Problem 1

1.

```
(xray) jason (master) pytorch-ssd $ python eval_ssd.py --net mb1-ssd  --dataset ./data/VOC2007test/ --trained_model models/mobilenet-v1-ssd-mp-0_67
5.pth --label_file models/voc-model-labels.txt
```

```
process image 4950
Load Image: 0.003893 seconds.
Inference time:  0.04917764663696289
Prediction: 0.067090 seconds.
process image 4951
Load Image: 0.004049 seconds.
Inference time:  0.048860788345336914
Prediction: 0.070375 seconds.


Average Precision Per-class:
aeroplane: 0.6742489426027927
bicycle: 0.7913672875238116
bird: 0.612096015101108
boat: 0.5616407126931772
bottle: 0.3471259064860268
bus: 0.7742298893362103
car: 0.7284171192326804
cat: 0.8360675520354323
chair: 0.5142295855384792
cow: 0.6244090341627014
diningtable: 0.7060025454924147
dog: 0.7849252606216821
horse: 0.8202146617282785
motorbike: 0.793578272243471
person: 0.7042670984734087
pottedplant: 0.40257147509774405
sheep: 0.6071252282334352
sofa: 0.7549120254763918
train: 0.8270992920206008
tvmonitor: 0.6459903029666852

Average Precision Across All Classes:0.6755259103533267
```

2.

```
Prediction: 0.045243 seconds.
process image 120
Load Image: 0.032561 seconds.
Inference time:  0.008483171463012695
Prediction: 0.049908 seconds.
process image 121
Load Image: 0.036689 seconds.
Inference time:  0.008579492568969727
Prediction: 0.048602 seconds.
process image 122
Load Image: 0.018957 seconds.
Inference time:  0.008281946182250977
Prediction: 0.051239 seconds.


Average Precision Per-class:
Handgun: 0.015981481468796188
Shotgun: 0.005171740154507875 4

Average Precision Across All Classes:0.010576610811652032
```

no finetuning

```
Inference time:  0.11897158622741699
Prediction: 0.123757 seconds.


Average Precision Per-class:
Handgun: 0.8270707717875698
Shotgun: 0.5643195135016551

Average Precision Across All Classes:0.6956951426
446125
```

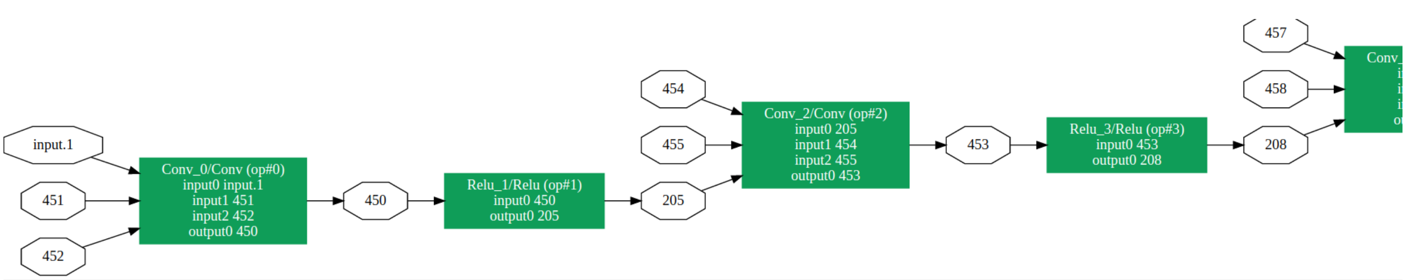
sorry it is hard to see pretrained handgun is 0.82 shotgun is 0.56

3.

```
(testssd) jason (master *) pytorch-ssd $ python convert_to_caffe2_models.py mb1-ssd mb1-ssd-Epoch-95
-Loss-2.7047196984291078.pth  models/open-images-model-labels.txt
```

[onnx file (https://drive.google.com/file/d/1hDzMHGq2f3GzQmGMdvr9mbuROuwqr3w1/view?usp=sharing)](https://drive.google.com/file/d/1hDzMHGq2f3GzQmGMdvr9mbuROuwqr3w1/view?usp=sharing)

4.



[link to svg (https://drive.google.com/file/d/1crto_mLPnFmmilJaels50hoENdgGhzul/view?usp=sharing)](https://drive.google.com/file/d/1crto_mLPnFmmilJaels50hoENdgGhzul/view?usp=sharing)

5.

```
(testssd) jason projects $ sudo docker run -it -v $(pwd):$(pwd) -p 127.0.0.1:9001:8001 mcr.microsoft
.com/onnxruntime/server --model_path $(pwd)/mb1-ssd.onnx
```

In [42]:
```
# Import some dependency libraries that we are going to need to run the

import numpy as np
import requests
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import onnx_ml_pb2 as onnx_ml_pb2
import predict_pb2 as predict_pb2
```

In [43]:
```
# Load the raw image

input_shape = (1, 3, 300, 300)
img = Image.open("handgun.jpg")
img = img.resize((300, 300), Image.BILINEAR)

# Let us see what the input image looks like
img
```

Out[43]:

In [44]:
```
img_data = np.array(img)
img_data = np.transpose(img_data, [2, 0, 1])
img_data = np.expand_dims(img_data, 0)
mean_vec = np.array([0.485, 0.456, 0.406])
stddev_vec = np.array([0.229, 0.224, 0.225])
norm_img_data = np.zeros(img_data.shape).astype('float32')
for i in range(img_data.shape[1]):
    norm_img_data[:,i,:,:] = (img_data[:,i,:,:]/255 - mean_vec[i]) / st
```

In [45]:
```
# Create request message to be sent to the ORT server

input_tensor = onnx_ml_pb2.TensorProto()
input_tensor.dims.extend(norm_img_data.shape)
input_tensor.data_type = 1
input_tensor.raw_data = norm_img_data.tobytes()

request_message = predict_pb2.PredictRequest()

# For your model, the inputs name should be something else customized b
request_message.inputs["input.1"].data_type = input_tensor.data_type
request_message.inputs["input.1"].dims.extend(input_tensor.dims)
request_message.inputs["input.1"].raw_data = input_tensor.raw_data

content_type_headers = ['application/x-protobuf', 'application/octet-st

for h in content_type_headers:
    request_headers = {
        'Content-Type': h,
        'Accept': 'application/x-protobuf'
    }
```

In [46]:
```
PORT_NUMBER = 9001 # Change appropriately if needed based on any change
inference_url = "http://127.0.0.1:" + str(PORT_NUMBER) + "/v1/models/de
response = requests.post(inference_url, headers=request_headers, data=r
```

In [46]:
```
PORT_NUMBER = 9001 # Change appropriately if needed based on any change
inference_url = "http://127.0.0.1:" + str(PORT_NUMBER) + "/v1/models/de
response = requests.post(inference_url, headers=request_headers, data=r
```


In [47]:
```
# Parse response message

response_message = predict_pb2.PredictResponse()
response_message.ParseFromString(response.content)

# For your model, the outputs names should be something else customized
boxes = np.frombuffer(response_message.outputs['boxes'].raw_data, dtype
#labels = np.frombuffer(response_message.outputs['labels'].raw_data, dt
scores = np.frombuffer(response_message.outputs['scores'].raw_data, dty

print('Boxes shape:', response_message.outputs['boxes'].dims)
#print('Labels shape:', response_message.outputs['labels'].dims)
print('Scores shape:', response_message.outputs['scores'].dims)
```

```
Boxes shape: [1, 3000, 4]
Scores shape: [1, 3000, 3]
```

```
response_message.ParseFromString(response.content)

# For your model, the outputs names should be something else customized
boxes = np.frombuffer(response_message.outputs['boxes'].raw_data, dtype
#labels = np.frombuffer(response_message.outputs['labels'].raw_data, dt
scores = np.frombuffer(response_message.outputs['scores'].raw_data, dty

print('Boxes shape:', response_message.outputs['boxes'].dims)
#print('Labels shape:', response_message.outputs['labels'].dims)
print('Scores shape:', response_message.outputs['scores'].dims)
```

```
Boxes shape: [1, 3000, 4]
Scores shape: [1, 3000, 3]
```

In [54]:

```
## Display image with bounding boxes and appropriate class

# Parse the list of class labels
classes = [line.rstrip('\n') for line in open('labels.txt')]

# Plot the bounding boxes on the image
plt.figure()
fig, ax = plt.subplots(1, figsize=(12,9))
ax.imshow(img)

resized_width = 300  # we resized the original image, remember ?
resized_height = 300
num_boxes = 6 # we limit displaying to just 10 boxes to avoid clogging
               # The results are already sorted based on box confidence

for c in range(num_boxes):
    base_index = c * 4
    y1, x1, y2, x2 = boxes[base_index] * resized_height, boxes[base_ind
    color = 'blue'
    box_h = (y2 - y1)
    box_w = (x2 - x1)
    bbox = patches.Rectangle((y1, x1), box_h, box_w, linewidth=2, edgec
    ax.add_patch(bbox)
    #plt.text(y1, x1, s=classes[labels[c] - 1], color='white', vertical
plt.axis('off')

# Save image
#plt.savefig("output/ssd_result.jpg", bbox_inches='tight', pad_inches=0
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```

6. Couldnt figure out what was wrong with my code for finding the

