**Learning TensorFlow: Using Convolutions with Complex Images**

**Overview**

The previous labs in this quest illustrate how an image classifier can be trained on simple datasets like Fashion MNIST. Fashion MNIST dataset has images of size 28x28, where the subject is centered. In this lab you'll go further and learn how an image classifier can be trained to recognize features in an image where the subject can be anywhere in the image! You'll train a classifier that identifies if the given image contains a horse or a human. The network will be trained to recognize features that help to determine which is which.

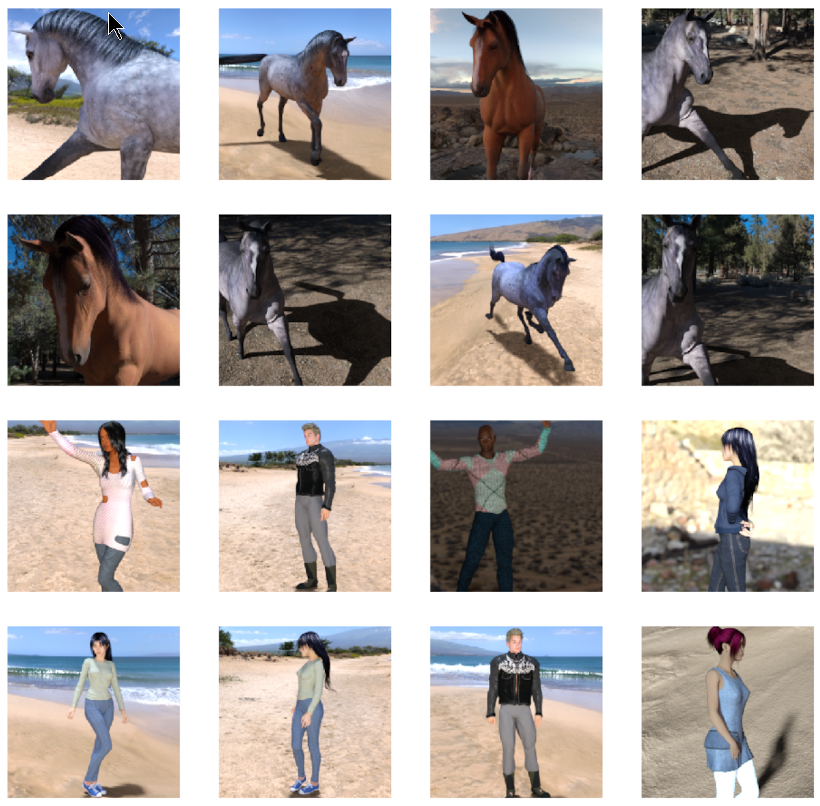
This tutorial demonstrates how to

1. Train a CNN model on Vertex AI using the SDK for Python
2. Deploy a custom image classification model for online prediction using Vertex AI

**Dataset**

You will train a neural network to classify images from a dataset called [Horses or Humans](https://www.tensorflow.org/datasets/catalog/horses_or_humans).

This dataset contains a large number of images of horses and humans. It has 1283 images of resolution of 300x300, as seen here:



In this notebook, you will use 1027 images to train the network and 256 images to evaluate how accurately the network learned to classify the images.

**Objective**

In this notebook, you will create a custom-trained CNN model from a Python script in a Docker container using the Vertex SDK for Python, and then do a prediction on the deployed model by sending data. Alternatively, you can create custom-trained models using gcloud command-line tool, or online using the Cloud Console.

The steps performed include:

* Create a Vertex AI custom job and train a CNN model.
* Deploy the Model resource to a serving Endpoint resource.
* Make a prediction.
* Undeploy the Model resource.

**Costs**

This tutorial uses billable components of Google Cloud (GCP):

* Vertex AI
* Cloud Storage

Learn about [Vertex AI pricing](https://cloud.google.com/vertex-ai/pricing) and [Cloud Storage pricing](https://cloud.google.com/storage/pricing), and use the [Pricing Calculator](https://cloud.google.com/products/calculator/) to generate a cost estimate based on your projected usage.

