Proof: Screenshots of how your code is successfully executed and generating required outputs, graphs and tables etc.

Note: proper screenshots also added in the report as a part of explanation.

Build_cnn

jupyter build_cnn Last Checkpoint: 9 hours ago

File Edit View Run Kernel Settings Help Trusted

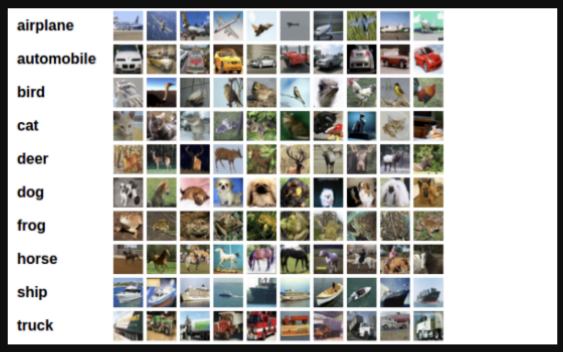
Code JupyterLab Python 3.12

Convolutional Neural Networks - Build Model

In this notebook, we build and train a **CNN** to classify images from the CIFAR-10 database.

- The code provided here are **almost** working. You are required to build up a CNN model and train it.
- Make sure you covered implementations of the **TODO**s in this notebook

The images in this database are small color images that fall into one of ten classes; some example images are pictured below.



Optional: Use CUDA if Available

Since these are color (32x32x3) images, it may prove useful to speed up your training time by using a GPU. CUDA is a parallel computing platform and CUDA Tensors are the same as typical Tensors, but they utilize GPU's for effcient parallel computation.

[1]:
```
import sys
print(sys.executable)
print(sys.version)
```

```
/opt/anaconda3/bin/python3.12
3.12.4 | packaged by Anaconda, Inc. | (main, Jun 18 2024, 10:07:17) [Clang 14.0.6 ]
```

[2]:
```
!pip install torch torchvision torchaudio
```

```
Requirement already satisfied: torch in /opt/anaconda3/lib/python3.12/site-packages (2.6.0)
Requirement already satisfied: torchvision in /opt/anaconda3/lib/python3.12/site-packages (0.21.0)
Requirement already satisfied: torchaudio in /opt/anaconda3/lib/python3.12/site-packages (2.6.0)
Requirement already satisfied: filelock in /opt/anaconda3/lib/python3.12/site-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions>=4.10.0 in /opt/anaconda3/lib/python3.12/site-packages (from torch) (4.11.0)
Requirement already satisfied: networkx in /opt/anaconda3/lib/python3.12/site-packages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in /opt/anaconda3/lib/python3.12/site-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /opt/anaconda3/lib/python3.12/site-packages (from torch) (2024.3.1)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.12/site-packages (from torch) (69.5.1)
Requirement already satisfied: sympy==1.13.1 in /opt/anaconda3/lib/python3.12/site-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/anaconda3/lib/python3.12/site-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.12/site-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /opt/anaconda3/lib/python3.12/site-packages (from torchvision) (10.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/anaconda3/lib/python3.12/site-packages (from jinja2->torch) (2.1.3)
```

[3]:
```
import torch
import numpy as np

# check if CUDA is available
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available.  Training on CPU ...')
else:
    print('CUDA is available!  Training on GPU ...')
```

```
/opt/anaconda3/lib/python3.12/site-packages/torch/utils/_pytree.py:185: FutureWarning: optree is installed but the version is too old to support PyTorch Dynamo in C++ pytree. C++ pytree support is disabled. Please consider upgrading optree using `python3 -m pip install --upgrade 'optree>=0.13.0'`.
  warnings.warn(
CUDA is not available.  Training on CPU ...
```

Load the Data

jupyter build_cnn Last Checkpoint: 9 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3.12

Load the Data

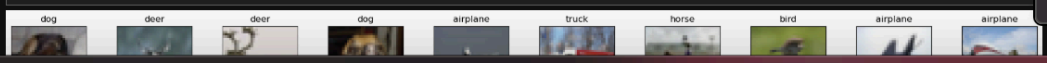
Downloading may take a minute. We load in the training and test data, split the training data into a training and validation set, then create DataLoaders for each of these sets of data.

[4]:
```
from torchvision import datasets
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler

# number of subprocesses to use for data loading
num_workers = 0
# how many samples per batch to load
batch_size = 20
# percentage of training set to use as validation
valid_size = 0.2

# convert data to a normalized torch.FloatTensor
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

# choose the training and test datasets
train_data = datasets.CIFAR10('data', train=True,
                              download=True, transform=transform)
test_data = datasets.CIFAR10('data', train=False,
                             download=True, transform=transform)

# obtain training indices that will be used for validation
num_train = len(train_data)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]

# define samplers for obtaining training and validation batches
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# prepare data loaders (combine dataset and sampler)
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
    sampler=train_sampler, num_workers=num_workers)
valid_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
    sampler=valid_sampler, num_workers=num_workers)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size,
    num_workers=num_workers)

# specify the image classes
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck']
```

Visualize a Batch of Training Data

[5]:
```
import matplotlib.pyplot as plt
%matplotlib inline

# helper function to un-normalize and display an image
def imshow(img):
    img = img / 2 + 0.5  # unnormalize
    plt.imshow(np.transpose(img, (1, 2, 0)))  # convert from Tensor image
```

[6]:
```
# obtain one batch of training images
dataiter = iter(train_loader)
#images, labels = dataiter.next() #python, torchvision version match issue
images, labels = next(dataiter)
images = images.numpy() # convert images to numpy for display

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(25, 4))
# display 20 images
for idx in np.arange(20):
    ax = fig.add_subplot(2, int(20/2), idx+1, xticks=[], yticks=[])
    imshow(images[idx])
    ax.set_title(classes[labels[idx]])
```



Sticki

localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/build_cnn/build_cnn.ipynb

jupyter build_cnn Last Checkpoint: 9 hours ago

File Edit View Run Kernel Settings Help

Trusted

Code

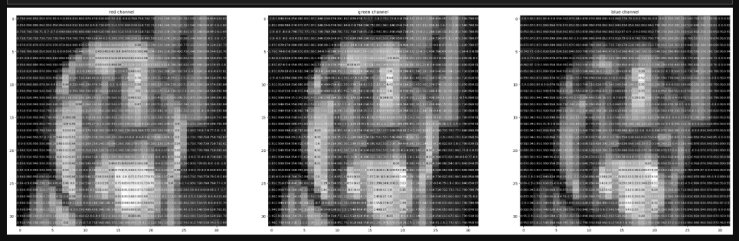
JupyterLab Python 3.12

```
for idx in np.arange(20):
    ax = fig.add_subplot(2, int(20/2), idx+1, xticks=[], yticks=[])
    imshow(images[idx])
    ax.set_title(classes[labels[idx]])
```

View an Image in More Detail

Here, we look at the normalized red, green, and blue (RGB) color channels as three separate, grayscale intensity images.

[7]:
```
rgb_img = np.squeeze(images[3])
channels = ['red channel', 'green channel', 'blue channel']

fig = plt.figure(figsize = (36, 36))
for idx in np.arange(rgb_img.shape[0]):
    ax = fig.add_subplot(1, 3, idx + 1)
    img = rgb_img[idx]
    ax.imshow(img, cmap='gray')
    ax.set_title(channels[idx])
    width, height = img.shape
    thresh = img.max()/2.5
    for x in range(width):
        for y in range(height):
            val = round(img[x][y],2) if img[x][y] !=0 else 0
            ax.annotate(str(val), xy=(y,x),
                    horizontalalignment='center',
                    verticalalignment='center', size=8,
                    color='white' if img[x][y]<thresh else 'black')
```



