**Feature Extraction in CNN**

[[Sharath S Hebbar](https://medium.com/@sharathhebbar24?source=post_page-----49f955a00cb--------------------------------)](https://medium.com/@sharathhebbar24?source=post_page-----49f955a00cb--------------------------------)

[Sharath S Hebbar](https://medium.com/@sharathhebbar24?source=post_page-----49f955a00cb--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fa5947dc8a8ee&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40sharathhebbar24%2Ffeature-extraction-in-cnn-49f955a00cb&user=Sharath+S+Hebbar&userId=a5947dc8a8ee&source=post_page-a5947dc8a8ee----49f955a00cb---------------------post_header-----------)

5 min read

·

Jul 9

152

[**GitHub**](https://github.com/SharathHebbar)[**LinkedIn**](https://www.linkedin.com/in/sharath-s-hebbar-14b541172/)[**Medium**](https://medium.com/@sharathhebbar24)[**Portfolio**](https://www.notion.so/ssh-blogs/Sharath-S-Hebbar-e7a58e16a1154467a74c1f1db1db0569)[**Substack**](https://sharathshebbar.substack.com/)

Often times you wonder what happens behind the scenes or what happens when we pass the input onto each layer.

The following code snippet illustrates how to gain insights into what happens behind the scenes and how the input is processed through each layer. For this demonstration, a Kaggle dataset called “Intel Image Classification” is used:

**Importing Libraries**

import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torchvision  
import torchvision.transforms as transforms  
from torchvision import datasets, models, transforms  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns

**Loading Intel Image Classification**

train\_data\_dir = '/kaggle/input/intel-image-classification/seg\_train/seg\_train'  
test\_data\_dir = '/kaggle/input/intel-image-classification/seg\_test/seg\_test'  
pred\_data\_dir = '/kaggle/input/intel-image-classification/seg\_pred/seg\_pred'

**Hyperparameter**

kernel\_size\_cnn = 3  
kernel\_size\_pool = 2  
stride1 = 1  
stride2 = 2  
padding = 1  
batch\_size = 256  
epochs = 50  
ratio = 64  
in\_channels = 3  
hidden\_channels1 = 64  
hidden\_channels2 = 128

Tune this as you need

**Finding the mean and standard deviation of the dataset**

train = datasets.ImageFolder(train\_data\_dir,  
 transform = transforms.Compose([  
 transforms.Resize(ratio),  
 transforms.RandomCrop(ratio),  
 transforms.ToTensor()  
 ]))  
  
train\_data\_loader = torch.utils.data.DataLoader(train, 64, shuffle=True, num\_workers=3, pin\_memory=True)  
  
def get\_mean\_std(data\_loader):  
 sum\_, squared\_sum, batches = 0, 0, 0  
 for data, \_ in data\_loader:  
 sum\_ += torch.mean(data, dim=([0, 2, 3]))  
 squared\_sum += torch.mean(data \*\* 2, dim=([0, 2, 3]))  
 batches += 1  
   
 mean = sum\_ / batches  
 std = (squared\_sum / batches - mean \*\* 2) \*\* 0.5  
 return mean, std  
  
mean, std = get\_mean\_std(train\_data\_loader)  
mean, std

**Transforms**

train\_transform = transforms.Compose([  
 transforms.RandomResizedCrop(ratio),  
 transforms.RandomHorizontalFlip(),  
 transforms.ToTensor(),  
 transforms.Normalize(mean, std)])  
  
test\_transform = transforms.Compose([  
 transforms.Resize(ratio),  
 transforms.CenterCrop(ratio),  
 transforms.ToTensor(),  
 transforms.Normalize(mean, std)])

**Preparing Dataset**

train\_datasets = datasets.ImageFolder(  
 os.path.join(train\_data\_dir),  
 transform=train\_transform  
)  
  
test\_datasets = datasets.ImageFolder(  
 os.path.join(test\_data\_dir),  
 transform=test\_transform  
)  
  
classes = len(train\_datasets.classes)  
  
val\_size = int(len(train\_datasets) \* 0.2)  
train\_size = len(train\_datasets) - val\_size  
  
train\_datasets, val\_datasets = torch.utils.data.random\_split(  
 train\_datasets,  
 [train\_size, val\_size]  
)  
  
  
train\_dataloaders = torch.utils.data.DataLoader(  
 train\_datasets,   
 batch\_size=batch\_size \* 2,   
 shuffle=True,  
 num\_workers=2,   
 pin\_memory=True  
)  
  
val\_dataloaders = torch.utils.data.DataLoader(  
 val\_datasets,  
 batch\_size=batch\_size \* 2,  
 num\_workers=2,  
 pin\_memory=True  
)  
  
test\_dataloaders = torch.utils.data.DataLoader(  
 test\_datasets,  
 batch\_size=batch\_size \* 2,  
 num\_workers=2,  
 pin\_memory=True  
)

**Invoking CUDA**

CUDA = torch.cuda.is\_available()  
CUDA

**Feature Representation**

from PIL import Image  
img = Image.open('/kaggle/input/intel-image-classification/seg\_train/seg\_train/forest/10007.jpg')  
image = train\_transform(img)  
image = image.unsqueeze(0)  
if CUDA:  
 image = image.cuda()  
image.shape

**Simple CNN Model**

class CNN11(nn.Module):  
 def \_\_init\_\_(self,  
 in\_channels,  
 hidden\_channels1,  
 hidden\_channels2,  
 out\_channels,  
 kernel\_size\_cnn=3,  
 kernel\_size\_pool=2,  
 stride1=1,  
 stride2=2,  
 padding=1 ):  
super().\_\_init\_\_()  
   
self.cnn1 = nn.Conv2d(  
 in\_channels=in\_channels,  
 out\_channels=hidden\_channels1,  
 kernel\_size=kernel\_size\_cnn,  
 stride=stride1,  
 padding=padding  
)  
  
 self.batchnorm1 = nn.BatchNorm2d(hidden\_channels1)  
 self.relu = nn.ReLU()  
 self.maxpool = nn.MaxPool2d(kernel\_size=kernel\_size\_pool)  
   
 self.cnn2 = nn.Conv2d(in\_channels=hidden\_channels1,  
 out\_channels=hidden\_channels2,  
 kernel\_size=kernel\_size\_cnn,  
 stride=stride1,  
 padding=padding)  
 self.batchnorm2 = nn.BatchNorm2d(hidden\_channels2)  
   
 self.fc1 = nn.Flatten()  
 self.dropout = nn.Dropout(0.4)  
 self.linear\_layer = 32768  
 self.fc2 = nn.Linear(self.linear\_layer, out\_channels)  
   
 def forward(self, x):  
 outputs = []  
 labels = []  
out = self.cnn1(x)  
 outputs.append(out)  
 labels.append('conv2d')  
out = self.batchnorm1(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
out = self.relu(out)  
 outputs.append(out)  
 labels.append('ReLU')  
out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('Maxpool')  
   
 out = self.cnn2(out)  
 outputs.append(out)  
 labels.append('conv2d')  
out = self.batchnorm2(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
out = self.relu(out)  
 outputs.append(out)  
 labels.append('ReLU')  
out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('Maxpool')  
   
 out = self.fc1(out)  
 out = self.relu(out)  
 out = self.fc2(out)  
 return outputs, labels

**Model**

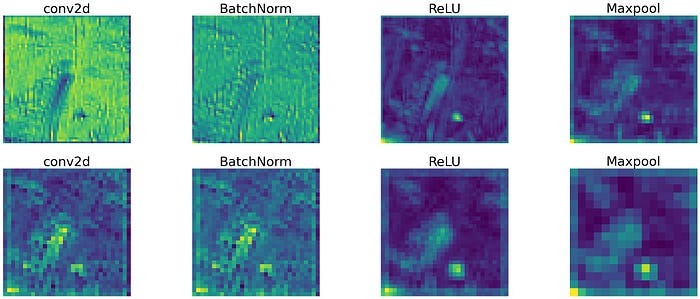
model11 = CNN11(in\_channels=in\_channels,  
 hidden\_channels1=hidden\_channels1,  
 hidden\_channels2=hidden\_channels2,  
 out\_channels=classes)  
if CUDA:  
 model11 = model11.cuda()  
outputs, labels = model11(image)

**preprocess**

processed = []  
for feature\_map in outputs:  
 feature\_map = feature\_map.squeeze(0)  
 gray\_scale = torch.sum(feature\_map, 0)  
 gray\_scale = gray\_scale / feature\_map.shape[0]  
 processed.append(gray\_scale.data.cpu().numpy())

**Display Image**

fig = plt.figure(figsize=(30, 50))  
for i in range(len(processed)):  
 a = fig.add\_subplot(8, 4, i + 1)  
 imgplot = plt.imshow(processed[i])  
 plt.axis('off')  
 a.set\_title(labels[i], fontsize=30)  
   
plt.savefig('feature\_maps1.jpg', bbox\_inches='tight')



Feature Extracted from CNN of every layer

**Try2: CNN with additional Layer**

class CNN2(nn.Module):  
 # in\_channels is the color channels in our case it is 3  
 def \_\_init\_\_(self, in\_channels, hidden\_channels1, hidden\_channels2, out\_channels,  
 kernel\_size\_cnn=3, kernel\_size\_pool=2,  
 stride1=1, stride2=2, padding=1):  
   
 super().\_\_init\_\_()  
 # padding\_value = (kernel\_size - 1) / 2 => (3–1)/ 2 => 1  
   
 self.cnn1 = nn.Conv2d(in\_channels=in\_channels,  
 out\_channels=hidden\_channels1,  
 kernel\_size=kernel\_size\_cnn,  
 stride=stride1,  
 padding=padding)  
 self.batchnorm1 = nn.BatchNorm2d(hidden\_channels1)  
 self.relu = nn.ReLU()  
 self.maxpool = nn.MaxPool2d(kernel\_size=kernel\_size\_pool)  
self.cnn2 = nn.Conv2d(in\_channels=hidden\_channels1,  
 out\_channels=hidden\_channels1,  
 kernel\_size=kernel\_size\_cnn,  
 stride=stride1,  
 padding=padding)  
 self.batchnorm2 = nn.BatchNorm2d(hidden\_channels2)  
self.fc1 = nn.Flatten()  
 self.dropout = nn.Dropout(0.4)  
 self.linear\_layer = 1024  
   
 self.fc2 = nn.Linear(self.linear\_layer, out\_channels)  
   
 def forward(self, x):  
   
 outputs = []  
 labels = []  
   
 out = self.cnn1(x)  
 outputs.append(out)  
 labels.append('conv2d')  
   
 out = self.batchnorm1(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
   
 out = self.relu(out)  
 outputs.append(out)  
 labels.append('ReLU')  
   
 out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('MaxPool')  
 # [256, 32, 32, 64]  
   
 out = self.cnn2(out)   
 labels.append('conv2d')  
 outputs.append(out)  
   
 out = self.batchnorm1(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
   
 out = self.relu(out)  
 labels.append('ReLU')  
 outputs.append(out)  
   
 out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('Maxpool')  
 # [256, 16, 16, 64]  
   
 out = self.cnn2(out)   
 labels.append('conv2d')  
 outputs.append(out)  
   
 out = self.batchnorm1(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
   
 out = self.relu(out)  
 labels.append('ReLU')  
 outputs.append(out)  
   
 out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('Maxpool')  
 # [256, 8, 8, 64]  
   
 out = self.cnn2(out)   
 labels.append('conv2d')  
 outputs.append(out)  
   
 out = self.batchnorm1(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
   
 out = self.relu(out)  
 labels.append('ReLU')  
 outputs.append(out)  
   
 out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('Maxpool')  
   
 # [256, 4, 4, 64]  
   
 out = self.fc1(out)  
 out = self.relu(out)  
 out = self.fc2(out)  
   
 return outputs, labels

**Model**

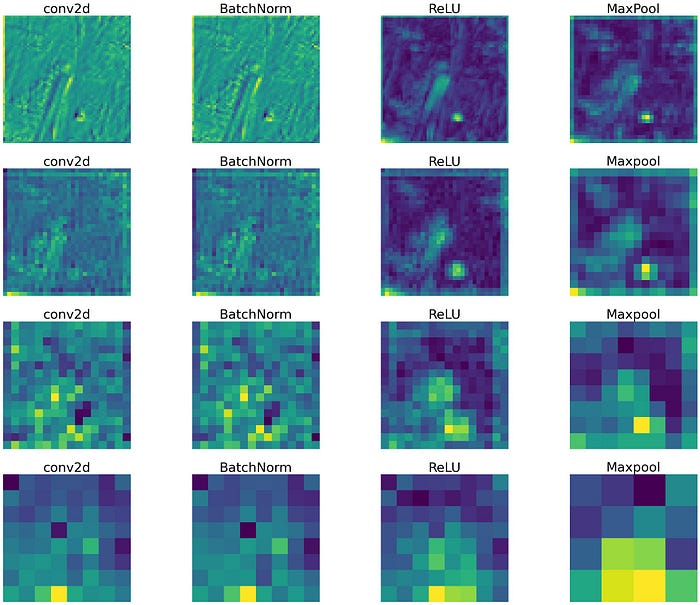
model1 = CNN2(in\_channels=in\_channels,  
 hidden\_channels1=hidden\_channels1,  
 hidden\_channels2=hidden\_channels2,  
 out\_channels=classes)  
if CUDA:  
 model1 = model1.cuda()  
outputs, labels = model1(image)

**Preprocess**

processed = []  
for feature\_map in outputs:  
 feature\_map = feature\_map.squeeze(0)  
 gray\_scale = torch.sum(feature\_map, 0)  
 gray\_scale = gray\_scale / feature\_map.shape[0]  
 processed.append(gray\_scale.data.cpu().numpy())

**Display Image**

fig = plt.figure(figsize=(30, 50))  
for i in range(len(processed)):  
 a = fig.add\_subplot(8, 4, i + 1)  
 imgplot = plt.imshow(processed[i])  
 plt.axis('off')  
 a.set\_title(labels[i], fontsize=30)  
   
plt.savefig('feature\_maps2.jpg', bbox\_inches='tight')



Feature Extracted from CNN of every layer

**Try 3: CNN without MaxPooling and stride of 2**

class CNN22(nn.Module):  
 # in\_channels is the color channels in our case it is 3  
 def \_\_init\_\_(self, in\_channels, hidden\_channels1, hidden\_channels2, out\_channels,  
 kernel\_size\_cnn=3, kernel\_size\_pool=2,  
 stride1=1, stride2=2, padding=1):  
   
 super().\_\_init\_\_()  
   
 self.cnn1 = nn.Conv2d(in\_channels=in\_channels,  
 out\_channels=hidden\_channels1,  
 kernel\_size=kernel\_size\_cnn,  
 stride=stride1,  
 padding=padding)  
 self.batchnorm1 = nn.BatchNorm2d(hidden\_channels1)  
 self.relu = nn.ReLU()  
 self.maxpool = nn.MaxPool2d(kernel\_size=kernel\_size\_pool)  
   
 self.cnn2 = nn.Conv2d(in\_channels=hidden\_channels1,  
 out\_channels=hidden\_channels1,  
 kernel\_size=kernel\_size\_cnn,  
 stride=stride1,  
 padding=padding)  
 self.batchnorm2 = nn.BatchNorm2d(hidden\_channels2)  
   
 self.fc1 = nn.Flatten()  
 self.dropout = nn.Dropout(0.4)  
 self.linear\_layer = 1024  
 self.fc2 = nn.Linear(self.linear\_layer, out\_channels)  
  
 def forward(self, x):  
   
 outputs = []  
 labels = []  
   
 out = self.cnn1(x)  
 outputs.append(out)  
 labels.append('conv2d')  
   
 out = self.batchnorm1(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
   
 out = self.relu(out)  
 outputs.append(out)  
 labels.append('ReLU')  
   
 out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('MaxPool')  
 # [256, 32, 32, 64]  
   
 out = self.cnn2(out)   
 labels.append('conv2d')  
 outputs.append(out)  
   
 out = self.batchnorm1(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
   
 out = self.relu(out)  
 labels.append('ReLU')  
 outputs.append(out)  
   
 out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('Maxpool')  
 # [256, 16, 16, 64]  
   
 out = self.cnn2(out)   
 labels.append('conv2d')  
 outputs.append(out)  
   
 out = self.batchnorm1(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
   
 out = self.relu(out)  
 labels.append('ReLU')  
 outputs.append(out)  
   
 out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('Maxpool')  
 # [256, 8, 8, 64]  
   
 out = self.cnn2(out)   
 labels.append('conv2d')  
 outputs.append(out)  
   
 out = self.batchnorm1(out)  
 outputs.append(out)  
 labels.append('BatchNorm')  
   
 out = self.relu(out)  
 labels.append('ReLU')  
 outputs.append(out)  
   
 out = self.maxpool(out)  
 outputs.append(out)  
 labels.append('Maxpool')  
   
 # [256, 4, 4, 64]  
   
 out = self.fc1(out)  
 out = self.relu(out)  
 out = self.fc2(out)  
   
 return outputs, labels

**Model**

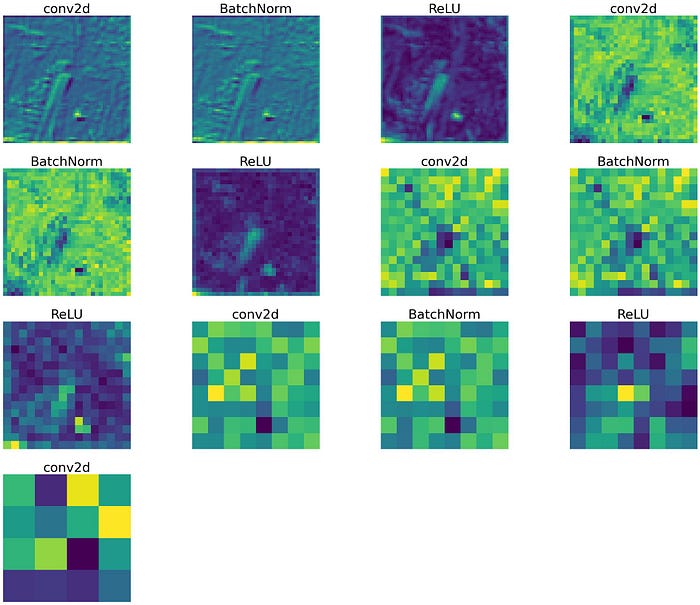
model1 = CNN3(in\_channels=in\_channels,  
 hidden\_channels1=hidden\_channels1,  
 hidden\_channels2=hidden\_channels2,  
 out\_channels=classes)  
if CUDA:  
 model1 = model1.cuda()  
outputs, labels = model1(image)

**Preprocess**

processed = []  
for feature\_map in outputs:  
 feature\_map = feature\_map.squeeze(0)  
 gray\_scale = torch.sum(feature\_map, 0)  
 gray\_scale = gray\_scale / feature\_map.shape[0]  
 processed.append(gray\_scale.data.cpu().numpy())

**Display Image**

fig = plt.figure(figsize=(30, 50))  
for i in range(len(processed)):  
 a = fig.add\_subplot(8, 4, i + 1)  
 imgplot = plt.imshow(processed[i])  
 plt.axis('off')  
 a.set\_title(labels[i], fontsize=30)  
   
plt.savefig('feature\_maps3.jpg', bbox\_inches='tight')



Feature Extracted from CNN of every layer

To conclude knowing the depth of every layer is important while building the model.

Reference:

**[ML-Project-list/kaggle-projects/Intel-Image-Classification/intel-image-classification-pytorch.ipynb…](https://github.com/SharathHebbar/ML-Project-list/blob/master/kaggle-projects/Intel-Image-Classification/intel-image-classification-pytorch.ipynb?source=post_page-----49f955a00cb--------------------------------" \t "_blank)**

[List of all ML projects. Contribute to SharathHebbar/ML-Project-list development by creating an account on GitHub.](https://github.com/SharathHebbar/ML-Project-list/blob/master/kaggle-projects/Intel-Image-Classification/intel-image-classification-pytorch.ipynb?source=post_page-----49f955a00cb--------------------------------" \t "_blank)

[github.com](https://github.com/SharathHebbar/ML-Project-list/blob/master/kaggle-projects/Intel-Image-Classification/intel-image-classification-pytorch.ipynb?source=post_page-----49f955a00cb--------------------------------" \t "_blank)

**[Intel-Image-Classification-Pytorch](https://www.kaggle.com/code/sharathshebbar/intel-image-classification-pytorch/edit/run/136121572?source=post_page-----49f955a00cb--------------------------------" \t "_blank)**

[Explore and run machine learning code with Kaggle Notebooks | Using data from Intel Image Classification](https://www.kaggle.com/code/sharathshebbar/intel-image-classification-pytorch/edit/run/136121572?source=post_page-----49f955a00cb--------------------------------" \t "_blank)

[www.kaggle.com](https://www.kaggle.com/code/sharathshebbar/intel-image-classification-pytorch/edit/run/136121572?source=post_page-----49f955a00cb--------------------------------" \t "_blank)

**[Welcome to PyTorch Tutorials - PyTorch Tutorials 2.0.1+cu117 documentation](https://pytorch.org/tutorials/?source=post_page-----49f955a00cb--------------------------------" \t "_blank)**

[Join the PyTorch developer community to contribute, learn, and get your questions answered.](https://pytorch.org/tutorials/?source=post_page-----49f955a00cb--------------------------------" \t "_blank)

[pytorch.org](https://pytorch.org/tutorials/?source=post_page-----49f955a00cb--------------------------------" \t "_blank)

**[Data-Science-and-ML/articles/feature extraction at main · SharathHebbar/Data-Science-and-ML](https://github.com/SharathHebbar/Data-Science-and-ML/tree/main/articles/feature%20extraction?source=post_page-----49f955a00cb--------------------------------" \t "_blank)**

[Data science and ML articles and codes. Contribute to SharathHebbar/Data-Science-and-ML development by creating an…](https://github.com/SharathHebbar/Data-Science-and-ML/tree/main/articles/feature%20extraction?source=post_page-----49f955a00cb--------------------------------" \t "_blank)

[github.com](https://github.com/SharathHebbar/Data-Science-and-ML/tree/main/articles/feature%20extraction?source=post_page-----49f955a00cb--------------------------------" \t "_blank)