```
for c in range(num_boxes):
    base_index = c * 4
    y1, x1, y2, x2 = boxes[base_index] * resized_height, boxes[base_ind
    color = 'blue'
    box_h = (y2 - y1)
    box_w = (x2 - x1)
    bbox = patches.Rectangle((y1, x1), box_h, box_w, linewidth=2, edgec
    ax.add_patch(bbox)
    #plt.text(y1, x1, s=classes[labels[c] - 1], color='white', vertical
plt.axis('off')

# Save image
#plt.savefig("output/ssd_result.jpg", bbox_inches='tight', pad_inches=0
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>
```

Problem 2

1.

GCP - vertex ai, i am including the deep learning frameworks as it is included in the listed link (https://cloud.google.com/deep-learning-vm/docs/images)

- tensorflow enterprise 2.x, 1.x
- pytorch - latest version
- pytorch XLA - latest
- Chainer - latest
- MXNet - latest
- CNTK - latest
- Caffe - latest

Microsoft - azure datascience vm link (https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/dsvm-tools-deep-learning-frameworks) azure ml link (https://docs.microsoft.com/en-us/azure/machine-learning/overview-what-is-azure-machine-learning)

- pytorch - version 1.9.0
- tensorflow - version 2.5

IBM - cloud pak - link (https://www.ibm.com/docs/en/cloud-paks/cp-data/3.0.1?topic=models-supported-frameworks)

- XGBOOST - 0.8, 0.9
- Tensorflow - 1.15, 2.1
- Caffe - 1.0
- Keras 2.2.5
- Pytorch, 1.1, 1.2, 1.3

Amazon - deepleanring containers link (https://github.com/aws/deep-learning-containers/blob/master/available_images.md)

- tensorflow
- pytorch 1.10.0, 1.9.0, etc...
- tensorflow - 2.7.0. 2.6.2, 2.6.0
- chainer - 4.0.0, 4.1.0, 5.0.0
- mxnet - 1.8.0, 1.7.0, etc..

2.

Google Vertex AI

- NVIDIA_TESLA_A100
- NVIDIA_TESLA_K80
- NVIDIA_TESLA_P4
- NVIDIA_TESLA_P100
- NVIDIA_TESLA_T4
- NVIDIA_TESLA_V100

Amazon

- NVIDIA_TESLA_A100
- NVIDIA_TESLA_M60
- NVIDIA_T4
- NVIDIA_TESLA_V100
- NVIDIA_A10G

IBM link (https://www.ibm.com/downloads/cas/RDPDBJ3X)

- NVIDIA_TESLA_M60
- NVIDIA_TESLA_K80
- NVIDIA_TESLA_P100
- NVIDIA_TESLA_V100

Microsoft Azure link (https://azure.microsoft.com/en-us/pricing/details/machine-learning/)

- NVIDIA_TESLA_K80
- NVIDIA_TESLA_P100
- NVIDIA_TESLA_V100
- NVIDIA_TESLA_M60
- NVIDIA_TESLA_P40
- NVIDIA_TESLA_T4
- NVIDIA_TESLA_A100

3.

Google

- Ai platform notebook
- container registry
- kubeflow pipelines
- clould build
- ai platform training
- ai platform prediction
- dataflow
- cloud storage
- cloud sql
- bigquery

Microsoft Azure ML

- Azure ML Pipelines
- Azure Container Instance
- Azure Kubernetes Service
- Azure ML datasets
- Interpretability
- Azure ML model registry
- Azure ML run
- Azure ML Supports alerts
- Azure ML provides monitoring
- Azure Data Factory

Amazon Sagemaker streamlines MLOps

- Pipelines supported
- Model Registry
- Projects - enables CI/CD across end to end lifecycle
- Model Monitor

IBM Cloud Pak

- WS watson studio
- Watson Machine Learning
- Watson Open Scale
- Watson Knowledge Catalog
- Data Virtualization
- CI/CD

4.

Amazon Sagemaker

- amazon sagemaker uses amazon cloudwatch which records application logs for 15 months

Google Cloud Vertex

- Cloud Logging service is used for application logging and resource logging

Microsoft Azure ML

- provides Azure Monitor Logs
- MLFlow can be used for logging

IBM

- platform logs which are used for logging app logs and recsource logs

5.

Google

- AI Platform Training Jobs provides method to monitor training jobs in progress for metrics like accuracy and throughput

Amazon Sagemaker

- Amazon Cloud Watch can be utilized in order to monitor training jobs
- Sagemaker Console can also be used for monitoring trianing jobs

Microsoft Azure ML

- Azure ML provides abilities to monitor runs link (https://docs.microsoft.com/en-us/azure/machine-learning/how-to-track-monitor-analyze-runs?tabs=python#monitor-run-performance)

IBM Cloud Pak (https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/ml_dlaas_working_with_new_models.html)

- Cloud Pak allows for monitoring of a training run by creating a training manifest file

6. Google AI

- autoscaling can be done through google compute engine clusters
- autoscaling can also be done for inference predicition as well through the google cloud ai platform

Amazon Sagemaker link (https://docs.aws.amazon.com/sagemaker/latest/dg/endpoint-auto-scaling.html)

- supports autoscaling in response to changes in workload, a scaling policy can be defined according to different mertrics

Microsoft Azure ML link (https://docs.microsoft.com/en-us/azure/machine-learning/how-to-autoscale-endpoints?tabs=azure-cli)

- Autoscale is supported which allows for autoscaling of a model which allows for setting vatrious rules for autoscaling

IBM link (https://www.ibm.com/docs/en/cloud-paks/cp-data/4.0?topic=cluster-scaling-services)

- using the scaleConfig setting in IBM Cloud Pak

7.

In [ ]:
```
#Google yaml file