TODO: Define the Network Architecture

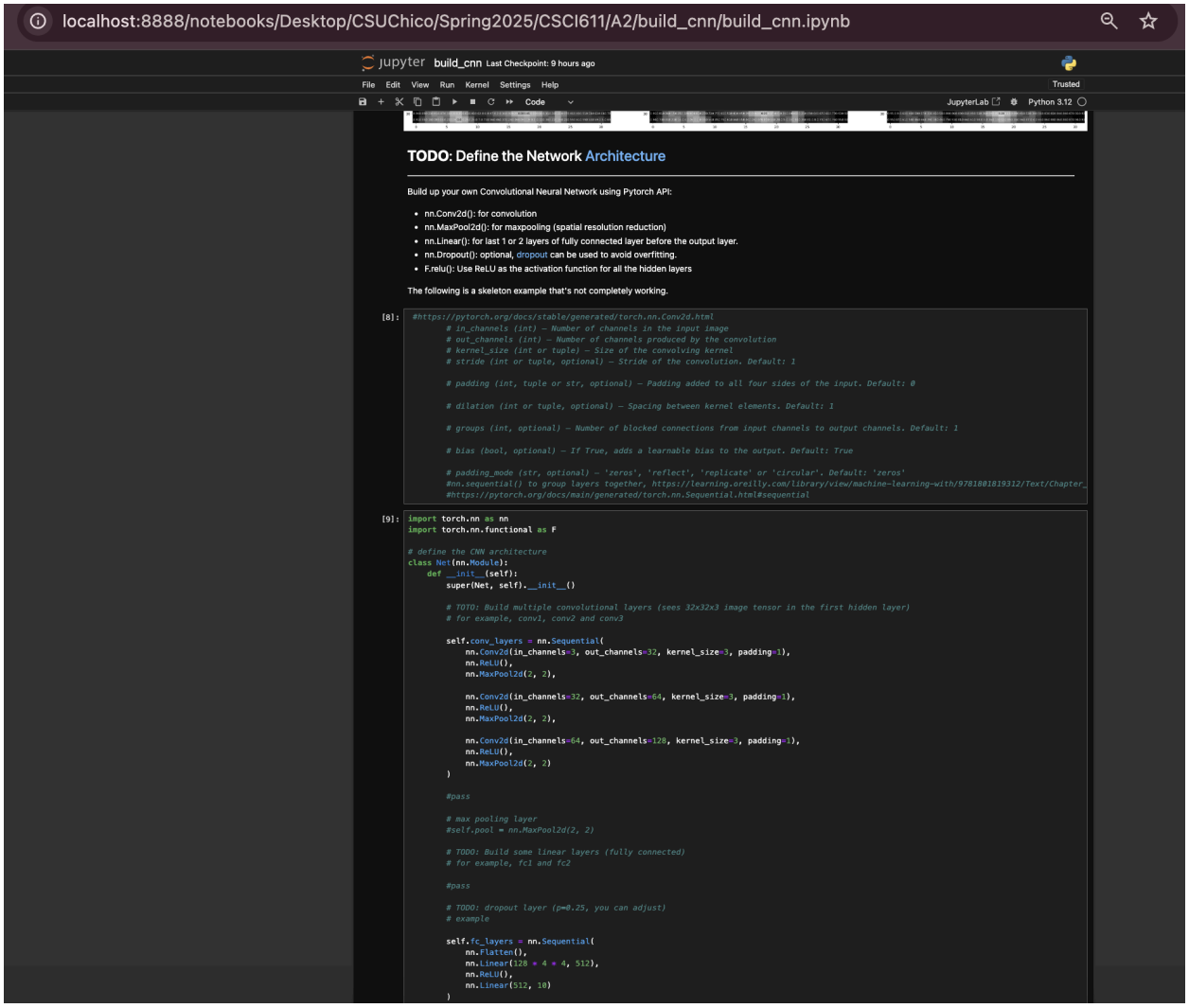
Build up your own Convolutional Neural Network using Pytorch API:

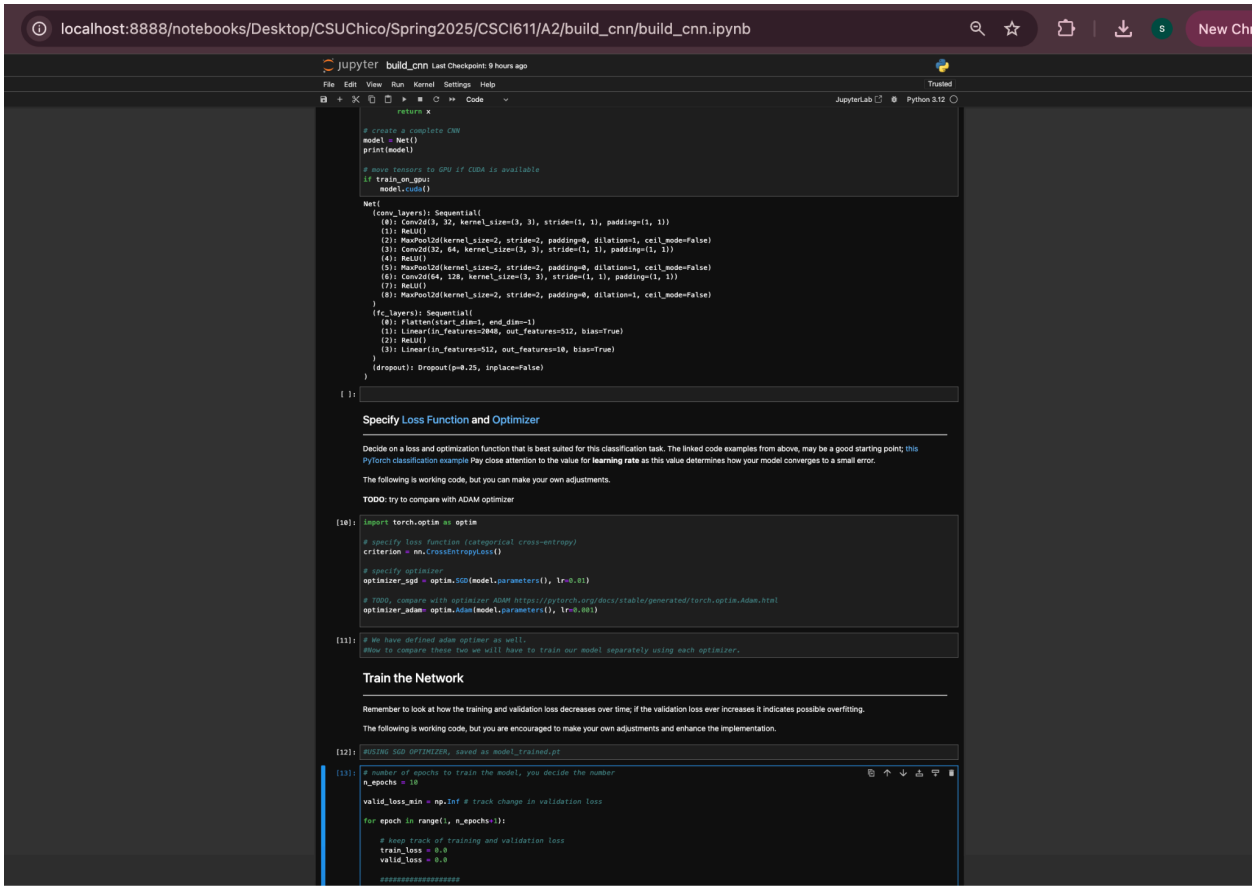
- nn.Conv2d(): for convolution
- nn.MaxPool2d(): for maxpooling (spatial resolution reduction)
- nn.Linear(): for last 1 or 2 layers of fully connected layer before the output layer.
- nn.Dropout(): optional, dropout can be used to avoid overfitting.
- F.relu(): Use ReLU as the activation function for all the hidden layers