**Setting up the environment**

**Installation**

**Note:** Disregard the errors and warnings associated with the installation.

[1]:



**!**pip install **--**user google**-**cloud**-**aiplatform

**!**pip install **--**user protobuf**==**3.19.**\***

Requirement already satisfied: google-cloud-aiplatform in /opt/conda/lib/python3.7/site-packages (1.22.1)

Requirement already satisfied: shapely<2.0.0 in /opt/conda/lib/python3.7/site-packages (from google-cloud-aiplatform) (1.8.5.post1)

Requirement already satisfied: google-cloud-bigquery<4.0.0dev,>=1.15.0 in /opt/conda/lib/python3.7/site-packages (from google-cloud-aiplatform) (3.6.0)

Requirement already satisfied: google-cloud-storage<3.0.0dev,>=1.32.0 in /opt/conda/lib/python3.7/site-packages (from google-cloud-aiplatform) (2.7.0)

Collecting packaging<22.0.0dev,>=14.3

Downloading packaging-21.3-py3-none-any.whl (40 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.8/40.8 kB 2.2 MB/s eta 0:00:00

Collecting protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5

Downloading protobuf-4.22.1-cp37-abi3-manylinux2014\_x86\_64.whl (302 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.4/302.4 kB 9.7 MB/s eta 0:00:00

Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.0 in /opt/conda/lib/python3.7/site-packages (from google-cloud-aiplatform) (1.22.2)

Requirement already satisfied: google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0 in /opt/conda/lib/python3.7/site-packages (from google-cloud-aiplatform) (2.10.1)

Requirement already satisfied: google-cloud-resource-manager<3.0.0dev,>=1.3.3 in /opt/conda/lib/python3.7/site-packages (from google-cloud-aiplatform) (1.9.0)

Requirement already satisfied: google-auth<3.0dev,>=1.25.0 in /opt/conda/lib/python3.7/site-packages (from google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (2.16.2)

Requirement already satisfied: requests<3.0.0dev,>=2.18.0 in /opt/conda/lib/python3.7/site-packages (from google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (2.28.2)

Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.56.2 in /opt/conda/lib/python3.7/site-packages (from google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (1.58.0)

Requirement already satisfied: grpcio<2.0dev,>=1.33.2 in /opt/conda/lib/python3.7/site-packages (from google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (1.51.3)

Requirement already satisfied: grpcio-status<2.0dev,>=1.33.2 in /opt/conda/lib/python3.7/site-packages (from google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (1.48.2)

Requirement already satisfied: google-cloud-core<3.0.0dev,>=1.6.0 in /opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery<4.0.0dev,>=1.15.0->google-cloud-aiplatform) (2.3.2)

Requirement already satisfied: python-dateutil<3.0dev,>=2.7.2 in /opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery<4.0.0dev,>=1.15.0->google-cloud-aiplatform) (2.8.2)

Requirement already satisfied: google-resumable-media<3.0dev,>=0.6.0 in /opt/conda/lib/python3.7/site-packages (from google-cloud-bigquery<4.0.0dev,>=1.15.0->google-cloud-aiplatform) (2.4.1)

Collecting google-cloud-resource-manager<3.0.0dev,>=1.3.3

Downloading google\_cloud\_resource\_manager-1.8.1-py2.py3-none-any.whl (235 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.7/235.7 kB 21.6 MB/s eta 0:00:00

Downloading google\_cloud\_resource\_manager-1.8.0-py2.py3-none-any.whl (235 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.3/235.3 kB 22.4 MB/s eta 0:00:00

Downloading google\_cloud\_resource\_manager-1.7.0-py2.py3-none-any.whl (235 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.3/235.3 kB 22.3 MB/s eta 0:00:00

Downloading google\_cloud\_resource\_manager-1.6.3-py2.py3-none-any.whl (233 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 233.8/233.8 kB 23.0 MB/s eta 0:00:00

Requirement already satisfied: grpc-google-iam-v1<1.0.0dev,>=0.12.4 in /opt/conda/lib/python3.7/site-packages (from google-cloud-resource-manager<3.0.0dev,>=1.3.3->google-cloud-aiplatform) (0.12.6)

Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/python3.7/site-packages (from packaging<22.0.0dev,>=14.3->google-cloud-aiplatform) (3.0.9)

Requirement already satisfied: pyasn1-modules>=0.2.1 in /opt/conda/lib/python3.7/site-packages (from google-auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (0.2.8)

Requirement already satisfied: cachetools<6.0,>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from google-auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (5.3.0)

