CV Studio

Transfer Learning with Convolutional Neural Networks For Classification with PyTorch and Computer Vision Learning Studio (CV Studio)

V 0.2

Project: Final_project_stop_signs

Training Run: CV-selfdriving-cars

Estimated time needed: **40** minutes

In this lab, you will train a deep neural network for image classification using transfer learning, the image dataset will automatically be download from your CV Studio account. Experiment with different hyperparameters.

Objectives

In this lab you will train a state of the art image classifier using and CV Studio, CV Studio is a fast, easy and collaborative open source image annotation tool for teams and individuals. In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a ConvNet on a very large dataset in the lab, then use this Network to train your model. We will use the Convolutional Network as a feature generator, only training the output layer. In general, 100-200 images will give you a good starting point, and it only takes about half an hour. Usually, the more images you add, the better your results, but it takes longer and the rate of improvement will decrease.

- Import Libraries and Define Auxiliary Functions
- Create Dataset Object
- Load Model and Train

Import Libraries and Define Auxiliary Functions

In [31]:
```
#! conda install -c pytorch torchvision
#! pip install skillsnetwork tqdm
#!pip install  skillsnetwork
```

Libraries for OS and Cloud

In [32]:
```
import os
import uuid
import shutil
import json
from botocore.client import Config
import ibm_boto3
import copy
from datetime import datetime
from skillsnetwork import cvstudio
```

Libraries for Data Processing and Visualization

In [33]:
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math
from matplotlib.pyplot import imshow
from tqdm import tqdm
from ipywidgets import IntProgress
import time
```

Deep Learning Libraries

In [34]:
```
import torch
import torchvision.models as models
from torch.utils.data import Dataset, DataLoader,random_split
from torch.optim import lr_scheduler
from torchvision import transforms
import torch.nn as nn
torch.manual_seed(0)
```

Out[34]:
```
<torch._C.Generator at 0x7e704868ec50>
```

Plot train cost and validation accuracy:

In [35]:
```
def plot_stuff(COST,ACC):
    fig, ax1 = plt.subplots()
    color = 'tab:red'
    ax1.plot(COST, color = color)
    ax1.set_xlabel('Iteration', color = color)
    ax1.set_ylabel('total loss', color = color)
    ax1.tick_params(axis = 'y', color = color)

    ax2 = ax1.twinx()
    color = 'tab:blue'
    ax2.set_ylabel('accuracy', color = color)  # we already handled the x-label wit
    ax2.plot(ACC, color = color)
    ax2.tick_params(axis = 'y', color = color)
    fig.tight_layout()  # otherwise the right y-label is slightly clipped

    plt.show()
```

Plot the transformed image:

In [36]:
```
def imshow_(inp, title=None):
    """Imshow for Tensor."""
    inp = inp .permute(1, 2, 0).numpy()
    print(inp.shape)
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)

    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)
    plt.show()
```

Compare the prediction and actual value:

In [37]:
```
def result(model,x,y):
    #x,y=sample
    z=model(x.unsqueeze_(0))
    _,yhat=torch.max(z.data, 1)

    if yhat.item()!=y:
        text="predicted: {} actual: {}".format(str(yhat.item()),y)
        print(text)
```

Define our device as the first visible cuda device if we have CUDA available:

In [38]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print("the device type is", device)
```

```
the device type is cpu
```

Load Data

In this section we will preprocess our dataset by changing the shape of the image, converting to tensor and normalizing the image channels. These are the default preprocessing steps for image data. In addition, we will perform data augmentation on the training dataset. The preprocessing steps for the test dataset is the same, but W do not prform data augmentation on the test dataset.

```
mean = [0.485, 0.456, 0.406]


std = [0.229, 0.224, 0.225]


composed = transforms.Compose([transforms.Resize((224, 224)),

transforms.RandomHorizontalFlip(),transforms.RandomRotation(degrees=5)
                                , transforms.ToTensor()
                                , transforms.Normalize(mean, std)])
```

Download the data:

In [39]:
```
# Get the Dataset
# Initialize the CV Studio Client
cvstudioClient = cvstudio.CVStudio()
# # Download All Images
cvstudioClient.downloadAll()
```

```
100%|████████| 197/197 [01:10<00:00,  2.81it/s]
```

We need to get our training and validation dataset. 90% of the data will be used for training.

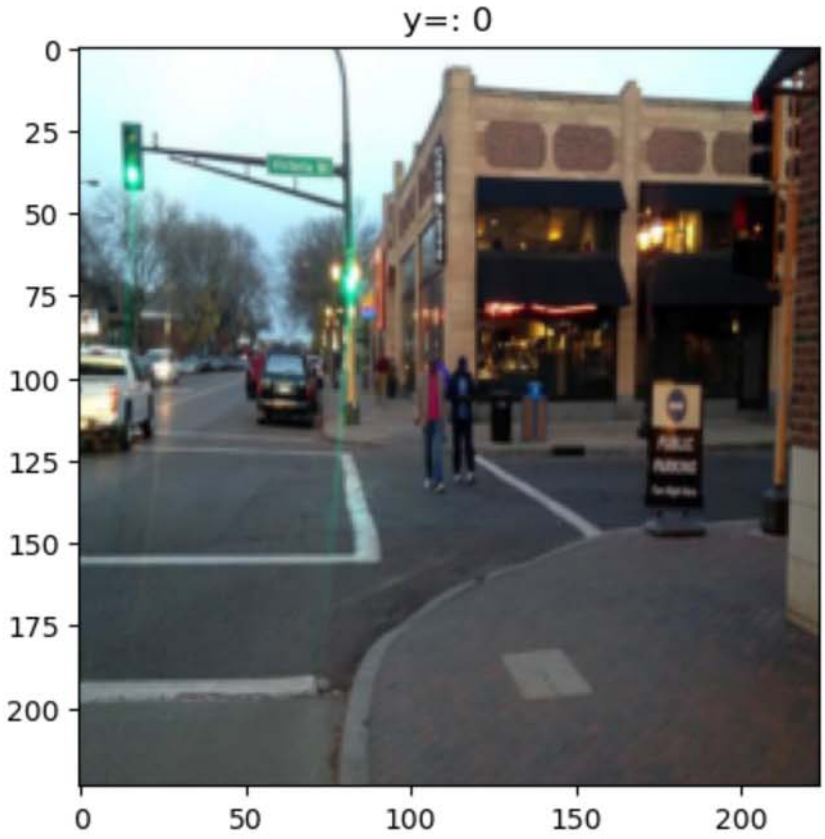
In [40]:
```
percentage_train=0.9
train_set=cvstudioClient.getDataset(train_test='train',percentage_train=percentage_
val_set=cvstudioClient.getDataset(train_test='test',percentage_train=percentage_tra
```

```
defult transform for pretrained model resnet18
this is the training set
defult transform for pretrained model resnet18
this is the test set
```

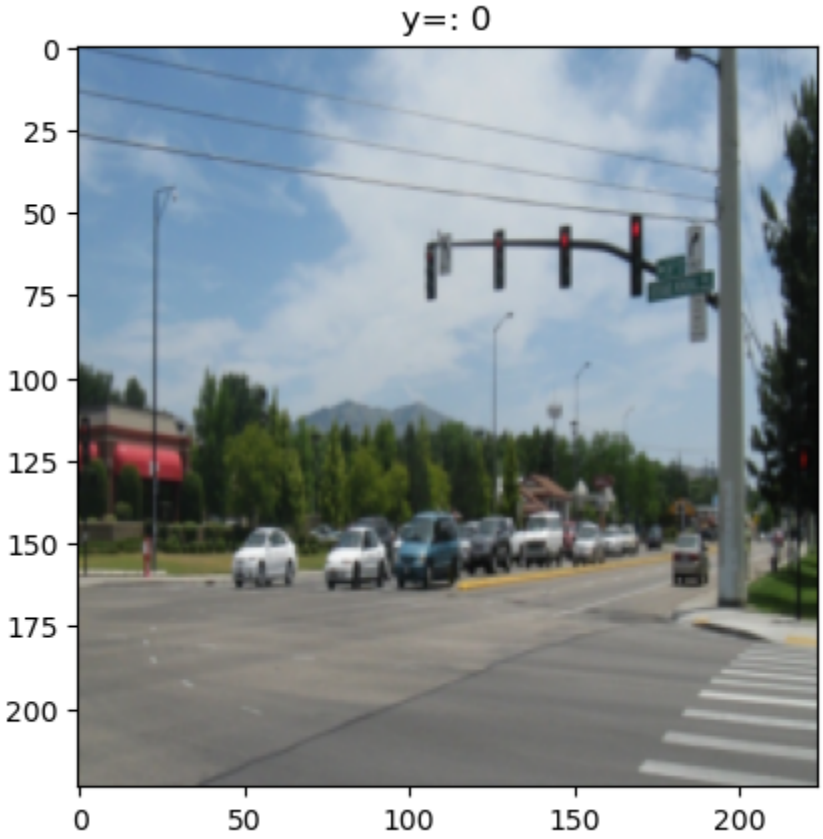
We can plot some of our dataset:

In [41]:
```
i=0

for x,y  in val_set:
    imshow_(x,"y=: {}".format(str(y.item())))
    i+=1
    if i==3:
        break
```

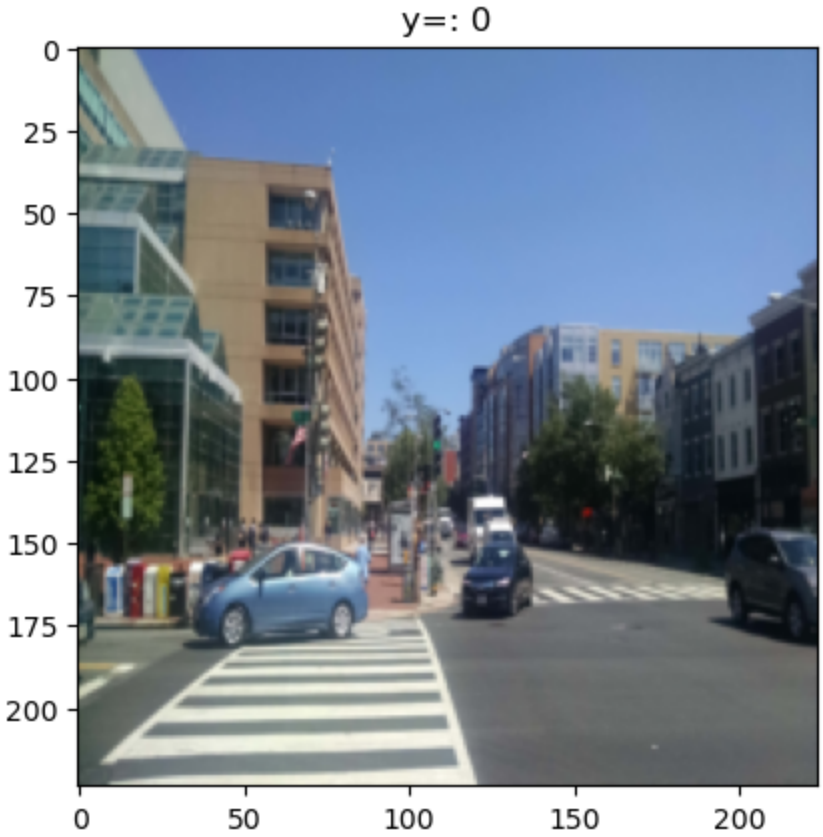
```
(224, 224, 3)
```




```
(224, 224, 3)
```




```
(224, 224, 3)
```



Hyperparameters

Experiment with different hyperparameters:

Epoch indicates the number of passes of the entire training dataset, here we will set the number of epochs to 10:

In [42]:
```
n_epochs=10
```

Batch size is the number of training samples utilized in one iteration. If the batch size is equal to the total number of samples in the training set, then every epoch has one iteration. In Stochastic Gradient Descent, the batch size is set to one. A batch size of 32--512 data points seems like a good value, for more information check out the following link.

In [43]:
```
batch_size=32
```

Learning rate is used in the training of neural networks. Learning rate is a hyperparameter with a small positive value, often in the range between 0.0 and 1.0.