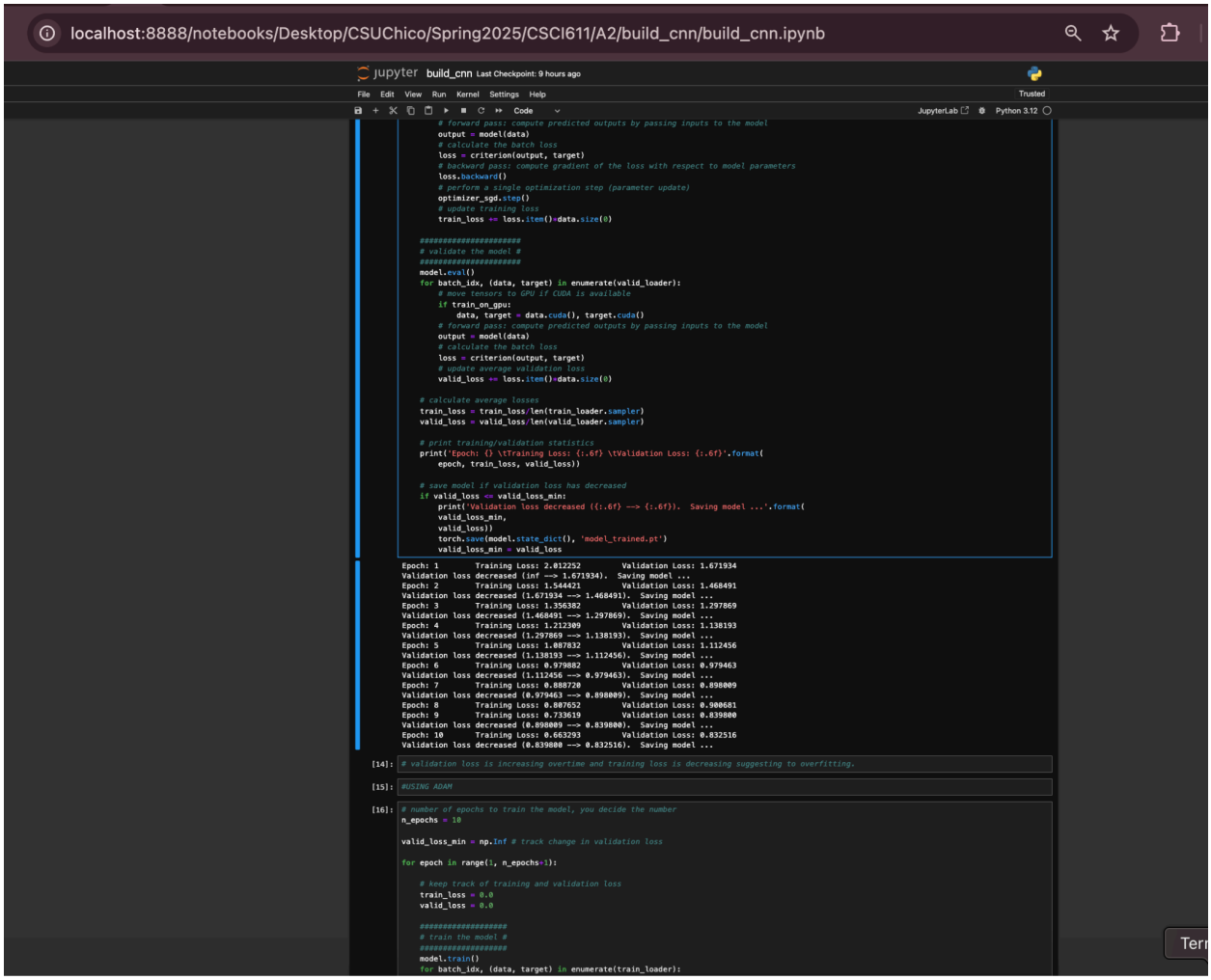
The following is a skeleton example that's not completely working.

[8]:
```
#https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
      # in_channels (int) – Number of channels in the input image
      # out_channels (int) – Number of channels produced by the convolution
      # kernel_size (int or tuple) – Size of the convolving kernel
      # stride (int or tuple, optional) – Stride of the convolution. Default: 1
```

Google


localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/build_cnn/build_cnn.ipynb
jupyter build_cnn Last Checkpoint: 9 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3.12
TODO: Define the Network Architecture
Build up your own Convolutional Neural Network using Pytorch API:
nn.Conv2d(): for convolution
nn.MaxPool2d(): for maxpooling (spatial resolution reduction)
nn.Linear(): for last 1 or 2 layers of fully connected layer before the output layer.
nn.Dropout(): optional, dropout can be used to avoid overfitting.
F.relu(): Use ReLU as the activation function for all the hidden layers
The following is a skeleton example that's not completely working.
[8]:
#https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
in_channels (int) – Number of channels in the input image
out_channels (int) – Number of channels produced by the convolution
kernel_size (int or tuple) – Size of the convolving kernel
stride (int or tuple, optional) – Stride of the convolution. Default: 1
padding (int, tuple or str, optional) – Padding added to all four sides of the input. Default: 0
dilation (int or tuple, optional) – Spacing between kernel elements. Default: 1
groups (int, optional) – Number of blocked connections from input channels to output channels. Default: 1
bias (bool, optional) – If True, adds a learnable bias to the output. Default: True
padding_mode (str, optional) – 'zeros', 'reflect', 'replicate' or 'circular'. Default: 'zeros'
#nn.sequential() to group layers together, https://learning.oreilly.com/library/view/machine-learning-with/9781801819312/Text/Chapter_
#https://pytorch.org/docs/main/generated/torch.nn.Sequential.html#sequential
[9]:
import torch.nn as nn
import torch.nn.functional as F
# define the CNN architecture
class Net(nn.Module):
def __init__(self):
super(Net, self).__init__()
# TOTO: Build multiple convolutional layers (sees 32x32x3 image tensor in the first hidden layer)
# for example, conv1, conv2 and conv3
self.conv_layers = nn.Sequential(
nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
nn.ReLU(),
nn.MaxPool2d(2, 2),
nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1),
nn.ReLU(),
nn.MaxPool2d(2, 2),
nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
nn.ReLU(),
nn.MaxPool2d(2, 2)
)
#pass
max pooling layer
#self.pool = nn.MaxPool2d(2, 2)
TODO: Build some linear layers (fully connected)
for example, fc1 and fc2
#pass
# TODO: dropout layer (p=0.25, you can adjust)
# example
self.fc_layers = nn.Sequential(
nn.Flatten(),
nn.Linear(128 * 4 * 4, 512),
nn.ReLU(),
nn.Linear(512, 10)
)


localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/build_cnn/build_cnn.ipynb
jupyter build_cnn Last Checkpoint: 9 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3.12
return x
# create a complete CNN
model = Net()
print(model)
# move tensors to GPU if CUDA is available
if train_on_gpu:
model.cuda()
Net(
(conv_layers): Sequential(
(0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(7): ReLU()
(8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(fc_layers): Sequential(
(0): Flatten(start_dim=1, end_dim=-1)
(1): Linear(in_features=2048, out_features=512, bias=True)
(2): ReLU()
(3): Linear(in_features=512, out_features=10, bias=True)
)
(dropout): Dropout(p=0.25, inplace=False)
)
[ ]:
Specify Loss Function and Optimizer
Decide on a loss and optimization function that is best suited for this classification task. The linked code examples from above, may be a good starting point; this PyTorch classification example Pay close attention to the value for learning rate as this value determines how your model converges to a small error.
The following is working code, but you can make your own adjustments.
TODO: try to compare with ADAM optimizer
[10]:
import torch.optim as optim
# specify loss function (categorical cross-entropy)
criterion = nn.CrossEntropyLoss()
# specify optimizer
optimizer_sgd = optim.SGD(model.parameters(), lr=0.01)
# TODO, compare with optimizer ADAM https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
optimizer_adam= optim.Adam(model.parameters(), lr=0.001)
[11]:
We have defined adam optimer as well.
#Now to compare these two we will have to train our model separately using each optimizer.
Train the Network
Remember to look at how the training and validation loss decreases over time; if the validation loss ever increases it indicates possible overfitting.
The following is working code, but you are encouraged to make your own adjustments and enhance the implementation.
[12]:
#USING SGD OPTIMIZER, saved as model_trained.pt
[13]:
# number of epochs to train the model, you decide the number
n_epochs = 10
valid_loss_min = np.Inf # track change in validation loss
for epoch in range(1, n_epochs+1):
# keep track of training and validation loss
train_loss = 0.0
valid_loss = 0.0
#################


localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/build_cnn/build_cnn.ipynb
jupyter build_cnn Last Checkpoint: 9 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3.12
# forward pass: compute predicted outputs by passing inputs to the model
output = model(data)
# calculate the batch loss
loss = criterion(output, target)
# backward pass: compute gradient of the loss with respect to model parameters
loss.backward()
# perform a single optimization step (parameter update)
optimizer_sgd.step()
# update training loss
train_loss += loss.item()*data.size(0)
######################
validate the model
######################
model.eval()
for batch_idx, (data, target) in enumerate(valid_loader):
# move tensors to GPU if CUDA is available
if train_on_gpu:
data, target = data.cuda(), target.cuda()
# forward pass: compute predicted outputs by passing inputs to the model
output = model(data)
# calculate the batch loss
loss = criterion(output, target)
# update average validation loss
valid_loss += loss.item()*data.size(0)
# calculate average losses
train_loss = train_loss/len(train_loader.sampler)
valid_loss = valid_loss/len(valid_loader.sampler)
# print training/validation statistics
print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
epoch, train_loss, valid_loss))
# save model if validation loss has decreased
if valid_loss <= valid_loss_min:
print('Validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(
valid_loss_min,
valid_loss))
torch.save(model.state_dict(), 'model_trained.pt')
valid_loss_min = valid_loss
Epoch: 1 Training Loss: 2.012252 Validation Loss: 1.671934
Validation loss decreased (inf --> 1.671934). Saving model ...
Epoch: 2 Training Loss: 1.544421 Validation Loss: 1.468491
Validation loss decreased (1.671934 --> 1.468491). Saving model ...
Epoch: 3 Training Loss: 1.356382 Validation Loss: 1.297869
Validation loss decreased (1.468491 --> 1.297869). Saving model ...
Epoch: 4 Training Loss: 1.212309 Validation Loss: 1.138193
Validation loss decreased (1.297869 --> 1.138193). Saving model ...
Epoch: 5 Training Loss: 1.087832 Validation Loss: 1.112456
Validation loss decreased (1.138193 --> 1.112456). Saving model ...
Epoch: 6 Training Loss: 0.979082 Validation Loss: 0.979463
Validation loss decreased (1.112456 --> 0.979463). Saving model ...
Epoch: 7 Training Loss: 0.888720 Validation Loss: 0.898009
Validation loss decreased (0.979463 --> 0.898009). Saving model ...
Epoch: 8 Training Loss: 0.807652 Validation Loss: 0.900681
Epoch: 9 Training Loss: 0.733619 Validation Loss: 0.839800
Validation loss decreased (0.898009 --> 0.839800). Saving model ...
Epoch: 10 Training Loss: 0.663293 Validation Loss: 0.832516
Validation loss decreased (0.839800 --> 0.832516). Saving model ...
[14]: # validation loss is increasing overtime and training loss is decreasing suggesting to overfitting.
[15]: #USING ADAM
[16]: # number of epochs to train the model, you decide the number
n_epochs = 10
valid_loss_min = np.Inf # track change in validation loss
for epoch in range(1, n_epochs+1):
# keep track of training and validation loss
train_loss = 0.0
valid_loss = 0.0
###################
train the model
###################
model.train()
for batch_idx, (data, target) in enumerate(train_loader):

```
    valid_loss = valid_loss/len(valid_loader.sampler)

    # print training/validation statistics
    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
        epoch, train_loss, valid_loss))

    # save model if validation loss has decreased
    if valid_loss <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
        valid_loss_min,
        valid_loss))
        torch.save(model.state_dict(), 'model_trained_with_adam.pt')
        valid_loss_min = valid_loss
```

```
Epoch: 1        Training Loss: 0.938161         Validation Loss: 0.876345
Validation loss decreased (inf --> 0.876345).  Saving model ...
Epoch: 2        Training Loss: 0.701895         Validation Loss: 0.780278
Validation loss decreased (0.876345 --> 0.780278).  Saving model ...
Epoch: 3        Training Loss: 0.537921         Validation Loss: 0.818282
Epoch: 4        Training Loss: 0.387152         Validation Loss: 0.845727
Epoch: 5        Training Loss: 0.269015         Validation Loss: 0.947663
Epoch: 6        Training Loss: 0.193520         Validation Loss: 1.201517
Epoch: 7        Training Loss: 0.161094         Validation Loss: 1.327034
Epoch: 8        Training Loss: 0.139349         Validation Loss: 1.340844
Epoch: 9        Training Loss: 0.134765         Validation Loss: 1.467664
Epoch: 10       Training Loss: 0.115975         Validation Loss: 1.565384
```

Load the Model with the Lowest Validation Loss

This is the model we will use for testing, which is the model we saved in the last step

[28]:
```
#test for SGD optimizer model
model.load_state_dict(torch.load('model_trained.pt'))
```

[28]:
```
<All keys matched successfully>
```

Test the Trained Network

Test your trained model on previously unseen data! Remember we have downloaded `train_data` and `test_data`. We will use `test_data` through `test_loader`.

A "good" result will be a CNN that gets around 70% (or more, try your best!) accuracy on these test images.

The following is working code, but you are encouraged to make your own adjustments and enhance the implementation.

[29]:
```
# track test loss
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

model.eval()
# iterate over test data
for batch_idx, (data, target) in enumerate(test_loader):
    # move tensors to GPU if CUDA is available
    if train_on_gpu:
        data, target = data.cuda(), target.cuda()
    # forward pass: compute predicted outputs by passing inputs to the model
    output = model(data)
    # calculate the batch loss
    loss = criterion(output, target)
    # update test loss
    test_loss += loss.item()*data.size(0)
    # convert output probabilities to predicted class
    _, pred = torch.max(output, 1)
    # compare predictions to true label
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else np.squeeze(correct_tensor.cpu().numpy())
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

# average test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(10):
    if class_total[i] > 0:
```

[28]:
```
<All keys matched successfully>
```

Test the Trained Network

Test your trained model on previously unseen data! Remember we have downloaded `train_data` and `test_data`. We will use `test_data` through `test_loader`.

A "good" result will be a CNN that gets around 70% (or more, try your best!) accuracy on these test images.

The following is working code, but you are encouraged to make your own adjustments and enhance the implementation.

[29]:
```
# track test loss
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

model.eval()
# iterate over test data
for batch_idx, (data, target) in enumerate(test_loader):
    # move tensors to GPU if CUDA is available
    if train_on_gpu:
        data, target = data.cuda(), target.cuda()
    # forward pass: compute predicted outputs by passing inputs to the model
    output = model(data)
    # calculate the batch loss
    loss = criterion(output, target)
    # update test loss
    test_loss += loss.item()*data.size(0)
    # convert output probabilities to predicted class
    _, pred = torch.max(output, 1)
    # compare predictions to true label
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else np.squeeze(correct_tensor.cpu().numpy())
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

# average test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(10):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            classes[i], 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
    100. * np.sum(class_correct) / np.sum(class_total),
    np.sum(class_correct), np.sum(class_total)))
```

```
Test Loss: 0.846499

Test Accuracy of airplane: 65% (652/1000)
Test Accuracy of automobile: 82% (828/1000)
Test Accuracy of  bird: 77% (779/1000)
Test Accuracy of   cat: 41% (413/1000)
Test Accuracy of  deer: 60% (605/1000)
Test Accuracy of   dog: 63% (631/1000)
Test Accuracy of  frog: 83% (836/1000)
Test Accuracy of horse: 70% (703/1000)
Test Accuracy of  ship: 84% (841/1000)
Test Accuracy of truck: 76% (768/1000)

Test Accuracy (Overall): 70% (7056/10000)
```

[ ]:

[24]:
```
#Testing with adam
model.load_state_dict(torch.load('model_trained_with_adam.pt'))
```

[24]:
```
<All keys matched successfully>
```

[25]:
```
# track test loss
```

jupyter build_cnn Last Checkpoint: 9 hours ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3.12

[ ]:

[24]:
```
#Testing with adam
model.load_state_dict(torch.load('model_trained_with_adam.pt'))
```

[24]:
```
<All keys matched successfully>
```

[25]:
```
# track test loss
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

model.eval()
# iterate over test data
for batch_idx, (data, target) in enumerate(test_loader):
    # move tensors to GPU if CUDA is available
    if train_on_gpu:
        data, target = data.cuda(), target.cuda()
    # forward pass: compute predicted outputs by passing inputs to the model
    output = model(data)
    # calculate the batch loss
    loss = criterion(output, target)
    # update test loss
    test_loss += loss.item()*data.size(0)
    # convert output probabilities to predicted class
    _, pred = torch.max(output, 1)
    # compare predictions to true label
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else np.squeeze(correct_tensor.cpu().numpy())
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

# average test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(10):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            classes[i], 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
    100. * np.sum(class_correct) / np.sum(class_total),
    np.sum(class_correct), np.sum(class_total)))
```

