loss: 0.3243
Epoch [3/20], Step [200/463], Loss: 0.3466
Epoch [3/20], Step [300/463], Loss: 0.6585
Epoch [3/20], Step [400/463], Loss: 1.0781
Epoch [4/20], Step [100/463], Loss: 0.3112
Epoch [4/20], Step [200/463], Loss: 0.4940
Epoch [4/20], Step [300/463], Loss: 0.5900
Epoch [4/20], Step [400/463], Loss: 0.1904
Epoch [5/20], Step [100/463], Loss: 0.5739
Epoch [5/20], Step [200/463], Loss: 0.6155
Epoch [5/20], Step [300/463], Loss: 0.3001
Epoch [5/20], Step [400/463], Loss: 0.3333
Epoch [6/20], Step [100/463], Loss: 0.5945
Epoch [6/20], Step [200/463], Loss: 0.2525
Epoch [6/20], Step [300/463], Loss: 0.8553
Epoch [6/20], Step [400/463], Loss: 0.6627
Epoch [7/20], Step [100/463], Loss: 0.4400
Epoch [7/20], Step [200/463], Loss: 0.6392
Epoch [7/20], Step [300/463], Loss: 1.1934
Epoch [7/20], Step [400/463], Loss: 0.1875
Epoch [8/20], Step [100/463], Loss: 0.7594
Epoch [8/20], Step [200/463], Loss: 0.5291
Epoch [8/20], Step [300/463], Loss: 0.7889
Epoch [8/20], Step [400/463], Loss: 0.6325
Epoch [9/20], Step [100/463], Loss: 0.1640
Epoch [9/20], Step [200/463], Loss: 1.0866
Epoch [9/20], Step [300/463], Loss: 0.9030
Epoch [9/20], Step [400/463], Loss: 1.5918
Epoch [10/20], Step [100/463], Loss: 0.7051
Epoch [10/20], Step [200/463], Loss: 2.0270
Epoch [10/20], Step [300/463], Loss: 0.2571
Epoch [10/20], Step [400/463], Loss: 1.0388
Epoch [11/20], Step [100/463], Loss: 0.2761
Epoch [11/20], Step [200/463], Loss: 0.6677
Epoch [11/20], Step [300/463], Loss: 0.7748
Epoch [11/20], Step [400/463], Loss: 0.7447
Epoch [12/20], Step [100/463], Loss: 1.9397
Epoch [12/20], Step [200/463], Loss: 1.0744
Epoch [12/20], Step [300/463], Loss: 0.2082
Epoch [12/20], Step [400/463], Loss: 2.4821
Epoch [13/20], Step [100/463], Loss: 1.6145
Epoch [13/20], Step [200/463], Loss: 1.7571
Epoch [13/20], Step [300/463], Loss: 1.2336
Epoch [13/20], Step [400/463], Loss: 1.3819
Epoch [14/20], Step [100/463], Loss: 0.5558
Epoch [14/20], Step [200/463], Loss: 1.9162
Epoch [14/20], Step [300/463], Loss: 0.7344
Epoch [14/20], Step [400/463], Loss: 0.0889
Epoch [15/20], Step [100/463], Loss: 0.6145
Epoch [15/20], Step [200/463], Loss: 0.2542
Epoch [15/20], Step [300/463], Loss: 0.3374
Epoch [15/20], Step [400/463], Loss: 1.1852
Epoch [16/20], Step [100/463], Loss: 0.4219
Epoch [16/20], Step [200/463], Loss: 0.5924
Epoch [16/20], Step [300/463], Loss: 0.4941
Epoch [16/20], Step [400/463], Loss: 0.7897
Epoch [17/20], Step [100/463], Loss: 1.4301
Epoch [17/20], Step [200/463], Loss: 1.1623
Epoch [17/20], Step [300/463], Loss: 4.7026
Epoch [17/20], Step [400/463], Loss: 0.3348
Epoch [18/20], Step [100/463], Loss: 0.2741
Epoch [18/20], Step [200/463], Loss: 0.6788
Epoch [18/20], Step [300/463], Loss: 0.8928
Epoch [18/20], Step [400/463], Loss: 0.2067
Epoch [19/20], Step [100/463], Loss: 0.1447
Epoch [19/20], Step [200/463], Loss: 0.0273
Epoch [19/20], Step [300/463], Loss: 0.6088
Epoch [19/20], Step [400/463], Loss: 1.0564
Epoch [20/20], Step [100/463], Loss: 1.4888
Epoch [20/20], Step [200/463], Loss: 1.6213
Epoch [20/20], Step [300/463], Loss: 0.9583
Epoch [20/20], Step [400/463], Loss: 0.8872
Finished Training Trainset
```



[9]:

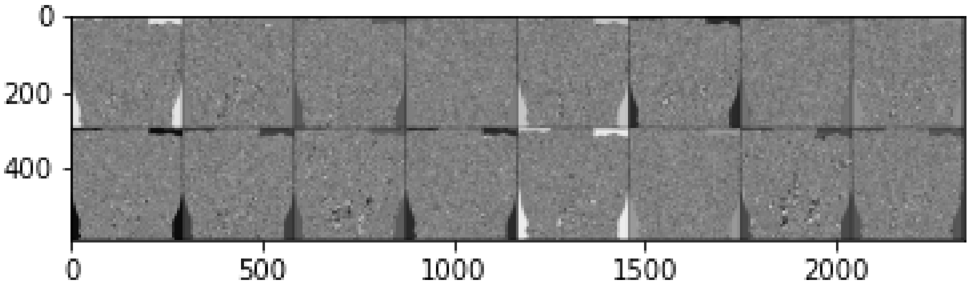
```
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == 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 0.8375059045819556
Validation Accuracy 0.8230569336168202
Train Accuracy 0.9561930475869052
```

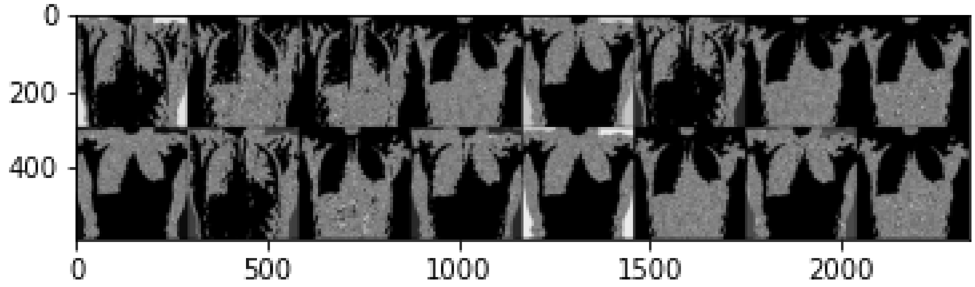
[16]:

```
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')
```

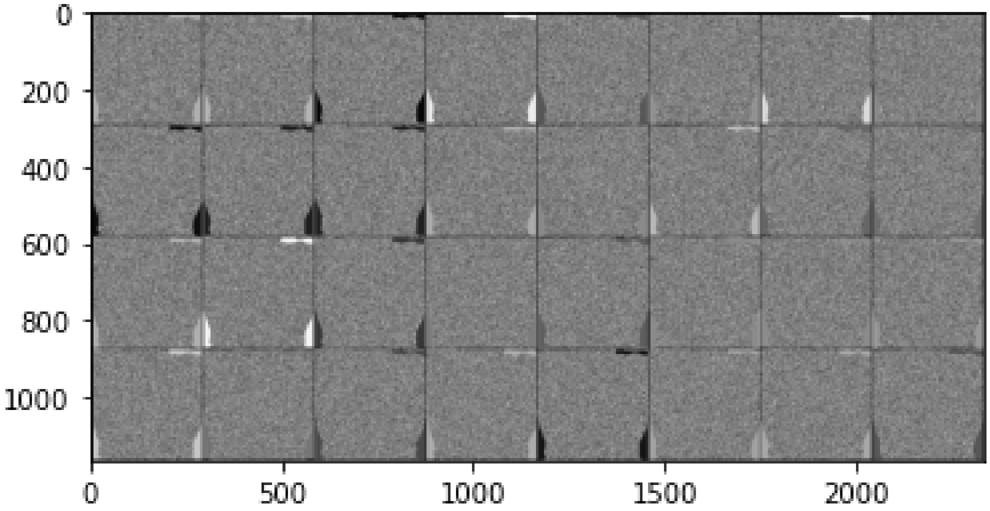
```
tensor([[-59.4509,  22.0876,  24.1623, -76.1220]])
tensor([[0, 2]])
layer 1: Convolution output
```



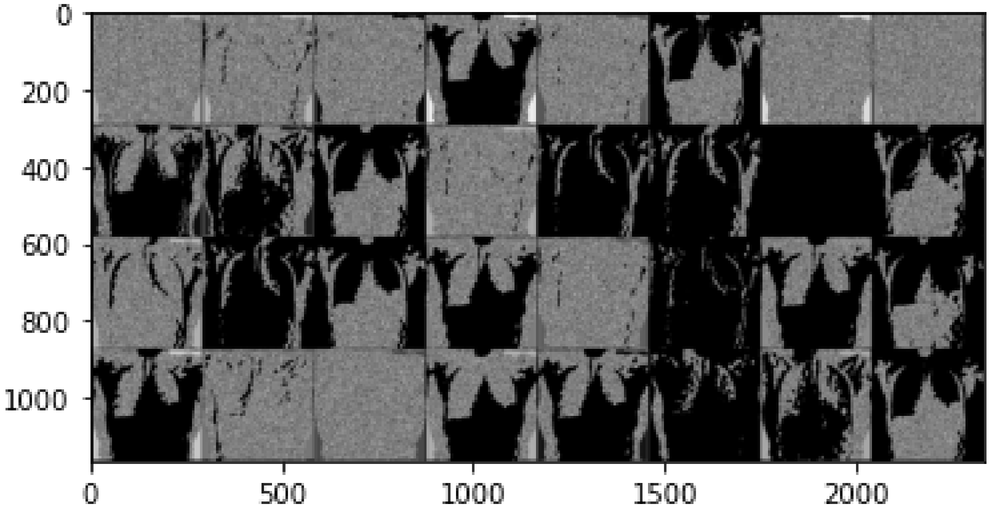
```
layer 1: Activation function output
```



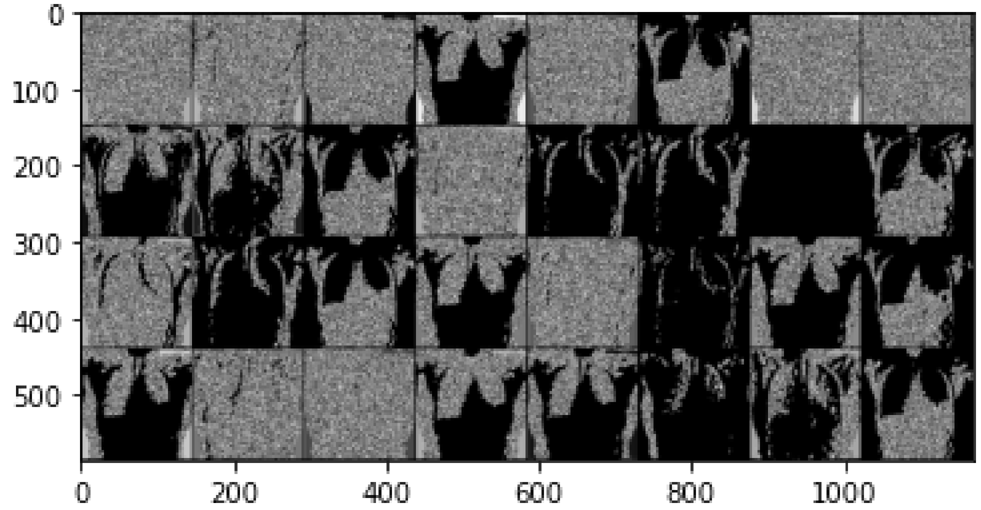
```
layer 2: Convolution output
```



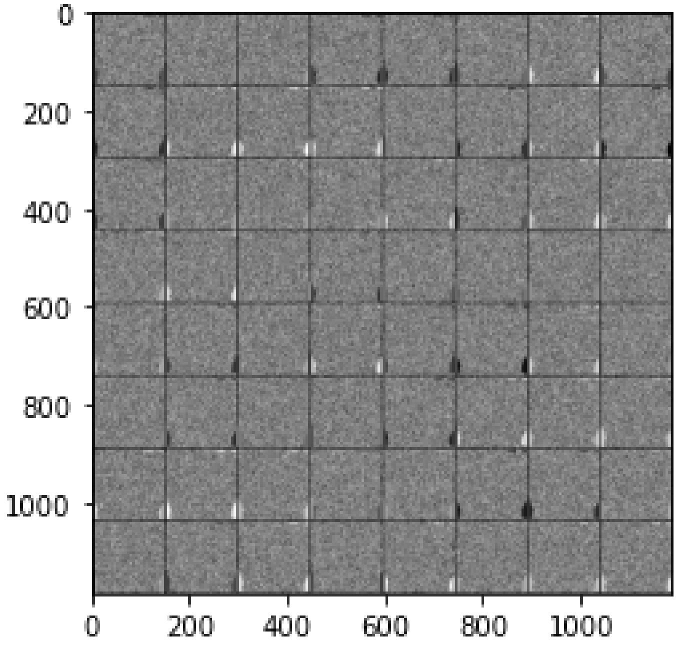
```
layer 2: Activation function output
```



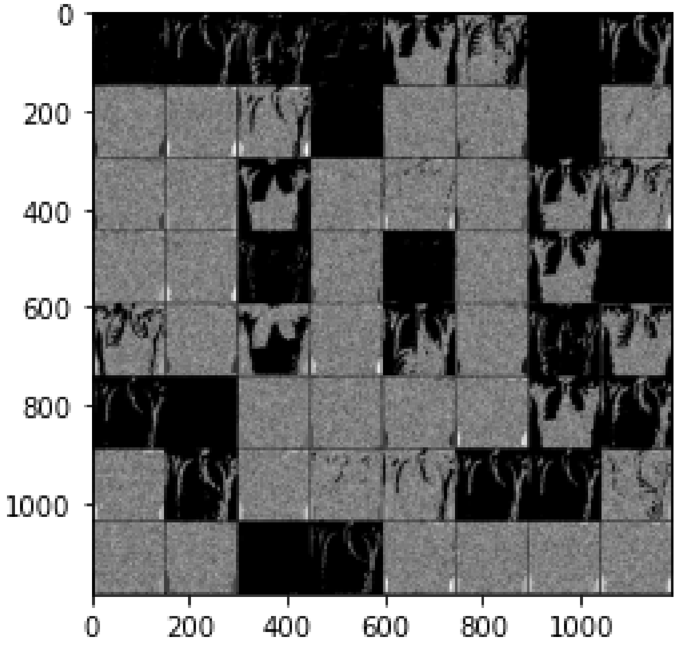
```
layer 2: Max-Pooling
```



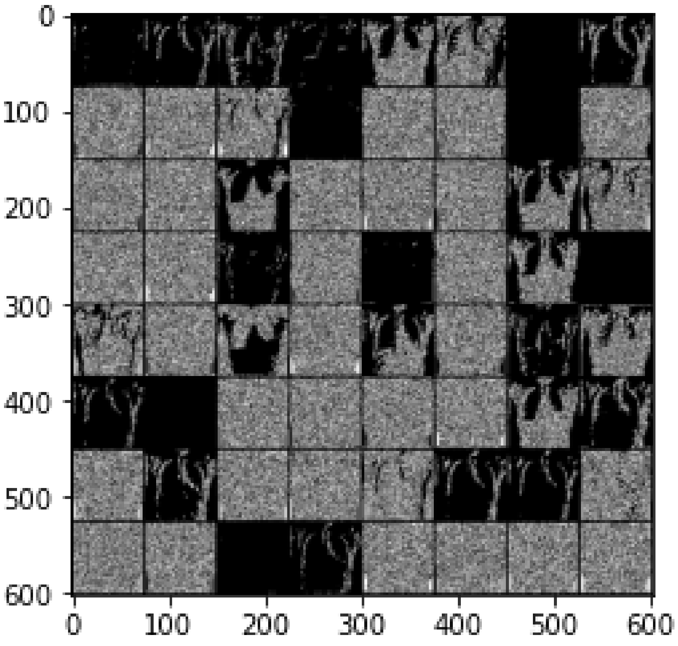
```
layer 3: Convolution output
```



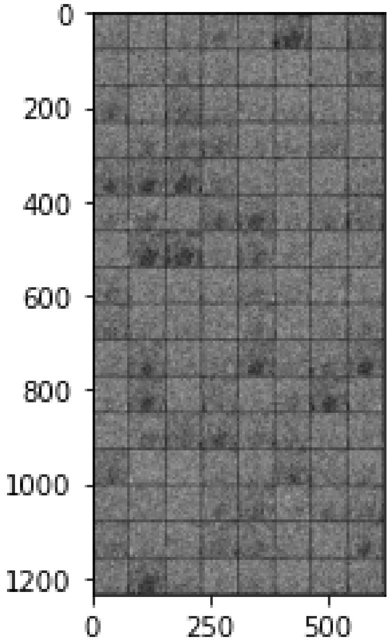
layer 3: Activation function output



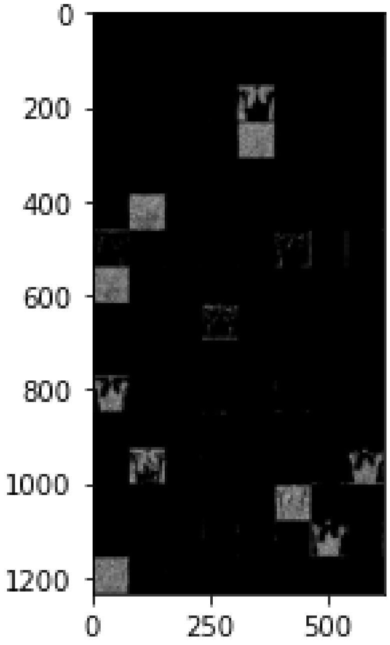
layer 3: Max-Pooling



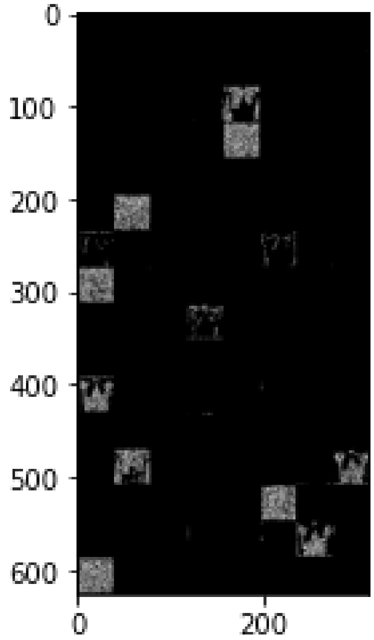
layer 4: Convolution output



layer 4: Activation function output



layer 4: Max-Pooling



```
[ ]:
```