trainingInput:
  jobId: 'dl-test'
  scaleTier: CUSTOM
  masterType: complex_model_m
  workerType: complex_model_m
  parameterServerType: large_model
  workerCount: 9
  parameterServerCount: 3
  runtimeVersion: '2.6'
  packageUris: 'package dir'
  pythonVersion: '3.7'
  region:
  scheduling:
    maxWaitTime: 3600s
    maxRunningTime: 7200s
```

In [ ]:
```
#Sagemaker yaml file
apiVersion: sagemaker.aws.amazon.com/v1
kind: TrainingJob
metadata:
  name: 'dl-test'
spec:
  hyperParameters:
    - name: max_depth
      value: "5"
  algorithmSpecification:
    trainingImage: 'some image'
    trainingInputMode: File
  roleArn: arn:aws:iam::123456789012:role/service-role/AmazonSageMake
r-ExecutionRole
  region: us-west-2
  outputDataConfig:
    s3OutputPath: ""
  resourceConfig:
    instanceCount: 1
    instanceType: ml.m4.xlarge
    volumeSizeInGB: 5
  stoppingCondition:
    maxRuntimeInSeconds: 7200
  inputDataConfig:
    - channelName: train
      dataSource:
        s3DataSource:
          s3DataType: S3Prefix
          s3Uri: ''
          s3DataDistributionType: FullyReplicated
      contentType: text/csv
      compressionType: None
    - channelName: validation
      dataSource:
        s3DataSource:
          s3DataType: S3Prefix
          s3Uri: ""
          s3DataDistributionType: FullyReplicated
      contentType: text/csv
      compressionType: None
  tags:
    - key: tagKey
      value: tagValue
```

In [ ]:
```
#Azure  ML
code:
  local_path: src
command: >-
  python main.py
  --dataset ${{inputs.iris_csv}}
  --C ${{inputs.C}}
  --kernel ${{inputs.kernel}}
  --coef0 ${{inputs.coef0}}
inputs:
  "some datsaset"
  C: 0.8
  kernel: "rbf"
  coef0: 0.1
environment: azureml:AzureML-pytorch-0.24-ubuntu18.04-py37-cpu:9
compute: azureml:cpu-cluster
display_name: dl-test
experiment_name: dltest
description: Train a deep learning model
```

https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/master/samples/xgboost-mnist-trainingjob.yaml (https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/master/samples/xgboost-mnist-trainingjob.yaml)

In [ ]:
```
# IBM Dataplatform
model_definition:
  framework:
    name: pytorch
    version: "1.1"
    runtimes:
      name: python
      version: "3.7"
  name: tf-mnist
  description: Simple MNIST model implemented in pytorch
  execution:
    command: python3 convolutional_network.py --trainImagesFile ${DAT
A_DIR}/train-images-idx3-ubyte.gz
      --trainLabelsFile ${DATA_DIR}/train-labels-idx1-ubyte.gz --test
ImagesFile ${DATA_DIR}/t10k-images-idx3-ubyte.gz
      --testLabelsFile ${DATA_DIR}/t10k-labels-idx1-ubyte.gz --learni
ngRate 0.001
      --trainingIters 2000000
    compute_configuration:
      name: v100
training_data_reference:
  name: MNIST image data files
  connection:
    endpoint_url: <auth-url>
    access_key_id: <username>
    secret_access_key: <password>
  source:
    bucket: mnist-training-data
  type: s3
training_results_reference:
  name: DL Model Storage
  connection:
    endpoint_url: <auth-url>
    access_key_id: <username>
    secret_access_key: <password>
  target:
    bucket: mnist-training-models
  type: s3
```

There are common fields specifying python version, region, dataset used, hyperparameters, compute instance specifications, and some of this is obfuscated within an app image in GCP.

Problem 3

Section 1

a

b

Deployment name *
minikf-jd3720

Zone
northamerica-northeast1-a

Machine type

Machine family

GENERAL-PURPOSE COMPUTE-OPTIMIZED

Machine types for common workloads, optimized for cost and flexibility

Series
N1

Powered by Intel Skylake CPU platform or one of its predecessors

Machine type
n1-standard-8 (8 vCPU, 30 GB memory)

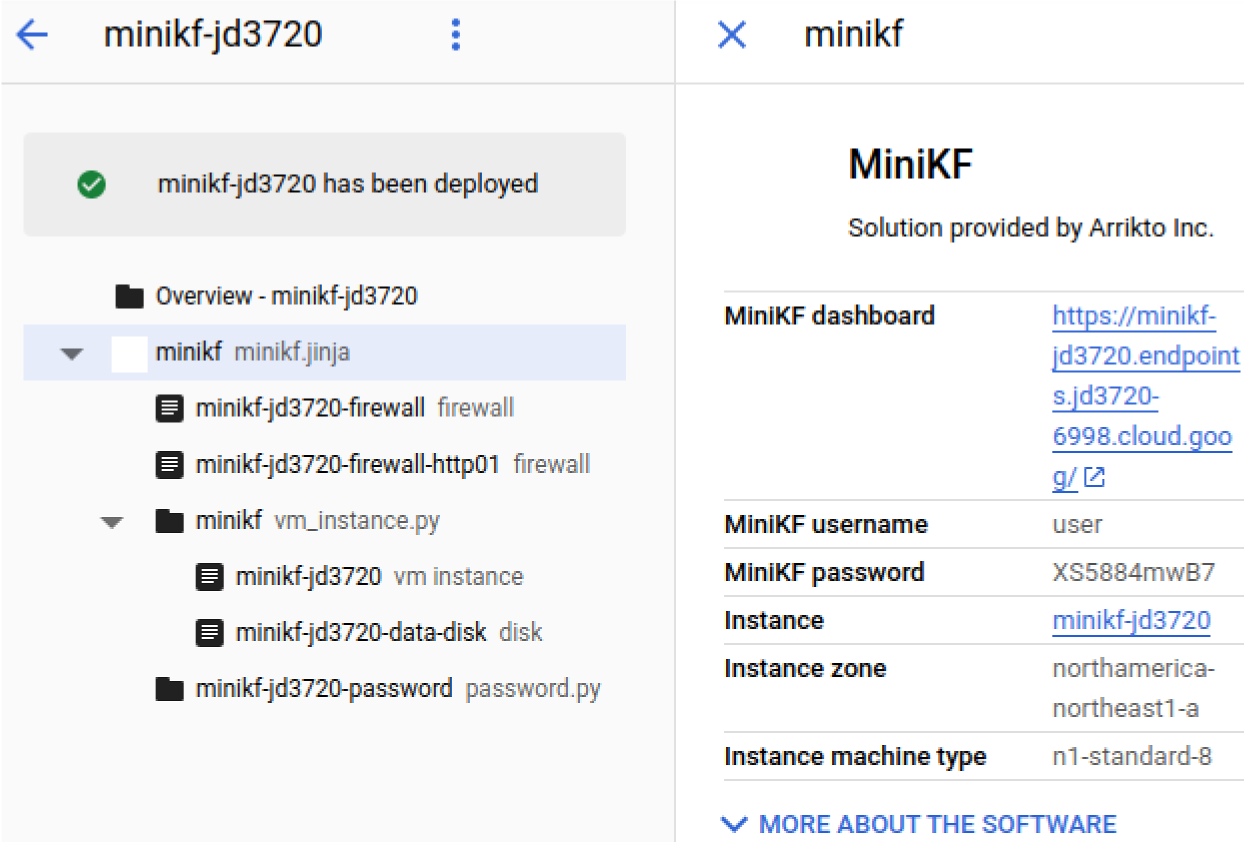
vCPU	Memory
8	30 GB

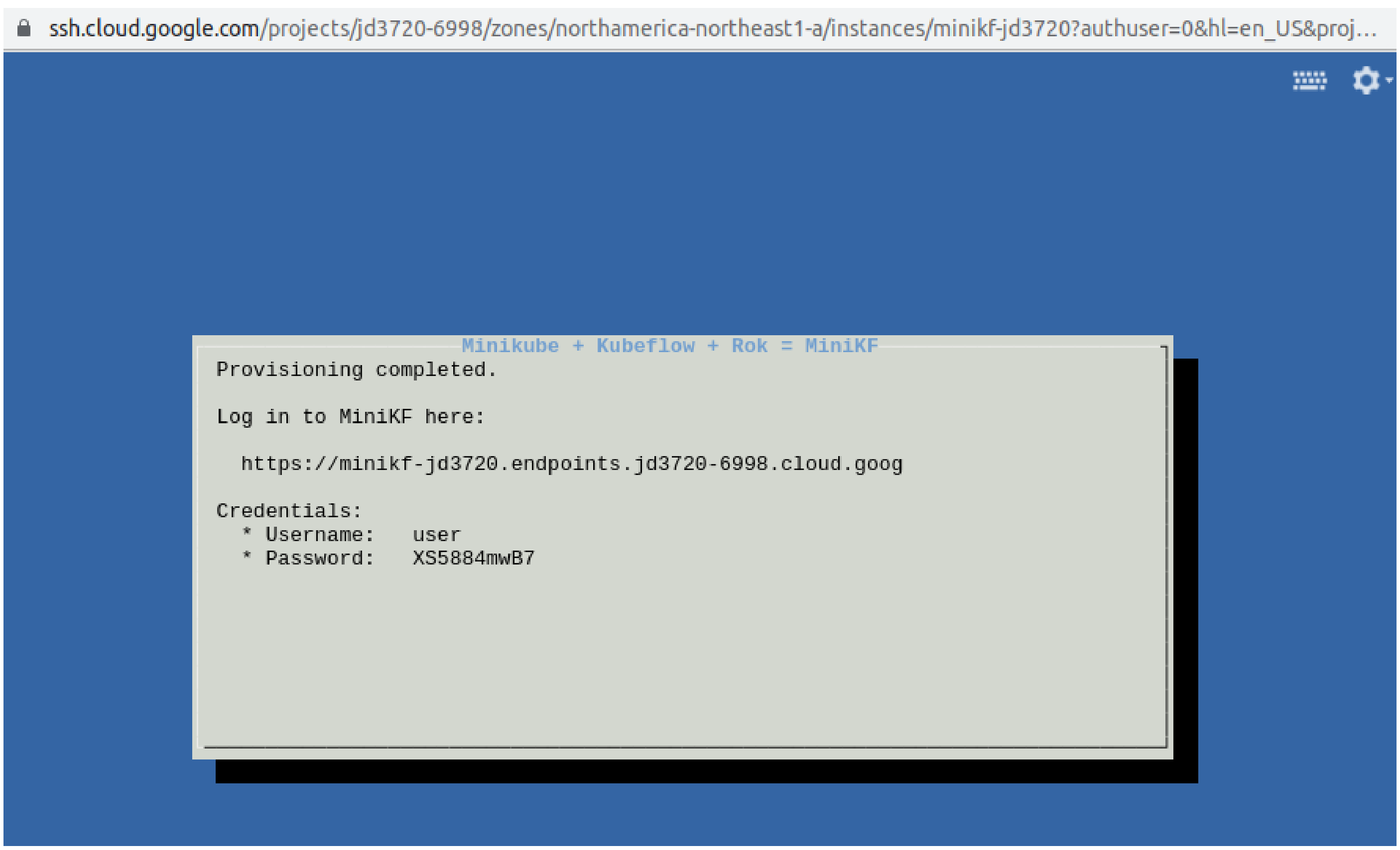
CPU PLATFORM AND GPU

Boot Disk

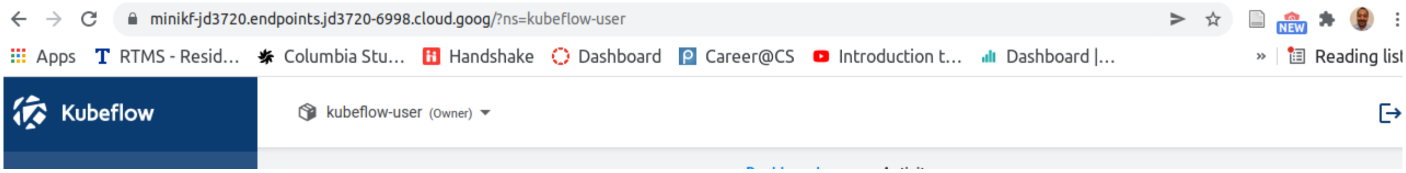
Boot disk type *
Standard Persistent Disk

Boot disk size in GB *
200


minikf-jd3720
minikf
minikf-jd3720 has been deployed
Overview - minikf-jd3720
minikf minikf.jinja
minikf-jd3720-firewall firewall
minikf-jd3720-firewall-http01 firewall
minikf vm_instance.py
minikf-jd3720 vm instance
minikf-jd3720-data-disk disk
minikf-jd3720-password password.py
MiniKF
Solution provided by Arrikto Inc.
MiniKF dashboard https://minikf-jd3720.endpoints.jd3720-6998.cloud.goog/
MiniKF username user
MiniKF password XS5884mwB7
Instance minikf-jd3720
Instance zone northamerica-northeast1-a
Instance machine type n1-standard-8
MORE ABOUT THE SOFTWARE

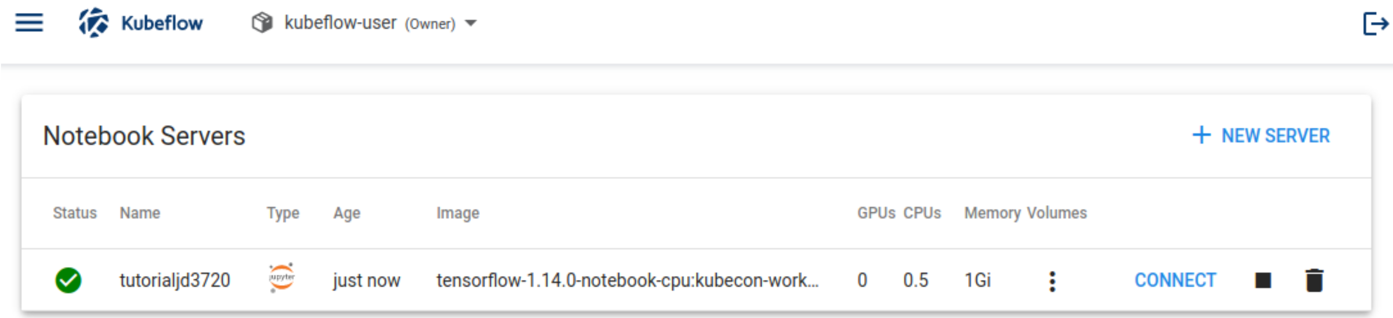

ssh.cloud.google.com/projects/jd3720-6998/zones/northamerica-northeast1-a/instances/minikf-jd3720?authuser=0&hl=en_US&proj...
Minikube + Kubeflow + Rok = MiniKF
Provisioning completed.
Log in to MiniKF here:
https://minikf-jd3720.endpoints.jd3720-6998.cloud.goog
Credentials:
* Username: user
* Password: XS5884mwB7

c


minikf-jd3720.endpoints.jd3720-6998.cloud.goog/?ns=kubeflow-user
Apps
RTMS - Resid...
Columbia Stu...
Handshake
Dashboard
Career@CS
Introduction t...
Dashboard |...
Reading list
Kubeflow
kubeflow-user (Owner)

Section 2

a



b