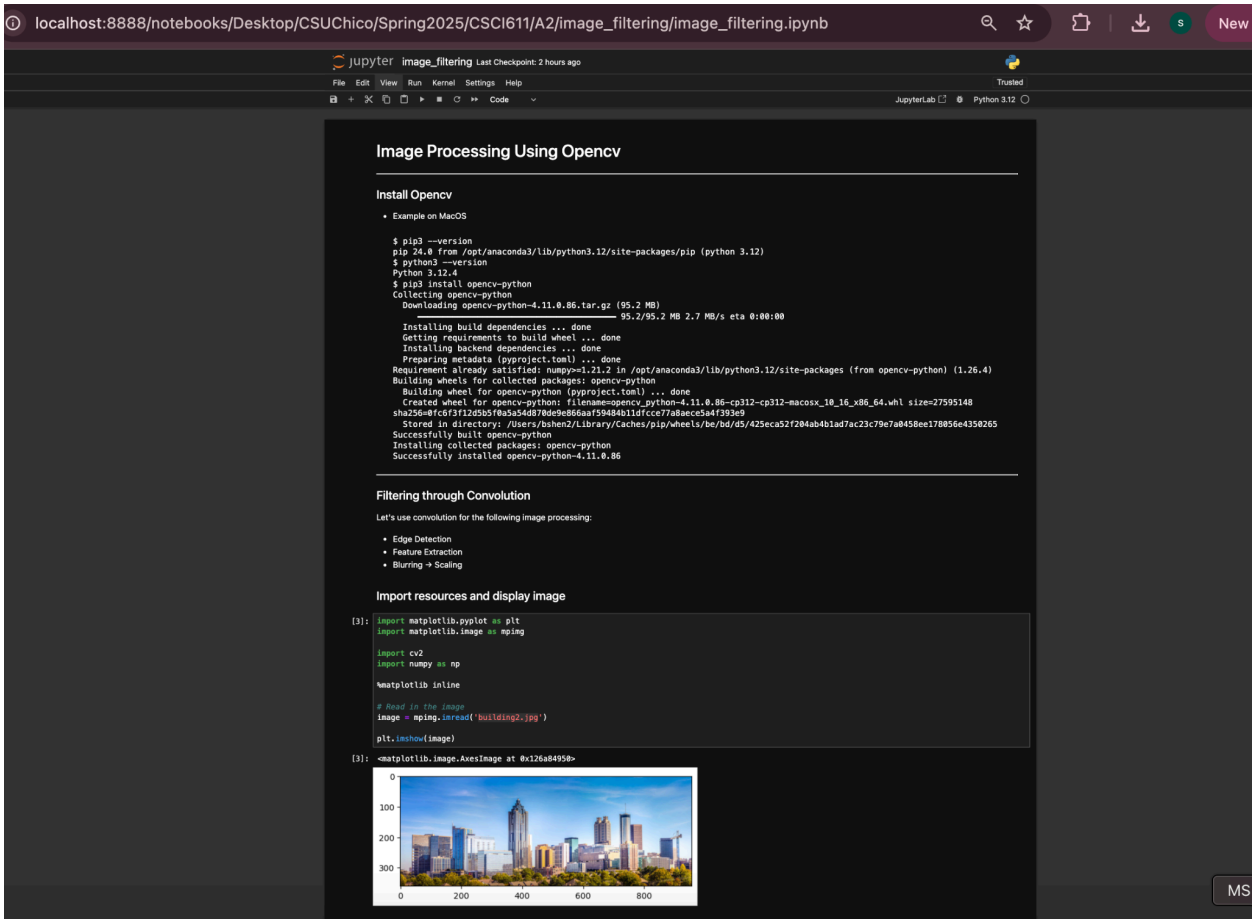
```
Test Loss: 0.791868

Test Accuracy of airplane: 74% (748/1000)
Test Accuracy of automobile: 82% (823/1000)
Test Accuracy of  bird: 49% (496/1000)
Test Accuracy of   cat: 67% (675/1000)
Test Accuracy of  deer: 67% (679/1000)
Test Accuracy of   dog: 52% (528/1000)
Test Accuracy of  frog: 85% (857/1000)
Test Accuracy of horse: 78% (786/1000)
Test Accuracy of  ship: 89% (892/1000)
Test Accuracy of truck: 84% (844/1000)

Test Accuracy (Overall): 73% (7328/10000)
```

Visualize Sample Test Results

The following is working code, but you are encouraged to make your own adjustments and enhance the visualization.

[60]:
```
# obtain one batch of test images
dataiter = iter(test_loader)
images, labels = next(dataiter)
images.numpy()

# move model inputs to cuda, if GPU available
if train_on_gpu:
    images = images.cuda()

# get sample outputs
```

Image_filtering



Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

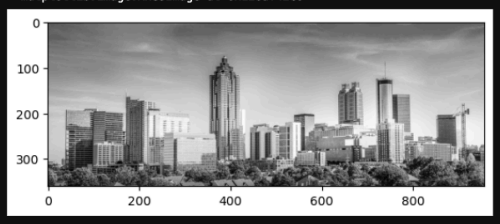
File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3.12

Convert the image to grayscale

[4]:
```
# Convert to grayscale for filtering
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

plt.imshow(gray, cmap='gray')
```

[4]: <matplotlib.image.AxesImage at 0x126af42c0>



Convolution Through a Kernel

Image convolution is a mathematical operation where a small matrix (called a kernel or filter) slides over an image, performing element-wise multiplications and summing the results to produce a new pixel value. This process helps extract features such as edges, textures, and patterns by emphasizing specific spatial structures in the image. Convolution is widely used in image processing and deep learning, particularly in convolutional neural networks (CNNs) for feature detection.

Here we practice applying image convolution using traditional image processing operations.

1. **Select a Kernel (Filter):** Choose a small matrix (e.g., 3x3 or 5x5) with predefined values for a specific operation (e.g., edge detection, blurring).
2. **Slide the Kernel Over the Image:** Move the kernel across the image, covering one region at a time.
3. **Compute the Element-Wise Product:** Multiply each value in the kernel with the corresponding pixel values in the image region.
4. **Sum the Products:** Add up all the multiplied values to obtain a single new pixel value.
5. **Store the Result:** Place the computed value into the corresponding location in the output image.
6. **Repeat for Entire Image:** Continue the process by shifting the kernel until every pixel has been processed.

Here is a common 3×3 kernel for edge detection, which detects edges in a specific direction, vertical and horizontal:

Vertical Edge Detection Kernel:

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Horizontal Edge Detection Kernel:

$$K_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

These kernels are convolved with an image to highlight vertical and horizontal edges, respectively.

Applying a filter like this to an image is a way of **taking (an approximation) of the derivative of the image** in the x or y direction, separately.

[5]:
```
# Create a custom kernel

# 3x3 array for edge detection
horizon = np.array([[ -1, -1, -1],
                    [  0,  0,  0],
                    [  1,  1,  1]])

## TODO: Create and apply a vertical edge detection operator
vertica = np.array([[ -1, 0, 1],
                    [ -1, 0, 1],
                    [ -1, 0, 1]])
```

Google C

jupyter image_filtering Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help Trusted

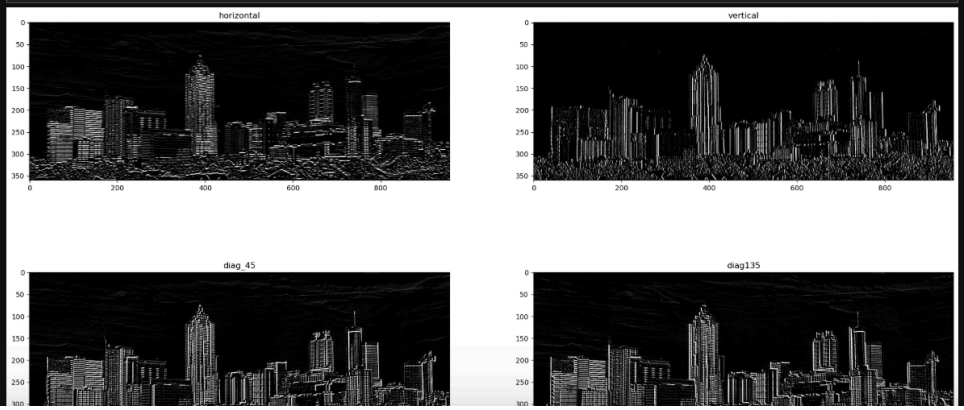
Code JupyterLab Python 3.12

[1 1 1]

These kernels are convolved with an image to highlight vertical and horizontal edges, respectively.

Applying a filter like this to an image is a way of **taking (an approximation) of the derivative of the image** in the x or y direction, separately.

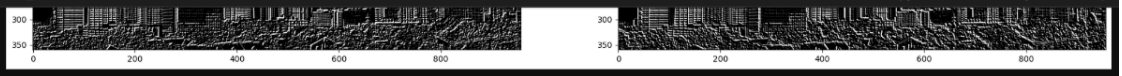
[5]:
```
# Create a custom kernel

# 3x3 array for edge detection
horizon = np.array([[ -1, -1, -1],
                    [  0,  0,  0],
                    [  1,  1,  1]])

## TODO: Create and apply a vertical edge detection operator
vertica = np.array([[ -1, 0, 1],
                    [ -1, 0, 1],
                    [ -1, 0, 1]])

diag_45 = np.array([[ -2, -1, 0],
                    [ -1,  0, 1],
                    [  0,  1, 2]])

diag135 = np.array([[ 0, -1, -2],
                    [ 1,  0, -1],
                    [ 2,  1,  0]])

fig = plt.figure(figsize=(24,24))
# Filter the image using filter2D, which has inputs: (grayscale image, bit-depth, kernel)
filtered_image = cv2.filter2D(gray, -1, horizon)
fig.add_subplot(2,2,1)
plt.imshow(filtered_image, cmap='gray')
plt.title('horizontal')

filtered_image2 = cv2.filter2D(gray, -1, vertica)
fig.add_subplot(2,2,2)
plt.imshow(filtered_image2, cmap='gray')
plt.title('vertical')

filtered_image3 = cv2.filter2D(gray, -1, diag_45)
fig.add_subplot(1,2,1)
plt.imshow(filtered_image3, cmap='gray')
plt.title('diag_45')

filtered_image4 = cv2.filter2D(gray, -1, diag135)
fig.add_subplot(1,2,2)
plt.imshow(filtered_image4, cmap='gray')
plt.title('diag135')

plt.show()
```



jupyter image_filtering Last Checkpoint: 2 hours ago

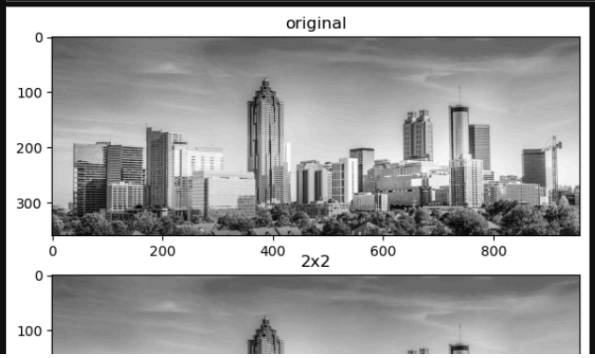
File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3.12



[6]:

```
# Create a custom kernel for blurring

# 2x2 kernel for averaging blurring
S2x2 = np.array([[ 1, 1],
                 [ 1, 1]])

# 3x3 kernel
S3x3 = np.array([[1, 1, 1],
                 [1, 1, 1],
                 [1, 1, 1]])
# 5x5 kernel
S5x5 = np.array([[1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1]])


fig = plt.figure(figsize=(48, 12))
fig.add_subplot(4,1,1)
plt.imshow(gray, cmap='gray')
plt.title('original')

# Filter the image using filter2D, which has inputs: (grayscale image, bit-depth, kernel)
#divide by 4 to normalize, ensures kernel elements equals thereby maintains the overall brightness
blurred_image = cv2.filter2D(gray, -1, S2x2/4.0)
fig.add_subplot(4,1,2)
plt.imshow(blurred_image, cmap='gray')
plt.title('2x2')

# TODO: blur image using a 3x3 average
blurred_image_3x3 = cv2.filter2D(gray, -1, S3x3/9.0)
fig.add_subplot(4,1,3)
plt.imshow(blurred_image_3x3, cmap='gray')
plt.title('3x3')

# TODO: blur image using a 5x5 average
blurred_image_5x5 = cv2.filter2D(gray, -1, S5x5/25.0)
fig.add_subplot(4,1,4)
plt.imshow(blurred_image_5x5, cmap='gray')
plt.title('5x5')

plt.show()
```



localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

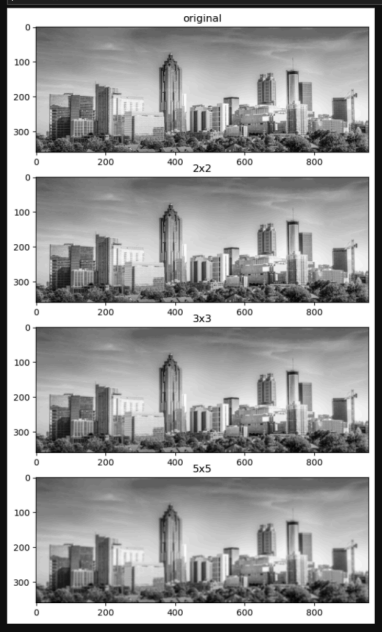
File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3.12

```
plt.title('5x5')

plt.show()
```



TODO

Other image processing/filtering you can try:

- Other Edge Detector (e.g. Sobel Operator) A common 3×3 kernel for edge detection is the **Sobel operator**, which detects edges in a specific direction. Below are the Sobel kernels for detecting vertical and horizontal edges:

Vertical Edge Detection Kernel:

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3.12

TODO

Other image processing/filtering you can try:

- Other Edge Detector (e.g. Sobel Operator) A common 3×3 kernel for edge detection is the **Sobel operator**, which detects edges in a specific direction. Below are the Sobel kernels for detecting vertical and horizontal edges:

Vertical Edge Detection Kernel:

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Horizontal Edge Detection Kernel:

$$K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

These kernels are convolved with an image to highlight vertical and horizontal edges, respectively.

- Corner Detection (use the kernels we discussed in slides)
- Scaling (after the blurring, can you pick one pixel out of the following?)
  - 2x2
  - 4x4
- Use other images of your choice
- For a challenge, see if you can put the image through a series of filters: first one that blurs the image (takes an average of pixels), and then one that detects the edges.

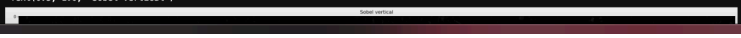
[7]:
```
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
plt.imshow(gray, cmap='gray')
plt.axis('off')
plt.show()
```

[8]:
```
#defining sobel kernels
#vertical
K_x = np.array([[-1, 0, 1],
                [-2, 0, 2],
                [-1, 0, 1]])

#horizontal
K_y = np.array([[-1, -2, -1],
                [ 0,  0,  0],
                [ 1,  2,  1]])
```

[9]:
```
#now that sobel is defined, i apply edge detection
#vertical detection
vertical_edges = cv2.filter2D(gray, -1, K_x)
horizontal_edges = cv2.filter2D(gray, -1, K_y)
```

[10]:
```
#sobel vertical

fig = plt.figure(figsize=(48, 12))
plt.imshow(vertical_edges, cmap='gray')
plt.title('Sobel vertical')
```

[10]:
```
Text(0.5, 1.0, 'Sobel vertical')
```



localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

Trusted

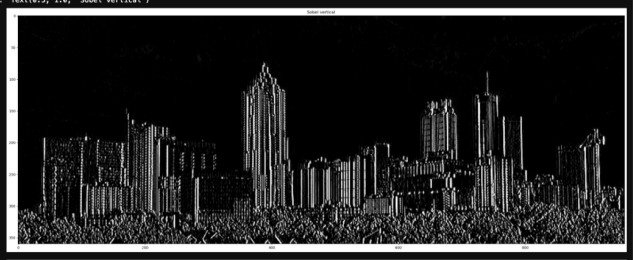
Code

JupyterLab Python 3.12

```
vertical_edges = cv2.filter2D(gray, -1, K_x)
horizontal_edges = cv2.filter2D(gray, -1, K_y)
```

[10]:
```
#sobel vertical

fig = plt.figure(figsize=(40, 12))
plt.imshow(vertical_edges, cmap='gray')
plt.title('Sobel vertical')
```

[10]: `Text(0.5, 1.0, 'Sobel vertical')`



[11]:
```
#sobel horizontal
fig = plt.figure(figsize=(40, 12))
plt.imshow(horizontal_edges, cmap='gray')
plt.title('Sobel horizontal')
```

[11]: `Text(0.5, 1.0, 'Sobel horizontal')`

[94]:
```
#Now work on corner detection
# https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#gac1fc35988188188880e370e2f709b4345
# cv.cornerHarris(  src, blockSize, ksize, k[, dst[, borderType]]  ) ->  dst

#load and convert image to gray
image=cv2.imread('building2-Copy1.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY).astype(np.float32)
plt.imshow(gray, cmap='gray')
plt.axis('off')
```

localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3.12

[94]:
```
#Now work on corner detection
# https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#gac1fc3598018010880e370e2f709b4345
# cv.cornerHarris(  src, blockSize, ksize, k[, dst[, borderType]]  ) ->  dst

#load and convert image to gray
image=cv2.imread('building2-Copy1.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY).astype(np.float32)
plt.imshow(gray, cmap='gray')
plt.axis('off')
plt.show()
```

