e-feature-maps-of-a-cnn-model-lab8

May 8, 2023

[1]:
```
!pip install torchsummary
```

```
Collecting torchsummary
  Downloading torchsummary-1.5.1-py3-none-any.whl (2.8 kB)
Installing collected packages: torchsummary
Successfully installed torchsummary-1.5.1
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommended to
use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

[2]:
```
# import libraries
import os
import sys
import time

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy_score

import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
import torchvision.transforms as T
from torchsummary import summary
```

[3]:
```
# select GPU if available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# device = "cpu"  # explicitly set to execute on CPU
```

[4]:
```
!nvidia-smi
```

```
Mon May  8 17:26:36 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.161.03   Driver Version: 470.161.03   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE…  Off   | 00000000:00:04.0 Off |                    0 |
| N/A   37C    P0    26W / 250W |      2MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

[5]:
```python
# transformation of the training/testing images
transformer = T.Compose([
    T.Resize((150,150)),
    T.RandomHorizontalFlip(),
    T.ToTensor(),  #0-255 to 0-1, numpy to tensors
    T.Normalize([0.5,0.5,0.5], # 0-1 to [-1,1] , formula (x-mean)/std
                         [0.5,0.5,0.5])
])


train_path = "/kaggle/input/intel-image-classification/seg_train/seg_train/"
test_path = "/kaggle/input/intel-image-classification/seg_test/seg_test/"

# load the training/testing images
train_set  = ImageFolder(train_path, transform = transformer)
test_set   = ImageFolder(test_path, transform = transformer)

# create DataLoader
batch_size  = 256
train_loader = DataLoader(train_set, batch_size = batch_size, shuffle = True)
test_loader  = DataLoader(test_set, batch_size = batch_size, shuffle = True)
```

[6]:
```python
# total test images, total train images
len(test_set), len(train_set)
```

[6]:
```
(3000, 14034)
```

[7]:
```
# categories and their labels
train_set.class_to_idx
```

[7]:
```
{'buildings': 0,
 'forest': 1,
 'glacier': 2,
 'mountain': 3,
 'sea': 4,
 'street': 5}
```

[8]:
```
# create model architecture
class CNN_Net(nn.Module):
    def __init__(self, is_print_toggle = False):
        super(CNN_Net, self).__init__()

        ##-----------------------------------------convolution␣
 ↪layers---------------------------------------

        ## 1st convolution - max_pooling block  ==> # np.floor((imahe_height +␣
 ↪2*padding - kernel_size)/stride) + 1
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=12, kernel_size=3,␣
 ↪stride=1, padding=1)
        # shape = (256, 12, 150, 150)
        self.bn1 = nn.BatchNorm2d(num_features=12)
        # shape = (256, 12, 150, 150)
        self.relu1     = nn.ReLU()
        # shape = (256, 12, 150, 150)
        self.max_pool1 = nn.MaxPool2d(kernel_size = 2)
        # So, after maxpool, 148/2 = 74 # shape = (256, 12, 75, 75)



        ## 2nd convolution - max_pooling block
        self.conv2 = nn.
 ↪Conv2d(in_channels=12,out_channels=20,kernel_size=3,stride=1,padding=1)
        # shape = (256, 20, 75, 75)
        self.relu2 = nn.ReLU()
        # shape = (256, 20, 75, 75)



        ## convolution layer 3
        self.conv3 = nn.Conv2d(in_channels=20,␣
 ↪out_channels=32,kernel_size=3,stride=1,padding=1)
        # shape = (256, 32, 75, 75)
        self.bn3 = nn.BatchNorm2d(num_features=32)
        # shape = (256, 32, 75, 75)
```

```
        self.relu3      = nn.ReLU()
        # shape = (256, 32, 75, 75)



    ␣
↪##-----------------------------------------------FCL-----------------------------------------
        estimated_features = 32 * 75 * 75 # output_channel * height * width =␣
↪180000 (from convolution layer 3)

        # output layer
        self.output = nn.Linear(in_features = estimated_features, out_features␣
↪= 6)

        self.print_toggle = is_print_toggle



    def forward(self, X):

        ##---------------------------------------convolution␣
↪layers---------------------------------------
        print(f"Input: {X.shape}") if self.print_toggle else None

        # 1st convolution-maxpooling block
        convolution_layer1 = self.relu1( self.bn1( self.conv1(X) ) )
        X = self.max_pool1( convolution_layer1 )
        print(f"conv1/max_pool: {X.shape}") if self.print_toggle else None

        # 2nd convolution-maxpooling block
        convolution_layer2 = self.relu2( self.conv2( X ) )
        print(f"Layer conv2/max_pool: {X.shape}") if self.print_toggle else None

        # 3rd convolution-maxpooling block
        X = self.relu3( self.bn3( self.conv3( convolution_layer2 ) ) )
        print(f"Layer conv3/max_pool: {X.shape}") if self.print_toggle else None



    ␣
↪##-----------------------------------------------FCL-----------------------------------------
        # reshape for linear layers
        nUnits = X.shape.numel()/X.shape[0]
        X = X.view(-1, int(nUnits))  # this shape should to be (256, 32 * 75 *␣
↪75) = (256, 180000)
        if self.print_toggle: print(f"Vectorize: {X.shape}")
```

```
        # Output layer
        X = self.output(X)
        if self.print_toggle: print(f"Layer output: {X.shape}")

        return X, convolution_layer1, convolution_layer2
```