```
groups: cannot find name for group ID 1337
jovyan@tutorialjd3720-0:~$ cd data
jovyan@tutorialjd3720-0:~/data$ git clone -b kubecon-workshop https://github.com/kubeflow-kale/examples
Cloning into 'examples'...
remote: Enumerating objects: 178, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 178 (delta 5), reused 4 (delta 4), pack-reused 169
Receiving objects: 100% (178/178), 927.77 KiB | 10.91 MiB/s, done.
Resolving deltas: 100% (82/82), done.
jovyan@tutorialjd3720-0:~/data$
```

c

Kale Deployment Panel

Enable

Pipeline Metadata

Select experiment
+ New Experiment

Experiment Name
jd3720

Pipeline Name
cifar10-classification

Pipeline Description
Sequential PyTorch pipeline to train a

Run

HP Tuning with Katib

SET UP KATIB JOB

Advanced Settings

COMPILE AND RUN

Terminal 1 | cifar10_classification.ipynb

Code

step: testwhole *depends on:*

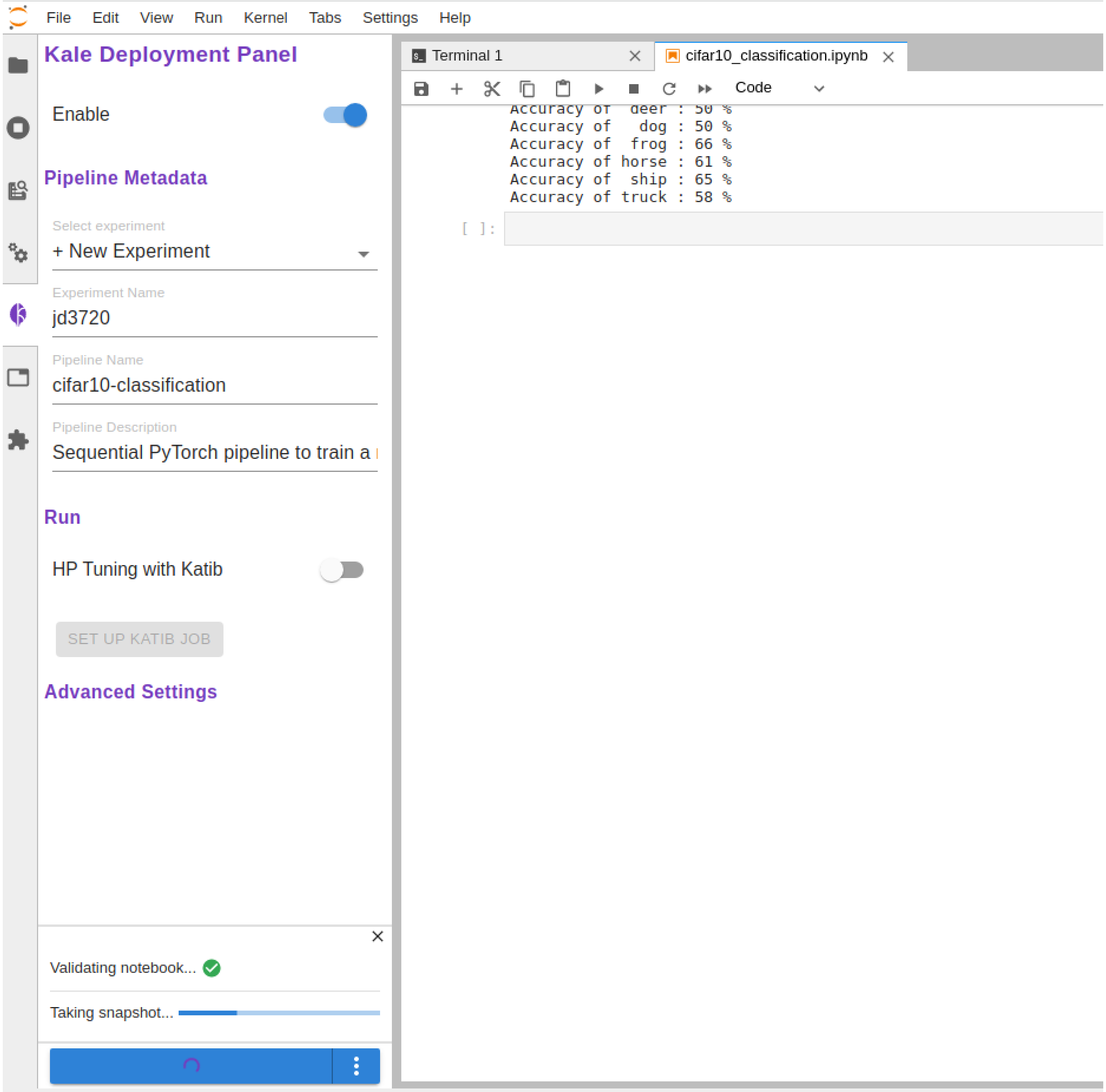
[19]:
```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data,
        total += labels.size(0)
        correct += (predicted == labels).sum()