In [44]:
```
lr=0.000001
```

Momentum is a term used in the gradient descent algorithm to improve training results:

In [45]:
```
momentum=0.9
```

If you set to `lr_scheduler=True` for every epoch use a learning rate scheduler changes the range of the learning rate from a maximum or minimum value. The learning rate usually decays over time.

In [46]:
```
lr_scheduler=True
base_lr=0.001
max_lr=0.01
```

Load Model and Train

This function will train the model

In [47]:
```
def train_model(model, train_loader,validation_loader, criterion, optimizer, n_epoc
    loss_list = []
    accuracy_list = []
    correct = 0
    #global:val_set
    n_test = len(val_set)
    accuracy_best=0
    best_model_wts = copy.deepcopy(model.state_dict())

    # Loop through epochs
        # Loop through the data in loader
    print("The first epoch should take several minutes")
    for epoch in tqdm(range(n_epochs)):

        loss_sublist = []
        # Loop through the data in loader

        for x, y in train_loader:
            x, y=x.to(device), y.to(device)
            model.train()

            z = model(x)
            loss = criterion(z, y)
            loss_sublist.append(loss.data.item())
            loss.backward()
            optimizer.step()

            optimizer.zero_grad()
        print("epoch {} done".format(epoch) )

        scheduler.step()
        loss_list.append(np.mean(loss_sublist))
        correct = 0


        for x_test, y_test in validation_loader:
            x_test, y_test=x_test.to(device), y_test.to(device)
            model.eval()
            z = model(x_test)
            _, yhat = torch.max(z.data, 1)
            correct += (yhat == y_test).sum().item()
        accuracy = correct / n_test
        accuracy_list.append(accuracy)
        if accuracy>accuracy_best:
            accuracy_best=accuracy
            best_model_wts = copy.deepcopy(model.state_dict())


        if print_:
            print('learning rate',optimizer.param_groups[0]['lr'])
            print("The validaion  Cost for each epoch " + str(epoch + 1) + ": " + s
            print("The validation accuracy for epoch " + str(epoch + 1) + ": " + st
    model.load_state_dict(best_model_wts)
    return accuracy_list,loss_list, model
```

Load the pre-trained model resnet18. Set the parameter pretrained to true.

In [48]:
```
model = models.resnet18(pretrained=True)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/torchvision/models/_u
tils.py:209: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and ma
y be removed in the future, please use 'weights' instead.
  f"The parameter '{pretrained_param}' is deprecated since 0.13 and may be removed i
n the future, "
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/torchvision/models/_u
tils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights'
are deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use `we
ights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /home/ju
pyterlab/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
  0%|          | 0.00/44.7M [00:00<?, ?B/s]
```

We will only train the last layer of the network set the parameter `requires_grad` to `False`, the network is a fixed feature extractor.

In [49]:
```
for param in model.parameters():
        param.requires_grad = False
```

Number of classes

In [50]:
```
n_classes=train_set.n_classes
n_classes
```

Out[50]: 2

Replace the output layer model.fc of the neural network with a nn.Linear object, to classify `n_classes` different classes. For the parameters in_features remember the last hidden layer has 512 neurons.

In [51]:
```
# Type your code here
model.fc = nn.Linear(512, n_classes)
```

Set device type

In [52]:
```
model.to(device)
```

Out[52]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=Fa
lse)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=Fa
lse)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
```

```
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
```

```
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=2, bias=True)
)
```

Cross-entropy loss, or log loss, measures the performance of a classification model combines LogSoftmax in one object class. It is useful when training a classification problem with C classes.

In [53]:
```
criterion = nn.CrossEntropyLoss()
```

Create a training loader and validation loader object.

In [54]:
```
train_loader = torch.utils.data.DataLoader(dataset=train_set , batch_size=batch_siz
validation_loader= torch.utils.data.DataLoader(dataset=val_set , batch_size=1)
```

Use the optim package to define an Optimizer that will update the weights of the model for us.

In [55]:
```
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=momentum)
```

We use Cyclical Learning Rates

In [56]:
```
if lr_scheduler:
    scheduler = torch.optim.lr_scheduler.CyclicLR(optimizer, base_lr=0.001, max_lr=
```

Now we are going to train model,for 500 images this take 25 minutes, depending on your dataset

In [57]:
```
start_datetime = datetime.now()
start_time=time.time()

accuracy_list,loss_list, model=train_model(model,train_loader , validation_loader,

end_datetime = datetime.now()
current_time = time.time()
elapsed_time = current_time - start_time
print("elapsed time", elapsed_time )
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
The first epoch should take several minutes
epoch 0 done
 10%|▉         | 1/10 [00:28<04:15, 28.42s/it]
learning rate 0.002800000000000002
The validaion  Cost for each epoch 1: 0.679717351992925
The validation accuracy for epoch 1: 0.6
epoch 1 done
 20%|██        | 2/10 [00:55<03:42, 27.76s/it]
learning rate 0.0046
The validaion  Cost for each epoch 2: 0.5436734904845556
The validation accuracy for epoch 2: 0.9
epoch 2 done
 30%|███       | 3/10 [01:22<03:12, 27.50s/it]
learning rate 0.006400000000000001
The validaion  Cost for each epoch 3: 0.3868098904689153
The validation accuracy for epoch 3: 0.85
epoch 3 done
 40%|████      | 4/10 [01:49<02:43, 27.23s/it]
learning rate 0.008199999999999999
The validaion  Cost for each epoch 4: 0.4068247253696124
The validation accuracy for epoch 4: 0.8
epoch 4 done
 50%|█████     | 5/10 [02:15<02:13, 26.80s/it]
```