[9]:
```
# create neural network
def create_network(is_print_toggle, learning_rate = 0.001):
    # create an model instance
    network = CNN_Net(is_print_toggle)

    # select loss function
    loss_fun = nn.CrossEntropyLoss()

    # select optimizer
    optimizer = torch.optim.Adam(network.parameters(), lr = learning_rate,␣
 ↪weight_decay=0.0001)

    return network, loss_fun, optimizer
```

[10]:
```
# test the model with one batch (sanity check)
network, loss_fun, optimizer = create_network(is_print_toggle = True)

X, y = iter(train_loader).next()
yHat = network(X)[0]  # here, we dont need the convolution layers' outputs

# check sizes of model outputs and target variable
print(' ')
print(yHat.shape)
print(y.shape)

# now let's compute the loss
loss = loss_fun(yHat, y)
print(' ')
print('Loss:')
print(loss)
```

```
Input: torch.Size([256, 3, 150, 150])
conv1/max_pool: torch.Size([256, 12, 75, 75])
Layer conv2/max_pool: torch.Size([256, 12, 75, 75])
Layer conv3/max_pool: torch.Size([256, 32, 75, 75])
Vectorize: torch.Size([256, 180000])
Layer output: torch.Size([256, 6])

torch.Size([256, 6])
torch.Size([256])
```

```
Loss:
tensor(1.8292, grad_fn=<NllLossBackward0>)
```

[11]:
```python
# summary of the model
network, loss_fun, optimizer = create_network(is_print_toggle = False)

if device == "cpu":
    summary(network.to(device), (3, 150, 150), device=device);
else:
    summary(network.to(device), (3, 150, 150));
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 12, 150, 150]             336
       BatchNorm2d-2         [-1, 12, 150, 150]              24
              ReLU-3         [-1, 12, 150, 150]               0
         MaxPool2d-4           [-1, 12, 75, 75]               0
            Conv2d-5           [-1, 20, 75, 75]           2,180
              ReLU-6           [-1, 20, 75, 75]               0
            Conv2d-7           [-1, 32, 75, 75]           5,792
       BatchNorm2d-8           [-1, 32, 75, 75]              64
              ReLU-9           [-1, 32, 75, 75]               0
           Linear-10                    [-1, 6]       1,080,006
================================================================
Total params: 1,088,402
Trainable params: 1,088,402
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.26
Forward/backward pass size (MB): 12.53
Params size (MB): 4.15
Estimated Total Size (MB): 16.94
----------------------------------------------------------------
```

[12]:
```python
# create a function to train the model
def train_the_model(network, loss_fun, optimizer, epochs = 100):
    # number of epochs
    epochs = epochs

    # send the model to GPU'
    network.to(device)

    # initialize losses
    train_loss     = torch.zeros(epochs)
    test_loss      = torch.zeros(epochs)
```

```
    train_accuracy = []
    test_accuracy  = []

    # loop over epochs
    for epoch in range(epochs):
        # To calcuate the execution time,
        start_time = time.time()

        # switch on the training mode
        network.train()

        batch_accuracy = []
        batch_loss     = []
        # loop over the training data batches
        for X, y in train_loader:
            # push X and y to GPU
            X = X.to(device)
            y = y.to(device)

            # forward propagation
            yHat = network(X)[0]  # here, we dont need the convolution layers'␣
↪outputs

            # compute loss
            loss = loss_fun(yHat, y)

            # backward propagation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # store the loss from this batch
            batch_loss.append(loss.item())

            # compute accuracy
            accuracy_percentage = accuracy_score(y.detach().cpu(), torch.
↪argmax(yHat, axis=1).detach().cpu()) * 100
            batch_accuracy.append( accuracy_percentage )
        # end of batch loop

        # now that we've trained through the batches, get their average train␣
↪accuracy
        train_acc = np.mean(batch_accuracy)
        train_accuracy.append( train_acc )

        # and get average losses across the batches
        train_loss[epoch] = np.mean( batch_loss )
```

```
        # switch to evaluation mode to calculate test accuracy
        network.eval()

        # # extract X,y from test dataloader
        # X, y = next(iter(test_loader))

        temp_test_loss = []  # to store loss temporarily
        temp_test_acc  = []  # to store accuracy temporarily
        # loop through the test batches
        for X, y in test_loader:
            # push the data to GPU
            X = X.to(device)
            y = y.to(device)

            with torch.no_grad():  # deactivate autograd
                # calculate prediction
                yHat = network(X)[0]  # here, we dont need the convolution␣
↪layers' outputs
                # calculate loss
                loss = loss_fun(yHat, y)
                # and store it
                temp_test_loss.append(loss.item())

                # calculate accuracy and store it
                accuracy_percentage = accuracy_score(y.detach().cpu(), torch.
↪argmax(yHat, axis=1).detach().cpu()) * 100
                temp_test_acc.append( accuracy_percentage )

        # calculate overall test loss and store it
        test_loss[epoch] = np.mean(temp_test_loss)

        # calculate the test accuracy
        test_acc = np.mean(temp_test_acc)
        test_accuracy.append(test_acc)

        # calculate end time
        end_time = time.time()

        # display a status message
        msg = f"Completed: {epoch + 1}/{epochs} - took: {(end_time -␣
↪start_time):.6f} sec - Train accuracy: {train_acc:.5f} - Test accuracy:␣
↪{test_acc:.5f}"
        sys.stdout.write("\n" + msg)


    # end epochs
```

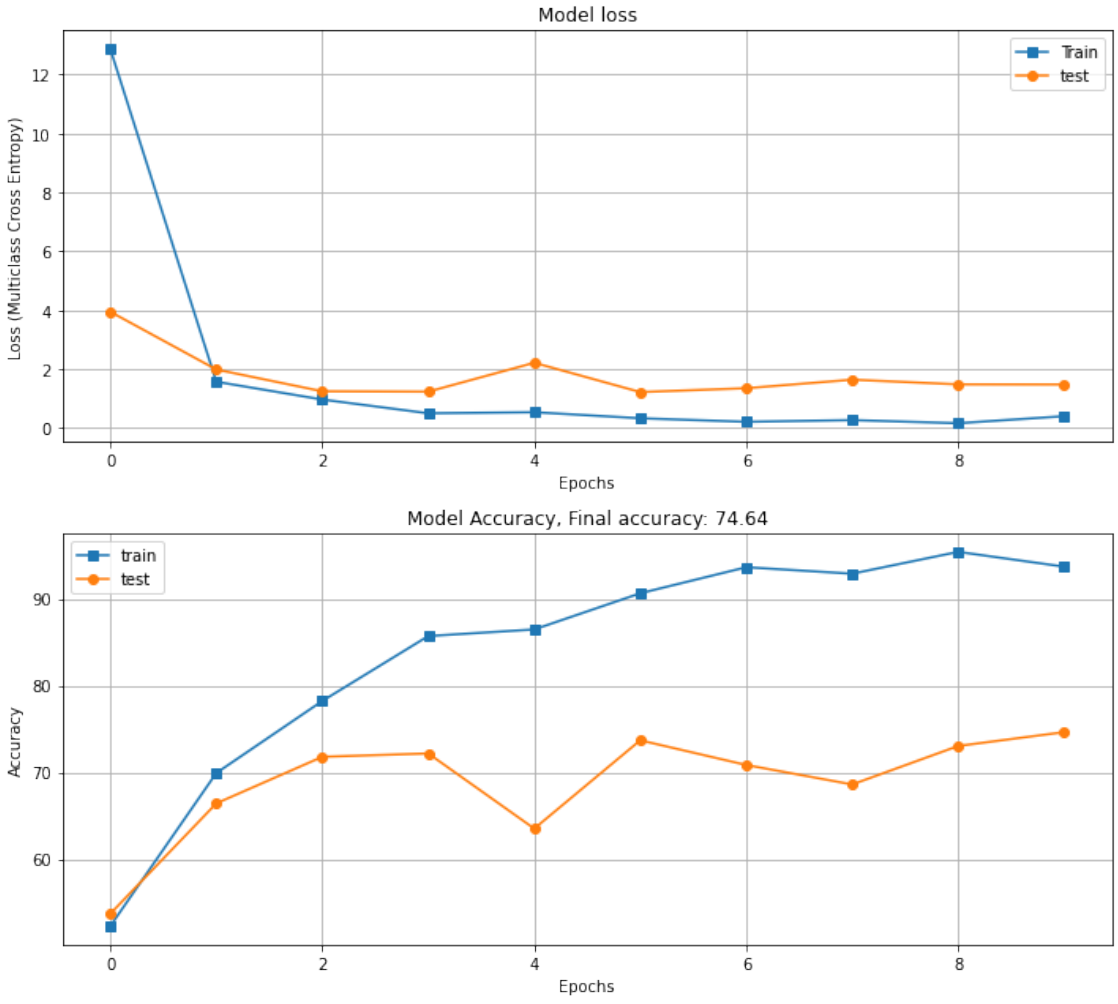
```
    # function output
    return train_loss, test_loss, train_accuracy, test_accuracy, network
```

[13]:
```
%%time


# create a neural network
network, loss_fun, optimizer = create_network(is_print_toggle = False)
# train the model
train_loss, test_loss, train_accuracy, test_accuracy, network =␣
 ↪train_the_model(network, loss_fun, optimizer, epochs = 10)
```

```
Completed: 1/10 - took: 88.125844 sec - Train accuracy: 52.29742 - Test
accuracy: 53.71518
Completed: 2/10 - took: 43.782111 sec - Train accuracy: 69.92796 - Test
accuracy: 66.42889
Completed: 3/10 - took: 44.413203 sec - Train accuracy: 78.23410 - Test
accuracy: 71.79857
Completed: 4/10 - took: 44.385618 sec - Train accuracy: 85.71659 - Test
accuracy: 72.18354
Completed: 5/10 - took: 44.781815 sec - Train accuracy: 86.47497 - Test
accuracy: 63.52327
Completed: 6/10 - took: 44.121289 sec - Train accuracy: 90.64516 - Test
accuracy: 73.69933
Completed: 7/10 - took: 44.227841 sec - Train accuracy: 93.64854 - Test
accuracy: 70.85598
Completed: 8/10 - took: 45.662771 sec - Train accuracy: 92.90192 - Test
accuracy: 68.60988
Completed: 9/10 - took: 44.171983 sec - Train accuracy: 95.42012 - Test
accuracy: 73.04688
Completed: 10/10 - took: 44.237915 sec - Train accuracy: 93.71199 - Test
accuracy: 74.64193CPU times: user 4min 56s, sys: 56 s, total: 5min 52s
Wall time: 8min 7s
```

[14]:
```
# display the accuracy of our model
fig, ax = plt.subplots(2, 1, figsize = (10, 9))

ax[0].plot(train_loss, "s-", label = "Train")
ax[0].plot(test_loss, "o-", label = "test")
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Loss (Multiclass Cross Entropy)")
ax[0].set_title("Model loss")
ax[0].legend()
ax[0].grid()
```

```
ax[1].plot(train_accuracy, "s-", label = "train")
ax[1].plot(test_accuracy, "o-", label = "test")
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Accuracy")
ax[1].set_title(f"Model Accuracy, Final accuracy: {test_accuracy[-1]:.2f}")
ax[1].legend()
ax[1].grid()


plt.tight_layout()
plt.show()
```

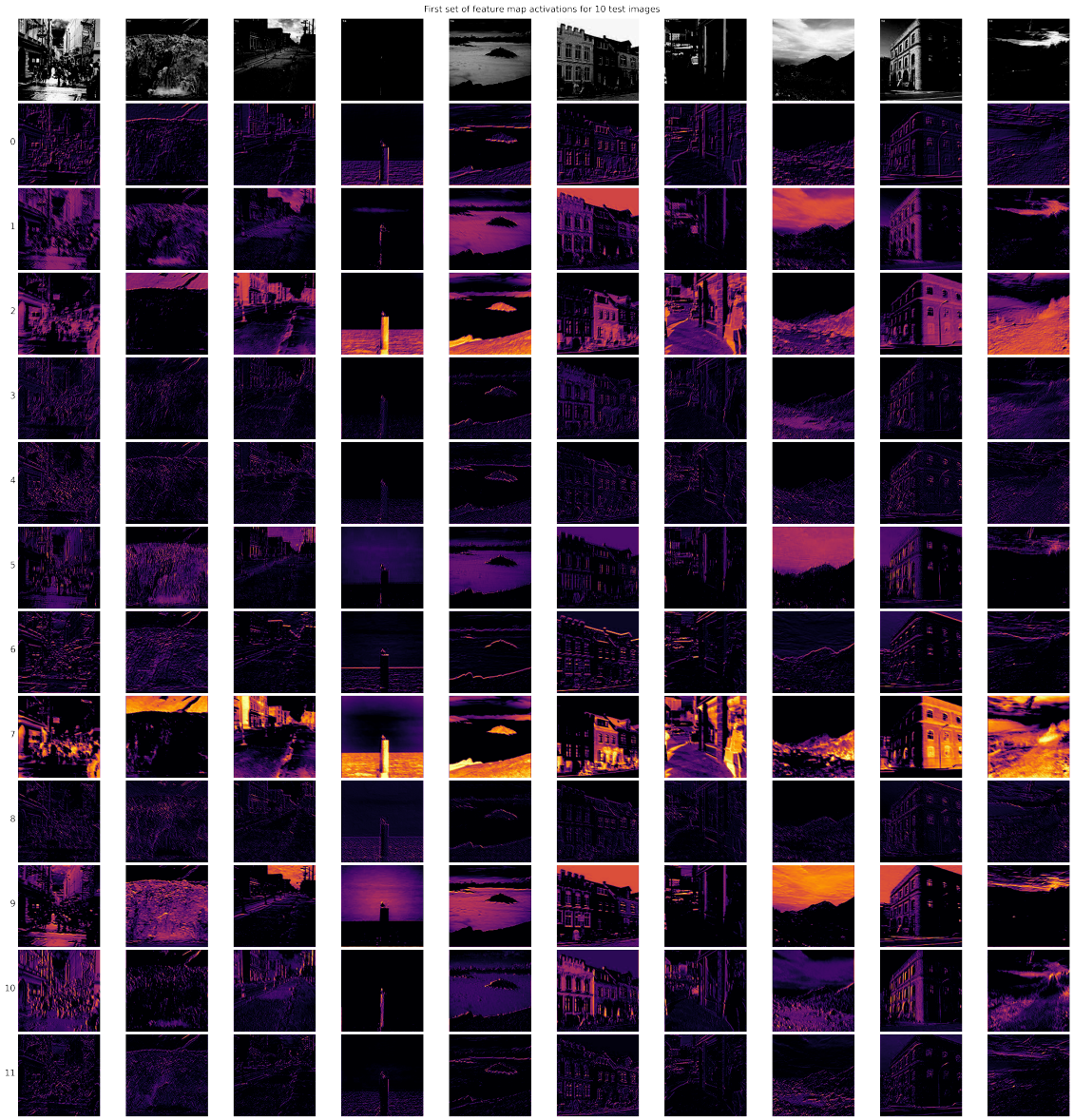


[15]:
```
### mow let's look though the convolution channels ###

# extract a bact from the test set
X, y = next(iter(test_loader))
```

```
# move the data to GPU
X = X.to(device)
y = y.to(device)

# prediction and out convolution layers
yHat, feature_maps1, feature_maps2= network(X)
# check the shape of the two feature maps
print(feature_maps1.shape, feature_maps2.shape)
```

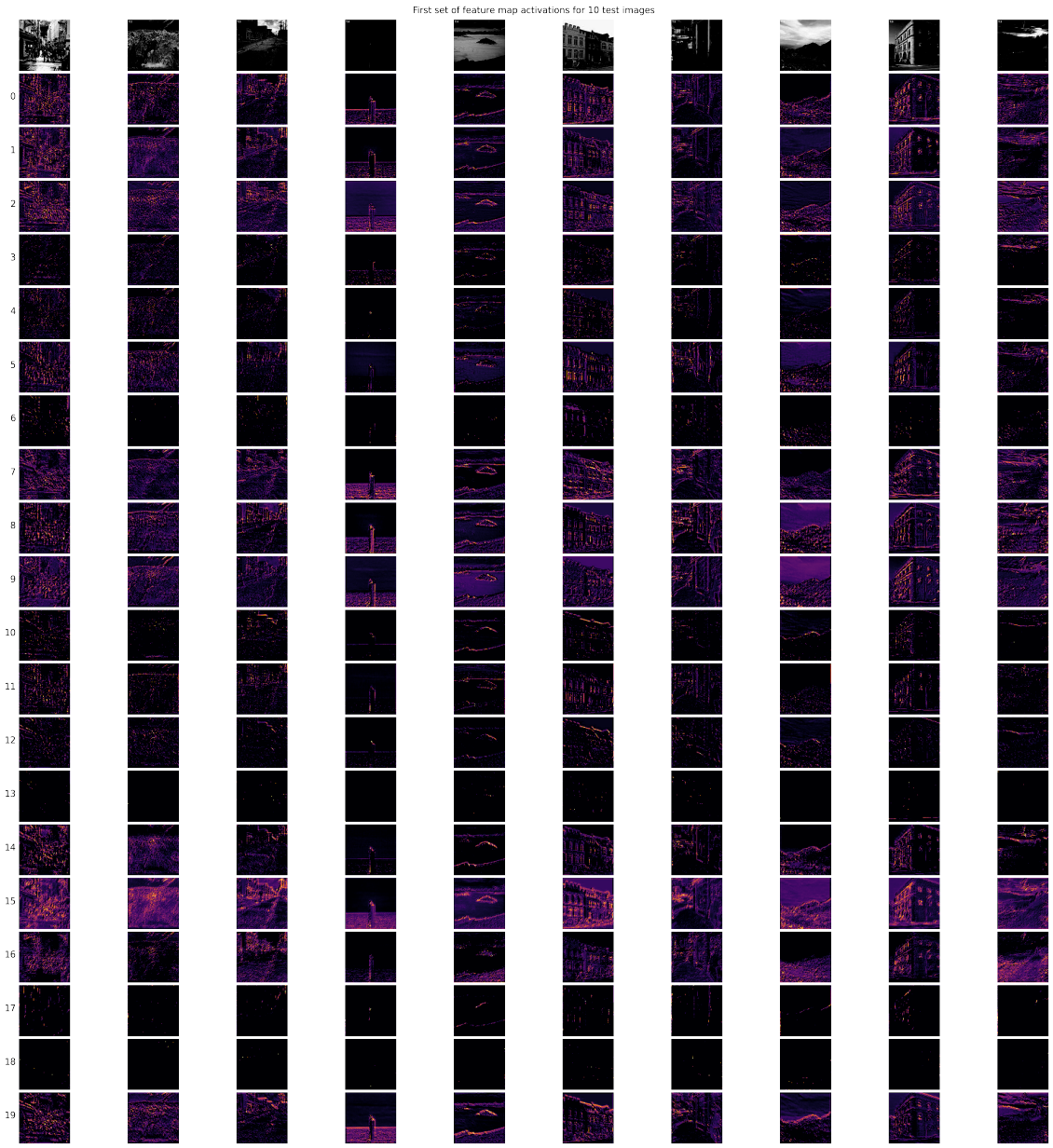
```
torch.Size([256, 12, 150, 150]) torch.Size([256, 20, 75, 75])
```

[16]:
```
# Feature maps from the convolution layer 1

number_of_images = 10

fig,axs = plt.subplots(feature_maps1.shape[1] + 1, number_of_images,␣
 ↪figsize=(70, 70))

for pici in range(number_of_images):

    # show the original picture
    img = X[pici, 0, :, :].detach().cpu()
    axs[0, pici].imshow(img, cmap="gray", vmin=0, vmax=1)
    axs[0, pici].axis('off')
    axs[0, pici].text(2,2,'T:%s'%int(y[pici].
 ↪item()),ha='left',va='top',color='w',fontweight='bold')

    for feati in range(feature_maps1.shape[1]):
        # extract the feature map from this image
        img = feature_maps1[pici,feati,:,:].detach().cpu()
        axs[feati+1,pici].imshow(img,cmap='inferno',vmin=0,vmax=torch.max(img)*.
 ↪9)
        axs[feati+1,pici].axis('off')
        axs[feati+1,pici].text(-5,feature_maps1.shape[2]/2,feati,ha='right',␣
 ↪fontsize=40) if pici==0 else None

plt.tight_layout()
plt.suptitle(f'First set of feature map activations for {number_of_images} test␣
 ↪images',x=.5,y=1.01, fontsize=40)

# save the plot
# plt.savefig('feature_maps1.png', dpi=300, bbox_inches='tight')
plt.show()
```



[18]:
```
# Feature maps from the convolution layer 2

number_of_images = 10

fig,axs = plt.subplots(feature_maps2.shape[1] + 1, number_of_images,␣
↪figsize=(70, 70))

for pici in range(number_of_images):

    # show the original picture
    img = X[pici, 0, :, :].detach().cpu()
```

```
    axs[0, pici].imshow(img, cmap="gray", vmin=0, vmax=1)
    axs[0, pici].axis('off')
    axs[0, pici].text(2,2,'T:%s'%int(y[pici].
 ↪item()),ha='left',va='top',color='w',fontweight='bold')

    for feati in range(feature_maps2.shape[1]):
        # extract the feature map from this image
        img = feature_maps2[pici,feati,:,:].detach().cpu()
        axs[feati+1,pici].imshow(img,cmap='inferno',vmin=0,vmax=torch.max(img)*.
 ↪9)
        axs[feati+1,pici].axis('off')
        axs[feati+1,pici].text(-5,feature_maps2.shape[2]/2,feati,ha='right',␣
 ↪fontsize=40) if pici==0 else None

plt.tight_layout()
plt.suptitle(f'First set of feature map activations for {number_of_images} test␣
 ↪images',x=.5,y=1.01, fontsize=40)

# save the plot
# plt.savefig('feature_maps2.png', dpi=300, bbox_inches='tight')
plt.show()
```


First set of feature map activations for 10 test images
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

[ ]: