EE 519: Deep Learning Theory & Fundamentals **Spring 2023**

Homework 2
Due: May 12, 2023, 11:59PM PT

Student Name: Andy Franck *Instructor Name: John Lipor*

Problem 1 `conv.py` (8 pts)

For the first problem of this assignment, you will complete a convolutional layer for your MyTorch package. Your task is to complete `conv.py` as detailed by the documentation in the corresponding file and the lecture notes. In particular, you will implement the forward and backward passes of a two-dimensional convolutional layer. For simplicity, we will ignore the bias term in this layer.

Turn in the output of the notebook `Conv2DTester.ipynb`.

[H]

Compare `forward()`

```
tic = time.time()
torch_out = net(Xt)
toc = time.time()
torch_time = toc - tic

tic = time.time()
my_out = my_net.forward(X)
toc = time.time()
my_time = toc - tic

print('Difference:', np.linalg.norm(my_out - torch_out.data.numpy()))
print('\nTorch Time:', torch_time, '\nMy Time:', my_time)
```

[22] ✓ 18.8s Python

```
Difference: 4.050074614665094e-05

Torch Time: 0.001573801040649414
My Time: 18.539093494415283
```

Compare `backward` and gradients

```
optimizer = torch.optim.SGD(net.parameters(), lr=0.1, momentum=0.0)
optimizer.zero_grad()
torch_loss_fn = nn.MSELoss()
torch_loss = torch_loss_fn(torch_out, Yt)
torch_loss.backward(retain_graph=True)
torch_dLdW = net.weight.grad.data

my_mse_fn = mynn.MSELoss()
my_mse = my_mse_fn.forward(torch_out.detach().numpy(), Yt.detach().numpy())
dLdO = my_mse_fn.backward()
dLdX = my_net.backward(dLdO)
my_dLdW = my_net.dLdW

print('Difference in dLdW:', np.linalg.norm(my_dLdW - torch_dLdW.data.numpy()))

# differences in dLdX
Xt = torch.tensor(X, requires_grad=True).float()
Xt.retain_grad()
Yt = torch.tensor(Y, requires_grad=True).float()
Yt.retain_grad()
torch_out = net(Xt)
torch_loss = torch_loss_fn(torch_out, Yt)
torch_loss.backward(retain_graph=True)
torch_dLdX = Xt.grad.data
torch_dLdO = Yt.grad.data

print('Difference in dLdX:', np.linalg.norm(torch_dLdX - dLdX))
```

✓ 1.3s Python

```
Difference in dLdW: 3.583056e-06
Difference in dLdX: 4.2715074690622777e-10
```

Problem 2 CNNs on CIFAR-10 (3 pts each)

The textbook focuses heavily on the Fashion-MNIST dataset since it is easy to quickly train a somewhat-accurate network on this dataset. However, this dataset may be too easy to develop an understanding of what works well and what doesn't in a CNN architecture. In this problem, we'll consider the more difficult (though still manageable) CIFAR-10 dataset, which is of similar size to Fashion-MNIST but has three

input channels and is considered a much more challenging dataset. This dataset can be accessed through `torchvision.datasets` in a manner similar to Fashion-MNIST.

Your task will be to train two CNNs on this dataset—one that utilizes a linear output layer and a second that uses global average pooling. You may make use of any available literature or blogs to gain insight into effective network architectures for this dataset. Your goal is to achieve the best accuracy you can while allowing time for the rest of the problems, which will utilize your networks. You **must** try both dropout and batch normalization for each architecture and report their impact on performance.

(a) Implement a CNN in the vein of LeNet, AlexNet, or VGG that uses one or more convolutional layers, followed by one or more dense layers. Turn in:

(a) final architecture

[H]
```
class model(nn.Module):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        #self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.LazyConv2d(96, kernel_size=11, stride=4, padding=1),
            nn.ReLU(), nn.MaxPool2d(kernel_size=3, stride=2),
            nn.LazyConv2d(256, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.LazyConv2d(384, kernel_size=3, padding=1), nn.ReLU(),
            nn.LazyConv2d(384, kernel_size=3, padding=1), nn.ReLU(),
            nn.LazyConv2d(256, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(kernel_size=3, stride=2), nn.Flatten(),
            nn.LazyLinear(4096), nn.ReLU(), nn.Dropout(p=0.5),
            nn.LazyLinear(4096), nn.ReLU(),nn.Dropout(p=0.5),
            nn.LazyLinear(num_classes))
```

(b) how I arrived at this

Answer:

To develop the basic architecture, I followed the basic LeNet design. However, I incorporated both batch normalization and dropout into my model architecture. Both batch normalization and dropout layers were determined by trial and error. I found the most performance increase when dropout was applied to the first dense layer. In some cases, applying dropout would cause the model to generate nan values. Batch normalization was most effective when applied after the grouping of three convolutional layers, so that is where I placed it.

(c) impact of batch normalization

Answer:

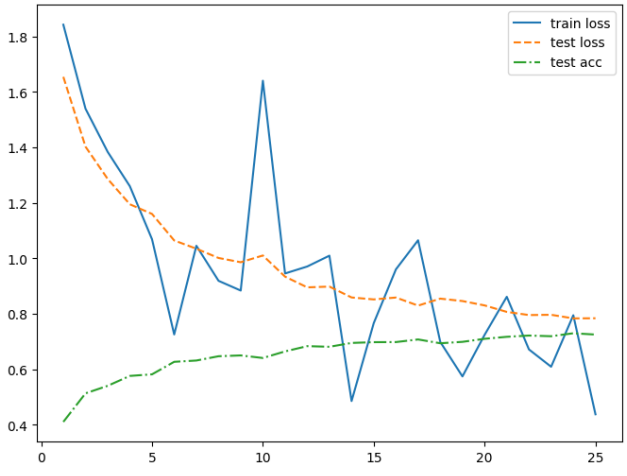
Batch normalization had a huge impact on the performance of the LeNet-like model. The performance increase from just one batch normalization layer was enormous. The model performance went from 61.2% to 68.0%.

(d) impact of dropout

Answer:

Dropout had a fairly large impact on the performance of the LeNet-like model. It allowed me to train for more epochs without overfitting, which in turn increased the performance of the model. The model performance went from 61.2% to 63.8%.

(e) plot of training and validation loss

[H]


(f) final validation accuracy **Final validation accuracy: 69.0%**

(b) Implement a CNN in the vein of NiN, ResNet, etc. that integrates nonlinearities within the convolutional portions and utilizes global average pooling. Turn in the same items as for part (a).

(a) final architecture

[H]
```python
def nin_block(out_channels, kernel_size, strides, padding):
    return nn.Sequential(
        nn.LazyConv2d(out_channels, kernel_size, strides, padding), nn.ReLU(),
        nn.LazyConv2d(out_channels, kernel_size=1), nn.ReLU(),
        nn.LazyConv2d(out_channels, kernel_size=1),
        #batch normalization implementation
        nn.LazyBatchNorm2d(),
        nn.ReLU())

class model(nn.Module):
    def __init__(self, lr=0.1, num_classes=10):
        super().__init__()
        #self.save_hyperparameters()
        self.net = nn.Sequential(
            nin_block(96, kernel_size=5, strides=3, padding=0),
            nn.MaxPool2d(3, stride=2),
            nin_block(256, kernel_size=3, strides=1, padding=2),
            nn.MaxPool2d(3, stride=2),
            nn.Dropout(0.5),
            nin_block(num_classes, kernel_size=3, strides=1, padding=1),
            nn.AdaptiveAvgPool2d((1, 1)),
            nn.Flatten())
```

(b) how I arrived at this

Answer:

Similarly to the LeNet-like model, I maainly followed the basic NiN design. Dropout was already incorporated into the model, but I experimented with removing it to compare results. It reduced performance considerably and the model overfit much earlier.

(c) impact of batch normalization

Answer:

Batch normalization had little impact on the performance of the model. The model with batch normalization had a final validation accuracy of 74.6% while the model without batch normalization had a final validation accuracy of 74.4%. The model with batch normalization also took longer to train than the model without batch normalization.

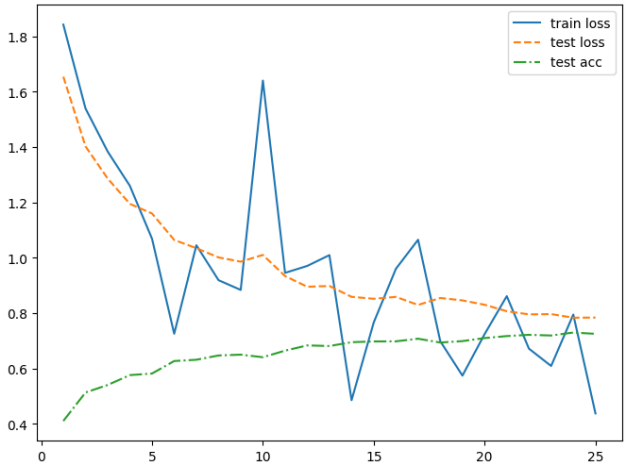
(d) impact of dropout

Answer:

Dropout didn't impact moder performance very much directy. Howevero, similarly to LeNet, it enabled me to train on more epochs without the model overfitting. The model with dropout had a

final validation accuracy of 74.6% while the model without dropout had a final validation accuracy of 70.1%. The model with dropout also took longer to train than the model without dropout.