print('Accuracy of the network on the 10000 te
    100 * correct / total))
```

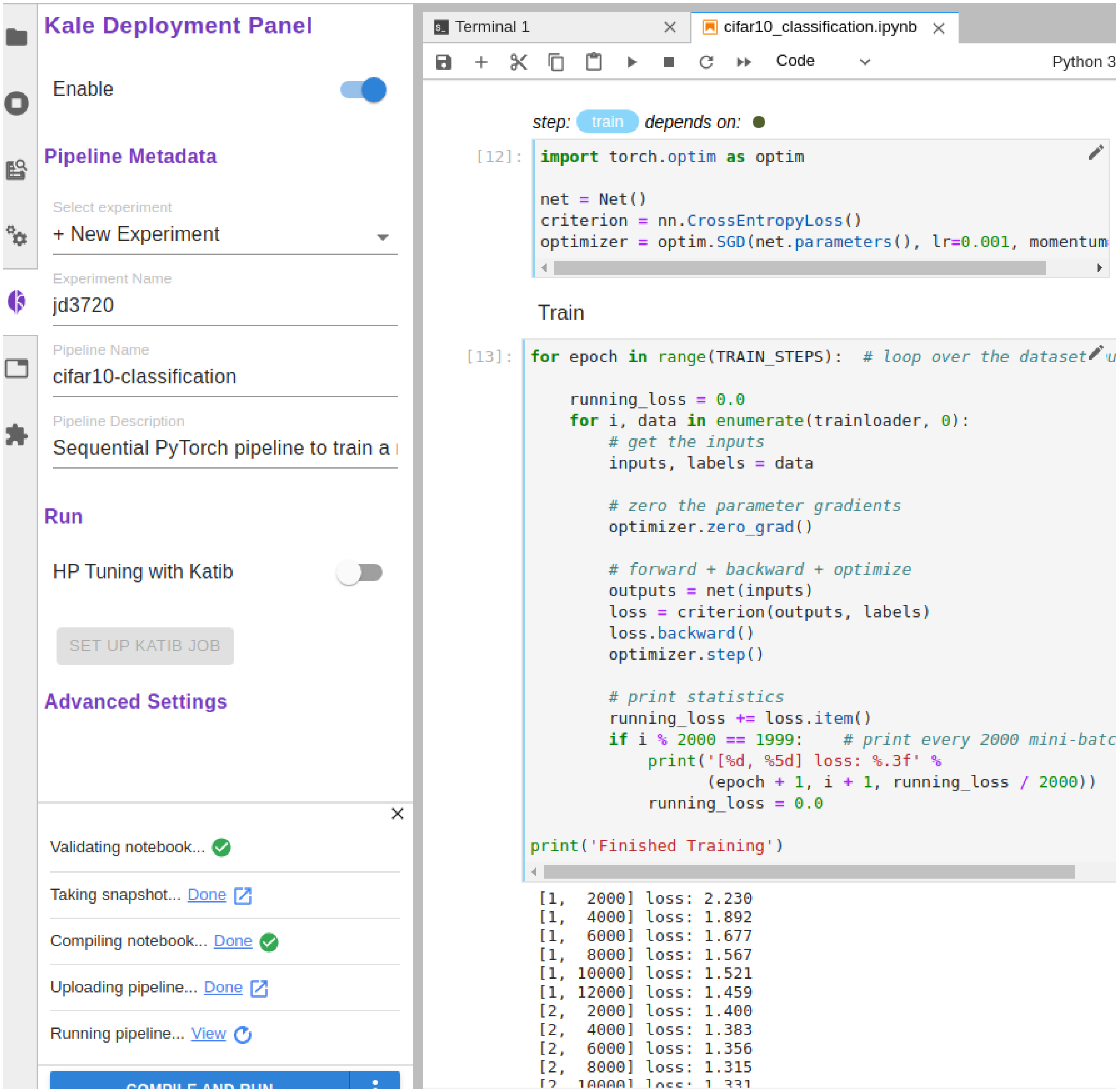
```
Accuracy of the network on the 10000 test imag
```

[20]:
```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item(
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / c
```

```
Accuracy of plane : 49 %
Accuracy of   car : 55 %
Accuracy of  bird : 34 %
Accuracy of   cat : 28 %
Accuracy of  deer : 35 %
Accuracy of   dog : 33 %
Accuracy of  frog : 84 %
Accuracy of horse : 57 %
Accuracy of  ship : 79 %
Accuracy of truck : 66 %
```

[ ]:


File Edit View Run Kernel Tabs Settings Help
Kale Deployment Panel
Enable
Pipeline Metadata
Select experiment
+ New Experiment
Experiment Name
jd3720
Pipeline Name
cifar10-classification
Pipeline Description
Sequential PyTorch pipeline to train a
Run
HP Tuning with Katib
SET UP KATIB JOB
Advanced Settings
Validating notebook...
Taking snapshot...
Terminal 1
cifar10_classification.ipynb
Code
Accuracy of dog : 50 %
Accuracy of frog : 66 %
Accuracy of horse : 61 %
Accuracy of ship : 65 %
Accuracy of truck : 58 %
[ ]:


Kale Deployment Panel
Enable
Pipeline Metadata
Select experiment
+ New Experiment
Experiment Name
jd3720
Pipeline Name
cifar10-classification
Pipeline Description
Sequential PyTorch pipeline to train a
Run
HP Tuning with Katib
SET UP KATIB JOB
Advanced Settings
Validating notebook...
Taking snapshot... Done
Compiling notebook... Done
Uploading pipeline... Done
Running pipeline... View
COMPILE AND RUN
Terminal 1
cifar10_classification.ipynb
Code
Python 3
step: train depends on:
[12]:
import torch.optim as optim
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum
Train
[13]:
for epoch in range(TRAIN_STEPS): # loop over the dataset
running_loss = 0.0
for i, data in enumerate(trainloader, 0):
# get the inputs
inputs, labels = data
# zero the parameter gradients
optimizer.zero_grad()
# forward + backward + optimize
outputs = net(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
# print statistics
running_loss += loss.item()
if i % 2000 == 1999: # print every 2000 mini-batc
print('[%d, %5d] loss: %.3f' %
(epoch + 1, i + 1, running_loss / 2000))
running_loss = 0.0
print('Finished Training')
[1, 2000] loss: 2.230
[1, 4000] loss: 1.892
[1, 6000] loss: 1.677
[1, 8000] loss: 1.567
[1, 10000] loss: 1.521
[1, 12000] loss: 1.459
[2, 2000] loss: 1.400
[2, 4000] loss: 1.383
[2, 6000] loss: 1.356
[2, 8000] loss: 1.315

Kale Deployment Panel

Enable

Pipeline Metadata

Select experiment
+ New Experiment

Experiment Name
jd3720

Pipeline Name
cifar10-classification

Pipeline Description
Sequential PyTorch pipeline to train a

Run

HP Tuning with Katib

SET UP KATIB JOB

Advanced Settings

Validating notebook...

Taking snapshot... Done

Compiling notebook... Done

Terminal 1 | cifar10_classification.ipynb

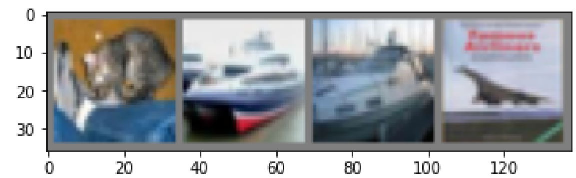
Code | Python 3

```
[1,  4000] loss: 1.892
[1,  6000] loss: 1.677
[1,  8000] loss: 1.567
[1, 10000] loss: 1.521
[1, 12000] loss: 1.459
[2,  2000] loss: 1.400
[2,  4000] loss: 1.383
[2,  6000] loss: 1.356
[2,  8000] loss: 1.315
[2, 10000] loss: 1.331
[2, 12000] loss: 1.293
Finished Training
```

Test on test data

step: testontest *depends on:*

[14]:
```
dataiter = iter(testloader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] f
```




```
GroundTruth:    cat  ship  ship plane
```

[15]:
```
outputs = net(images)
```

[*]:
```
_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```

```
Predicted:    cat  ship  ship plane
```

Kale Deployment Panel

Enable

Pipeline Metadata

Select experiment
+ New Experiment

Experiment Name
jd3720

Pipeline Name
cifar10-classification

Pipeline Description
Sequential PyTorch pipeline to train a

Run

HP Tuning with Katib

SET UP KATIB JOB

Advanced Settings

Validating notebook...

Taking snapshot... Done

Compiling notebook... Done

Uploading pipeline... Done

Running pipeline... View

COMPILE AND RUN

Terminal 1 | cifar10_classification.ipynb

Code | Python 3

```
Predicted:    cat  ship  ship plane
```

Performance on whole dataset

step: testwhole *depends on:*

[*]:
```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 54 %
```

[*]:
```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i
```

```
Accuracy of plane : 66 %
Accuracy of   car : 62 %
Accuracy of  bird : 42 %
Accuracy of   cat : 23 %
Accuracy of  deer : 50 %
Accuracy of   dog : 50 %
Accuracy of  frog : 66 %
Accuracy of horse : 61 %
Accuracy of  ship : 65 %
```

Kale Deployment Panel

Enable

Pipeline Metadata

Select experiment
+ New Experiment

Experiment Name
jd3720

Pipeline Name
cifar10-classification

Pipeline Description
Sequential PyTorch pipeline to train a

Run

HP Tuning with Katib

SET UP KATIB JOB

Advanced Settings

Validating notebook...

Taking snapshot... Done

Compiling notebook... Done

Terminal 1 | cifar10_classification.ipynb

Markdown | Python

visual 3/3

```
Downloading
on.tar.gz t
100.0%
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

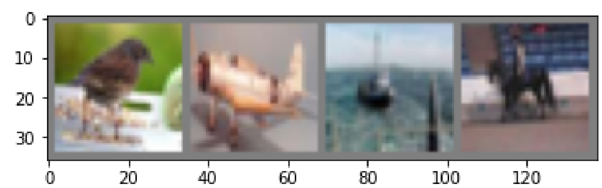
```
#### Visualize dataset
```

functions

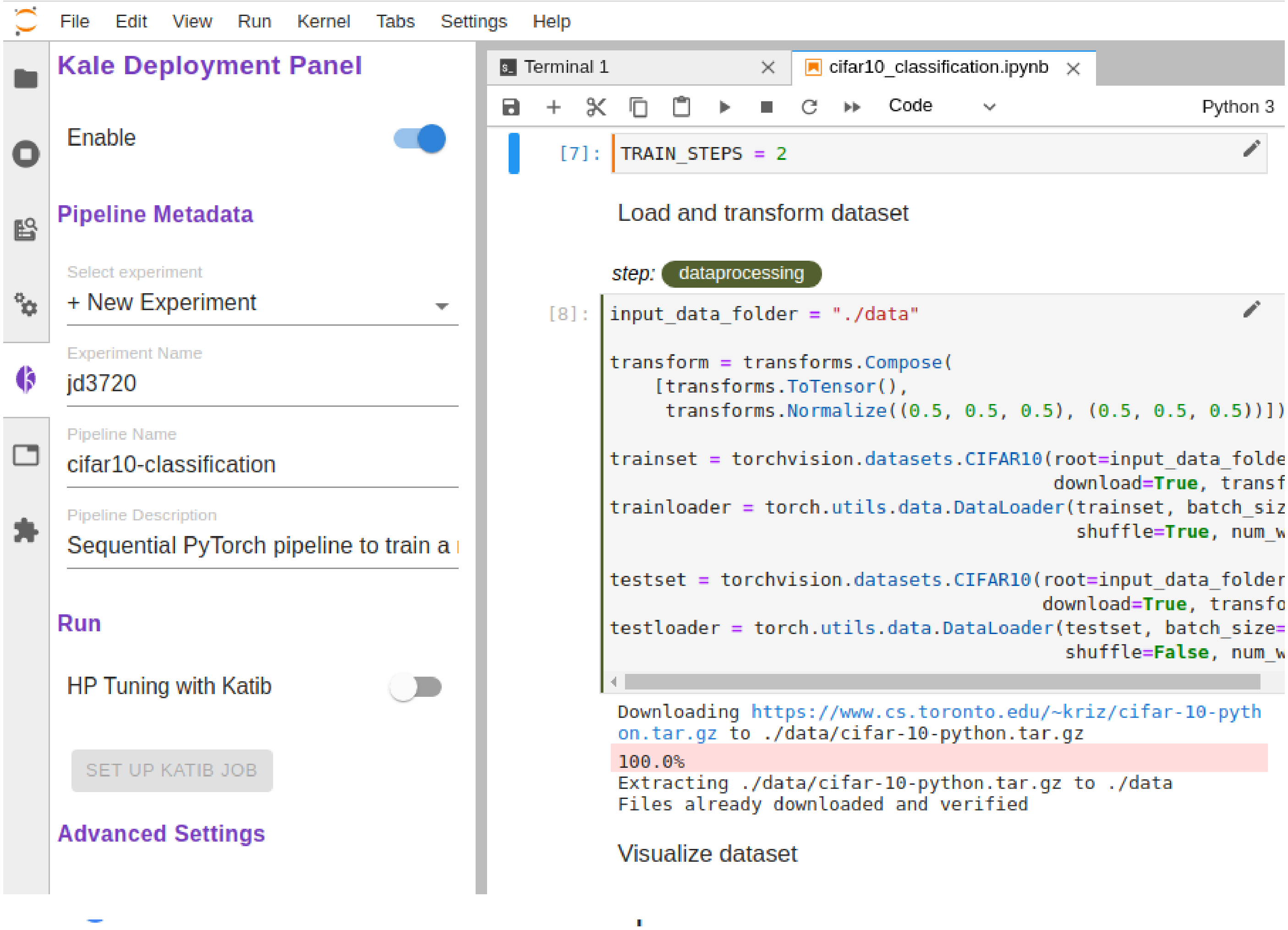
[9]:
```python
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

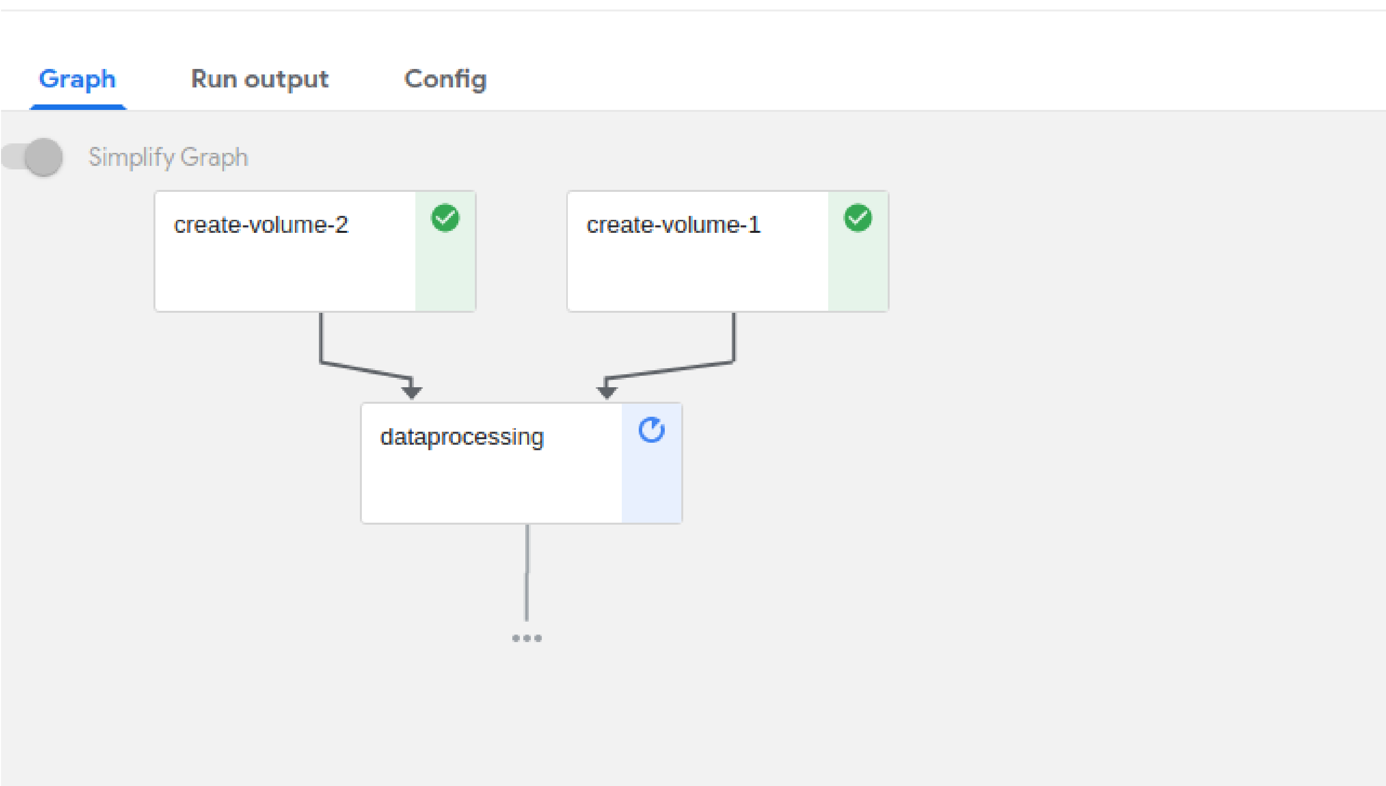
skip

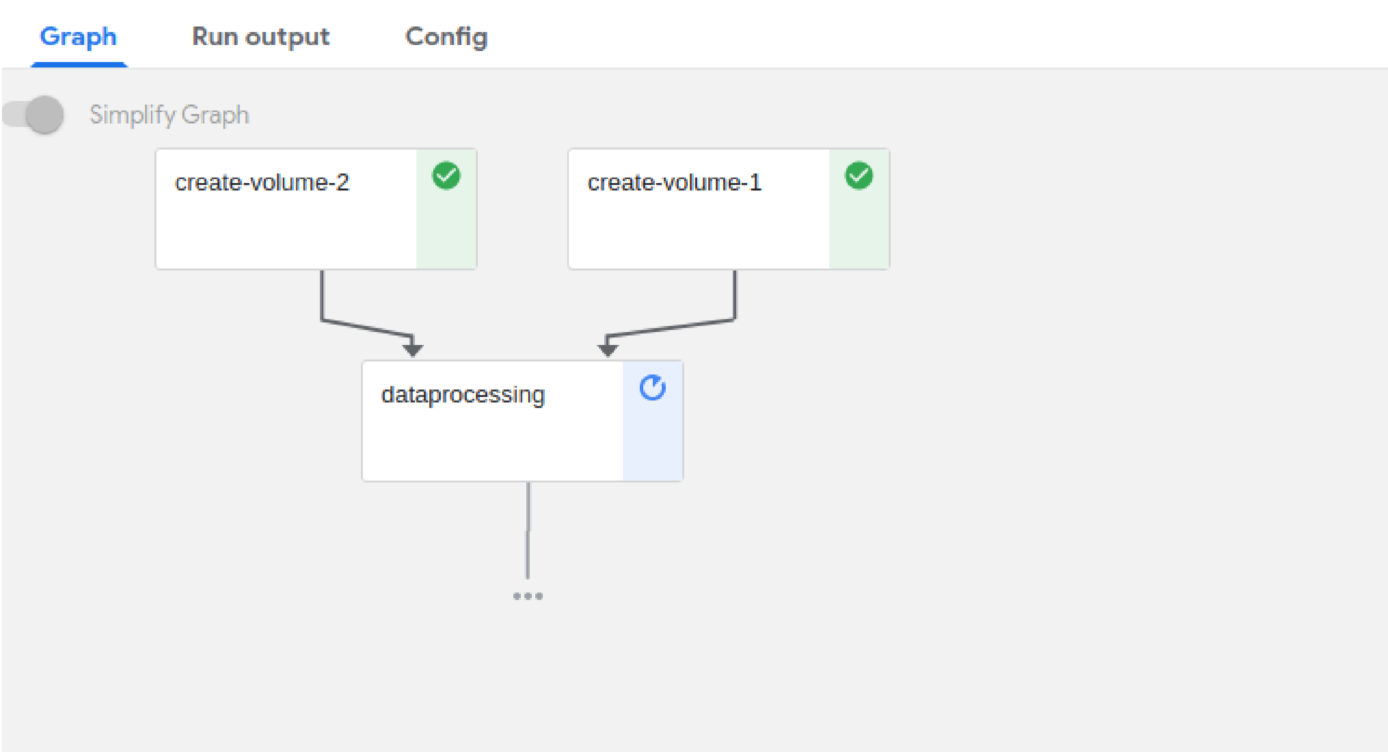
[10]:
```python
# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(
```



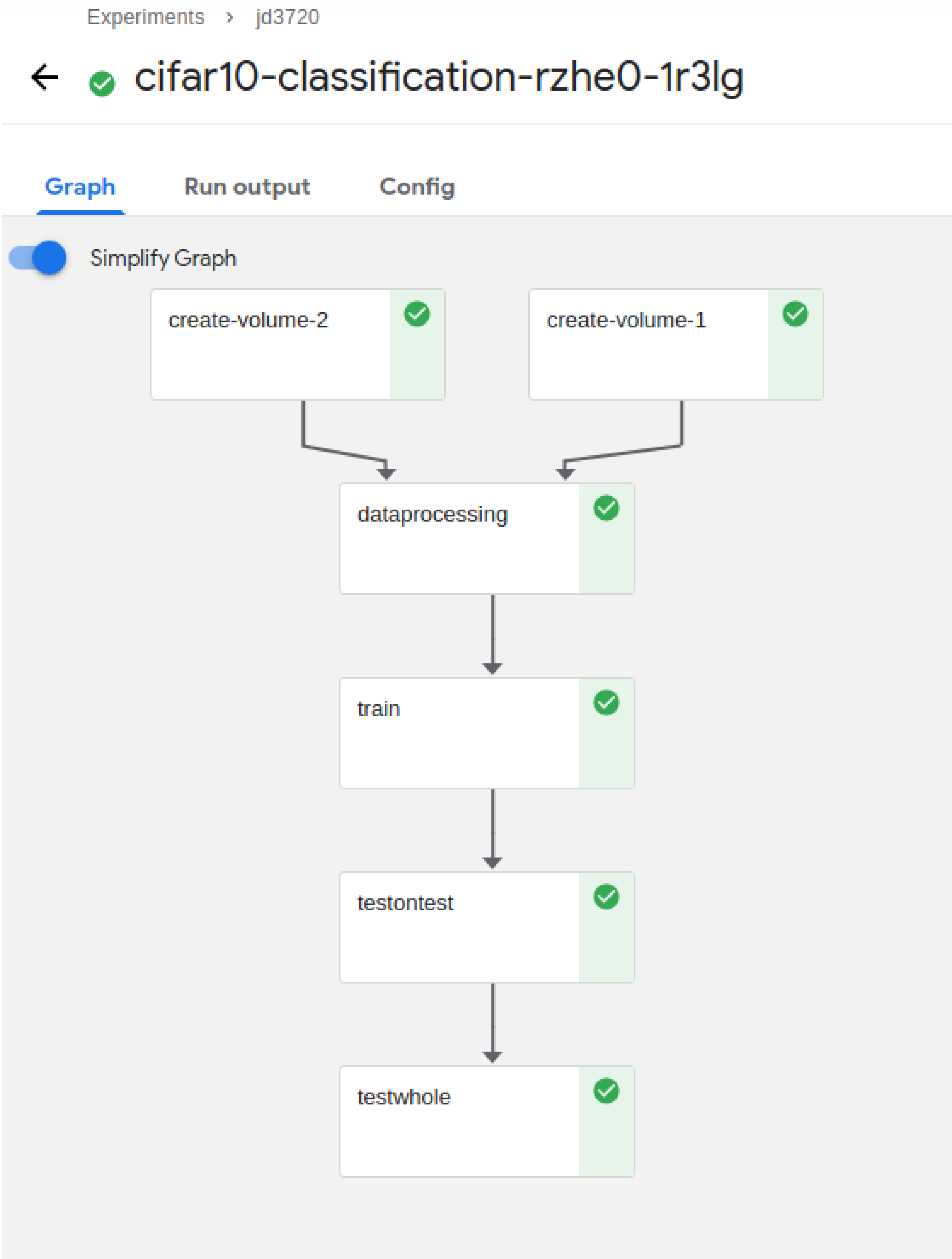

```
bird plane  ship horse
```


File Edit View Run Kernel Tabs Settings Help
Kale Deployment Panel
Enable
Pipeline Metadata
Select experiment
+ New Experiment
Experiment Name
jd3720
Pipeline Name
cifar10-classification
Pipeline Description
Sequential PyTorch pipeline to train a
Run
HP Tuning with Katib
SET UP KATIB JOB
Advanced Settings
Terminal 1
cifar10_classification.ipynb
Code
Python 3
[7]: TRAIN_STEPS = 2
Load and transform dataset
step: dataprocessing
[8]: input_data_folder = "./data"
transform = transforms.Compose(
[transforms.ToTensor(),
transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
trainset = torchvision.datasets.CIFAR10(root=input_data_folde
download=True, transf
trainloader = torch.utils.data.DataLoader(trainset, batch_siz
shuffle=True, num_w
testset = torchvision.datasets.CIFAR10(root=input_data_folder
download=True, transfo
testloader = torch.utils.data.DataLoader(testset, batch_size=
shuffle=False, num_w
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-pyth
on.tar.gz to ./data/cifar-10-python.tar.gz
100.0%
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
Visualize dataset


Graph
Run output
Config
Simplify Graph
create-volume-2
create-volume-1
dataprocessing


Graph
Run output
Config
Simplify Graph
create-volume-2
create-volume-1
dataprocessing

d


Experiments › jd3720
cifar10-classification-rzhe0-1r3lg
Graph
Run output
Config
Simplify Graph
create-volume-2
create-volume-1
dataprocessing
train
testontest
testwhole

testwhole

Rok autosnapshot

Rok has successfully created a snapshot for step `testwhole`.
To **explore the execution state** at the **beginning** of this step follow the instructions below:
1. View the snapshot in the Rok UI.
2. Copy the Rok URL.
3. Create a new Notebook Server by using this Rok URL to autofill the form.