Requirement already satisfied: rsa<5,>=3.1.4 in /opt/conda/lib/python3.7/site-packages (from google-auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (4.9)

Requirement already satisfied: six>=1.9.0 in /opt/conda/lib/python3.7/site-packages (from google-auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (1.16.0)

Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in /opt/conda/lib/python3.7/site-packages (from google-resumable-media<3.0dev,>=0.6.0->google-cloud-bigquery<4.0.0dev,>=1.15.0->google-cloud-aiplatform) (1.5.0)

Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (2022.12.7)

Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (3.4)

Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (1.26.14)

Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (2.1.1)

Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /opt/conda/lib/python3.7/site-packages (from pyasn1-modules>=0.2.1->google-auth<3.0dev,>=1.25.0->google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform) (0.4.8)

Installing collected packages: protobuf, packaging, google-cloud-resource-manager

ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.

tensorflow 2.8.4 requires protobuf<3.20,>=3.9.2, but you have protobuf 4.22.1 which is incompatible.

tensorflow-transform 0.14.0 requires protobuf<4,>=3.7, but you have protobuf 4.22.1 which is incompatible.

tensorflow-metadata 0.14.0 requires protobuf<4,>=3.7, but you have protobuf 4.22.1 which is incompatible.

tensorboardx 2.6 requires protobuf<4,>=3.8.0, but you have protobuf 4.22.1 which is incompatible.

google-cloud-vision 3.4.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-videointelligence 1.16.3 requires protobuf<4.0.0dev, but you have protobuf 4.22.1 which is incompatible.

google-cloud-spanner 3.28.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-pubsub 2.15.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-language 1.3.2 requires protobuf<4.0.0dev, but you have protobuf 4.22.1 which is incompatible.

google-cloud-dlp 3.12.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-datastore 1.15.5 requires protobuf<4.0.0dev, but you have protobuf 4.22.1 which is incompatible.

google-cloud-bigtable 1.7.3 requires protobuf<4.0.0dev, but you have protobuf 4.22.1 which is incompatible.

google-cloud-artifact-registry 1.8.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

apache-beam 2.45.0 requires protobuf<4,>3.12.2, but you have protobuf 4.22.1 which is incompatible.

Successfully installed google-cloud-resource-manager-1.6.3 packaging-21.3 protobuf-4.22.1

Collecting protobuf==3.19.\*

Downloading protobuf-3.19.6-cp37-cp37m-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl (1.1 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 9.2 MB/s eta 0:00:00:00:010:01

Installing collected packages: protobuf

Attempting uninstall: protobuf

Found existing installation: protobuf 4.22.1

Uninstalling protobuf-4.22.1:

Successfully uninstalled protobuf-4.22.1

ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.

google-cloud-vision 3.4.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-spanner 3.28.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-pubsub 2.15.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-dlp 3.12.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-artifact-registry 1.8.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-api-core 2.10.1 requires protobuf<5.0.0dev,>=3.20.1, but you have protobuf 3.19.6 which is incompatible.

Successfully installed protobuf-3.19.6

[2]:



**!** pip3 install protobuf**==**3.20.1

Collecting protobuf==3.20.1

Downloading protobuf-3.20.1-cp37-cp37m-manylinux\_2\_5\_x86\_64.manylinux1\_x86\_64.whl (1.0 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 14.0 MB/s eta 0:00:0000:010:01

Installing collected packages: protobuf

Attempting uninstall: protobuf

Found existing installation: protobuf 3.19.6

Uninstalling protobuf-3.19.6:

Successfully uninstalled protobuf-3.19.6

ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.

google-cloud-resource-manager 1.6.3 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

tensorflow 2.8.4 requires protobuf<3.20,>=3.9.2, but you have protobuf 3.20.1 which is incompatible.