[104]:
```
# #using cv2.cornerHarris

dst = cv2.cornerHarris(gray, blockSize=2, ksize=3, k=0.04)
dst = cv2.dilate(dst, None)

gray = cv2.cvtColor(np.uint8(gray), cv2.COLOR_GRAY2RGB)

# Mark detected corners in red
gray[dst > 0.01 * dst.max()] = [255, 0, 0]  # Red color in (B, G, R)

# Display the grayscale image with red corners
plt.imshow(gray)
plt.axis('off')
plt.show()
```

[86]:
```
#corner detection using lecture slide example kernels.
#load image, define kernels for top right and left as well as bottopm right and left corners,
# apply each kernel using conv, ad threshold to detect strong corners, display the corners.
image=cv2.imread('building2-Copy1.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY).astype(np.float32)
plt.imshow(gray, cmap='gray')
plt.axis('off')
plt.show()
```

Google

localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

Trusted

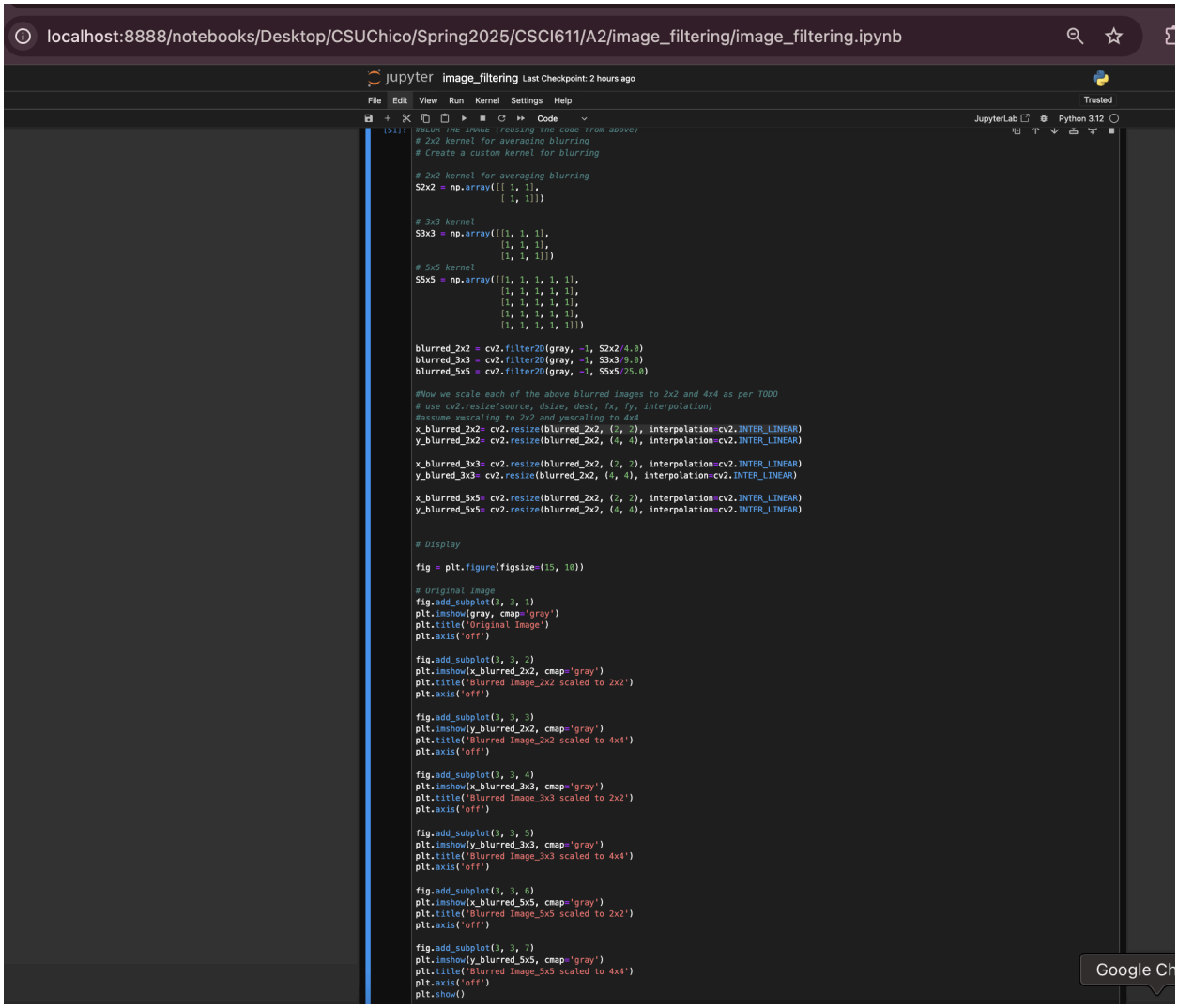
Code

JupyterLab Python 3.12

[112]:
```
#defining kernels
kernels = [
    np.array([[1,  1,  0],
              [0,  1,  1],
              [0,  0,  0]], dtype=np.float32),

    np.array([[1,  1,  0],
              [1,  1,  0],
              [0,  0,  0]], dtype=np.float32),

    np.array([[0,  0,  0],
              [1,  1,  0],
              [0,  1,  1]], dtype=np.float32),

    np.array([[0,  0,  0],
              [0,  1,  1],
              [1,  1,  1]], dtype=np.float32),
]

corner_response = np.zeros_like(gray)

# filter2d for each kernel
for kernel in kernels:
    response = cv2.filter2D(gray, -1, kernel)
    corner_response = np.maximum(corner_response, response)

#thershold for string corner detection
threshold = 0.9 * corner_response.max()
corners = (corner_response > threshold).astype(np.uint8) * 255

image[corners > 0] = [0, 0, 255]
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.title("Corner Detection using Kernels from Lecture Slide")
plt.show()
```

[109]:
```
#scalling(after the blurring, can you pick one pixel out of the following?)
#steps: load the image, blur
image=cv2.imread('building2-Copy1.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY).astype(np.float32)
plt.imshow(gray, cmap='gray')
plt.axis('off')
plt.show()
```

[51]:
```
#BLUR THE IMAGE (reusing the code from above)
# 2x2 kernel for averaging blurring
# Create a custom kernel for blurring
```


localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb
jupyter image_filtering Last Checkpoint: 2 hours ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3.12
[51]:
#BLUR THE IMAGE (reusing the code from above)
# 2x2 kernel for averaging blurring
# Create a custom kernel for blurring
# 2x2 kernel for averaging blurring
S2x2 = np.array([[1, 1],
[1, 1]])
# 3x3 kernel
S3x3 = np.array([[1, 1, 1],
[1, 1, 1],
[1, 1, 1]])
# 5x5 kernel
S5x5 = np.array([[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1]])
blurred_2x2 = cv2.filter2D(gray, -1, S2x2/4.0)
blurred_3x3 = cv2.filter2D(gray, -1, S3x3/9.0)
blurred_5x5 = cv2.filter2D(gray, -1, S5x5/25.0)
#Now we scale each of the above blurred images to 2x2 and 4x4 as per TODO
use cv2.resize(source, dsize, dest, fx, fy, interpolation)
#assume x=scaling to 2x2 and y=scaling to 4x4
x_blurred_2x2= cv2.resize(blurred_2x2, (2, 2), interpolation=cv2.INTER_LINEAR)
y_blurred_2x2= cv2.resize(blurred_2x2, (4, 4), interpolation=cv2.INTER_LINEAR)
x_blurred_3x3= cv2.resize(blurred_2x2, (2, 2), interpolation=cv2.INTER_LINEAR)
y_blured_3x3= cv2.resize(blurred_2x2, (4, 4), interpolation=cv2.INTER_LINEAR)
x_blurred_5x5= cv2.resize(blurred_2x2, (2, 2), interpolation=cv2.INTER_LINEAR)
y_blurred_5x5= cv2.resize(blurred_2x2, (4, 4), interpolation=cv2.INTER_LINEAR)
# Display
fig = plt.figure(figsize=(15, 10))
# Original Image
fig.add_subplot(3, 3, 1)
plt.imshow(gray, cmap='gray')
plt.title('Original Image')
plt.axis('off')
fig.add_subplot(3, 3, 2)
plt.imshow(x_blurred_2x2, cmap='gray')
plt.title('Blurred Image_2x2 scaled to 2x2')
plt.axis('off')
fig.add_subplot(3, 3, 3)
plt.imshow(y_blurred_2x2, cmap='gray')
plt.title('Blurred Image_2x2 scaled to 4x4')
plt.axis('off')
fig.add_subplot(3, 3, 4)
plt.imshow(x_blurred_3x3, cmap='gray')
plt.title('Blurred Image_3x3 scaled to 2x2')
plt.axis('off')
fig.add_subplot(3, 3, 5)
plt.imshow(y_blurred_3x3, cmap='gray')
plt.title('Blurred Image_3x3 scaled to 4x4')
plt.axis('off')
fig.add_subplot(3, 3, 6)
plt.imshow(x_blurred_5x5, cmap='gray')
plt.title('Blurred Image_5x5 scaled to 2x2')
plt.axis('off')
fig.add_subplot(3, 3, 7)
plt.imshow(y_blurred_5x5, cmap='gray')
plt.title('Blurred Image_5x5 scaled to 4x4')
plt.axis('off')
plt.show()
Google Ch

localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

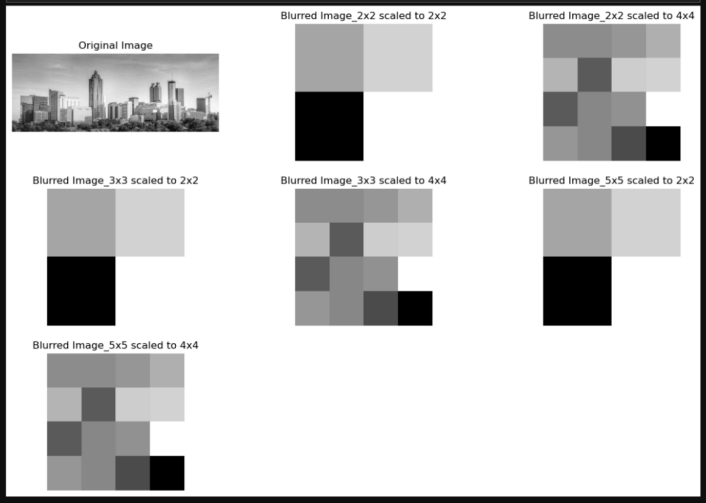
File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3.12

```
plt.imshow(x_blurred_5x5, cmap='gray')
plt.title('Blurred Image_5x5 scaled to 2x2')
plt.axis('off')

fig.add_subplot(3, 3, 7)
plt.imshow(y_blurred_5x5, cmap='gray')
plt.title('Blurred Image_5x5 scaled to 4x4')
plt.axis('off')
plt.show()
```



[53]:
```
# Use other images of your choice
# For a challenge, see if you can put the image through a series of filters: first one that blurs the image (takes an average of pixels),
# and then one that detects the edges.
```

[61]:
```
image_2 = cv2.imread('OtherImage.jpg')
image_2 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_2)
```

[61]: `<matplotlib.image.AxesImage at 0x134bd27b8>`

localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

Trusted

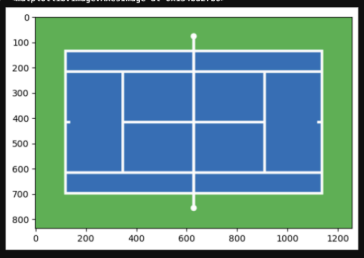
Code

JupyterLab Python 3.12

[53]:
```
# Use other images of your choice
# For a challenge, see if you can put the image through a series of filters: first one that blurs the image (takes an average of pixels),
# and then one that detects the edges.
```

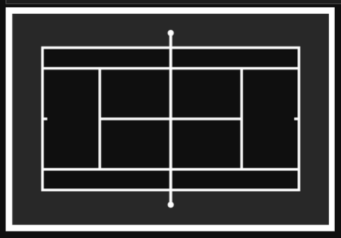
[61]:
```
image_2 = cv2.imread('OtherImage.jpg')
image_2 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_2)
```

[61]:
```
<matplotlib.image.AxesImage at 0x134bd27b0>
```



[56]:
```
#NOW I WILL PERFORM A SERIES OF FILTERS ON THIS IMAGE
```

[69]:
```
#1. corner detection using openCV Shi-Tomasi https://docs.opencv.org/3.4/d4/d8c/tutorial_py_shi_tomasi.html
img = cv2.imread('OtherImage.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY).astype(np.float32)
plt.imshow(gray, cmap='gray')
plt.axis('off')
plt.show()
```



[71]:
```
#https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga1d6bb77486c8f92d79c8793ad995d541

corners = cv2.goodFeaturesToTrack(gray, maxCorners=100, qualityLevel=0.01, minDistance=10)
corners = np.int0(corners)

# Mark the detected corners on the original image
for i in corners:
    x, y = i.ravel()
    cv2.circle(image_rgb, (x, y), 5, (255, 0, 0), -1)

# Display the image with detected corners
plt.imshow(image_rgb)
```

localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

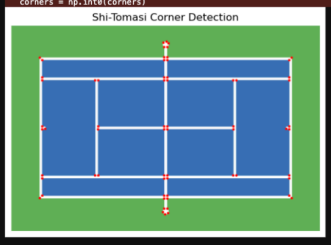
Trusted

Code

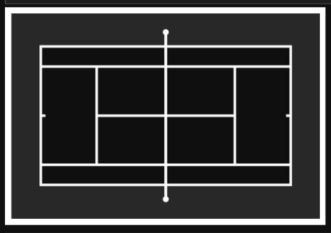
JupyterLab Python 3.12

[71]:
```
#https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga1d6bb77486c8f92d79c8793ad995d541

corners = cv2.goodFeaturesToTrack(gray, maxCorners=100, qualityLevel=0.01, minDistance=10)
corners = np.int0(corners)

# Mark the detected corners on the original image
for i in corners:
    x, y = i.ravel()
    cv2.circle(image_rgb, (x, y), 5, (255, 0, 0), -1)

# Display the image with detected corners
plt.imshow(image_rgb)
plt.title("Shi-Tomasi Corner Detection")
plt.axis("off")
plt.show()
```

```
/var/folders/rc/7hcf34jn40g30rgtz1vjrrnw0000gn/T/ipykernel_24281/1412757133.py:4: DeprecationWarning: `np.int0` is a deprecated alias for `n
p.intp`.  (Deprecated NumPy 1.24)
  corners = np.int0(corners)
```



[73]:
```
#2. Blurring this Image but instead of using custom kernels i will try using OpenCV fucntion blur()
#https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga8c45db9afe636703801b0b2e440fce37
img = cv2.imread('OtherImage.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY).astype(np.float32)
plt.imshow(gray, cmap='gray')
plt.axis('off')
plt.show()
```



[80]:
```
blurred = cv2.blur(gray, (12, 12))
plt.imshow(blurred, cmap='gray')
plt.axis('off')
plt.title("Blurred Image using cv2.blur()")
```

Termina

localhost:8888/notebooks/Desktop/CSUChico/Spring2025/CSCI611/A2/image_filtering/image_filtering.ipynb

jupyter image_filtering Last Checkpoint: 2 hours ago

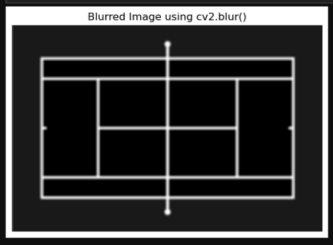
File Edit View Run Kernel Settings Help

Trusted

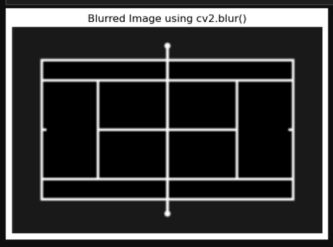
Code

JupyterLab Python 3.12

[80]:
```
blurred = cv2.blur(gray, (12, 12))
plt.imshow(blurred, cmap='gray')
plt.axis('off')
plt.title("Blurred Image using cv2.blur()")
plt.show()
```



[79]:
```
blurred = cv2.blur(gray, (9, 9))
plt.imshow(blurred, cmap='gray')
plt.axis('off')
plt.title("Blurred Image using cv2.blur()")
plt.show()
```



[81]:
```
#takes an average of pixels
#using OpneCV mean()
# https://docs.opencv.org/4.x/d2/de8/group__core__array.html#ga191389f8a0e58180bb13a727782cd461
mean_value = cv2.mean(gray)

print("Mean Pixel Value of the Image:", mean_value[0])
```

```
Mean Pixel Value of the Image: 143.39767489257306
```

[ ]: