

Alibaba Cloud

Machine Learning Platform for
AI
DSW notebook modeling

Document Version: 20210419

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).
5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.
6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
 Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 Danger: Resetting will result in the loss of user configuration data.
 Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
 Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	 Notice: If the weight is set to 0, the server no longer receives new requests.
 Note	A note indicates supplemental instructions, best practices, tips, and other content.	 Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click Settings> Network> Set network type .
Bold	Bold formatting is used for buttons, menus, page names, and other UI elements.	Click OK .
<code>Courier font</code>	Courier font is used for commands	Run the <code>cd /d C:/window</code> command to enter the Windows system folder.
<i>Italic</i>	Italic formatting is used for parameters and variables.	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] or [a b]	This format is used for an optional value, where only one item can be selected.	<code>ipconfig [-all -t]</code>
{ } or {a b}	This format is used for a required value, where only one item can be selected.	<code>switch {active stand}</code>

Table of Contents

1.Overview	05
2.Authorization	09
3.Work with DSW	12
3.1. Create an instance	12
3.2. Manage instances	13
3.3. Work with the development environments of DSW	14
3.4. Read data from and write data to OSS	19
3.5. Read data from and write data to MaxCompute tables	24
3.6. Export notebooks	25
4.Best practices	27
4.1. Use WebIDE to debug code online	27
4.2. Use CNN to classify images	30
4.3. Use RNN to recognize surnames	36
4.4. Use EasyVision to detect targets	45

1. Overview

Data Science Workshop (DSW) of Machine Learning Platform for AI (PAI) is an integrated development environment (IDE) in the cloud. DSW provides interactive development environments for developers of different levels. This topic describes the features, instance types, and supported zones of DSW editions. The editions are Individual Edition, GPU On-sale Edition, and Explorer Edition.

DSW integrates open source JupyterLab and provides plug-ins for customized development. You can directly launch Notebook to write, debug, and run Python code without O&M configurations. DSW also provides a variety of computing resources and supports heterogeneous data sources. DSW offers all-in-one machine learning services. You can use EASCMDB to deploy models trained in DSW as RESTful APIs to provide online services.

Features

- Supports real-time resource monitoring. Monitors CPU or GPU utilization during algorithm development.
- Supports a variety of data sources, such as MaxCompute, Object Storage Service (OSS), and Apsara File Storage NAS.
- Allows you to write and execute SQL statements.
- Supports multiple resource types, including a variety of CPUs and GPU models.
- Allows you to switch among different types of resources, which reduces resource usage costs.
- Provides built-in big data development packages and algorithm libraries, and allows you to install third-party libraries.

Editions

DSW provides the following editions: Individual Edition, GPU On-sale Edition, and Explorer Edition. The following table describes the differences among the editions.

Feature	Individual Edition	GPU On-sale Edition	Explorer Edition
GPUs	Supported.	Supported.	Supported.
Pay-as-you-go billing method	Supported.	Not supported.	Free of charge.
Subscription billing method	Not supported.	Supported.	Free of charge.
Memory and number of CPU cores	No upper limit. You can specify the memory and CPU cores as needed.	No upper limit. You can specify the memory and CPU cores as needed.	2 vCPUs + 4 GB of memory
Instance storage	No upper limit. You can specify storage space as needed.	No upper limit. You can specify storage space as needed.	5 GB

Feature	Individual Edition	GPU On-sale Edition	Explorer Edition
Network access	No limit.	DSW instances of this edition cannot access the Internet.	No limit on CPU-accelerated DSW instances. However, GPU-accelerated DSW instances cannot access the Internet.
Root permissions	Supported.	Not supported.	Not supported.
Deploy a runtime from a specified image	Supported.	Not supported.	Not supported.

Individual Edition

DSW Individual Edition is upgraded from DSW 2.0. This edition is developed based on cloud-native technologies of Alibaba Cloud, such as Docker and Kubernetes. It provides open and AI-assisted development environments for you to train models with high elasticity. The following section describes the features, instance types, and supported zones and images of this edition.

- Features
 - Reduces the time that is required to create a DSW instance by 65%, compared with DSW 2.0. Does not charge additional fees for elastic IP addresses (EIPs) and Server Load Balancer (SLB) instances.
 - Allows you to start and stop DSW instances on demand, save images with one click, and restore development environments.
 - Provides integrated development environments:
 - Provides built-in big data development packages and algorithm packages, and grants sudo permissions to you for installing third-party libraries.
 - Provides built-in JupyterLab plug-ins to improve development efficiency, such as Git and TensorBoard.
 - Provides official images that support different versions of mainstream computing frameworks, such as TensorFlow and PyTorch.
 - Provides built-in WebIDE that allows you to install all plug-ins.
 - Supports basic features of PAI, including EasyVision, AutoML, TAO, and CommonIO. EasyVision is a computer vision algorithm tool. AutoML can help you automatically tune parameters. TAO is used to optimize compilation. CommonIO allows you to read data from MaxCompute tables.
 - Supports root permissions.

- Instance types and zones

The following table describes the instance types and supported zones of CPU-accelerated DSW instances.

Instance type	vCPUs	Memory (GiB)	Bandwidth (Gbit/s)	System disk (GB)	Region
---------------	-------	--------------	--------------------	------------------	--------

Instance type	vCPUs	Memory (GiB)	Bandwidth (Gbit/s)	System disk (GB)	Region
ecs.c6.large	2	4	1	128	<ul style="list-style-type: none"> China (Beijing) China (Shanghai) China (Hangzhou) China (Shenzhen)
ecs.g6.large	2	8	1	128	
ecs.g6.xlarge	4	16	1.5	256	
ecs.g6.2xlarge	8	32	2.5	500	
ecs.g6.4xlarge	16	64	5	500	
ecs.g6.8xlarge	32	128	10	500	

The following table describes the instance types and supported zones of GPU-accelerated DSW instances.

Instance type	vCPUs	Memory (GiB)	GPU	Bandwidth (Gbit/s)	System disk (GB)	Region
ecs.gn6e-c12g1.12xlarge	48	368	4*NVIDIA V100	16	500	<ul style="list-style-type: none"> China (Beijing) China (Shanghai) China (Hangzhou) China (Shenzhen)
'ecs.gn5-c4g1.xlarge	4	30	1*NVIDIA P100	3	256	
ecs.gn5-c8g1.2xlarge	8	60	1*NVIDIA P100	3	500	
'ecs.gn5-c8g1.4xlarge	16	120	2*NVIDIA P100	5	500	
ecs.gn5-c28g1.7xlarge	28	112	1*NVIDIA P100	5	500	

• Images

Image name	Description
py27_cuda90_tf1.12_ubuntu	Supports GPU-based TensorFlow 1.12.
py27_cpu_tf1.12_ubuntu	Supports CPU-based TensorFlow 1.12.
py36_cuda101_tf2.1_torch1.4_ubuntu	Supports GPU-based TensorFlow 2.1 and PyTorch 1.4.
py36_cpu_tf2.1_torch1.4_ubuntu	Supports CPU-based TensorFlow 2.1 and PyTorch 1.4.

Image name	Description
py36_cuda100_paitf1.12_alios	Supports GPU-based PAI-TensorFlow 1.12.
py36_cpu_paitf1.12_alios	Supports CPU-based PAI-TensorFlow 1.12.
py36_cuda100_tf1.15_ubuntu	Supports GPU-based TensorFlow 1.15.
py36_cpu_tf1.15_ubuntu	Supports CPU-based TensorFlow 1.15.

GPU On-sale Edition

GPU On-sale Edition is upgraded from DSW 1.0. This edition is developed based on the Apsara big data platform of Alibaba Cloud and greatly reduces the costs of running DSW instances. However, this edition does not support Internet access, root permissions, or sudo operations. Purchase with caution. The following section describes the features, instance types, and supported zones of this edition.

- Features
 - Supports real-time resource monitoring. Monitors CPU or GPU utilization during algorithm development.
 - Supports a variety of data sources, such as MaxCompute, OSS, and Apsara File Storage NAS.
 - Allows you to write and execute SQL statements.
 - Supports multiple resource types, including a variety of CPUs and GPU models.
 - Allows you to switch among different types of resources, which reduces resource usage costs.
 - Provides built-in big data development packages and algorithm libraries, and allows you to install third-party libraries.
- Instance types and zones

Instance type	Region
P100	China (Beijing)
M40	China (Shanghai)

Explorer Edition

Explorer Edition is upgraded from DSW 2.0. This edition is developed based on cloud-native technologies of Alibaba Cloud, such as Docker and Kubernetes. It provides open and AI-assisted development environments for you to train models with high elasticity. Explorer Edition is free of charge. You can click [Explorer Edition](#) to try it. For more information about how to use this edition, see [DSW user guide](#).

2. Authorization

This topic shows you how to authorize RAM users to manage Data Science Workshop (DSW) instances and assign service-linked roles to DSW.

Context

For more information about the structure and syntax of permission policies, see [Policy structure and syntax](#).

Authorize RAM users

You can use your Alibaba Cloud account to authorize RAM users to manage DSW instances. Then, the RAM users can create, start, stop, and delete DSW instances.

1. Log on to the [Resource Access Management \(RAM\) console](#).
2. Create a custom policy.
 - i. In the left-side navigation pane, choose **Permissions > Policies**.
 - ii. On the **Policies** page, click **Create Policy**.
 - iii. On the **Create Custom Policy** page, set the parameters.

Parameter	Description
Policy Name	Enter DSW_Notebook_Access.
Note	Enter DSW access policy.
Configuration Mode	Select Script .
	<div><p>Add the following content to Policy Document:</p><pre>{ "Statement": [{ "Action": ["notebook:CreateInstance", "notebook:StartInstance", "notebook:StopInstance", "notebook>EditInstance", "notebook>ListInstance"], "Effect": "Allow", "Resource": "*" }], "Version": "1" }</pre><p>Action indicates the granted permissions, including:</p><ul style="list-style-type: none">■ notebook:CreateInstance: creates DSW instances.■ notebook:StartInstance: starts DSW instances.</div>

Parameter	Description
Policy Document	■ notebook:StopInstance: stops DSW instances.
	■ notebook>EditInstance: modifies DSW instances.
	■ notebook>ListInstance: views all DSW instances.
	Resource indicates resources that a RAM user is authorized to manage. You can set this parameter in the following ways:
	■ Authorize the RAM user to manage all DSW instances in a specified region. <pre>"Resource": "acs:notebook:cn-beijing:*:*:notebook/*"</pre>
	■ Authorize the RAM user to manage a specified instance, such as the DSW instance hhdemo in the following example: <pre>"Resource": "acs:notebook:*:*:notebook/hhdemo"</pre>
	■ Authorize the RAM user to manage all DSW instances. <pre>"Resource": ""</pre>
	For more information about other permissions, see Policy elements .

- iv. Click **OK**.
3. Grant permissions to a RAM user.
 - i. In the left-side navigation pane, choose **Identities > Users**.
 - ii. On the **Users** page, click **Add Permissions** in the **Actions** column.
 - iii. (Optional) In the **Add Permissions** dialog box, click **Custom Policy**.
 - iv. In the **Select Policy** section, enter **DSW_Notebook_Access**.
 - v. Click **DSW_Notebook_Access** in the **Authorization Policy Name** column. The policy is then displayed in the **Selected** section.
 - vi. Click **OK**.