grpc-google-iam-v1 0.12.6 requires protobuf!=3.20.0,!=3.20.1,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

googleapis-common-protos 1.58.0 requires protobuf!=3.20.0,!=3.20.1,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

google-cloud-vision 3.4.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-vision 3.4.0 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

google-cloud-spanner 3.28.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-spanner 3.28.0 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

google-cloud-pubsub 2.15.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-pubsub 2.15.0 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

google-cloud-monitoring 2.14.1 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

google-cloud-dlp 3.12.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-dlp 3.12.0 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

google-cloud-bigquery 3.6.0 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

google-cloud-bigquery-storage 2.16.2 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

google-cloud-artifact-registry 1.8.0 requires google-api-core[grpc]!=2.0.\*,!=2.1.\*,!=2.10.\*,!=2.2.\*,!=2.3.\*,!=2.4.\*,!=2.5.\*,!=2.6.\*,!=2.7.\*,!=2.8.\*,!=2.9.\*,<3.0.0dev,>=1.34.0, but you have google-api-core 2.10.1 which is incompatible.

google-cloud-artifact-registry 1.8.0 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

google-cloud-aiplatform 1.22.1 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1 which is incompatible.

Successfully installed protobuf-3.20.1

**Restart the kernel**

Once you've installed everything, you need to restart the notebook kernel so it can find the packages.

[3]:



**import** os

​

**if** **not** os.getenv("IS\_TESTING"):

*# Automatically restart kernel after installs*

**import** IPython

​

app **=** IPython.Application.instance()

app.kernel.do\_shutdown(**True**)

**Set your project ID**

**If you don't know your project ID**, you may be able to get your project ID using gcloud.

[1]:



**import** os

​

PROJECT\_ID **=** ""

​

**if** **not** os.getenv("IS\_TESTING"):

*# Get your Google Cloud project ID from gcloud*

shell\_output**=!**gcloud config list **--**format 'value(core.project)' 2**>/**dev**/**null

PROJECT\_ID **=** shell\_output[0]

print("Project ID: ", PROJECT\_ID)

Project ID: qwiklabs-gcp-02-bd07dc32b9c1

**Timestamp**

If you are in a live tutorial session, you might be using a shared test account or project. To avoid name collisions between users on resources created, you create a timestamp for each instance session, and append it onto the name of resources you create in this tutorial.

[2]:



**from** datetime **import** datetime

​

TIMESTAMP **=** datetime.now().strftime("%Y%m%d%H%M%S")

**Create a Cloud Storage bucket**

**The following steps are required, regardless of your notebook environment.**

When you submit a training job using the Cloud SDK, you upload a Python package containing your training code to a Cloud Storage bucket. Vertex AI runs the code from this package. In this tutorial, Vertex AI also saves the trained model that results from your job in the same bucket. Using this model artifact, you can then create Vertex AI model and endpoint resources in order to serve online predictions.

Set the name of your Cloud Storage bucket below. It must be unique across all Cloud Storage buckets.

You may also change the REGION variable, which is used for operations throughout the rest of this notebook. Make sure to [choose a region where Vertex AI services are available](https://cloud.google.com/vertex-ai/docs/general/locations#available_regions). You may not use a Multi-Regional Storage bucket for training with Vertex AI.

[3]:



BUCKET\_NAME **=** "gs://[your-bucket-name]"

REGION **=** "us-central1" *# @param {type:"string"}*

[4]:



PROJECT\_ID

[4]:

'qwiklabs-gcp-02-bd07dc32b9c1'

[5]:



**if** BUCKET\_NAME **==** "" **or** BUCKET\_NAME **is** **None** **or** BUCKET\_NAME **==** "gs://[your-bucket-name]":

BUCKET\_NAME **=** "gs://" **+** PROJECT\_ID

**Only if your bucket doesn't already exist**: Run the following cells to create your Cloud Storage bucket.

[6]:



BUCKET\_NAME

[6]:

'gs://qwiklabs-gcp-02-bd07dc32b9c1'

[7]:



**!** gsutil mb **-**l $REGION $BUCKET\_NAME

Creating gs://qwiklabs-gcp-02-bd07dc32b9c1/...

**Check progress**

Go back to lab manual and Click *Check my progress* under **Create a Cloud Storage Bucket** section to verify that the bucket is created.

**Set up variables**

Next, set up some variables used throughout the tutorial.

**Import Vertex SDK for Python**

Import the Vertex SDK for Python into your Python environment and initialize it.

[8]:



**import** os

**import** sys

​

**from** google.cloud **import** aiplatform

**from** google.cloud.aiplatform **import** gapic **as** aip

​

aiplatform.init(project**=**PROJECT\_ID, location**=**REGION, staging\_bucket**=**BUCKET\_NAME)

**Set hardware accelerators**

Here to run a container image on a CPU, we set the variables TRAIN\_GPU/TRAIN\_NGPU and DEPLOY\_GPU/DEPLOY\_NGPU to (None, None) since this notebook is meant to be run in a Qwiklab environment where GPUs cannot be provisioned.

