What are field-programmable gate arrays (FPGA) and how to deploy

06/03/2020 • 10 minutes to read • +5

In this article

Prerequisites
What are FPGAs
Deploy models on FPGAs
1. Define the TensorFlow model
2. Convert the model
3. Containerize and deploy the model
4. Consume the deployed model
Clean up resources
Secure FPGA web services
Next steps

APPLIES TO: Basic edition Enterprise edition (Upgrade to Enterprise edition)

This article provides an introduction to field-programmable gate arrays (FPGA), and shows you how to deploy your models using Azure Machine Learning to an Azure FPGA.

Prerequisites

- An Azure subscription. If you do not have one, you will need to create a pay-as-you-go account (free Azure accounts are not eligible for FPGA quota).
- Azure CLI
- FPGA quota. Use the Azure CLI to check whether you have quota:

Azure CLI | Copy | Try It

```
az vm list-usage --location "eastus" -o table --query "[?localName=='Standard PBS Family vCPUs']"
```

> **Tip**
>
> The other possible locations are `southeastasia`, `westeurope`, and `westus2`.

The command returns text similar to the following:

```text
CurrentValue    Limit    LocalName
-------------   -------  -------------------------
0               6        Standard PBS Family vCPUs
```

Make sure you have at least 6 vCPUs under **CurrentValue**.

If you do not have quota, then submit a request at https://aka.ms/accelerateAI.

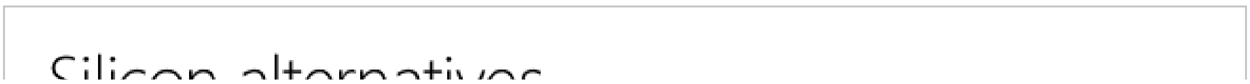
- An Azure Machine Learning workspace and the Azure Machine Learning SDK for Python installed. For more information, see Create a workspace.
- The Python SDK for hardware-accelerated models:

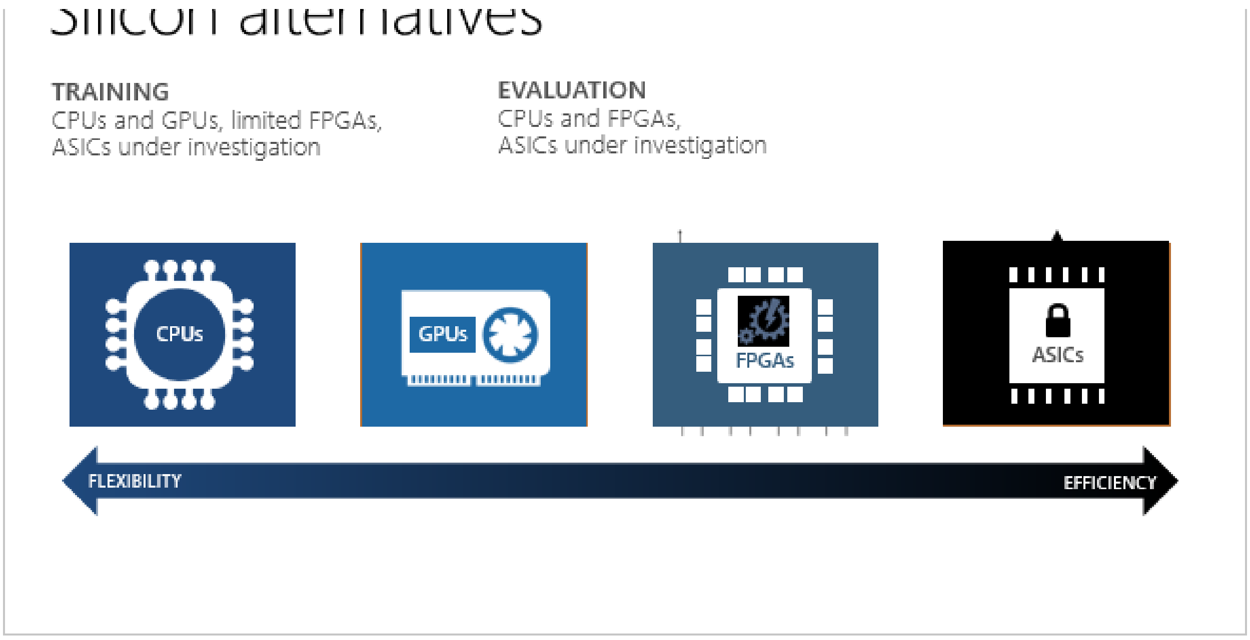
```Bash
pip install --upgrade azureml-accel-models[cpu]
```

What are FPGAs

FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects. The interconnects allow these blocks to be configured in various ways after manufacturing. Compared to other chips, FPGAs provide a combination of programmability and performance.

The following diagram and table show how FPGAs compare to other processors.





Processor	Abbreviation	Description
Application-specific integrated circuits	ASICs	Custom circuits, such as Google's TensorFlow Processor Units (TPU), provide the highest efficiency. They can't be reconfigured as your needs change.
Field-programmable gate arrays	FPGAs	FPGAs, such as those available on Azure, provide performance close to ASICs. They are also flexible and reconfigurable over time, to implement new logic.
Graphics processing units	GPUs	A popular choice for AI computations. GPUs offer parallel processing capabilities, making it faster at image rendering than CPUs.
Central processing units	CPUs	General-purpose processors, the performance of which isn't ideal for graphics and video processing.

FPGAs on Azure are based on Intel's FPGA devices, which data scientists and developers use to accelerate real-time AI calculations. This FPGA-enabled architecture offers performance, flexibility, and scale, and is available on Azure.

FPGAs make it possible to achieve low latency for real-time inference (or model scoring) requests. Asynchronous requests (batching) aren't needed. Batching can cause latency, because more data needs to be processed. Implementations of neural processing units don't require batching; therefore the latency can be many times lower, compared to CPU and GPU processors.

Reconfigurable power

You can reconfigure FPGAs for different types of machine learning models. This flexibility makes it easier to accelerate the applications based on the most optimal numerical precision and memory model being used. Because FPGAs are reconfigurable, you can stay current with the requirements of rapidly changing AI algorithms.

FPGA support in Azure

Microsoft Azure is the world's largest cloud investment in FPGAs. Microsoft uses FPGAs for DNN evaluation, Bing search ranking, and software defined networking (SDN) acceleration to reduce latency, while freeing CPUs for other tasks.

Azure FPGAs are integrated with Azure Machine Learning. Azure can parallelize pre-trained deep neural networks (DNN) across FPGAs to scale out your service. The DNNs can be pre-trained, as a deep featurizer for transfer learning, or fine-tuned with updated weights.

FPGAs on Azure supports:

- Image classification and recognition scenarios
- TensorFlow deployment (requires Tensorflow 1.x)
- Intel FPGA hardware

These DNN models are currently available:

- ResNet 50
- ResNet 152
- DenseNet-121
- VGG-16
- SSD-VGG

FPGAs are available in these Azure regions:

- East US
- Southeast Asia
- West Europe
- West US 2

> ⓘ **Important**
>
> To optimize latency and throughput, your client sending data to the FPGA model should be in one of the regions above (the one you deployed the model to).

The **PBS Family of Azure VMs** contains Intel Arria 10 FPGAs. It will show as "Standard PBS Family vCPUs" when you check your Azure quota allocation. The PB6 VM has six

PBS Family vCPUs" when you check your Azure quota allocation. The PB6 VM has six vCPUs and one FPGA, and it will automatically be provisioned by Azure ML as part of deploying a model to an FPGA. It is only used with Azure ML, and it cannot run arbitrary bitstreams. For example, you will not be able to flash the FPGA with bitstreams to do encryption, encoding, etc.

Deploy models on FPGAs

You can deploy a model as a web service on FPGAs with Azure Machine Learning Hardware Accelerated Models. Using FPGAs provides ultra-low latency inference, even with a single batch size. Inference, or model scoring, is the phase where the deployed model is used for prediction, most commonly on production data.

Deploying a model to an FPGA involves the following steps:

- Define the TensorFlow model
- Convert the model to ONNX
- Deploy the model to the cloud or an edge device
- Consume the deployed model

In this sample, you create a TensorFlow graph to preprocess the input image, make it a featurizer using ResNet 50 on an FPGA, and then run the features through a classifier trained on the ImageNet data set. Then, the model is deployed to an AKS cluster.

1. Define the TensorFlow model

Use the Azure Machine Learning SDK for Python to create a service definition. A service definition is a file describing a pipeline of graphs (input, featurizer, and classifier) based on TensorFlow. The deployment command automatically compresses the definition and graphs into a ZIP file, and uploads the ZIP to Azure Blob storage. The DNN is already deployed to run on the FPGA.

1. Load Azure Machine Learning workspace

Python Copy

```python
import os
import tensorflow as tf

from azureml.core import Workspace

ws = Workspace.from_config()

print(ws.name, ws.resource_group, ws.location, ws.subscription_id,
sep='\n')
```

2. Preprocess image. The input to the web service is a JPEG image. The first step is to decode the JPEG image and preprocess it. The JPEG images are treated as strings and the result are tensors that will be the input to the ResNet 50 model.

Python

```python
# Input images as a two-dimensional tensor containing an arbitrary
number of images represented a strings
import azureml.accel.models.utils as utils
tf.reset_default_graph()

in_images = tf.placeholder(tf.string)
image_tensors = utils.preprocess_array(in_images)
print(image_tensors.shape)
```

3. Load featurizer. Initialize the model and download a TensorFlow checkpoint of the quantized version of ResNet50 to be used as a featurizer. You may replace "QuantizedResnet50" in the code snippet below with by importing other deep neural networks:

    - QuantizedResnet152
    - QuantizedVgg16
    - Densenet121

Python

```python
from azureml.accel.models import QuantizedResnet50
save_path = os.path.expanduser('~/models')
model_graph = QuantizedResnet50(save_path, is_frozen=True)
feature_tensor = model_graph.import_graph_def(image_tensors)
print(model_graph.version)
print(feature_tensor.name)
print(feature_tensor.shape)
```

4. Add a classifier. This classifier has been trained on the ImageNet data set. Examples for transfer learning and training your customized weights are available in the set of sample notebooks.

Python

```python
classifier_output = model_graph.get_default_classifier(feature_tensor)
print(classifier_output)
```

5. Save the model. Now that the preprocessor, ResNet 50 featurizer, and the classifier have been loaded, save the graph and associated variables as a model.

have been loaded, save the graph and associated variables as a model.

Python

```python
model_name = "resnet50"
model_save_path = os.path.join(save_path, model_name)
print("Saving model in {}".format(model_save_path))

with tf.Session() as sess:
    model_graph.restore_weights(sess)
    tf.saved_model.simple_save(sess, model_save_path,
                               inputs={'images': in_images},
                               outputs={'output_alias':
classifier_output})
```

6. Save input and output tensors. The input and output tensors that were created during the preprocessing and classifier steps will be needed for model conversion and inference.

Python

```python
input_tensors = in_images.name
output_tensors = classifier_output.name

print(input_tensors)
print(output_tensors)
```

Important

Save the input and output tensors because you will need them for model conversion and inference requests.

The available models and the corresponding default classifier output tensors are below, which is what you would use for inference if you used the default classifier.

- Resnet50, QuantizedResnet50

  Python

  ```python
  output_tensors = "classifier_1/resnet_v1_50/predictions/Softmax:0"
  ```

- Resnet152, QuantizedResnet152

  Python

  ```python
  output_tensors = "classifier/resnet_v1_152/predictions/Softmax:0"
  ```

- Densenet121, QuantizedDensenet121

Python Copy

```python
output_tensors = "classifier/densenet121/predictions/Softmax:0"
```

- Vgg16, QuantizedVgg16

Python Copy

```python
output_tensors = "classifier/vgg_16/fc8/squeezed:0"
```

- SsdVgg, QuantizedSsdVgg

Python Copy

```python
output_tensors = ['ssd_300_vgg/block4_box/Reshape_1:0',
'ssd_300_vgg/block7_box/Reshape_1:0',
'ssd_300_vgg/block8_box/Reshape_1:0',
'ssd_300_vgg/block9_box/Reshape_1:0',
'ssd_300_vgg/block10_box/Reshape_1:0',
'ssd_300_vgg/block11_box/Reshape_1:0',
'ssd_300_vgg/block4_box/Reshape:0',
'ssd_300_vgg/block7_box/Reshape:0',
'ssd_300_vgg/block8_box/Reshape:0',
'ssd_300_vgg/block9_box/Reshape:0',
'ssd_300_vgg/block10_box/Reshape:0',
'ssd_300_vgg/block11_box/Reshape:0']
```

2. Convert the model

Before deploying the model to FPGAs, you have to convert it to ONNX format.

1. Register the model by using the SDK with the ZIP file in Azure Blob storage. Adding tags and other metadata about the model helps you keep track of your trained models.

   Python Copy

   ```python
   from azureml.core.model import Model

   registered_model = Model.register(workspace=ws,
                                     model_path=model_save_path,
                                     model_name=model_name)

   print("Successfully registered: ", registered_model.name,
         registered_model.description, registered_model.version, sep='\t')
   ```

   If you've already registered a model and want to load it, you may retrieve it.

Python

```python
from azureml.core.model import Model
model_name = "resnet50"
# By default, the latest version is retrieved. You can specify the
version, i.e. version=1
registered_model = Model(ws, name="resnet50")
print(registered_model.name, registered_model.description,
      registered_model.version, sep='\t')
```

2. Convert the TensorFlow graph to the Open Neural Network Exchange format (ONNX). You will need to provide the names of the input and output tensors, and these names will be used by your client when you consume the web service.

Python

```python
from azureml.accel import AccelOnnxConverter

convert_request = AccelOnnxConverter.convert_tf_model(
    ws, registered_model, input_tensors, output_tensors)

# If it fails, you can run wait_for_completion again with
show_output=True.
convert_request.wait_for_completion(show_output=False)

# If the above call succeeded, get the converted model
converted_model = convert_request.result
print("\nSuccessfully converted: ", converted_model.name,
converted_model.url, converted_model.version,
      converted_model.id, converted_model.created_time, '\n')
```

3. Containerize and deploy the model

Create Docker image from the converted model and all dependencies. This Docker image can then be deployed and instantiated. Supported deployment targets include AKS in the cloud or an edge device such as Azure Data Box Edge. You can also add tags and descriptions for your registered Docker image.

Python

```python
from azureml.core.image import Image
from azureml.accel import AccelContainerImage

image_config = AccelContainerImage.image_configuration()
# Image name must be lowercase
image_name = "{}-image".format(model_name)

image = Image.create(name=image_name,
```

```
                     models=[converted_model],
                     image_config=image_config,
                     workspace=ws)
image.wait_for_creation(show_output=False)
```

List the images by tag and get the detailed logs for any debugging.

Python Copy

```python
for i in Image.list(workspace=ws):
    print('{}(v.{} [{}]) stored at {} with build log {}'.format(
        i.name, i.version, i.creation_state, i.image_location,
i.image_build_log_uri))
```

Deploy to AKS Cluster

1. To deploy your model as a high-scale production web service, use Azure Kubernetes Service (AKS). You can create a new one using the Azure Machine Learning SDK, CLI, or Azure Machine Learning studio.

   Python Copy

```python
from azureml.core.compute import AksCompute, ComputeTarget

# Specify the Standard_PB6s Azure VM and location. Values for location
may be "eastus", "southeastasia", "westeurope", or "westus2". If no
value is specified, the default is "eastus".
prov_config = AksCompute.provisioning_configuration(vm_size =
"Standard_PB6s",
                                                    agent_count = 1,
                                                    location =
"eastus")

aks_name = 'my-aks-cluster'
# Create the cluster
aks_target = ComputeTarget.create(workspace=ws,
                                  name=aks_name,

provisioning_configuration=prov_config)
```

   The AKS deployment may take around 15 minutes. Check to see if the deployment succeeded.

   Python Copy

```python
aks_target.wait_for_completion(show_output=True)
```

```
print(aks_target.provisioning_state)
print(aks_target.provisioning_errors)
```

2. Deploy the container to the AKS cluster.

Python Copy

```python
from azureml.core.webservice import Webservice, AksWebservice

# For this deployment, set the web service configuration without
enabling auto-scaling or authentication for testing
aks_config =
AksWebservice.deploy_configuration(autoscale_enabled=False,
                                                num_replicas=1,
                                                auth_enabled=False)

aks_service_name = 'my-aks-service'

aks_service = Webservice.deploy_from_image(workspace=ws,
                                           name=aks_service_name,
                                           image=image,

deployment_config=aks_config,

deployment_target=aks_target)
aks_service.wait_for_deployment(show_output=True)
```

Deploy to a local edge server

All Azure Data Box Edge devices contain an FPGA for running the model. Only one model can be running on the FPGA at one time. To run a different model, just deploy a new container. Instructions and sample code can be found in this Azure Sample.

4. Consume the deployed model

The Docker image supports gRPC and the TensorFlow Serving "predict" API. Use the sample client to call into the Docker image to get predictions from the model. Sample client code is available:

- Python
- C#

If you want to use TensorFlow Serving, you can download a sample client.

Python Copy

```python
# Using the grpc client in Azure ML Accelerated Models SDK package
from azureml.accel import PredictionClient

address = aks_service.scoring_uri
ssl_enabled = address.startswith("https")
address = address[address.find('/')+2:].strip('/')
port = 443 if ssl_enabled else 80

# Initialize Azure ML Accelerated Models client
client = PredictionClient(address=address,
                          port=port,
                          use_ssl=ssl_enabled,
                          service_name=aks_service.name)
```

Since this classifier was trained on the ImageNet data set, map the classes to human-readable labels.

Python

```python
import requests
classes_entries = requests.get(

"https://raw.githubusercontent.com/Lasagne/Recipes/master/examples/resnet50/imagenet_classes.txt").text.splitlines()

# Score image with input and output tensor names
results = client.score_file(path="./snowleopardgaze.jpg",
                            input_name=input_tensors,
                            outputs=output_tensors)

# map results [class_id] => [confidence]
results = enumerate(results)
# sort results by confidence
sorted_results = sorted(results, key=lambda x: x[1], reverse=True)
# print top 5 results
for top in sorted_results[:5]:
    print(classes_entries[top[0]], 'confidence:', top[1])
```

Clean up resources

Delete your web service, image, and model (must be done in this order since there are dependencies).

Python

```python
aks_service.delete()
aks_target.delete()
image.delete()
registered_model.delete()
```

```
converted_model.delete()
```

Secure FPGA web services

To secure your FPGA web services, see the Secure web services document.

Next steps

Check out these notebooks, videos, and blogs:

- Several sample notebooks
- Hyperscale hardware: ML at scale on top of Azure + FPGA: Build 2018 (video)
- Inside the Microsoft FPGA-based configurable cloud (video)
- Project Brainwave for real-time AI: project home page
- Automated optical inspection system
- Land cover mapping