```
learning rate 0.010000000000000002
The validaion  Cost for each epoch 5: 0.45595018814007443
The validation accuracy for epoch 5: 0.65
epoch 5 done
 60%|██████    | 6/10 [02:42<01:47, 26.76s/it]
learning rate 0.008199999999999999
The validaion  Cost for each epoch 6: 0.432115625590086
The validation accuracy for epoch 6: 0.65
epoch 6 done
 70%|███████   | 7/10 [03:10<01:21, 27.08s/it]
learning rate 0.006400000000000001
The validaion  Cost for each epoch 7: 0.27116487671931583
The validation accuracy for epoch 7: 0.95
epoch 7 done
 80%|████████  | 8/10 [03:37<00:54, 27.05s/it]
learning rate 0.0046
The validaion  Cost for each epoch 8: 0.15089798097809157
The validation accuracy for epoch 8: 0.8
epoch 8 done
 90%|█████████ | 9/10 [04:06<00:27, 27.82s/it]
learning rate 0.002800000000000002
The validaion  Cost for each epoch 9: 0.23049759740630785
The validation accuracy for epoch 9: 0.9
epoch 9 done
100%|██████████| 10/10 [04:34<00:00, 27.46s/it]
learning rate 0.001
The validaion  Cost for each epoch 10: 0.19474557787179947
The validation accuracy for epoch 10: 0.8
elapsed time 274.7476177215576
```

Now run the following to report back the results of the training run to CV Studio

In [58]:
```
parameters = {
    'epochs': n_epochs,
    'learningRate': lr,
    'momentum':momentum,
    'percentage used training':percentage_train,
    "learningRatescheduler": {"lr_scheduler":lr_scheduler,"base_lr":base_lr, "max_l


}
result = cvstudioClient.report(started=start_datetime, completed=end_datetime, para

if result.ok:
    print('Congratulations your results have been reported back to CV Studio!')
```

```
Congratulations your results have been reported back to CV Studio!
```

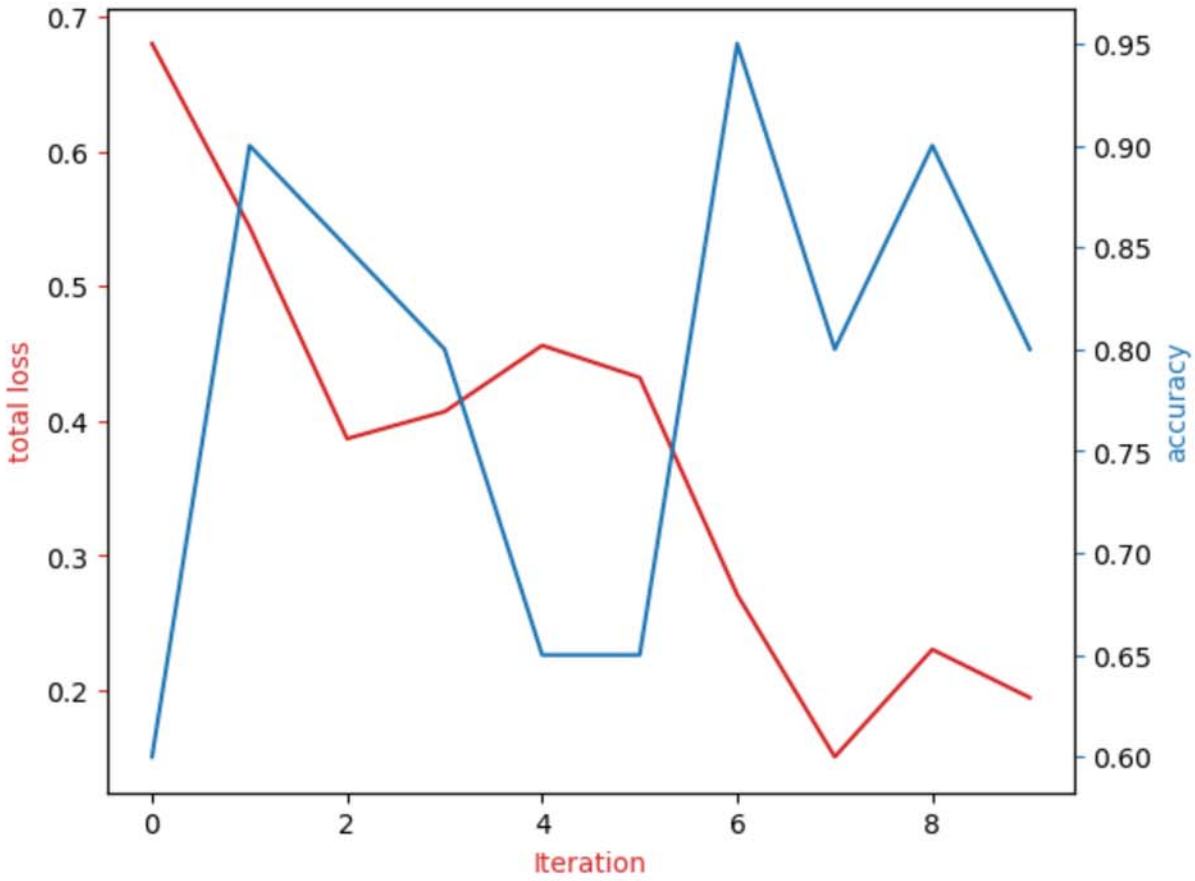
Save the model to model.pt

In [59]:
```
# Save the model to model.pt
torch.save(model.state_dict(), 'model.pt')

# Save the model and report back to CV Studio
result = cvstudioClient.uploadModel('model.pt', {'numClasses': n_classes})
```

```
File Uploaded
```

Plot train cost and validation accuracy, you can improve results by getting more data.

In [60]:
```
plot_stuff(loss_list,accuracy_list)
```



Load the model that performs best:

In [61]:
```
model = models.resnet18(pretrained=True)
model.fc = nn.Linear(512, n_classes)
model.load_state_dict(torch.load( "model.pt"))
model.eval()
```

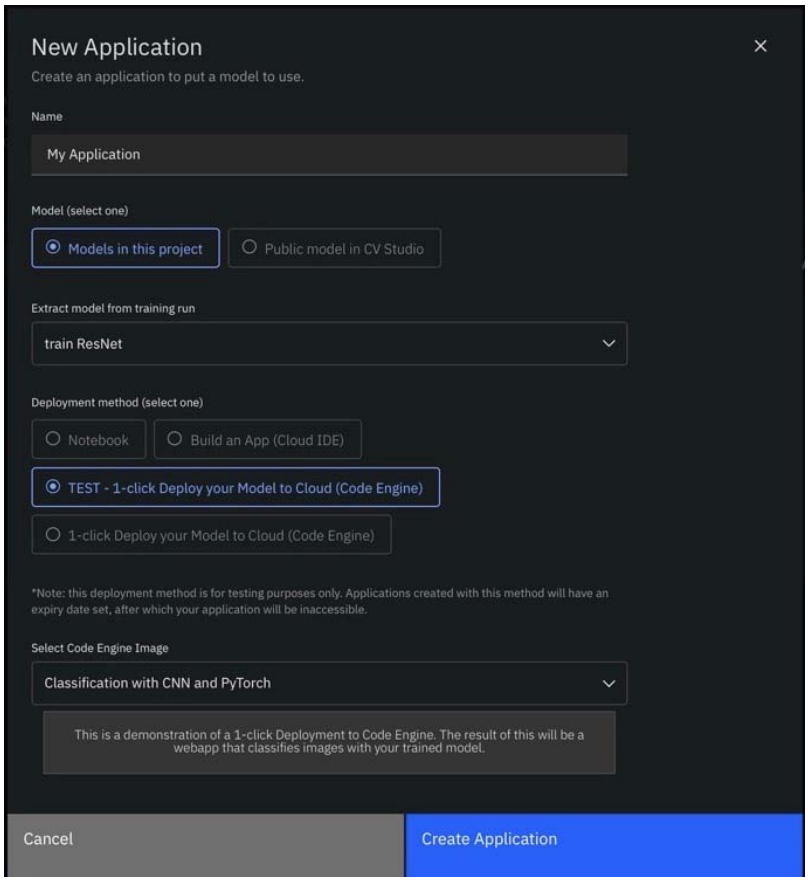
Out[61]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=Fa
lse)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=Fa
lse)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
```

```
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
```

```
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=2, bias=True)
)
```

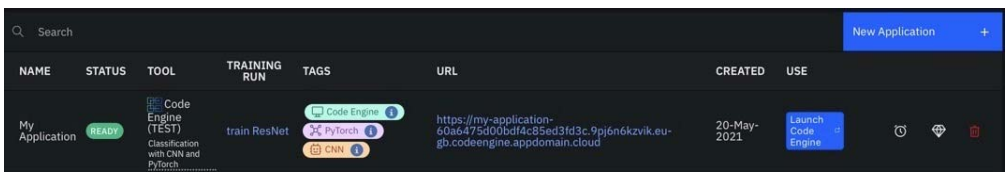
What's Next

You can also deploy your model via Web Application or Web App . This allows users to interact with your model like a website. They can upload the image with a user interface and view the results. Let's see how we can deploy a web app in CV Studio. In CV Studio, go to the use model section and select New Application. Fill out the window as follows, giving your model a name and selecting the Model in this project, select **TEST - 1-click Deploy your Model to Cloud (Code Engine)** and select the model from the training run as shown here:

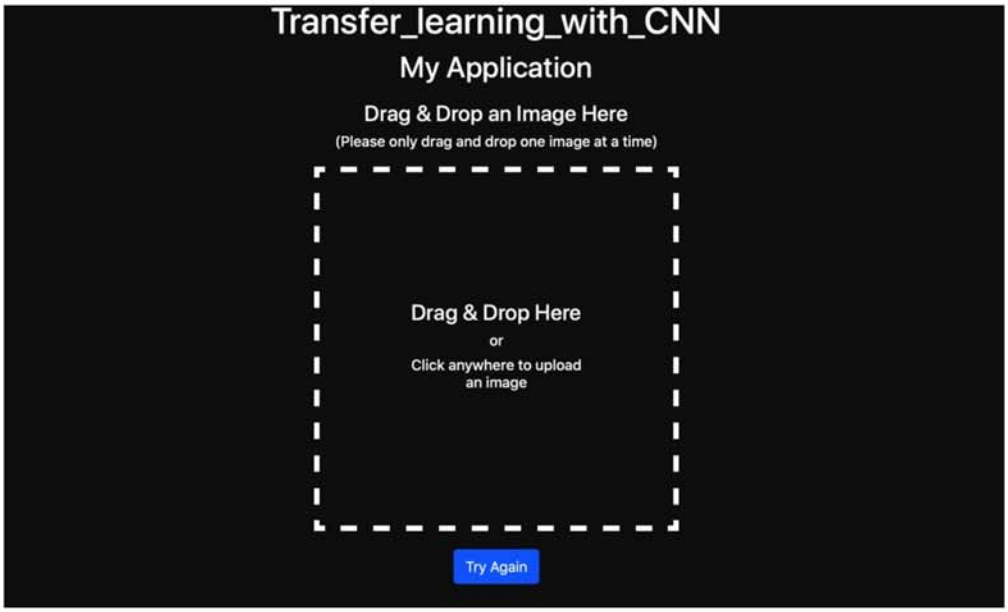


Then once the window is filled out press the Create Application button and your model will begin deploying.

Wait until the status changes from "deploying" to "ready". Once the status changes to ready, your application is ready for you to use!



You can press the URL to go to your web application.



Authors

Joseph Santarcangelo,has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-05-25	0.3	Yasmine	Modifies Multiple Areas
2021-05-25	0.3	Kathy	Modified Multple Areas.
2021-03-08	0.2	Joseph	Modified Multiple Areas
2021-02-01	0.1	Joseph	Modified Multiple Areas

In [2]:
```
from PIL import Image
import torch
from torchvision import transforms
```

In [14]:
```
import torch.nn as nn
import torch
from torchvision import models

# Example using pretrained ResNet18 (change based on your model)
model = models.resnet18()
model.fc = nn.Linear(model.fc.in_features, 2)  # 2 classes: Stop and Not Stop

# Load the saved weights
model.load_state_dict(torch.load('model.pt', map_location='cpu'))
model.eval()
```

Out[14]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=Fa
lse)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=Fa
lse)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), b
ias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
```

```
bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
```

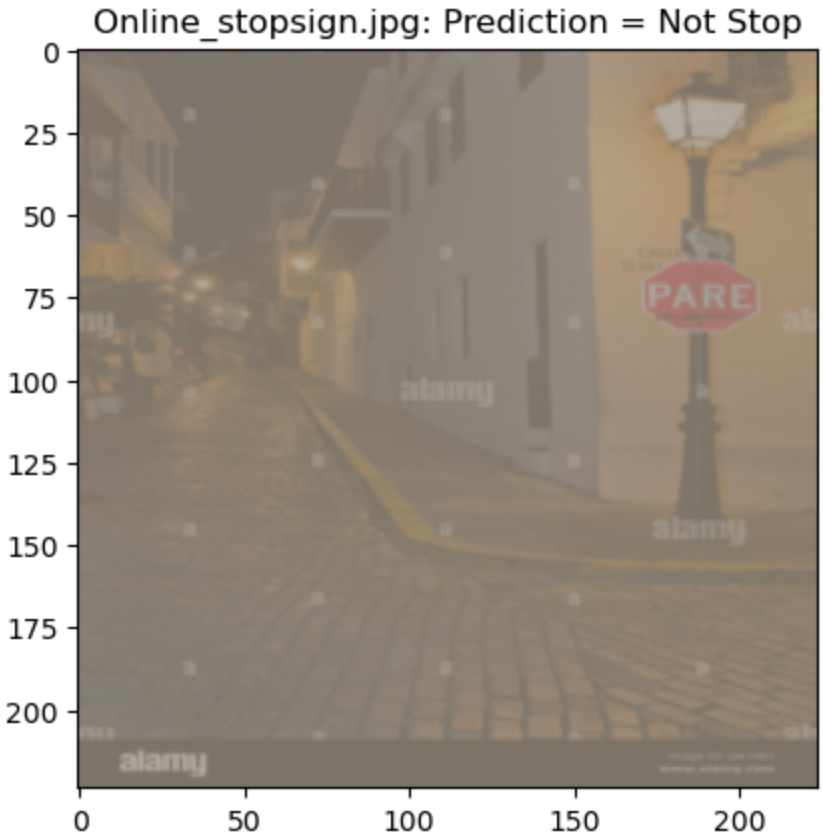
```
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=2, bias=True)
)
```

In [63]:
```
imageNames = ['stop_1.jpg','stop_2.jpg','not_stop_1.jpg']
for imageName in imageNames:
    image = Image.open(imageName)
    transform = composed = transforms.Compose([transforms.Resize((224, 224)), trans
    x = transform(image)
    z=model(x.unsqueeze_(0))
    _,yhat=torch.max(z.data, 1)
    # print(yhat)
    prediction = "Stop"
    if yhat == 1:
        prediction ="Not Stop"
    imshow_(transform(image),imageName+": Prediction = "+prediction)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipykernel_381/4203486423.py in <module>
      1 imageNames = ['stop_1.jpg','stop_2.jpg','not_stop_1.jpg']
      2 for imageName in imageNames:
----> 3     image = Image.open(imageName)
      4     transform = composed = transforms.Compose([transforms.Resize((224, 22
4)), transforms.ToTensor()])
      5     x = transform(image)

~/conda/envs/python/lib/python3.7/site-packages/PIL/Image.py in open(fp, mode, forma
ts)
   2902
   2903     if filename:
-> 2904         fp = builtins.open(filename, "rb")
   2905         exclusive_fp = True
   2906

FileNotFoundError: [Errno 2] No such file or directory: 'stop_1.jpg'
```

In [64]:
```
imageNames = ['Online_stopsign.jpg','online_without_stop_sign_bridge.jpg','street_w
for imageName in imageNames:
    image = Image.open(imageName)
    transform = composed = transforms.Compose([transforms.Resize((224, 224)), trans
    x = transform(image)
    z=model(x.unsqueeze_(0))
    _,yhat=torch.max(z.data, 1)
    # print(yhat)
    prediction = "Stop"
    if yhat == 1:
        prediction ="Not Stop"
    imshow_(transform(image),imageName+": Prediction = "+prediction)
```

```
(224, 224, 3)
```

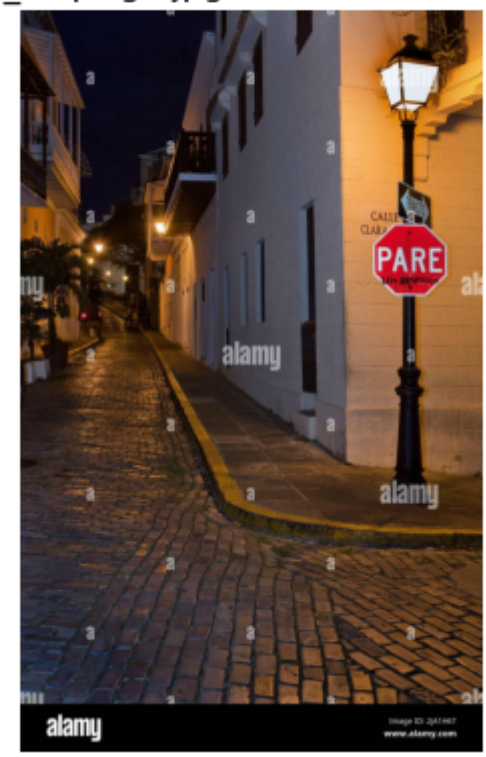



```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipykernel_381/4144860413.py in <module>
      1 imageNames = ['Online_stopsign.jpg','online_without_stop_sign_bridge.jpg','s
treet_without_stopsign.webp']
      2 for imageName in imageNames:
----> 3     image = Image.open(imageName)
      4     transform = composed = transforms.Compose([transforms.Resize((224, 22
4)), transforms.ToTensor()])
      5     x = transform(image)

~/conda/envs/python/lib/python3.7/site-packages/PIL/Image.py in open(fp, mode, forma
ts)
   2902
   2903     if filename:
-> 2904         fp = builtins.open(filename, "rb")
   2905         exclusive_fp = True
   2906

FileNotFoundError: [Errno 2] No such file or directory: 'online_without_stop_sign_br
idge.jpg'
```

In [65]:
```
import torch
from torchvision import transforms, models
from PIL import Image
import matplotlib.pyplot as plt

# Load model architecture and weights (adjust if needed)
model = models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(model.fc.in_features, 2)  # Assuming binary classificati
model.load_state_dict(torch.load('model.pt', map_location='cpu'))
model.eval()

# Define transform
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

# Images to test
imageNames = ['Online_stopsign.jpg', 'online_without_stopsign_bridge.jpg', 'street_

# Loop through images
for imageName in imageNames:
    image = Image.open(imageName).convert('RGB')
    x = transform(image).unsqueeze(0)  # Add batch dimension

    with torch.no_grad():
        z = model(x)
        _, yhat = torch.max(z.data, 1)

    prediction = "Stop" if yhat.item() == 0 else "Not Stop"

    # Show image with prediction
    plt.imshow(image)
    plt.title(f"{imageName}: Prediction = {prediction}")
    plt.axis('off')
    plt.show()
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/torchvision/models/_u
tils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights'
are deprecated since 0.13 and may be removed in the future. The current behavior is
equivalent to passing `weights=None`.
  warnings.warn(msg)
```

Online_stopsign.jpg: Prediction = Not Stop



online_without_stopsign_bridge.jpg: Prediction = Stop

In [101…
```
import os
print(os.getcwd())
```

```
/resources/labs/cvstudio/final-project-stop-signs/runs/cv-selfdriving-cars
```

In [102…
```
import os

folder = '/resources/labs/cvstudio/final-project-stop-signs/runs/cv-selfdriving-car

files = os.listdir(folder)
print(f"Files in '{folder}':")
for f in files:
    print(f)
```

```
Files in '/resources/labs/cvstudio/final-project-stop-signs/runs/cv-selfdriving-car
s':
.cvstudio.json
.ipynb_checkpoints
Online_stopsign.jpg
images
missclassified.png
model.pt
not_stop
online_without_stopsign_bridge.jpg
stop
street_without_stopsign.webp
test_set_stop_not_stop
train-classification-cnn-pytorch.ipynb
```

In [100…
```python
import os

folder = 'stop'  # or the path you want to check

files = os.listdir(folder)
print(f"Files in '{folder}':")
for f in files:
    print(f)
```

```
Files in 'stop':
1.jpg
2.jpg
3.jpg
4.png
```

In [86]:
```python
import os

folder = 'not_stop'  # or the path you want to check

files = os.listdir(folder)
print(f"Files in '{folder}':")
for f in files:
    print(f)
```

```
Files in 'not_stop':
101.jpg
102.jpg
103.jpg
104.jpg
105.jpg
106.jpg
107.jpg
108.jpg
109.jpg
110.jpg
111.jpg
112.jpg
113.jpg
114.jpg
115.jpeg
116.jpg
117.JPG
118.JPG
119.jpg
120.jpg
121.JPG
122.jpg
123.jpg
124.jpg
125.JPG
126.JPG
127.jpg
128.jpg
129.jpg
130.jpg
131.jpeg
132.jpg
133.jpg
134.jpg
135.jpg
136.jpg
137.jpg
138.JPG
139.jpg
140.jpg
141.jpg
142.jpg
143.jpg
144.jpg
145.jpg
146.jpg
147.jpg
148.jpg
149.jpg
150.png
151.jpg
152.jpg
153.jpeg
154.jpg
155.jpg
```

156.jpg
157.jpg
158.jpg
159.jpg
160.jpg
161.jpg
162.jpg
163.jpg
164.jpg
165.jpg
166.jpg
167.jpg
168.JPG
169.jpg
170.png
171.jpeg
172.jpg
173.jpg
174.jpg
175.jpg
176.jpg
177.jpg
178.jpg
179.jpg
180.jpg
181.jpg
182.jpg
183.jpg
184.jpg
185.jpg
186.jpg
187.jpg
188.jpg
189.jpg
190.jpg
191.jpg
192.jpg
193.jpg
194.jpg
195.jpg
196.jpg
197.jpg
198.jpeg
199.jpg
200.jpg

In [93]:
```python
import torch
from torchvision import transforms, models
from PIL import Image
import matplotlib.pyplot as plt
import os

# Device setup (GPU if available)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Load model architecture and weights
model = models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(model.fc.in_features, 2)  # Binary classification
model.load_state_dict(torch.load('model.pt', map_location=device))
model.to(device)
model.eval()

# Define transform with normalization (common for ResNet)
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

# Folder path and image names
folder = '/resources/labs/cvstudio/final-project-stop-signs/runs/cv-selfdriving-car
image_names = ['1.jpg', '2.jpg', '3.jpg', '4.png']

# Loop through images
for name in image_names:
    image_path = os.path.join(folder, name)
    image = Image.open(image_path).convert('RGB')
    x = transform(image).unsqueeze(0).to(device)  # Add batch and send to device

    with torch.no_grad():
        output = model(x)
        _, predicted = torch.max(output.data, 1)

    prediction = "Not Stop" if predicted.item() == 0 else "Stop"

    # Show image with prediction
    plt.imshow(image)
    plt.title(f"{name}: Prediction = {prediction}")
    plt.axis('off')
    plt.show()
```

1.jpg: Prediction = Stop

2.jpg: Prediction = Stop

3.jpg: Prediction = Stop

4.png: Prediction = Stop

In [96]:
```
import torch
from torchvision import transforms, models
from PIL import Image
import matplotlib.pyplot as plt
import os

# Load model architecture and weights
model = models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(model.fc.in_features, 2)  # Adjust based on your model
model.load_state_dict(torch.load('model.pt', map_location='cpu'))
model.eval()

# Define transform
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

# Folder path and image names
folder = '/resources/labs/cvstudio/final-project-stop-signs/runs/cv-selfdriving-car
image_names = ['101.jpg', '102.jpg', '103.jpg']

# Loop through images
for name in image_names:
    image_path = os.path.join(folder, name)
    image = Image.open(image_path).convert('RGB')
    x = transform(image).unsqueeze(0)

    with torch.no_grad():
        output = model(x)
        _, predicted = torch.max(output.data, 1)

    prediction = "Stop" if predicted.item() == 0 else "Not Stop"