Note: If you happen to be running this notebook from your personal GCP account, set the variables TRAIN\_GPU/TRAIN\_NGPU and DEPLOY\_GPU/DEPLOY\_NGPU to use a container image supporting a GPU and the number of GPUs allocated to the virtual machine (VM) instance. For example, to use a GPU container image with 4 Nvidia Tesla K80 GPUs allocated to each VM, you would specify:

(aip.AcceleratorType.NVIDIA\_TESLA\_K80, 4)

See the [locations where accelerators are available](https://cloud.google.com/vertex-ai/docs/general/locations#accelerators).

[9]:



TRAIN\_GPU, TRAIN\_NGPU **=** (**None**, **None**)

DEPLOY\_GPU, DEPLOY\_NGPU **=** (**None**, **None**)

**Set pre-built containers**

There are two ways you can train a custom model using a container image:

* **Use a Google Cloud prebuilt container**. If you use a prebuilt container, you will additionally specify a Python package to install into the container image. This Python package contains your code for training a custom model.
* **Use your own custom container image**. If you use your own container, the container needs to contain your code for training a custom model.

Here you will use pre-built containers provided by Vertex AI to run training and prediction.

For the latest list, see [Pre-built containers for training](https://cloud.google.com/vertex-ai/docs/training/pre-built-containers) and [Pre-built containers for prediction](https://cloud.google.com/vertex-ai/docs/predictions/pre-built-containers)

[10]:



TRAIN\_VERSION **=** "tf-cpu.2-8"

DEPLOY\_VERSION **=** "tf2-cpu.2-8"

​

TRAIN\_IMAGE **=** "us-docker.pkg.dev/vertex-ai/training/{}:latest".format(TRAIN\_VERSION)

DEPLOY\_IMAGE **=** "us-docker.pkg.dev/vertex-ai/prediction/{}:latest".format(DEPLOY\_VERSION)

​

print("Training:", TRAIN\_IMAGE, TRAIN\_GPU, TRAIN\_NGPU)

print("Deployment:", DEPLOY\_IMAGE, DEPLOY\_GPU, DEPLOY\_NGPU)

Training: us-docker.pkg.dev/vertex-ai/training/tf-cpu.2-8:latest None None

Deployment: us-docker.pkg.dev/vertex-ai/prediction/tf2-cpu.2-8:latest None None

**Set machine types**

Next, set the machine types to use for training and prediction.

* Set the variables TRAIN\_COMPUTE and DEPLOY\_COMPUTE to configure your compute resources for training and prediction.
* machine type
  + n1-standard: 3.75GB of memory per vCPU
  + n1-highmem: 6.5GB of memory per vCPU
  + n1-highcpu: 0.9 GB of memory per vCPU
* vCPUs: number of [2, 4, 8, 16, 32, 64, 96 ]

*Note: The following is not supported for training:*

* standard: 2 vCPUs
* highcpu: 2, 4 and 8 vCPUs

*Note: You may also use n2 and e2 machine types for training and deployment, but they do not support GPUs*.

[11]:



MACHINE\_TYPE **=** "n1-standard"

​

VCPU **=** "4"

TRAIN\_COMPUTE **=** MACHINE\_TYPE **+** "-" **+** VCPU

print("Train machine type", TRAIN\_COMPUTE)

​

MACHINE\_TYPE **=** "n1-standard"

​

VCPU **=** "4"

DEPLOY\_COMPUTE **=** MACHINE\_TYPE **+** "-" **+** VCPU

print("Deploy machine type", DEPLOY\_COMPUTE)

Train machine type n1-standard-4

Deploy machine type n1-standard-4

**Design and Train the Convolutional Neural Network**

Now you are ready to design and train your own custom-trained CNN model with Horses or Humans data.

**Training script**

In the next cell, you will write the contents of the training script, task.py. In summary:

* Get the directory where to save the model artifacts from the environment variable AIP\_MODEL\_DIR. This variable is set by the training service.
* Loads Horses or Humans dataset from TF Datasets (tfds).
* Builds a CNN model using tf.keras model API. (Please see code comments for details about the layers in the CNN).
* Compiles the model (compile()).
* Trains the model (fit()) with epochs.
* Saves the trained model (save(MODEL\_DIR)) to the specified model directory.