Assign service-linked roles

If you are a first-time user of DSW, you must first assign service-linked roles to DSW so that DSW can access the required resources.

1. Go to the **Notebook Models** page.
 - i. Log on to the [Machine Learning Platform for AI \(PAI\) console](#).
 - ii. In the left-side navigation pane, choose **Model Training > DSW-Notebook Service**.

2. Click **Create Instance**.
3. In the **Role Authorization** dialog box, click **Authorize Now**.
4. On the **Cloud Resource Access Authorization** page, click **Agree to Authorize**. Service-linked roles are automatically assigned to DSW and displayed on the **Cloud Resource Access Authorization** page.

3. Work with DSW

3.1. Create an instance

You must create Data Science Workshop (DSW) instances before you use DSW to build Notebook models. This topic describes how to create a DSW instance.

Prerequisites


If you are a first-time user of DSW, you must first assign service-linked roles to DSW so that DSW can access the required resources. For more information, see [Assign service-linked roles](#).

If you want to create a DSW instance as a RAM user, you must first use your Alibaba Cloud account to authorize the RAM user. For more information, see [Authorize RAM users](#).

Procedure

1. Log on to the [Machine Learning Platform for AI console](#).
2. In the left-side navigation pane, choose **Model Training** > **DSW-Notebook Service**.
3. In the upper-left corner of the page, select the region where you want to create the instance.
4. On the **Notebook Modeling** page, click **Create Instance**.
5. In the **Configure Instance** step, set the parameters that are described in the following table.

Parameter	Description
Instance Name	The name cannot exceed 27 characters in length. It can contain only letters, digits, and underscores (_).
Instance Version	The following editions are supported: <ul style="list-style-type: none">◦ DSW Individual Edition: This edition is upgraded from DSW 2.0. It is developed based on cloud-native technologies of Alibaba Cloud, such as Docker and Kubernetes. It provides open and AI-assisted development environments for you to train models with high elasticity.◦ DSW On-sale Edition: This edition is upgraded from DSW 1.0. It is developed based on the Apsara big data platform of Alibaba Cloud and greatly reduces the costs of running DSW instances. However, this edition does not support Internet access, root permissions, or sudo operations. Purchase with caution.
Region & Zone	Instances in different regions cannot communicate with each other. Select a region that is close to your clients to reduce network latency and improve connection quality.
Billing Method	A DSW Individual Edition instance supports only the pay-as-you-go billing method. A DSW On-sale Edition instance supports only the subscription billing method.
Network	This parameter is available only to DSW On-sale Edition instances. The default setting is Classic Network , and cannot be changed.

Parameter	Description
Instance	This parameter is available only to DSW Individual Edition instances. You can create a CPU- or GPU-accelerated instance. For more information about instance specifications, see Individual Edition .
Resource Type	This parameter is available only to DSW On-sale Edition instances. If the number of remaining GPU cards is insufficient for your business needs, go to the buy page to purchase GPU cards.
Storage	<p>The instance comes with a system disk storage as a temporary storage. After the instance is stopped or deleted, this storage is cleared. If a permanent storage is required, you must mount a created Apsara File Storage NAS file system to the instance. For more information about how to create an Apsara File Storage NAS file system, see Manage file systems.</p> <div>  Note After an Apsara File Storage NAS file system is mounted, the instance uses this permanent storage, instead of the temporary storage, to store data. </div>
Select Image	Multiple Python, TensorFlow, and PyTorch versions of images are supported. For more information about supported images, see Individual Edition .
VPC	<p>This parameter is available only to DSW Individual Edition instances. You can select a VPC to deploy the DSW instance. If you set the VPC parameter, you must also set the vSwitch and Security Group parameters.</p> <p>You can select an existing VPC to deploy the DSW instance. You can also click Create VPC next to VPC and create a VPC to deploy the DSW instance.</p>
vSwitch	If you set the VPC parameter, you must also set the vSwitch and Security Group parameters.
Security Group	You can select an existing vSwitch and an existing security group to deploy the DSW instance. You can also click Create vSwitch and Create Security Group and create a vSwitch and a security group to deploy the DSW instance.

- Click **Confirm Order**.
- After you confirm the order, select **Machine Learning DSW Terms of Service** and click **Create Instance**.

3.2. Manage instances

This topic shows you how to manage Data Science Workshop (DSW) instances. You can start, stop, and delete DSW instances.

Prerequisites

- If you want to manage DSW instances as a RAM user, you must first use your Alibaba Cloud account to authorize the RAM user. For more information, see [Authorization](#).
- A DSW instance is created. For more information, see [Create an instance](#).

Go to DSW

1. Log on to the [Machine Learning Platform for AI console](#).
2. In the left-side navigation pane, choose **Model Training** > **DSW-Notebook Service**.

Stop an instance

1. Click **DSW-Notebook Service**.
2. On the **Notebook Models** page, find the instance that you want to stop, and click **Stop** in the **Actions** column.
3. In the dialog box that appears, select a stop method. You can use the following methods to support DSW instances:
 - **Stop after Saving Environment**: We recommend that you select this method if you have modified the default environment, for example, a software package or a pip package is installed.
 - **Stop without Saving**: Generally, you can select **Stop without Saving** if you do not modify the default environment.

After the instance is stopped, the status in the **Status** column changes to **Stopped**. If the instance is a pay-as-you-go instance, the system stops charging you for the instance. Make sure that the status of the DSW instances that are not in use is **Stopped**. Otherwise, the system continues to charge you for the instances.

Start an instance

You can manually start an instance that is in the Stopped state.

1. Click **DSW-Notebook Service**.
2. On the **Notebook Models** page, find the instance that you want to start, and click **Start** in the **Actions** column.

After the instance is started, the status in the **Status** column changes to **Running**. If the instance is a pay-as-you-go instance, the system starts to charge you for the instance. We recommend that you stop DSW instances after you complete model training. Otherwise, the system continues to charge you for the instances.

Delete an instance

You can delete instances that are not in use. After an instance is deleted, its data cannot be recovered.

1. Click **DSW-Notebook Service**.
2. On the **Notebook Models** page, find the instance that you want to delete, and click **Delete** in the **Actions** column.
3. In the **Delete** message, click **OK**.

3.3. Work with the development environments of DSW

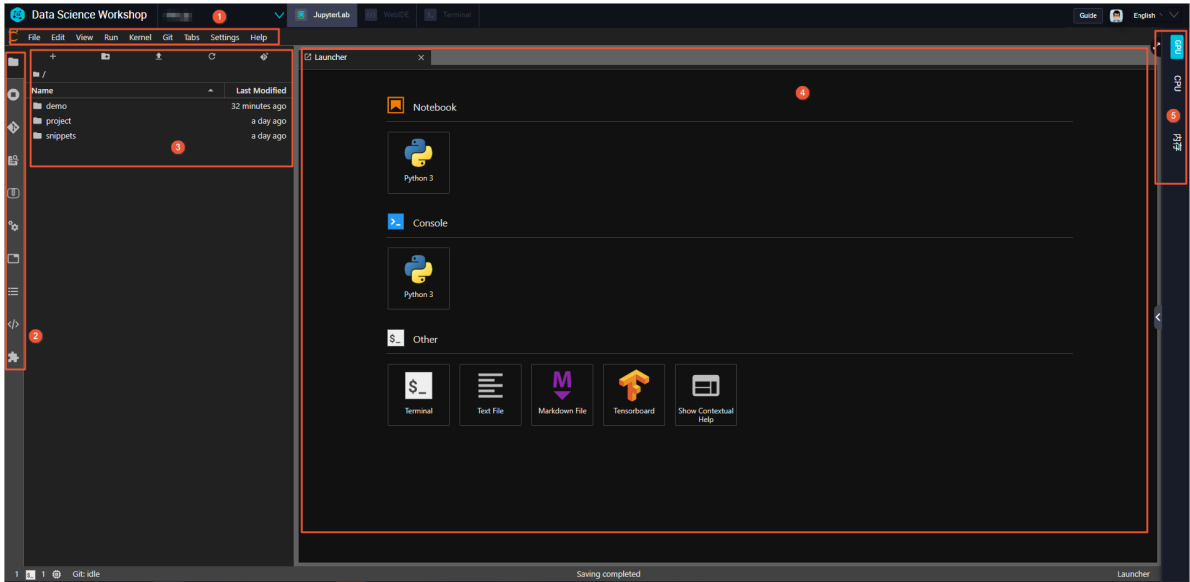
This topic shows you how to work with the development environments of Data Science Workshop (DSW), including how to use user interfaces, run preset Notebook cases, upload data, and manage third-party libraries.

Prerequisites

A DSW instance is created. For more information, see [Create an instance](#).

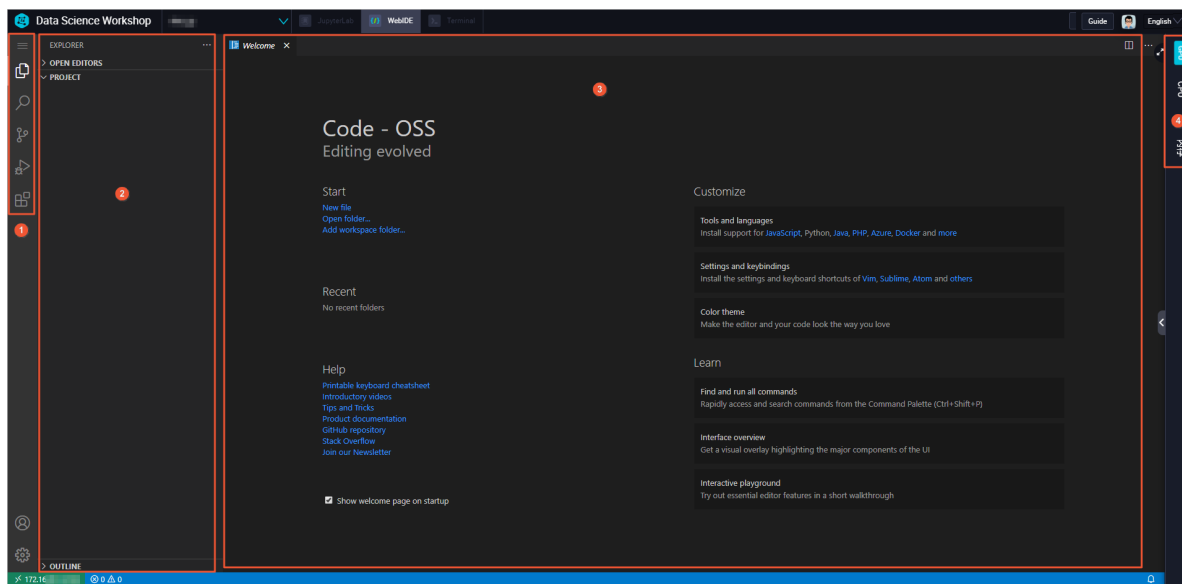
User interfaces

- JupyterLab



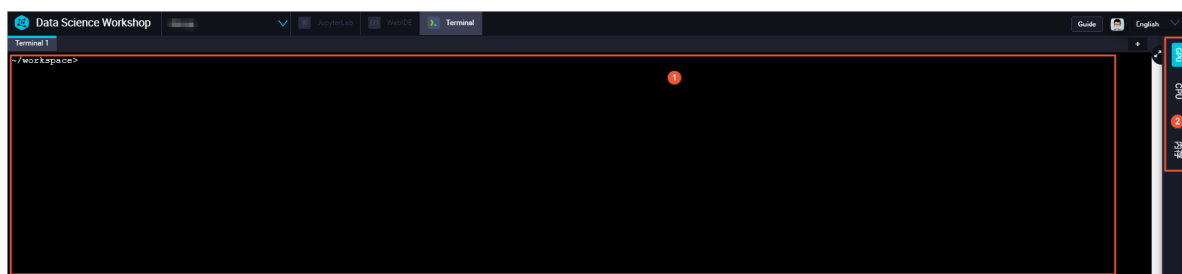
No.	Description
1	Top navigation bar
2	Left-side navigation pane
3	File browser
4	Main work area
5	Resource usage monitoring area

- WebIDE



No.	Description
1	Left-side navigation pane
2	File browser
3	Main work area
4	Resource usage monitoring area

- Terminal





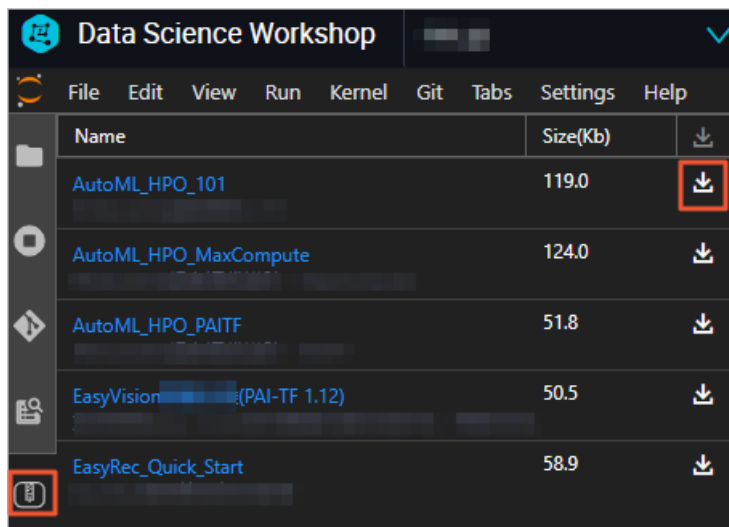
No.	Description
1	Main work area
2	Resource usage monitoring area

Run preset Notebook cases

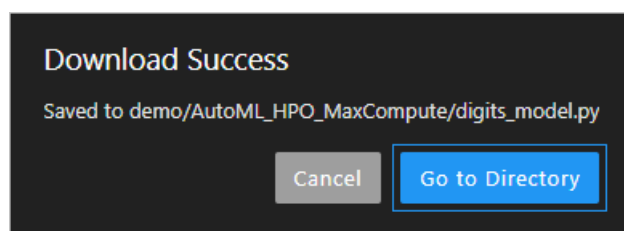
If you are a first-time user of DSW, we recommend that you use one of the preset cases to familiarize yourself with related features.


1. Go to the development environment of DSW.
 - i. Log on to the [Machine Learning Platform for AI console](#).
 - ii. In the left-side navigation pane, choose **Model Training** > **DSW-Notebook Service**.

- iii. In the upper-left corner of the page, select the region that you want.
 - iv. (Optional) In the search box on the **Notebook Modeling** page, enter the name or keywords of a Data Science Workshop (DSW) instance to search for the DSW instance.
 - v. Find the DSW instance and click **Launch DSW** in the **Actions** column.
2. Download a preset case.
 - i. In the left-side navigation pane of the Data Science Workshop page, click the  icon.
 - ii. Find the case that you want to download, such as **AutoML_HPO_101**, and click the  icon next to the case.





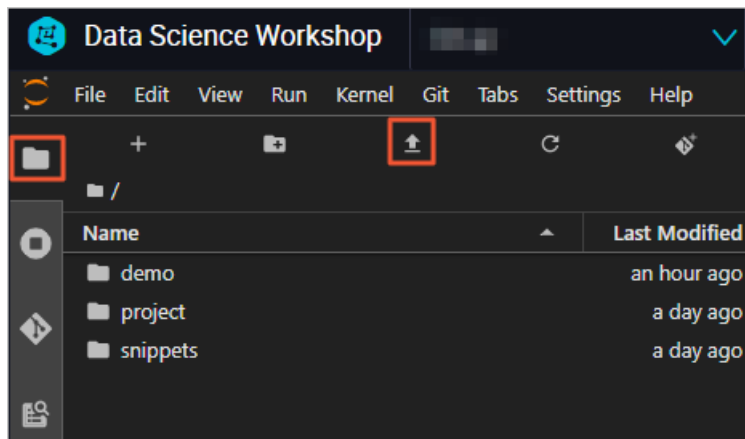
3. Open the model file of the downloaded case. The **AutoML_HPO_101** case is used as an example.
 - i. Go to the directory in which the downloaded case is stored. The downloaded **AutoML_HPO_101** case is stored in the `/demo/AutoML_HPO_101/` directory. You can go to the directory by using one of the following methods:
 - After you download the case, click **Go to Directory** in the **Download Success** dialog box.



- Use the left-side navigation pane of the Data Science Workshop page.
 - a. In the left-side navigation pane of the Data Science Workshop page, click the  icon.
 - b. Double-click **Demo** in the **Name** column.
 - c. Double-click **AutoML_HPO_101** in the **Name** column.
 - ii. In the storage directory, double-click **AutoML_HPO_101.ipynb** in the **Name** column to open the file.
4. In the **AutoML_HPO_101.ipynb** file, you can view the use principle of the case and perform tasks as instructed.

Upload files

1. In the left-side navigation pane of the JupyterLab Notebook programming environment of DSW, click the  icon.
2. Click the  icon in the toolbar to upload files. Resumable upload is supported.



Manage third-party libraries

If you use a Python development environment, you can perform the following operations to manage third-party libraries in the Terminal interface:

- Install a third-party library

```
pip install --user <yourLibraryName>
```

You must replace `<yourLibraryName>` with the name of the third-party library that you want to install. For example, you can run the `pip install --user sklearn` command to install a sklearn library.

- View third-party libraries


```
pip list
```

You can view all third-party libraries that you have installed.

- Remove a third-party library

```
pip uninstall <yourLibraryName>
```


You must replace `<yourLibraryName>` with the name of the third-party library that you want to remove.

 **Note** You can remove only the third-party libraries that are installed by yourself.

tensorflow-gpu cannot be removed. You can only run the update command to install a specified version of tensorflow-gpu. The specified version must be compatible with the Compute Unified Device Architecture (CUDA) version of the DSW instance that you are using. Subscription DSW instances use CUDA 10. Pay-as-you-go DSW instances use CUDA 9.

```
pip install --upgrade --user tensorflow-gpu=<versionNumber>
```

You must replace `<versionNumber>` with the version number of tensorflow-gpu that you want to install.

 **Notice** Do not upgrade pip. Otherwise, the installation may fail.

DSW provides the following development environments: Python2, Python3, PyTorch, and TensorFlow2.0. By default, third-party libraries are installed in Python3. If you want to install a third-party library in another development environment, you must manually switch to the environment where you want to install the third-party library.

```
# Install a third-party library in Python2.
source activate python2
pip install --user <yourLibraryName>
# Install a third-party library in TensorFlow2.0.
source activate tf2
pip install --user <yourLibraryName>
```

You must replace `<yourLibraryName>` with the name of the third-party library that you want to install.

3.4. Read data from and write data to OSS

This topic describes how to use Object Storage Service (OSS) SDK for Python, OSS IO for TensorFlow, and OSS API for PyTorch to read data from and write data to OSS.

Context

OSS is a secure, cost-effective, and highly reliable cloud storage service provided by Alibaba Cloud. It enables you to store a large amount of data in the cloud. By default, Data Science Workshop (DSW) instances are attached with Network Attached Storage (NAS) file systems. You can also use DSW instances with OSS if you require larger storage.

OSS SDK for Python

In most cases, you can use OSS SDK for Python to read data from and write data to OSS. For more information, see . DSW has preinstalled OSS2 for Python packages. The following code block describes how to read data from or write data to OSS:

1. Authentication and initialization.

```
import oss2
auth = oss2.Auth('<your_AccessKey_ID>', '<your_AccessKey_Secret>')
bucket = oss2.Bucket(auth, 'http://oss-cn-beijing-internal.aliyuncs.com', '<your_bucket_name>')
```

Set the following parameters based on your requirements.

Parameter	Description
<code><your_AccessKey_ID></code>	The AccessKey ID of your Alibaba Cloud account.
<code><your_AccessKey_Secret></code>	The AccessKey secret of your Alibaba Cloud account.

Parameter	Description
<code>http://oss-cn-beijing-internal.aliyuncs.com</code>	<p>The endpoint of OSS. Select an endpoint based on the region where your DSW instances are deployed.</p> <ul style="list-style-type: none">◦ Pay-as-you-go instances deployed in China (Beijing): <code>oss-cn-beijing.aliyuncs.com</code>◦ Subscription instances deployed in China (Beijing): <code>oss-cn-beijing-internal.aliyuncs.com</code>◦ GPU P100 instances and CPU instances deployed in China (Shanghai): <code>oss-cn-shanghai.aliyuncs.com</code>◦ GPU M40 instances deployed in China (Shanghai): <code>oss-cn-shanghai-internal.aliyuncs.com</code>
<code><your_bucket_name></code>	The name of the OSS bucket. It cannot start with <code>oss://</code> .

2. Read data from and write data to OSS.

```
#Read a file from OSS.
result = bucket.get_object('<your_file_path/your_file>')
print(result.read())
#Read data by range.
result = bucket.get_object('<your_file_path/your_file>', byte_range=(0, 99))
#Write data to OSS.
bucket.put_object('<your_file_path/your_file>', '<your_object_content>')
#Append a file.
result = bucket.append_object('<your_file_path/your_file>', 0, '<your_object_content>')
result = bucket.append_object('<your_file_path/your_file>', result.next_position, '<your_object_content>')
```

`<your_file_path/your_file>` indicates the OSS path from which the data is to be read and to which the data is to be written. `<your_object_content>` indicates the content that you want to append to a file. Set the parameters based on your requirements.

OSS IO for TensorFlow

DSW provides the `tensorflow_io.oss` module. TensorFlow users can use the module to read data from and write data to OSS. Therefore, users do not need to frequently copy data or model files when training models in TensorFlow.

1. Import the `tensorflow_io.oss` module, and concatenate OSS bucket URLs.

```
import tensorflow as tf
import tensorflow_io.oss
access_id="<your_Access_Key_ID>"
access_key="<your_Access_Key_Secret>"
host = "oss-cn-beijing-internal.aliyuncs.com"
bucket="oss://<your_bucket_name>"
oss_bucket_root="{}/x01id={}/x02key={}/x02host={}/".format(bucket, access_id, access_key, host)
```

Set the following parameters based on your requirements.

Parameter	Description
<your_AccessKey_ID>	The AccessKey ID of your Alibaba Cloud account.
<your_AccessKey_Secret>	The AccessKey secret of your Alibaba Cloud account.
oss-cn-beijing-internal.aliyuncs.com	<p>The endpoint of OSS. Select an endpoint based on the region where your DSW instances are deployed.</p> <ul style="list-style-type: none"> Pay-as-you-go instances deployed in China (Beijing): oss-cn-beijing.aliyuncs.com Subscription instances deployed in China (Beijing): oss-cn-beijing-internal.aliyuncs.com GPU P100 instances and CPU instances deployed in China (Shanghai): oss-cn-shanghai.aliyuncs.com GPU M40 instances deployed in China (Shanghai): oss-cn-shanghai-internal.aliyuncs.com
<your_bucket_name>	The name of the OSS bucket. It cannot start with oss://.

2. You can use one of the following methods to read data from or write data to OSS. TensorFlow 1.0 API is used as an example in this topic.

- Use `GFile` to read text files from and write text files to OSS.

```
oss_file = oss_bucket_root + "test.txt"
with tf.gfile.GFile(oss_file, "w") as f:
    f.write("<your_context>")
with tf.gfile.GFile(oss_file, "r") as f:
    print(f.read())
```

<your_context> indicates the content that you want to write to OSS. Set the parameter based on your requirements.

- Use `TextLineDataset` to read data from OSS.

```
#Test textline reader op.
oss_file = oss_bucket_root + "test.txt"
dataset = tf.data.TextLineDataset([oss_file])
iterator = dataset.make_initializable_iterator()
a = iterator.get_next()
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    sess.run(iterator.initializer)
    print(sess.run(a))
```

OSS API for Python

For PyTorch users, DSW provides OSS API for Python to read data from and write data to OSS.


You can store training data, logs, and model files in OSS.

- Load training data

You can store training data in an OSS bucket. The path and labels of the data must be stored in an index file in the same OSS bucket. You can customize `DataSet` and call the `DataLoader` API in PyTorch to read data through multiple threads in parallel. The following code block shows an example:


```
import io
import oss2
import PIL
import torch
class OSSDataset(torch.utils.data.dataset.Dataset):
    def __init__(self, endpoint, bucket, auth, index_file):
        self._bucket = oss2.Bucket(auth, endpoint, bucket)
        self._indices = self._bucket.get_object(index_file).read().split(',')
    def __len__(self):
        return len(self._indices)
    def __getitem__(self, index):
        img_path, label = self._indices(index).strip().split(':')
        img_str = self._bucket.get_object(img_path)
        img_buf = io.BytesIO()
        img_buf.write(img_str.read())
        img_buf.seek(0)
        img = Image.open(img_buf).convert('RGB')
        img_buf.close()
        return img, label
dataset = OSSDataset(endpoint, bucket, index_file)
data_loader = torch.utils.data.DataLoader(
    dataset,
    batch_size=batch_size,
    num_workers=num_loaders,
    pin_memory=True)
```

`endpoint` indicates the endpoint of OSS. `bucket` indicates the name of the OSS bucket. `auth` indicates the objects that are authenticated. `index_file` indicates the path of the index file. Set the parameters based on your requirements.

 **Note** In this topic, samples in the index file are separated with commas (,). The sample path and labels are separated with colons (:).

- Write logs to OSS

You can compile a `StreamHandler` to write log data to OSS.

 **Note** You cannot write a log through multiple threads in parallel.

```
import oss2
import logging
class OSSLoggingHandler(logging.StreamHandler):
    def __init__(self, endpoint, bucket, auth, log_file):
        OSSLoggingHandler.__init__(self)
        self._bucket = oss2.Bucket(auth, endpoint, bucket)
        self._log_file = log_file
        self._pos = self._bucket.append_object(self._log_file, 0, "")
    def emit(self, record):
        msg = self.format(record)
        self._pos = self._bucket.append_object(self._log_file, self._pos.next_position, msg)
oss_handler = OSSLoggingHandler(endpoint, bucket, log_file)
logging.basicConfig(
    stream=oss_handler,
    format='%(asctime)s] [%(levelname)s] [%(process)d#%(threadName)s] ' +
        '[(filename)s:%(lineno)d] %(message)s',
    level=logging.INFO)
```

`endpoint` indicates the endpoint of OSS. `bucket` indicates the name of the OSS bucket. `auth` indicates the objects that are authenticated. `log_file` indicates the path where you want to store the log file. Set the parameters based on your requirements.

- Save or load models

You can use OSS2 API for Python to save or load PyTorch models. For more information about how to save or load models by using PyTorch, see [PyTorch](#).

- Save a model

```
from io import BytesIO
import torch
import oss2
# bucket_name
bucket_name = "<your_bucket_name>"
bucket = oss2.Bucket(auth, endpoint, bucket_name)
buffer = BytesIO()
torch.save(model.state_dict(), buffer)
bucket.put_object("<your_model_path>", buffer.getvalue())
```

`endpoint` indicates the endpoint of OSS. `bucket` indicates the name of the OSS bucket. It cannot start with `oss://`. `auth` indicates the objects that are authenticated. `<your_model_path>` indicates the path where you want to store the model. Set the parameters based on your requirements.

- Load a model

```
from io import BytesIO
import torch
import oss2
bucket_name = "<your_bucket_name>"
bucket = oss2.Bucket(auth, endpoint, bucket_name)
buffer = BytesIO(bucket.get_object("<your_model_path>").read())
model.load_state_dict(torch.load(buffer))
```

`endpoint` indicates the endpoint of OSS. `bucket` indicates the name of the OSS bucket. It cannot start with `oss://`. `auth` indicates the objects that are authenticated. `log_file` indicates the path where the model is stored. Set the parameters based on your requirements.

3.5. Read data from and write data to MaxCompute tables

This topic shows you how to use PyODPS to read data from and write data to MaxCompute tables.

PyODPS

You can use PyODPS to read data from and write data to MaxCompute or Machine Learning Studio. PyODPS is an SDK for Python provided by Alibaba Cloud. For more information, see [PyODPS documentation](#).

1. Install PyODPS. In the Terminal interface of Data Science Workshop (DSW), run the following command:

```
pip install --user pyodps
```

2. Read data from MaxCompute tables. In this example, read the first 10 rows of a table from a MaxCompute project.

```
from odps import ODPS
from odps.df import DataFrame
o = ODPS('<your_AccessKey_ID>', '<your_AccessKey_Secret>', project='<your_MaxCompute_project>',
        endpoint='http://service-all.ext.odps.aliyun-inc.com/api')
users = DataFrame(o.get_table('<your_table_name>'))
print(users.head(10))
```

Set the parameters that are described in the following table based on your requirements.

Parameter	Description
<your_AccessKey_ID>	The AccessKey ID of your Alibaba Cloud account.
<your_AccessKey_Secret>	The AccessKey secret of your Alibaba Cloud account.
<your_MaxCompute_project>	The name of the MaxCompute project.

Parameter	Description
<code>http://service-all.ext.odps.aliyun-inc.com/api</code>	The endpoint of GPU M40 instances deployed in the China (Shanghai) region and subscription P100 instances deployed in the China (Beijing) region. For DSW instances in other regions, the endpoint is <code>http://service.cn.maxcompute.aliyun.com/api</code> .
<code><your_table_name></code>	The name of the MaxCompute table.

3.6. Export notebooks

You can export notebooks as local files in various formats so that you can view and share the notebooks.

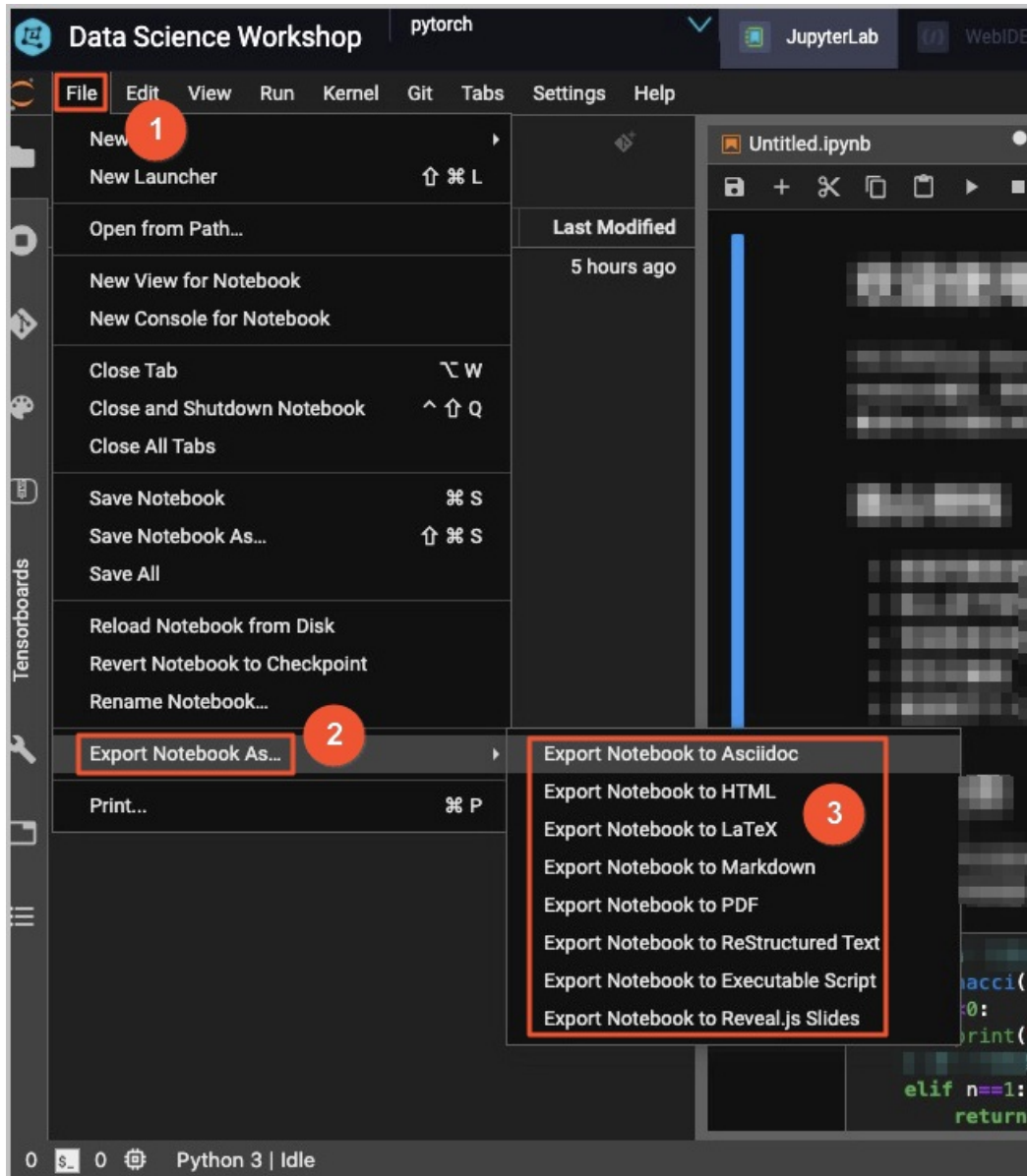
Context

Data Science Workshop (DSW) of Machine Learning Platform for AI (PAI) allows you to export notebooks as the following types of local files:

- AsciiDoc files in the .asciidoc format
- HTML files in the .html format
- LaTeX files in the .tex format
- Markdown files in the .md format
- Portable document format (PDF) files in the .pdf format
- reStructuredText files in the .rst format
- Executable scripts in the .py format
- Reveal.js slides in the .html format

Procedure

1. Go to the development environment of DSW.
 - i. Log on to the **PAI console**.
 - ii. In the left-side navigation pane, choose **Model Training > DSW-Notebook Service**.
 - iii. On the Notebook Models page, select the region of the instance that you want to manage in the upper-left corner.
 - iv. On the page that appears, find the instance that you want to manage and click **Launch DSW** in the **Actions** column.
2. Open the notebook that you want to export.
3. In the top navigation bar, choose **File > Export Notebook As...** and select a format from the shortcut menu. Then, the notebook is exported as a file in the selected format.



Note The file formats in which you can export a notebook are configured in nbconvert. For more information, see [nbconvert: Convert Notebooks to other formats](#).

4. Best practices

4.1. Use WebIDE to debug code online

If your code does not run as expected, you must locate the cause. To locate the cause, you can carefully check the code. You can also enable the debug logging feature and use an online debugging tool to check the variables and logic of the code based on debug logs. This topic describes how to use WebIDE of Data Science Workshop (DSW) to debug the Python code in the sample notebook that is provided by DSW of Machine Learning Platform for AI (PAI).

Context

DSW provides a few examples to demonstrate how to tune hyperparameters by using AutoML. The code in the example "Quick start to hyperparameter tuning by using AutoML" is often stuck at the `tuner.fit()` method. The following figure shows the error logs in an output cell of the sample notebook.

```
[*]: your_job_id = tuner.fit()
    your_job_id

2020-04-18,04:03:58.306 [INFO] Missing algorithm, using default setting
"initial_trials_num": 4, "stop_when_exception": true}}
2020-04-18,04:03:58.307 [INFO] Missing earllystop, using default setting

[*]: tuner.status()

[*]: tuner.info()

[*]: tuner.log()

[*]: # tuner.kill(your_job_id)

[*]: row = tuner.get_best_config()
    print(row['Hyperparams'])
```

The error logs do not provide enough information for you to locate the cause. You can check the code of tuner and debug the code. To check the code of tuner, perform the following steps:

1. Find the location where the class of tuner is defined.

The class of tuner is AutoTuner and is defined in the pai.automl.hpo Python package, as shown in the following figure.

```
[1]: from pai.automl.hpo import *

n = hyperparam.create(type='Categorical',
                      name='n', candidates=[10, 20, 30, 40, 50, 60, 70, 80, 90])
s = hyperparam.create(type='Categorical',
                      name='s', candidates=[4, 3, 2])
d = hyperparam.create(type='Categorical',
                      name='d', candidates=[2, 3, 4, 5])
lr = hyperparam.create(type='Categorical',
                      name='lr', candidates=[0.01, 0.02, 0.04, 0.08, 0.16, 0.32])
params = [n, s, d, lr]
```

2. Search for the AutoTuner class to obtain the path of the AutoTuner class in DSW. In this example, the path is `/home/admin/.local/lib/python3.6/site-packages/pai/automl`. Open the file of the AutoTuner class in WebIDE as a project, as shown in the following figure.

```
hpo > automl > AutoTuner > _init_
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238

def fit(self):
    """start training in given mode.

    Returns:
        a job id.
    """
    self._fit_param_check('fit()')

    self._job_id = self._get_client().submit_job(
        yaml.safe_dump(self._to_dict()), self._user_args)

    return self._job_id

def is_active(self, job_id=None):
    """check if the job is still active.

    Returns:
        True, if active, otherwise False.
    """
    job_id = self._job_id if job_id is None else job_id
    return self._get_client().is_active(job_id)

def status(self, job_id=None):
    """get job status.

    Args:
        job_id: optional: str
        job_id: for local it's optional
        job_id: for service mode, you must provide an id

    Returns:
        job status
    """
    job_id = self._job_id if job_id is None else job_id
    return self._get_client().query_job_status(job_id)
```

Combining all of the code is time-consuming. You can use a debugger to set a breakpoint at the position where the code is stuck. This helps you diagnose issues. You can use WebIDE to debug programs that are run inside or outside WebIDE. To debug a program that is run outside WebIDE, you must add the Python Tools for Visual Studio Debug Server (PTVSD) code to the program. After the PTVSD code is added to a program, the program is paused at the PTVSD code until a debugger is attached to the program. In this topic, the code in the sample notebook is run in JupyterLab and WebIDE is used to debug the code step by step.

Procedure

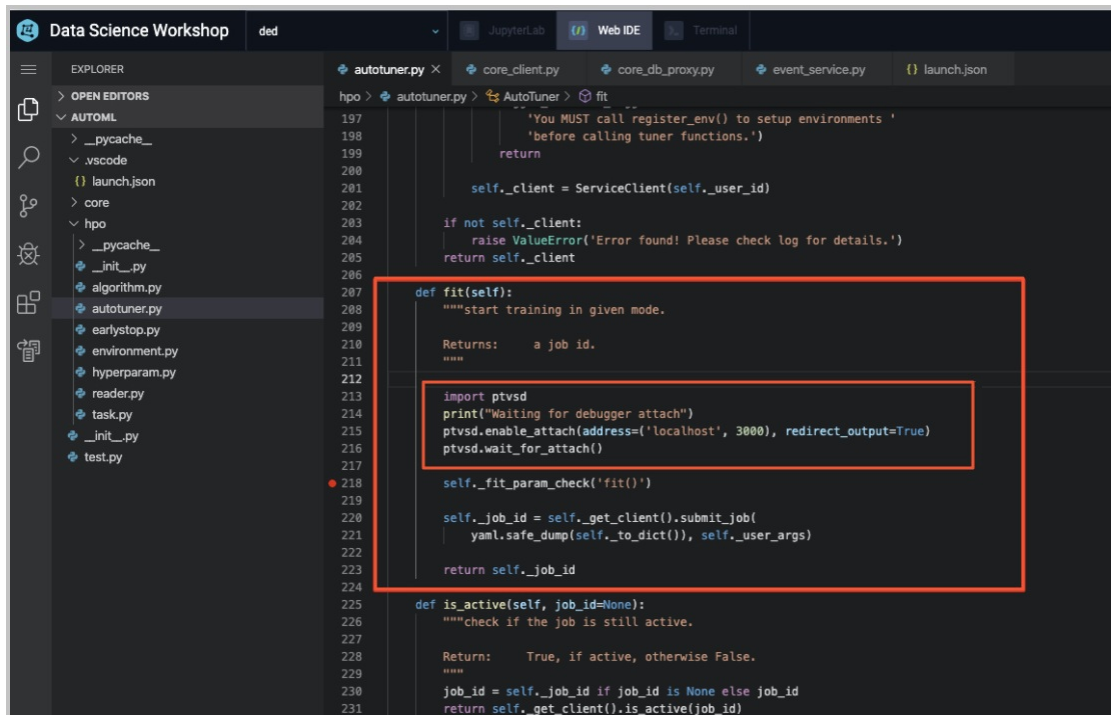
1. Add the PTVSD code to the program that you want to debug.
 - i. (Optional) If you have not installed PTVSD, run the following code to install it:

```
pip install --upgrade ptvsd.
```

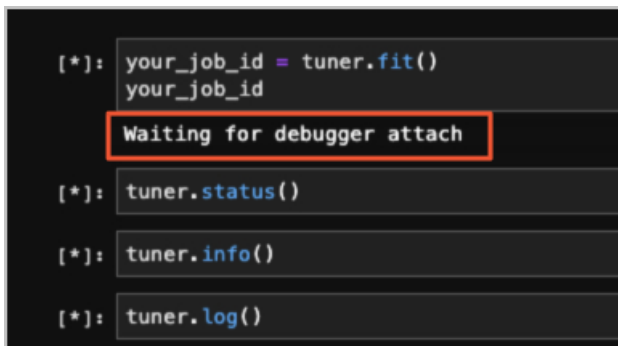
- ii. Add the following code to the program that you want to debug:

```
import ptvsd
print("Waiting for debugger attach")
# Allow other computers to attach to ptvsd at this IP address and port.
ptvsd.enable_attach(address=('192.0.2.1', 3000), redirect_output=True)
# Pause the program until a remote debugger is attached
ptvsd.wait_for_attach()
```

In the PTVSD code, `192.0.2.1` indicates the IP address of the server on which the program is run, and `3000` indicates the port on which the debugger listens. In this example, add the PTVSD code to the `autotuner.py` file in `/home/admin/.local/lib/python3.6/site-packages/pai/automl/hpo/` in WebIDE. The following figure shows the position of the PTVSD code in the `autotuner.py` file.



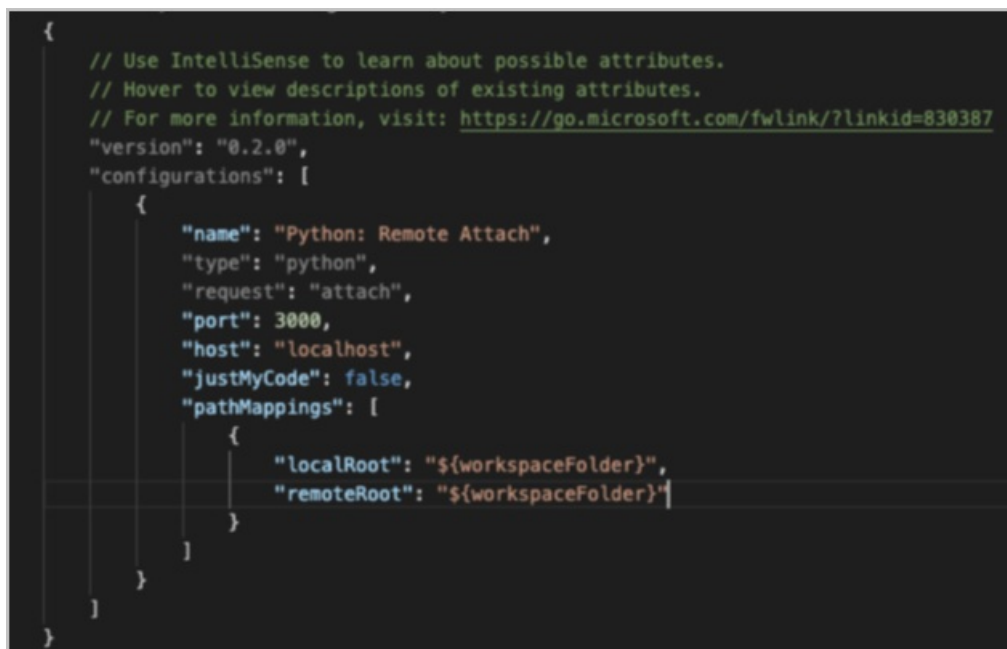
- iii. Rerun the code in the sample notebook in JupyterLab. Then, view the running result in WebIDE.



```
[*]: your_job_id = tuner.fit()
your_job_id
Waiting for debugger attach
[*]: tuner.status()
[*]: tuner.info()
[*]: tuner.log()
```

If `Waiting for debugger attach` is displayed below the cell of `tuner.fit()`, the debugger configuration is in effect.

2. In WebIDE, connect the debugger to PTVSD. To connect the debugger to PTVSD, you must add the debugger configuration to the `launch.json` file in the `.vscode` folder, as shown in the following figure.



```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python: Remote Attach",
      "type": "python",
      "request": "attach",
      "port": 3000,
      "host": "localhost",
      "justMyCode": false,
      "pathMappings": [
        {
          "localRoot": "${workspaceFolder}",
          "remoteRoot": "${workspaceFolder}"
        }
      ]
    }
  ]
}
```

The debugger configuration includes the following key parameters:

- `host`: the IP address of the server on which the debugger resides. In this example, the debugger and the program that you want to debug reside on the same server. Therefore, set the `host` parameter to `localhost`.
- `port`: the port on which the debugger listens. This port is also used to connect to PTVSD. In this example, set the `port` parameter to `3000`.
- `justMyCode`: specifies whether to enable the Just My Code feature. Set this parameter to `false`. Otherwise, the breakpoint does not take effect.

4.2. Use CNN to classify images

This topic describes how to use a Convolutional Neural Network (CNN) in Data Science Workshop (DSW) of Machine Learning Platform for AI to classify images. The dataset used in this topic is CIFAR10.

Training a Classifier

This is it. You have seen how to define neural networks, compute loss and make updates to the weights of the network. Now you might be thinking,

What about data?

Generally, when you have to deal with image, text, audio or video data, you can use standard python packages that load data into a numpy array. Then you can convert this array into a `torch.*Tensor` :

- For images, packages such as Pillow, OpenCV are useful
- For audio, packages such as scipy and librosa
- For text, either raw Python or Cython based loading, or NLTK and SpaCy are useful

Specifically for vision, we have created a package called `torchvision` , that has data loaders for common datasets such as Imagenet, CIFAR10, MNIST, etc. and data transformers for images, viz., `torchvision.datasets` and `torch.utils.data.DataLoader` . This provides a huge convenience and avoids writing boilerplate code.

For this tutorial, we will use the CIFAR10 dataset. It has the classes: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', and 'truck'. The images in CIFAR-10 are of size 3x32x32, i.e. 3-channel color images of 32x32 pixels in size.

Training an image classifier

1. Load and normalizing the CIFAR10 training and test datasets using `torchvision` .

Using torchvision, it is extremely easy to load CIFAR10.

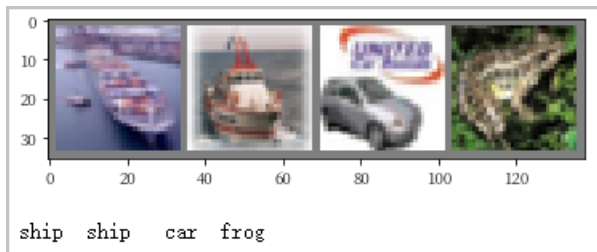
```
%matplotlib inline
import torch
import torchvision
import torchvision.transforms as transforms
```

The output of torchvision datasets are PILImage images of range [0, 1]. We transform them to Tensors of normalized range [-1, 1].

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                         download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                           shuffle=True, num_workers=2)
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                         download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                          shuffle=False, num_workers=2)
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

Let us show some of the training images, for fun.


```
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()
# 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(4)))
```



2. Define a Convolutional Neural Network.

```
import torch.nn as nn
import torch.nn.functional as F
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
net = Net()
```

3. Define a Loss function and optimizer. Let's use a Classification Cross-Entropy loss and SGD with momentum.

```
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```


4. Train the network on the training data. This is when things start to get interesting. We simply have to loop over our data iterator, and feed the inputs to the network and optimize.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Assuming that we are on a CUDA machine, this should print a CUDA device:
print(device)
```

The output is shown below.

```
cuda:0
```

Use the GPU:

```
net.to(device)
```

The output is shown below.

```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

```
for epoch in range(2): # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        # inputs, labels = data
        inputs, labels = data[0].to(device), data[1].to(device)
        # 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-batches
            print('[%d, %5d] loss: %.3f %'
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
    print('Finished Training')
```

The output is shown below.

```
[1, 2000] loss: 2.192
[1, 4000] loss: 1.882
[1, 6000] loss: 1.698
[1, 8000] loss: 1.564
[1, 10000] loss: 1.513
[1, 12000] loss: 1.464
[2, 2000] loss: 1.391
[2, 4000] loss: 1.363
[2, 6000] loss: 1.331
[2, 8000] loss: 1.296
[2, 10000] loss: 1.302
[2, 12000] loss: 1.287
Finished Training
```

Let's quickly save our trained model:

```
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

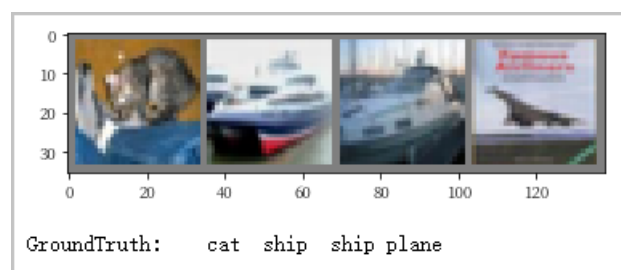
For more details on saving PyTorch models see [Serialization semantics](#).

5. Test the network on the test data.


We have trained the network for 2 passes over the training dataset. But we need to check if the network has learnt anything at all. We will check this by predicting the class label that the neural network outputs, and checking it against the ground-truth. If the prediction is correct, we add the sample to the list of correct predictions.

Okay, first step. Let us display an image from the test set to get familiar.

```
dataiter = iter(testloader)
images, labels = dataiter.next()
# print images.
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
```



Next, let's load back in our saved model.

 **Note** Saving and re-loading the model wasn't necessary here, we only did it to illustrate how to do so.

```
net = Net()
net.load_state_dict(torch.load(PATH))
```

Okay, now let us see what the neural network thinks these examples above are:

```
outputs = net(images)
```

The outputs are energies for the 10 classes. The higher the energy for a class, the more the network thinks that the image is of the particular class. So, let's get the index of the highest energy:

```
_, predicted = torch.max(outputs, 1)
print('Predicted: ', ''.join('%5s' % classes[predicted[j]]
                              for j in range(4)))
```

The output is shown below.

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

The results seem pretty good.

Let us look at how the network performs on the whole dataset.

```
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))
```

The output is shown below.

```
Accuracy of the network on the 10000 test images: 55 %
```

That looks way better than chance, which is 10% accuracy (randomly picking a class out of 10 classes). Seems like the network learnt something.

Hmmm, what are the classes that performed well, and the classes that did not perform well:

```
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]))
```

The output is shown below.

```
Accuracy of plane : 75 %  
Accuracy of car : 77 %  
Accuracy of bird : 35 %  
Accuracy of cat : 29 %  
Accuracy of deer : 42 %  
Accuracy of dog : 52 %  
Accuracy of frog : 46 %  
Accuracy of horse : 69 %  
Accuracy of ship : 61 %  
Accuracy of truck : 61 %
```

4.3. Use RNN to recognize surnames

This topic describes how to use a Recurrent Neural Network (RNN) in Data Science Workshop (DSW) of Machine Learning Platform for AI to recognize surnames. RNN can predict which language that a person speaks by recognizing how the surname of the person is spelled.

Context

A character-level RNN reads words as a series of characters - outputting a prediction and "hidden state" at each step, feeding its previous hidden state into each next step. We take the final prediction to be the output, i.e. which class the word belongs to.

Specifically, we'll train on a few thousand surnames from 18 languages of origin, and predict which language a name is from based on the spelling:

```
$ python predict.py Hinton  
(-0.47) Scottish  
(-1.52) English  
(-3.57) Irish  
$ python predict.py Schmidhuber  
(-0.19) German  
(-2.48) Czech  
(-2.68) Dutch
```

Preparing the Data

Download the data and extract it to the current directory. Included in the data/names directory are 18 text files named as "[Language].txt". Each file contains a bunch of names, one name per line, mostly romanized (but we still need to convert from Unicode to ASCII).

We'll end up with a dictionary of lists of names per language, {language: [names ...]}. The generic variables "category" and "line" (for language and name in our case) are used for later extensibility. In [7]:

```

from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
def findFiles(path): return glob.glob(path)
print(findFiles('data/names/*.txt'))
import unicodedata
import string
all_letters = string.ascii_letters + ".,;"
n_letters = len(all_letters)
# Turn a Unicode string to plain ASCII, thanks to https://stackoverflow.com/a/518232/2809427
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters
    )
print(unicodeToAscii('Ślusàrski'))
# Build the category_lines dictionary, a list of names per language
category_lines = {}
all_categories = []
# Read a file and split into lines
def readLines(filename):
    lines = open(filename, encoding='utf-8').read().strip().split("\n")
    return [unicodeToAscii(line) for line in lines]
for filename in findFiles('data/names/*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
    all_categories.append(category)
    lines = readLines(filename)
    category_lines[category] = lines
n_categories = len(all_categories)

```

The output is shown below.

```

['data/names/Greek.txt', 'data/names/Korean.txt', 'data/names/English.txt', 'data/names/Russian.txt', 'data/
names/Japanese.txt', 'data/names/German.txt', 'data/names/Scottish.txt', 'data/names/Arabic.txt', 'data/na
mes/Czech.txt', 'data/names/Vietnamese.txt', 'data/names/Polish.txt', 'data/names/Portuguese.txt', 'data/n
ames/Italian.txt', 'data/names/Dutch.txt', 'data/names/Irish.txt', 'data/names/Spanish.txt', 'data/names/Chi
nese.txt', 'data/names/French.txt']
Slusarski

```

Now we have `category_lines` a dictionary mapping each category (language) to a list of lines (names). We also kept track of `all_categories` (just a list of languages) and `n_categories` for later reference.

```
print(category_lines['Italian'][:5])
```

The output is shown below.

```
['Abandonato', 'Abatangelo', 'Abatantuono', 'Abate', 'Abategiovanni']
```

Turning Names into Tensors

To represent a single letter, we use a "one-hot vector" of size `<1 x n_letters>`. A one-hot vector is filled with 0s except for a 1 at index of the current letter. e.g. `"b" = <0 1 0 0 ... >`. To make a word we join a bunch of those into a 2D matrix `<line_length x 1 x n_letters>`. That extra 1 dimension is because PyTorch assumes everything is in batches - we're just using a batch size of 1 here.

The output is shown below.

This RNN module (mostly copied from the PyTorch for Torch users tutorial, see example 2-Recurrent Net in [NN PACKAGE](#)) is just 2 linear layers which operate on an input and hidden state, with a LogSoftmax layer after the output.

```
import torch.nn as nn
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)
    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden
    def initHidden(self):
        return torch.zeros(1, self.hidden_size)
n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```

To run a step of this network we need to pass an input (in our case, the Tensor for the current letter) and a previous hidden state (which we initialize as zeros at first). We'll get back the output (probability of each language) and a next hidden state (which we keep for the next step).

```
input = letterToTensor('A')
hidden = torch.zeros(1, n_hidden)
output, next_hidden = rnn(input, hidden)
```

For the sake of efficiency we don't want to be creating a new Tensor for every step, so we will use `lineToTensor` instead of `letterToTensor` and use slices. This could be further optimized by pre-computing batches of Tensors.

```
input = lineToTensor('Albert')
hidden = torch.zeros(1, n_hidden)
output, next_hidden = rnn(input[0], hidden)
print(output)
```

The output is shown below.

```
tensor([[[-2.8313, -2.8603, -2.9229, -2.8841, -2.8769, -2.9459, -2.8800, -2.9424,
          -2.8405, -2.9202, -2.9026, -2.9292, -2.9909, -2.8454, -2.9096, -2.8061,
          -2.8472, -2.9105]], grad_fn=<LogSoftmaxBackward>])
```

As you can see the output is a `<1 x n_categories>` Tensor, where every item is the likelihood of that category (higher is more likely).

Training

1. Preparing for Training

Before going into training we should make a few helper functions. The first is to interpret the output of the network, which we know to be a likelihood of each category. We can use `Tensor.topk` to get the index of the greatest value:

```
def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i[0].item()
    return all_categories[category_i], category_i
print(categoryFromOutput(output))
```

The output is shown below.

```
('Spanish', 15)
```

We will also want a quick way to get a training example (a name and its language):

```
import random
def randomChoice(l):
    return l[random.randint(0, len(l) - 1)]
def randomTrainingExample():
    category = randomChoice(all_categories)
    line = randomChoice(category_lines[category])
    category_tensor = torch.tensor([all_categories.index(category)], dtype=torch.long)
    line_tensor = lineToTensor(line)
    return category, line, category_tensor, line_tensor
for i in range(10):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    print('category =', category, '/ line =', line)
```

The output is shown below.

```
category = Arabic / line = Fakhoury
category = Vietnamese / line = Bui
category = Czech / line = Buchta
category = Arabic / line = Basara
category = Dutch / line = Sevriens
category = Czech / line = Cerda
category = Russian / line = Chajengin
category = Vietnamese / line = Vuong
category = Russian / line = Davydov
category = Dutch / line = Simonis
```

2. Training the Network

Now all it takes to train this network is show it a bunch of examples, have it make guesses, and tell it if it's wrong.

For the loss function `nn.NLLLoss` is appropriate, since the last layer of the RNN is `nn.LogSoftmax`.

```
criterion = nn.NLLLoss()
```

Each loop of training will:

- i. Create input and target tensors
- ii. Create a zeroed initial hidden state
- iii. Read each letter in and
- iv. Keep hidden state for next letter
- v. Compare final output to target

- vi. Back-propagate
- vii. Return the output and loss

```
learning_rate = 0.005 # If you set this too high, it might explode. If too low, it might not learn
def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()
    rnn.zero_grad()
    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)
    loss = criterion(output, category_tensor)
    loss.backward()
    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)
    return output, loss.item()
```

Now we just have to run that with a bunch of examples. Since the `train` function returns both the output and loss we can print its guesses and also keep track of loss for plotting. Since there are 1000s of examples we print only every `print_every` examples, and take an average of the loss.

```
import time
import math
n_iters = 100000
print_every = 5000
plot_every = 1000
# Keep track of losses for plotting
current_loss = 0
all_losses = []
def timeSince(since):
    now = time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)
start = time.time()
for iter in range(1, n_iters + 1):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output, loss = train(category_tensor, line_tensor)
    current_loss += loss
    # Print iter number, loss, name and guess
    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = '✓' if guess == category else 'X (%s)' % category
        print('%d %d%% (%s) %.4f %s / %s %s' % (iter, iter / n_iters * 100, timeSince(start), loss, line, guess, correct))
    # Add current loss avg to list of losses
    if iter % plot_every == 0:
        all_losses.append(current_loss / plot_every)
        current_loss = 0
```

The output is shown below.

```

5000 5% (0m 11s) 2.8952 Kinnaird / Spanish X (English)
10000 10% (0m 23s) 2.1054 Wojewodka / Russian X (Polish)
15000 15% (0m 35s) 0.7664 Rudaski / Polish ✓
20000 20% (0m 46s) 0.9732 Yee / Chinese ✓
25000 25% (0m 58s) 1.7527 Alesio / Portuguese X (Italian)
30000 30% (1m 10s) 0.3432 Shadid / Arabic ✓
35000 35% (1m 22s) 0.1572 Bartalotti / Italian ✓
40000 40% (1m 34s) 1.0907 Saliba / Arabic ✓
45000 45% (1m 46s) 0.1409 O'Connell / Irish ✓
50000 50% (1m 57s) 3.0025 Martell / Scottish X (German)
55000 55% (2m 9s) 2.6584 Atalian / Irish X (Russian)
60000 60% (2m 21s) 1.9061 Tasse / Japanese X (French)
65000 65% (2m 32s) 0.4886 Fernandez / Spanish ✓
70000 70% (2m 44s) 0.4260 Acerbi / Italian ✓
75000 75% (2m 56s) 2.0236 Longworth / Scottish X (English)
80000 80% (3m 8s) 2.4320 Oquendo / Italian X (Spanish)
85000 85% (3m 19s) 1.0514 Pesek / Czech ✓
90000 90% (3m 31s) 1.5673 Holzer / German ✓
95000 95% (3m 43s) 2.7059 Martin / Arabic X (French)
100000 100% (3m 55s) 2.2362 Rosenfeld / English X (German)

```

Plotting the Results

Plotting the historical loss from `all_losses` shows the network learning:

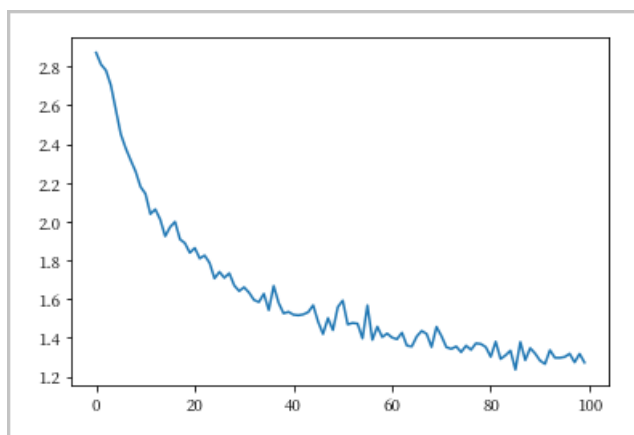
```

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
plt.figure()
plt.plot(all_losses)

```

The output is shown below.

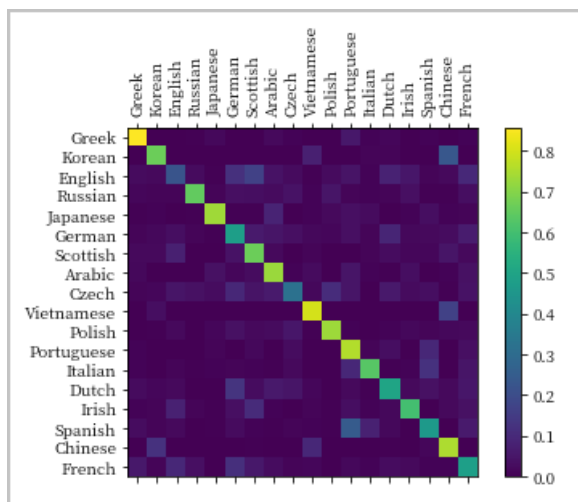
```
[<matplotlib.lines.Line2D at 0x7f3ce43b6e80>]
```



Evaluating the Results

To see how well the network performs on different categories, we will create a confusion matrix, indicating for every actual language (rows) which language the network guesses (columns). To calculate the confusion matrix a bunch of samples are run through the network with `evaluate()`, which is the same as `train()` minus the `backprop`.

```
# Keep track of correct guesses in a confusion matrix
confusion = torch.zeros(n_categories, n_categories)
n_confusion = 10000
# Just return an output given a line
def evaluate(line_tensor):
    hidden = rnn.initHidden()
    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)
    return output
# Go through a bunch of examples and record which are correctly guessed
for i in range(n_confusion):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output = evaluate(line_tensor)
    guess, guess_i = categoryFromOutput(output)
    category_i = all_categories.index(category)
    confusion[category_i][guess_i] += 1
# Normalize by dividing every row by its sum
for i in range(n_categories):
    confusion[i] = confusion[i] / confusion[i].sum()
# Set up plot
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(confusion.numpy())
fig.colorbar(cax)
# Set up axes
ax.set_xticklabels([''] + all_categories, rotation=90)
ax.set_yticklabels([''] + all_categories)
# Force label at every tick
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
# sphinx_gallery_thumbnail_number = 2
plt.show()
```



You can pick out bright spots off the main axis that show which languages it guesses incorrectly, e.g. Chinese for Korean, and Spanish for Italian. It seems to do very well with Greek, and very poorly with English (perhaps because of overlap with other languages).

Running on User Input

```
def predict(input_line, n_predictions=3):
    print('\n> %s' % input_line)
    with torch.no_grad():
        output = evaluate(lineToTensor(input_line))
        # Get top N categories
        topv, topi = output.topk(n_predictions, 1, True)
        predictions = []
        for i in range(n_predictions):
            value = topv[0][i].item()
            category_index = topi[0][i].item()
            print('{0.2f} %s' % (value, all_categories[category_index]))
            predictions.append([value, all_categories[category_index]])
    predict('Dovesky')
    predict('Jackson')
    predict('Satoshi')
```

The output is shown below.

```
> Dovesky
(-0.30) Russian
(-1.76) Czech
(-3.54) English
> Jackson
(-0.07) Scottish
(-3.31) English
(-4.89) Russian
> Satoshi
(-0.84) Japanese
(-1.88) Italian
(-2.12) Arabic
```

```
predict('yuze')
```

The output is shown below.

```
> yuze
(-1.61) Japanese
(-1.64) French
(-2.11) English
```

The final versions of the scripts is in [the Practical PyTorch repo](#), split the above code into a few files:

- *data.py* (loads files)
- *model.py* (defines the RNN)
- *train.py* (runs training)

Run *train.py* to train and save the network.

- `predict.py` (runs `predict()` with command line arguments)

Run `predict.py` with a name to view predictions:

```
$ python predict.py Hazaki
(-0.42) Japanese
(-1.39) Polish
(-3.51) Czech
```

- `server.py` (serve prediction as a JSON API with `bottle.py`)

4.4. Use EasyVision to detect targets

Machine Learning Platform for AI (PAI) provides EasyVision, which is an enhanced algorithm framework for visual intelligence. EasyVision provides a variety of model training and prediction capabilities. You can use EasyVision to train and apply computer vision (CV) models for your CV applications. This topic describes how to use EasyVision in Data Science Workshop (DSW) of PAI.

Prerequisites

A development environment with the following software versions is prepared:

- Python 2.7, or Python 3.4 or later
- TensorFlow 1.8 or later, or PAI-TensorFlow

Step 1: Prepare data

1. Use one of the following methods to download the Pascal dataset:

```
# Method 1: Use osscmd.
osscmd downloadallobject oss://pai-vision-data-hz/data/voc0712_tfrecord/ data/voc0712_tfrecord --h
ost=oss-cn-zhangjiakou.aliyuncs.com
# Method 2: Use ossutil. To use ossutil to download the Pascal dataset, you must set the host paramete
r to oss-cn-zhangjiakou.aliyuncs.com in the configuration file.
ossutil cp -r oss://pai-vision-data-hz/data/voc0712_tfrecord/ data/voc0712_tfrecord
```

2. Download the ResNet50 pre-trained model.

```
mkdir -p pretrained_models/
ossutil cp -r oss://pai-vision-data-hz/pretrained_models/resnet_v1d_50/ pretrained_models/resnet_v1
d_50
```

Step 2: Start a training task

- Single-machine mode

```
import easy_vision
easy_vision.train_and_evaluate(easy_vision.RFCN_SAMPLE_CONFIG)
```

In the training process, the model is evaluated every 5,000 rounds of training.

- Multi-machine mode

Use multiple servers to train the model. Make sure that each server has at least two GPUs. In multi-machine mode, you must start the following child processes:

- `ps`: the parameter server.

- master: the master that writes summaries, saves checkpoints, and periodically evaluates the model.
- worker: the worker that processes specific data.

Run the following script to start a training task.

```
#-*- encoding:utf-8 -*-
import multiprocessing
import sys
import os
import easy_vision
import json
import logging
import subprocess
import time
# train config under distributed settings
config=easy_vision.RFCN_DISTRIBUTE_SAMPLE_CONFIG
# The configuration of the cluster.
TF_CONFIG={'cluster':{
    'ps': ['localhost:12921'],
    'master': ['localhost:12922'],
    'worker': ['localhost:12923']
}}
def job(task, gpu):
    task_name = task['type']
    # redirect python log and tf log to log_file_name
    # [logs/master.log, logs/worker.log, logs/ps.log]
    log_file_name = "logs/%s.log" % task_name
    TF_CONFIG['task'] = task
    os.environ['TF_CONFIG'] = json.dumps(TF_CONFIG)
    os.environ['CUDA_VISIBLE_DEVICES'] = gpu
    train_cmd = 'python -m easy_vision.python.train_eval --pipeline_config_path %s' % config
    logging.info('%s > %s 2>&1 ' % (train_cmd, log_file_name))
    with open(log_file_name, 'w') as lfile:
        return subprocess.Popen(train_cmd.split(' '), stdout= lfile, stderr=subprocess.STDOUT)
if __name__ == '__main__':
    procs = {}
    # start ps job on cpu
    task = {'type':'ps', 'index':0}
    procs['ps'] = job(task, "")
    # start master job on gpu 0
    task = {'type':'master', 'index':0}
    procs['master'] = job(task, '0')
    # start worker job on gpu 1
    task = {'type':'worker', 'index':0}
    procs['worker'] = job(task, '1')
    num_worker = 2
    for k, proc in procs.items():
        logging.info('%s pid: %d' % (k, proc.pid))
    task_failed = None
    task_finish_cnt = 0
    task_has_finished = {k:False for k in procs.keys()}
    while True:
        for k, proc in procs.items():
```

```

if proc.poll() is None:
    if task_failed is not None:
        logging.error('task %s failed, %s quit' % (task_failed, k))
        proc.terminate()
    if k != 'ps':
        task_has_finished[k] = True
        task_finish_cnt += 1
        logging.info('task_finish_cnt %d' % task_finish_cnt)
    else:
        if not task_has_finished[k]:
            #process quit by itself
            if k != 'ps':
                task_finish_cnt += 1
                task_has_finished[k] = True
            logging.info('task_finish_cnt %d' % task_finish_cnt)
        if proc.returncode != 0:
            logging.error('%s failed' % k)
            task_failed = k
        else:
            logging.info('%s run successfully' % k)
if task_finish_cnt >= num_worker:
    break
time.sleep(1)

```

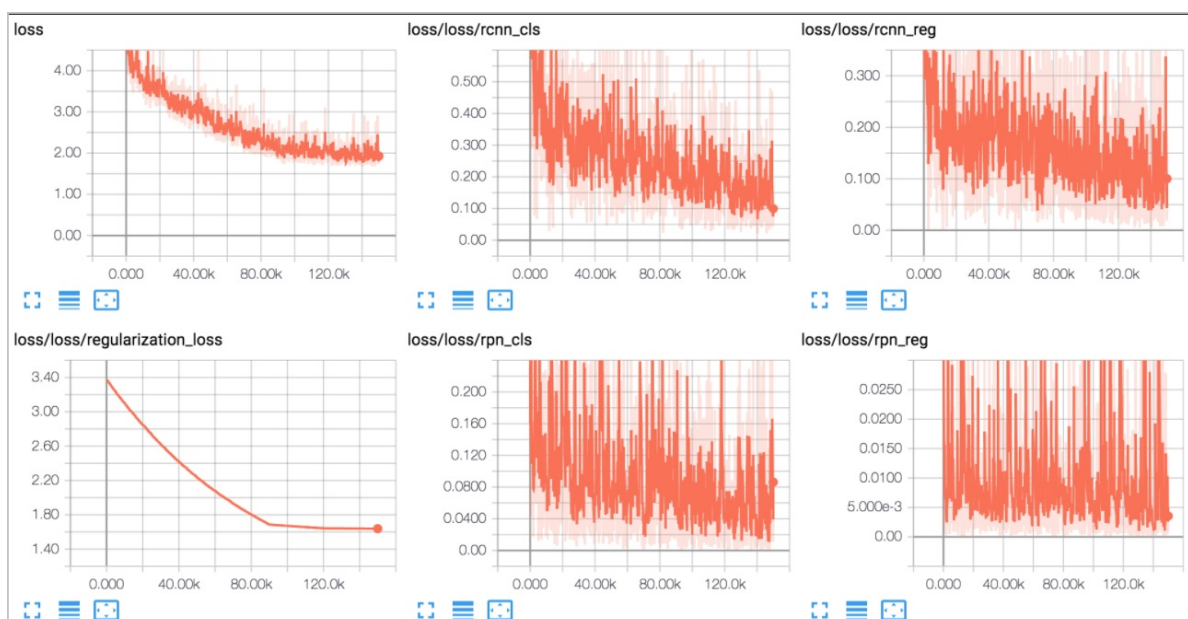
Step 3: Use TensorBoard to monitor the training task

The checkpoints and event files of the model are saved in the *pascal_resnet50_rfcn_model* directory. You can run the following command in TensorBoard to view the loss and mean average precision (mAP) of the training:

```
tensorboard --port 6006 --logdir pascal_resnet50_rfcn_model [ --host 0.0.0.0 ]
```

In TensorBoard, you can view the following information:

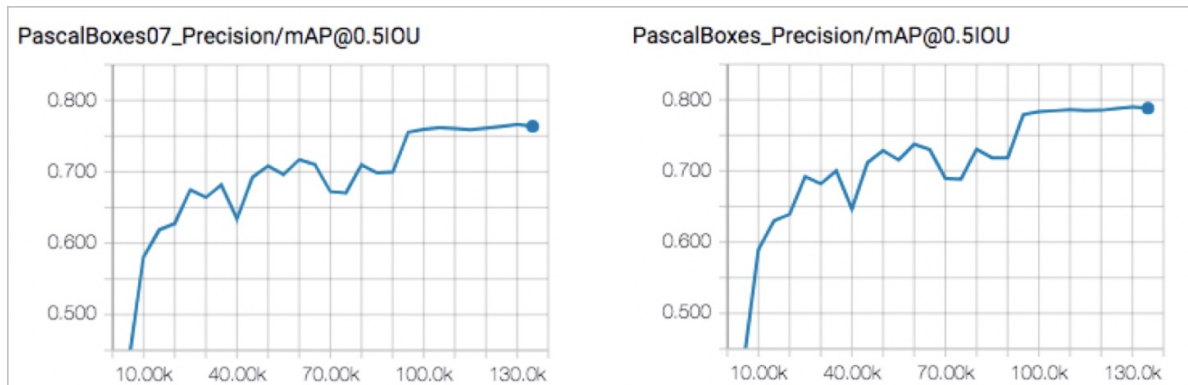
- Training loss



TensorBoard provides the following metrics about the training loss:

- loss: the total loss of the training.
- loss/loss/rcnn_cls: the classification loss.
- loss/loss/rcnn_reg: the regression loss.
- loss/loss/regularization_loss: the regularization loss.
- loss/loss/rpn_cls: the classification loss of region proposal network (RPN).
- loss/loss/rpn_reg: the regression loss of RPN.

- Test mAP



In the preceding figure, PascalBoxes07 and PascalBoxes are used as metrics to calculate the test mAP. PascalBoxes07 is commonly used in studies.

Step 4: Test and evaluate the model

After the training task is completed, you can test and evaluate the trained model.

- Use other datasets to test the model. Then, check the detection result of each image.

```
import easy_vision
test_filelist = 'path/to/filelist.txt' # each line is a image file path
detect_results = easy_vision.predict(easy_vision.RFCN_SAMPLE_CONFIG, test_filelist=test_filelist)
```

The detection result of each image is returned in the `detect_results` parameter in the format of `[detection_boxes, box_probability, box_class]`. In the format, `detection_boxes` and `box_class` indicate the location and category of the detected object. `box_probability` indicates the confidence level of the detection result.

- Evaluate the trained model.

```
import easy_vision
eval_metrics = easy_vision.evaluate(easy_vision.RFCN_SAMPLE_CONFIG)
```

The `eval_metrics` parameter indicates evaluation metrics, including PascalBoxes07, PascalBoxes, `global_step`, and the following loss metrics: `loss`, `loss/loss/rcnn_cls`, `loss/loss/rcnn_reg`, `loss/loss/rpn_cls`, `loss/loss/rpn_reg`, and `loss/loss/total_loss`. The following examples show the metrics:

- PascalBoxes07 Metric

```
PascalBoxes07_PerformanceByCategory/AP@0.5IOU/aeroplane = 0.74028647
PascalBoxes07_PerformanceByCategory/AP@0.5IOU/bicycle = 0.77216494
.....
PascalBoxes07_PerformanceByCategory/AP@0.5IOU/train = 0.771075
PascalBoxes07_PerformanceByCategory/AP@0.5IOU/tvmonitor = 0.70221454
PascalBoxes07_Precision/mAP@0.5IOU = 0.6975172
```

- PascalBoxes Metric

```
PascalBoxes_PerformanceByCategory/AP@0.5IOU/aeroplane = 0.7697732
PascalBoxes_PerformanceByCategory/AP@0.5IOU/bicycle = 0.80088705
.....
PascalBoxes_PerformanceByCategory/AP@0.5IOU/train = 0.8002225
PascalBoxes_PerformanceByCategory/AP@0.5IOU/tvmonitor = 0.72775906
PascalBoxes_Precision/mAP@0.5IOU = 0.7182514
```

- global_step and loss

```
global_step = 75000
loss = 0.51076376
loss/loss/rcnn_cls = 0.23392382
loss/loss/rcnn_reg = 0.12589474
loss/loss/rpn_cls = 0.13748208
loss/loss/rpn_reg = 0.013463326
loss/loss/total_loss = 0.51076376
```

Step 5: Export the model

Run the following script to export the model as a SavedModel file:

```
import easy_vision
easy_vision.export(export_dir, pipeline_config_path, checkpoint_path)
```

After you run the preceding script, a model directory is created in the `export_dir` directory. The name of the model directory contains the UNIX timestamp that indicates the time when the directory is created. All checkpoints of the model are exported to a SavedModel file in the model directory.

Step 6: Evaluate the SavedModel file

Run the following script to evaluate the exported SavedModel file. All metrics of the model are contained in the evaluation result file and logs.

```
from easy_vision.python.main import predictor_evaluate
predictor_evaluate(predictor_eval_config)
```

In the preceding code, `predictor_eval_config` specifies the .proto file that is used for the evaluation. For more information, see [Protocol Documentation](#). You can also use the following files for evaluation:

- `detector_eval.config` for target detection
- `text_detector_eval.config` for text detection
- `text_recognizer_eval.config` for text recognition
- `text_spotter_eval.config` for end-to-end text recognition

Step 7: Deploy the model as a service

Save the SavedModel file in Object Storage Service (OSS) and use the file to deploy a service in Elastic Algorithm Service (EAS). For more information, see [Create a service](#).