(e) plot of training and validation loss



[H]

(f) final validation accuracy
Final validation accuracy: 74.6%

Problem 3 Feature Maps and Filters (3 pts each)

As discussed in class, interpreting CNNs is largely performed by visualizing their different elements. In this problem, your task is to visualize both the intermediate feature maps (outputs of convolutional layers) as well as some of the learned filters for each of the two networks trained in Problem 2. For each network, turn in:

(a) images of subplots of feature maps for multiple convolutional layers. Each image should contain a number of subplots corresponding to the number of output channels for a given convolutional layer, and you should create one image per convolutional layer for no more than three layers.

Model 1 (LeNet-like):

Figure 1: LeNet-like model. Both images are of two seperate layers of the model. The maps correspond to a correctly labeled airplane.

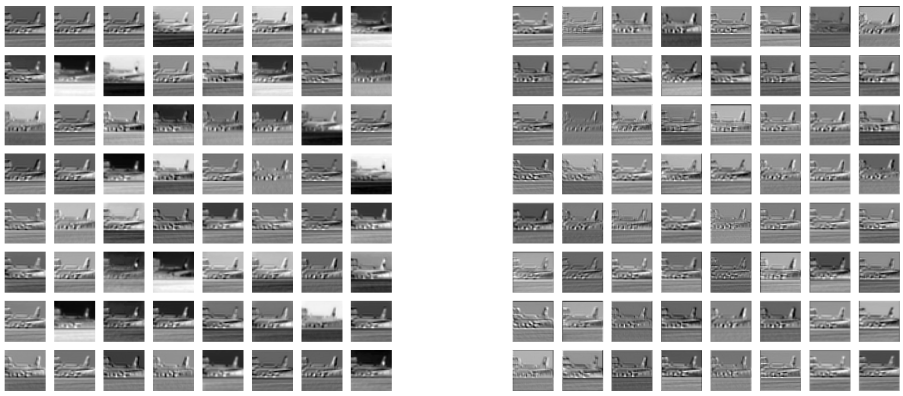
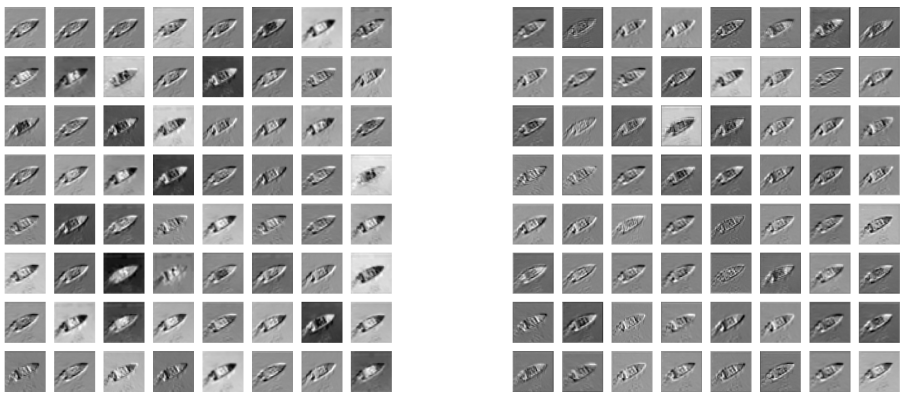


Figure 2: LeNet-like model. Both images are of two seperate layers of the model. The maps correspond to an incorrectly labeled ship.



Model 2 (NiN-like):

Figure 3: NiN-like model. Both images are of two seperate layers of the model. The maps correspond to a correctly labeled dog.

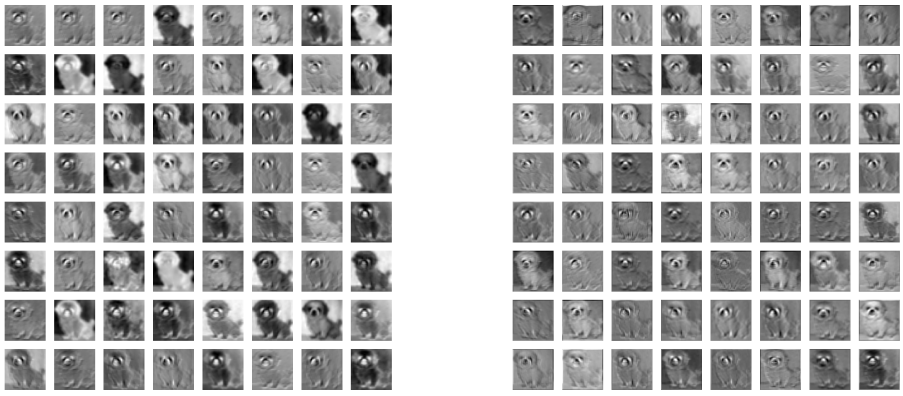
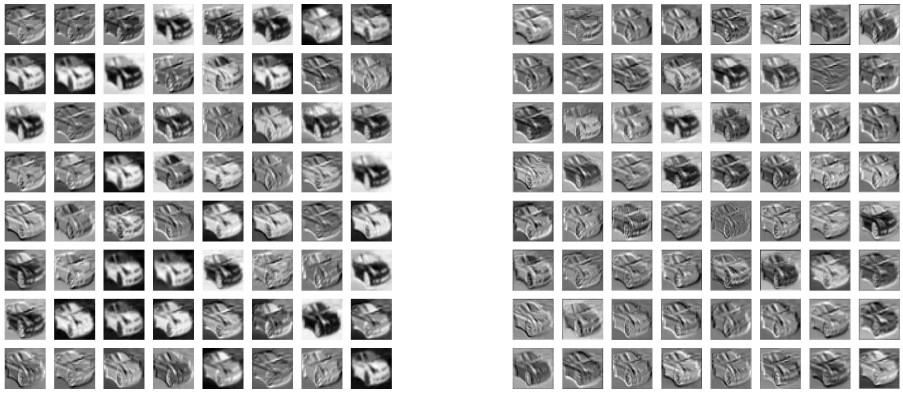
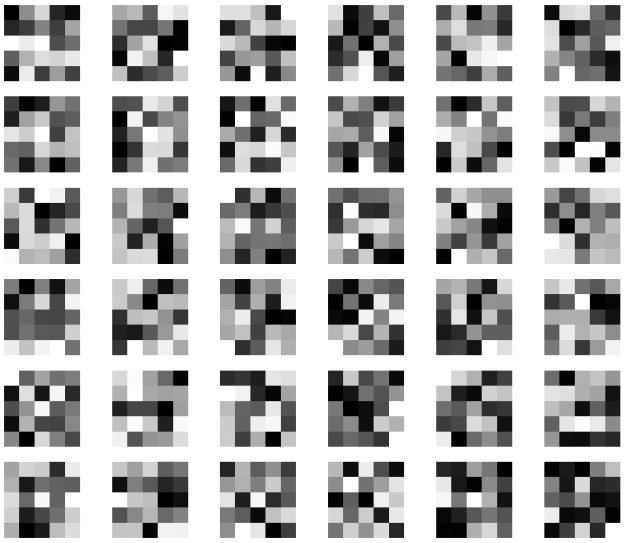


Figure 4: NiN-like model. Both images are of two seperate layers of the model. The maps correspond to an incorrectly labeled car.

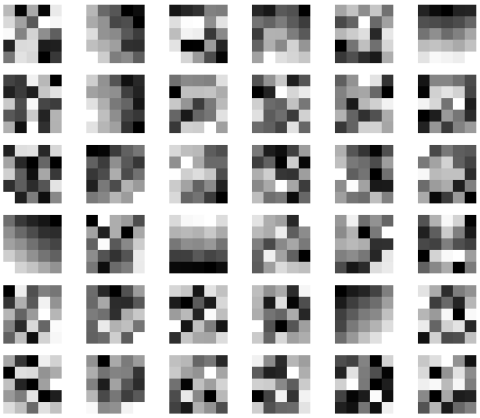


(b) images of subplots of the learned filters for multiple convolutional layers, following the above specifications. Note whether your network ever learns filters similar to the well-known Gabor filters.

Model 1 (LeNet-like):



Model 2 (NiN-like):



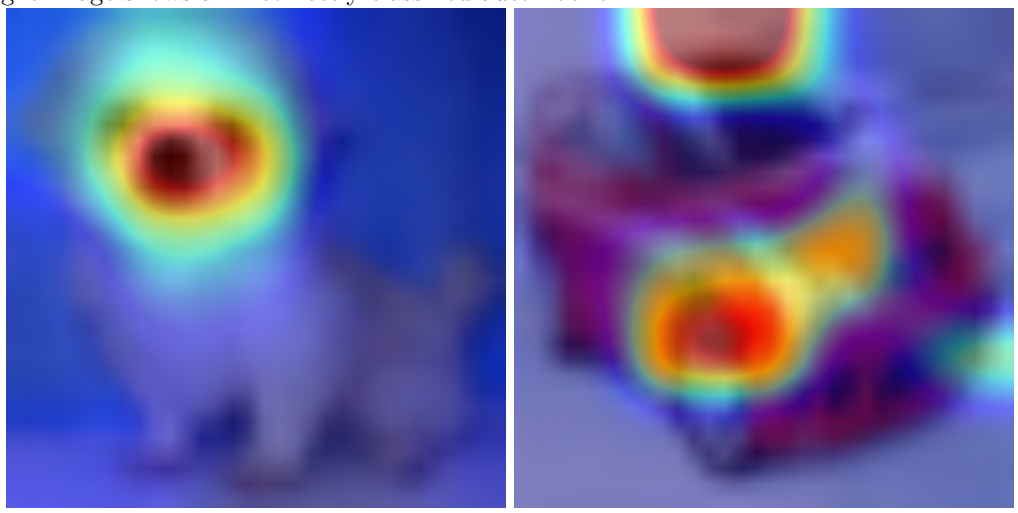
Problem 4 Class Activation Maps (5 pts)

In this problem, you will integrate class activation maps (CAMs) as defined in the paper Learning Deep Features for Discriminative Localization. You may use any code you find online, including the authors' own GitHub repository. The article here may also be helpful.

Turn in some sample CAM images from each architecture developed in Problem 2. Be sure to display images that are correctly classified as well as incorrectly classified, as well as a brief interpretation of what the CAMs show.

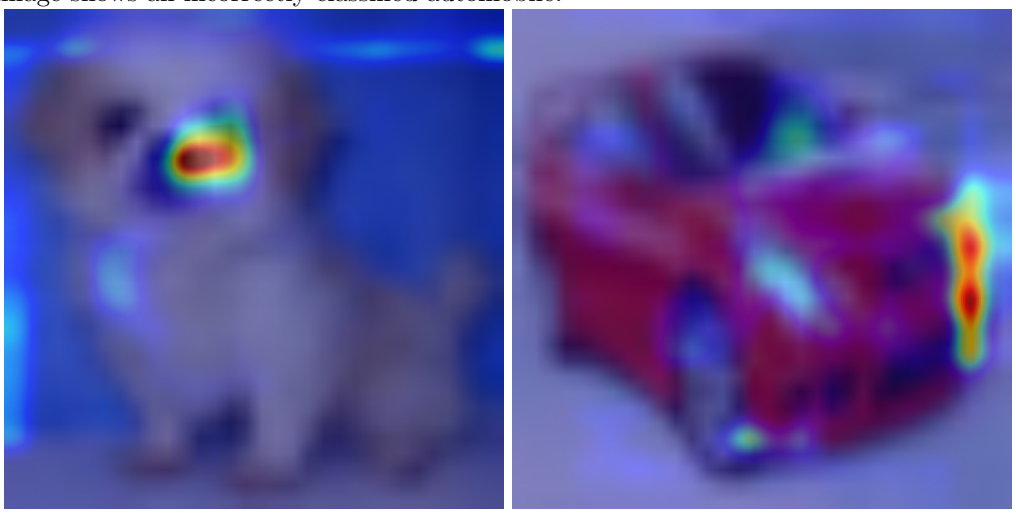
Model 1 (LeNet-like):

Figure 5: Class activation maps for the LeNet-like model. The left image shows a correctly classified dog, and the right image shows an incorrectly classified automobile.



Model 2 (NiN-like):

Figure 6: Class activation maps for the NiN-like model. The left image shows a correctly classified dog, and the right image shows an incorrectly classified automobile.



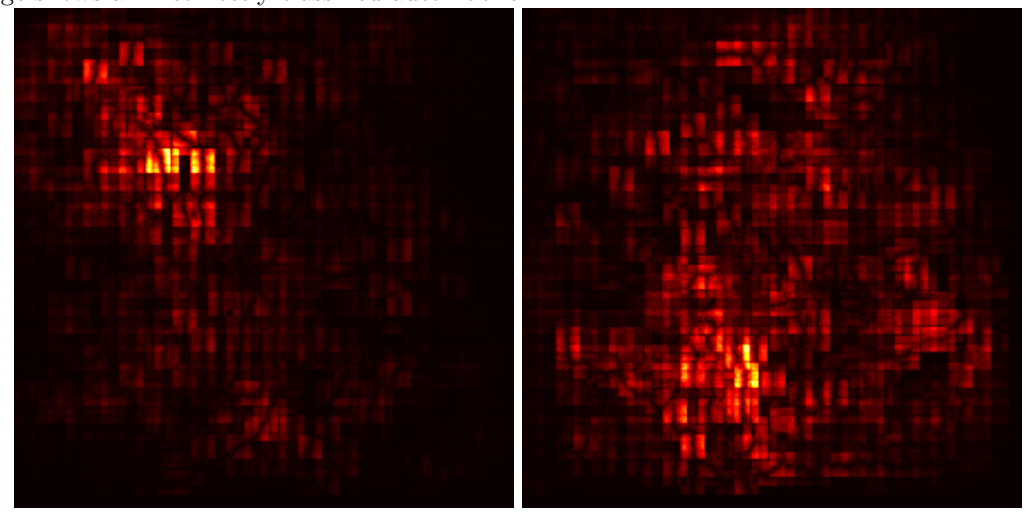
Problem 5 Guided Backpropagation (5 pts)

In this problem, you will produce saliency maps as defined in the paper Striving for Simplicity: The All Convolutional Net. You may use any code you find online. The article here may also be helpful.

Turn in some sample saliency map images from each architecture developed in Problem 2. Be sure to display images that are correctly classified as well as incorrectly classified, as well as a brief interpretation of what the saliency maps show.

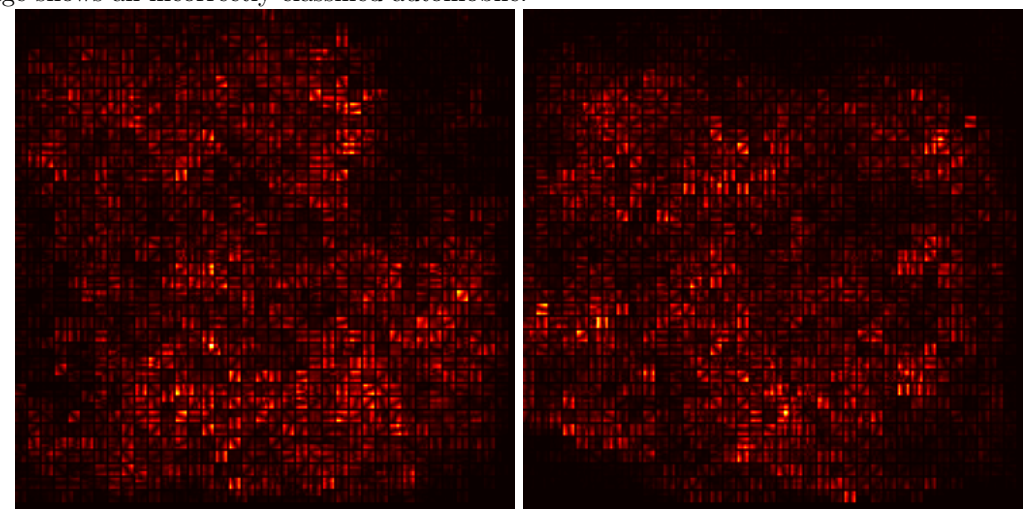
Model 1 (LeNet-like):

Figure 7: Saliency maps for the LeNet-like model. The left image shows a correctly classified dog, and the right image shows an incorrectly classified automobile.



Model 2 (NiN-like):

Figure 8: Saliency maps for the NiN-like model. The left image shows a correctly classified dog, and the right image shows an incorrectly classified automobile.



Problem 6 Reflection on Datasets (5 pts)

Read Ch. 8 of the textbook *Patterns, Predictions, and Actions* available here, which discusses datasets and their impact on machine learning. In the `interesting-reading` channel on Slack, state (1) one thing you learned from a technical perspective regarding datasets (i.e., from the material up to the section *Harms associated with data*), (2) one thing you learned about the harms associated with data and how we might overcome these harms, and (3) one question you have after reading the chapter.

Completed in Slack.