    # Show image with prediction
    plt.imshow(image)
    plt.title(f"{name}: Prediction = {prediction}")
    plt.axis('off')
    plt.show()
```

101.jpg: Prediction = Stop

102.jpg: Prediction = Not Stop

103.jpg: Prediction = Stop

In [67]:
```
import os
from PIL import Image
import torch
from torchvision import transforms

# Your model should be already defined and loaded
# Example: model = torch.load('model.pth')
# model.eval()

# Define transform
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

# Folder path
folder_path = 'test_set_stop_not_stop'

# List all image files (filter out non-image files)
image_names = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.jpeg', '

for image_name in image_names:
    image_path = os.path.join(folder_path, image_name)
    image = Image.open(image_path).convert('RGB')

    x = transform(image)
    x = x.unsqueeze(0)  # Add batch dimension

    with torch.no_grad():
        z = model(x)
        _, yhat = torch.max(z.data, 1)

    prediction = "Stop"
    if yhat.item() == 1:
        prediction = "Not Stop"

    # Display result
    print(f"{image_name}: Prediction = {prediction}")
```

```
not_stop_1.jpeg: Prediction = Stop
not_stop_2.jpeg: Prediction = Stop
not_stop_3.jpeg: Prediction = Not Stop
not_stop_4.jpeg: Prediction = Stop
stop_1.jpeg: Prediction = Not Stop
stop_2.jpeg: Prediction = Not Stop
stop_3.jpeg: Prediction = Not Stop
```

In [95]:
```
import torch
from torchvision import transforms, models
from PIL import Image
import matplotlib.pyplot as plt

# Load model architecture and weights (adjust if needed)
model = models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(model.fc.in_features, 2)  # Assuming binary classificati
model.load_state_dict(torch.load('model.pt', map_location='cpu'))
model.eval()

# Define transform
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

# Images to test
imageNames = ['Online_stopsign.jpg', 'online_without_stopsign_bridge.jpg', 'street_

# Loop through images
for imageName in imageNames:
    image = Image.open(imageName).convert('RGB')
    x = transform(image).unsqueeze(0)  # Add batch dimension

    with torch.no_grad():
        z = model(x)
        _, yhat = torch.max(z.data, 1)

    prediction = "Not Stop" if yhat.item() == 0 else "Stop"

    # Show image with prediction
    plt.imshow(image)
    plt.title(f"{imageName}: Prediction = {prediction}")
    plt.axis('off')
    plt.show()
```

Online_stopsign.jpg: Prediction = Stop

online_without_stopsign_bridge.jpg: Prediction = Not Stop

street_without_stopsign.webp: Prediction = Not Stop

In [104…

```python
#Model Retraining
import torch
from torchvision import transforms, models
from PIL import Image
import torch.nn as nn
import torch.optim as optim

# ----------------------------
# 📥 INPUTS:
# - A pre-trained model ('model.pt')
# - One misclassified image: 'misclassified.jpg'
# - Correct label: 0 ('Stop' class)
# ----------------------------

# Set device (GPU if available)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# ----------------------------
# 🔧 MODEL SETUP:
# Load the model architecture and existing weights
# ----------------------------
model = models.resnet18(pretrained=False)
model.fc = nn.Linear(model.fc.in_features, 2)  # Assuming binary classification
model.load_state_dict(torch.load('model.pt', map_location=device))
model.to(device)
model.train()  # Set model to training mode for fine-tuning

# ----------------------------
# 🔄 TRANSFORM:
# Resize, convert to tensor, normalize (ResNet18 expects ImageNet-style normalizati
# ----------------------------
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

# ----------------------------
# 📷 LOAD MISCLASSIFIED IMAGE:
# Apply transform and add batch dimension
# ----------------------------
image_path = '/resources/labs/cvstudio/final-project-stop-signs/runs/cv-selfdriving
correct_label = 0  # 0 = Stop, 1 = Not Stop

image = Image.open(image_path).convert('RGB')
x = transform(image).unsqueeze(0).to(device)  # Shape: [1, 3, 224, 224]
y = torch.tensor([correct_label], dtype=torch.long).to(device)

# ----------------------------
# 🎯 TRAINING SETUP:
# Loss function and optimizer
# ----------------------------
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)  # Low LR for fine-tuning
```

```python
# ----------------------------
# 🔁 FINE-TUNE THE MODEL:
# Just a few steps to adjust weights based on this single image
# ----------------------------
for step in range(10):
    optimizer.zero_grad()
    output = model(x)
    loss = criterion(output, y)
    loss.backward()
    optimizer.step()
    print(f"Step {step + 1}, Loss: {loss.item():.6f}")

# ----------------------------
# 💾 OUTPUT:
# Save the fine-tuned model
# ----------------------------
torch.save(model.state_dict(), 'model_finetuned.pt')

# ----------------------------
# 🧪 VERIFY CORRECTION:
# Use the updated model to re-predict the same image
# ----------------------------
model.eval()
with torch.no_grad():
    output = model(x)
    _, predicted = torch.max(output.data, 1)

label = "Stop" if predicted.item() == 0 else "Not Stop"
print(f"✅ Prediction after fine-tuning: {label}")
```

```
Step 1, Loss: 0.767158
Step 2, Loss: 0.591651
Step 3, Loss: 0.477177
Step 4, Loss: 0.397080
Step 5, Loss: 0.338294
Step 6, Loss: 0.292971
Step 7, Loss: 0.257820
Step 8, Loss: 0.229238
Step 9, Loss: 0.204678
Step 10, Loss: 0.183850
✅ Prediction after fine-tuning: Stop
```

In [105…

```
import matplotlib.pyplot as plt
from PIL import Image

# Load the original image for visualization
image = Image.open('missclassified.png').convert('RGB')

# Reapply the transform
x = transform(image).unsqueeze(0).to(device)

# Inference after fine-tuning
model.eval()
with torch.no_grad():
    output = model(x)
    _, predicted = torch.max(output.data, 1)

# Map predicted class to label
prediction_label = "Stop" if predicted.item() == 0 else "Not Stop"

# Visualize the image with prediction as annotation
plt.imshow(image)
plt.title(f"Prediction after fine-tuning: {prediction_label}", fontsize=14)
plt.axis('off')
plt.show()
```