---

testwhole

```
2021-12-03 20:22:00 Kale marshalling          [INF

2021-12-03 20:22:05 Kale marshalling          [INF

Accuracy of the network on the 10000 test images:

Accuracy of plane : 78 %
Accuracy of   car : 63 %
Accuracy of  bird : 59 %
Accuracy of   cat : 26 %
Accuracy of  deer : 48 %
Accuracy of   dog : 37 %
Accuracy of  frog : 63 %
Accuracy of horse : 43 %
Accuracy of  ship : 56 %
Accuracy of truck : 36 %
```

dataprocessing

Rok autosnapshot

Rok has successfully created a snapshot for step `dataprocessing`.
To **explore the execution state** at the **beginning** of this step follow the instructions below:
1. View the snapshot in the Rok UI.
2. Copy the Rok URL.
3. Create a new Notebook Server by using this Rok URL to autofill the form.

dataprocessing

```
Files already downloaded and verified
Files already downloaded and verified
2021-12-03 20:16:22 Kale marshalling          [INF
2021-12-03 20:16:22 Kale marshalling          [INF
```

Rok autosnapshot

Rok has successfully created a snapshot for step `train`.
To **explore the execution state** at the **beginning** of this step follow the instructions below:
1. View the snapshot in the Rok UI.
2. Copy the Rok URL.
3. Create a new Notebook Server by using this Rok URL to autofill the form.

---

train

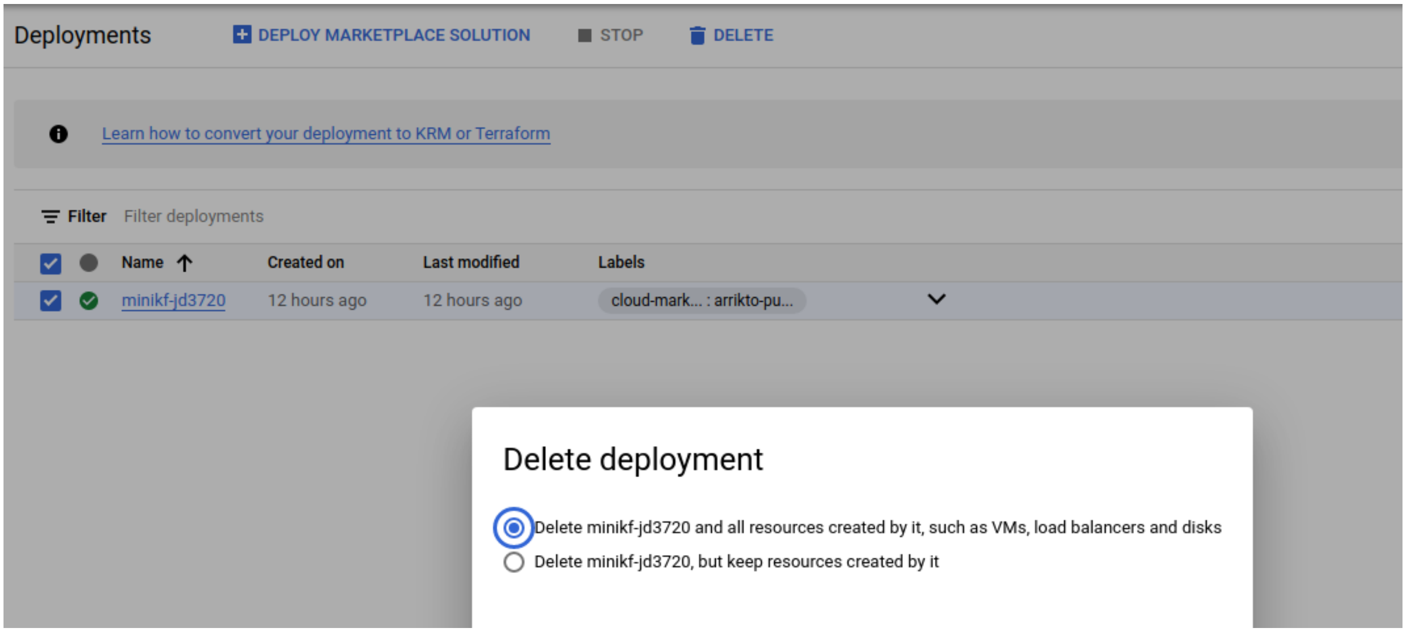
```
[1,  6000] loss: 1.633
[1,  8000] loss: 1.530
[1, 10000] loss: 1.487
[1, 12000] loss: 1.463
[2,  2000] loss: 1.379
[2,  4000] loss: 1.360
[2,  6000] loss: 1.351
[2,  8000] loss: 1.331
[2, 10000] loss: 1.304
[2, 12000] loss: 1.264
Finished Training
2021-12-03 20:20:11 Kale marshalling
```

testontest

```
2021-12-03 20:21:01 Kale marshalling         [INF
2021-12-03 20:21:08 Kale marshalling         [INF
```


```
GroundTruth:    cat  ship  ship plane
Predicted:   bird  ship  ship plane
2021-12-03 20:21:10 Kale marshalling         [INF
2021-12-03 20:21:10 Kale marshalling         [INF
```

e


Deployments
DEPLOY MARKETPLACE SOLUTION
STOP
DELETE
Learn how to convert your deployment to KRM or Terraform
Filter
Filter deployments
Name
Created on
Last modified
Labels
minikf-jd3720
12 hours ago
12 hours ago
cloud-mark... : arrikto-pu...
Delete deployment
Delete minikf-jd3720 and all resources created by it, such as VMs, load balancers and disks
Delete minikf-jd3720, but keep resources created by it

Problem 4

1.

Episodic tasks are those that have an end state or terminal state. An episodic task can be a game like checkers which has a clear end for an agent. Continuous task is when there is no end state and the task doesn't end. An example of a continuous task is a surveillance task where the agent must constantly monitor and identify intruders or changes to the environment.

2.

Exploration in RL means taking a action that has not been taken before while exploitation is taking an action which leads to the greatest rewards(best action) given the learners knowledge at this point in time.

Epsilon greedy aids in keeping the network from getting stuck in sub-optimal policies while at the same time allowing the network to build off of the best policy found at this point in time. As such epsilon greedy aids in exploring the state space.

Epsilomn should follow a schedule during deep RL training as the network doesnt have any knowledge of the best course of action and exploitation can't be done. As epsilon should be high in the beginning and become smaller. In the beginning exploring is almost always done and then once learnings have taken place exploitation can occur given the network has now learned. As such a small epsilon is needed for enabling some exploration.

3.

Initialize replay memory D to capacity N

- init replay memory

Initialize action-value function Q with random weights

- init Q function

for episode = 1, M do:

- loop over episodes

Initialise sequence $s_1 = x_1$ and preprocessed sequenced $\varphi_1 = \varphi(s_1)$

- init start state and history

for t = 1, T do:

- start until terminal state is reached

With probability epsilon select a random action a_t otherwise select $a_t = maxaQ * (\varphi(s_t), a; \theta)$

- use a epsilon greedy policy where the exploit/exploarition takes place

Execute action a_t in emulator and observe reward r*t and image x*(t+1):

- action and take into account reward and the next state

Set $s_(t+1) = s_t, a_t, x_(t+1)$ and preprocess $\varphi(t+1) = \varphi(s(t+1))$

- update state

Store transition $(\varphi_t, a_t, r_t, \varphi_t + 1)$ in D

- store the new experience

Sample random minibatch of transitions $(\varphi j, a_j, r_j, \varphi(j+1))$ from D

- sample a minibatch of experiences from the replay memory

Set y_j = {r_j | r_j + λ max Q}:

- set the target value equal to the reward of the state s_j for terminal states or set the target value equal to the reward + future discounted rewards for non terminal states

perform a gradient descent step

- finally update the Q network

4

The target Q networks keeps the target distribution stable so that the learner is not always trying to fit to a moving distribution.

5

Deep learning algorithms expect data to be independent but in online Reinforcement Learning it is that the data is interdependent because future steps are depedent on previous steps. As such experience replay stores previous steps and the learner is able to randomly sample from this distribution which solves the problem of correlated data and non-stationary distributions

6

Insteaad of drawing experiences at random from memory, experience which carry the most information and provide the learned with the best signal are chosen.

7

Similar

- various actors
- shared replay memory
- each respective actor have their own instance of the environment

Differences

- ape-x uses prioritized experience replay, Gorilla doesn't
- gorilla uses parmameter server and ape-x doesnt
- gorilla has multiple learners on multiple GPUS while apex has a single learned on a GPU