[12]:



**%%**writefile task.py

*# Training Horses vs Humans using CNN*

​

**import** tensorflow\_datasets **as** tfds

**import** tensorflow **as** tf

**from** tensorflow.python.client **import** device\_lib

**import** argparse

**import** os

**import** sys

tfds.disable\_progress\_bar()

​

parser **=** argparse.ArgumentParser()

​

parser.add\_argument('--epochs', dest**=**'epochs',

default**=**10, type**=**int,

help**=**'Number of epochs.')

​

args **=** parser.parse\_args()

​

print('Python Version = {}'.format(sys.version))

print('TensorFlow Version = {}'.format(tf.\_\_version\_\_))

print('TF\_CONFIG = {}'.format(os.environ.get('TF\_CONFIG', 'Not found')))

print('DEVICES', device\_lib.list\_local\_devices())

​

*# Define batch size*

BATCH\_SIZE **=** 32

​

*# Load the dataset*

datasets, info **=** tfds.load('horses\_or\_humans', with\_info**=True**, as\_supervised**=True**)

​

*# Normalize and batch process the dataset*

ds\_train **=** datasets['train'].map(**lambda** x, y: (tf.cast(x, tf.float32)**/**255.0, y)).batch(BATCH\_SIZE)

​

​

*# Build the Convolutional Neural Network*

model **=** tf.keras.models.Sequential([

*# Note the input shape is the desired size of the image 300x300 with 3 bytes color*

tf.keras.layers.Conv2D(16, (3,3), activation**=**'relu', input\_shape**=**(300, 300, 3)),

tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(32, (3,3), activation**=**'relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64, (3,3), activation**=**'relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64, (3,3), activation**=**'relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64, (3,3), activation**=**'relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(512, activation**=**'relu'),

*# Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('horses') and 1 for the other ('humans')*

tf.keras.layers.Dense(1, activation**=**'sigmoid')

])

​

model.compile(optimizer **=** tf.keras.optimizers.RMSprop(),

loss **=** tf.keras.losses.BinaryCrossentropy(),

metrics**=**[tf.keras.metrics.BinaryAccuracy()])

​

​

​

*# Train and save the model*

MODEL\_DIR **=** os.getenv("AIP\_MODEL\_DIR")

​

model.fit(ds\_train, epochs**=**args.epochs)

model.save(MODEL\_DIR)

Writing task.py

**Define the command args for the training script**

Prepare the command-line arguments to pass to your training script.

* args: The command line arguments to pass to the corresponding Python module. In this example, they will be:
  + "--epochs=" + EPOCHS: The number of epochs for training.

[13]:



JOB\_NAME **=** "custom\_job\_" **+** TIMESTAMP

MODEL\_DIR **=** "{}/{}".format(BUCKET\_NAME, JOB\_NAME)

​

EPOCHS **=** 8

​

CMDARGS **=** [

"--epochs=" **+** str(EPOCHS),

]

**Train the model**

Define your custom training job on Vertex AI.

Use the CustomTrainingJob class to define the job, which takes the following parameters:

* display\_name: The user-defined name of this training pipeline.
* script\_path: The local path to the training script.
* container\_uri: The URI of the training container image.
* requirements: The list of Python package dependencies of the script.
* model\_serving\_container\_image\_uri: The URI of a container that can serve predictions for your model — either a prebuilt container or a custom container.

Use the run function to start training, which takes the following parameters:

* args: The command line arguments to be passed to the Python script.
* replica\_count: The number of worker replicas.
* model\_display\_name: The display name of the Model if the script produces a managed Model.
* machine\_type: The type of machine to use for training.
* accelerator\_type: The hardware accelerator type.
* accelerator\_count: The number of accelerators to attach to a worker replica.

The run function creates a training pipeline that trains and creates a Model object. After the training pipeline completes, the run function returns the Model object.

You can read more about the CustomTrainingJob.run API [here](https://googleapis.dev/python/aiplatform/latest/aiplatform.html?highlight=customtraining#google.cloud.aiplatform.CustomTrainingJob.run)

[14]:



job **=** aiplatform.CustomTrainingJob(

display\_name**=**JOB\_NAME,

script\_path**=**"task.py",

container\_uri**=**TRAIN\_IMAGE,

requirements**=**["tensorflow\_datasets==4.6.0"],

model\_serving\_container\_image\_uri**=**DEPLOY\_IMAGE,

)

​

MODEL\_DISPLAY\_NAME **=** "horsesvshumans-" **+** TIMESTAMP

​

*# Start the training*

**if** TRAIN\_GPU:

model **=** job.run(

model\_display\_name**=**MODEL\_DISPLAY\_NAME,

args**=**CMDARGS,

replica\_count**=**1,

machine\_type**=**TRAIN\_COMPUTE,

accelerator\_type**=**TRAIN\_GPU.name,

accelerator\_count**=**TRAIN\_NGPU,

)

**else**:

model **=** job.run(

model\_display\_name**=**MODEL\_DISPLAY\_NAME,

args**=**CMDARGS,

replica\_count**=**1,

machine\_type**=**TRAIN\_COMPUTE,

accelerator\_count**=**0,

)

Training script copied to:

gs://qwiklabs-gcp-02-bd07dc32b9c1/aiplatform-2023-03-18-13:33:09.215-aiplatform\_custom\_trainer\_script-0.1.tar.gz.

Training Output directory:

gs://qwiklabs-gcp-02-bd07dc32b9c1/aiplatform-custom-training-2023-03-18-13:33:09.526

View Training:

<https://console.cloud.google.com/ai/platform/locations/us-central1/training/6585339764294025216?project=117287216777>

CustomTrainingJob projects/117287216777/locations/us-central1/trainingPipelines/6585339764294025216 current state:

PipelineState.PIPELINE\_STATE\_PENDING

CustomTrainingJob projects/117287216777/locations/us-central1/trainingPipelines/6585339764294025216 current state:

PipelineState.PIPELINE\_STATE\_PENDING

View backing custom job:

<https://console.cloud.google.com/ai/platform/locations/us-central1/training/2928997409008648192?project=117287216777>

CustomTrainingJob projects/117287216777/locations/us-central1/trainingPipelines/6585339764294025216 current state:

PipelineState.PIPELINE\_STATE\_RUNNING

CustomTrainingJob projects/117287216777/locations/us-central1/trainingPipelines/6585339764294025216 current state:

PipelineState.PIPELINE\_STATE\_RUNNING

CustomTrainingJob projects/117287216777/locations/us-central1/trainingPipelines/6585339764294025216 current state:

PipelineState.PIPELINE\_STATE\_RUNNING

CustomTrainingJob projects/117287216777/locations/us-central1/trainingPipelines/6585339764294025216 current state:

PipelineState.PIPELINE\_STATE\_RUNNING

CustomTrainingJob projects/117287216777/locations/us-central1/trainingPipelines/6585339764294025216 current state:

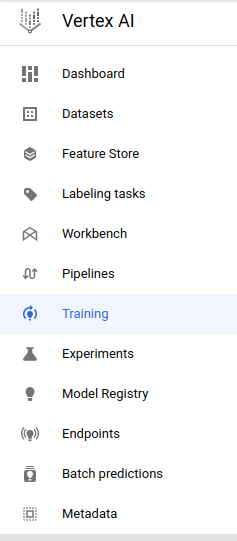
PipelineState.PIPELINE\_STATE\_RUNNING

CustomTrainingJob run completed. Resource name: projects/117287216777/locations/us-central1/trainingPipelines/6585339764294025216

Model available at projects/117287216777/locations/us-central1/models/6665734267810611200

The Training pipeline will take an average of **12 to 14** minutes to finish.

To view the training pipeline status, you have to navigate to **Vertex AI** ➞ **Training**

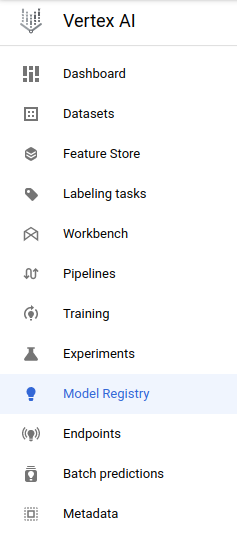


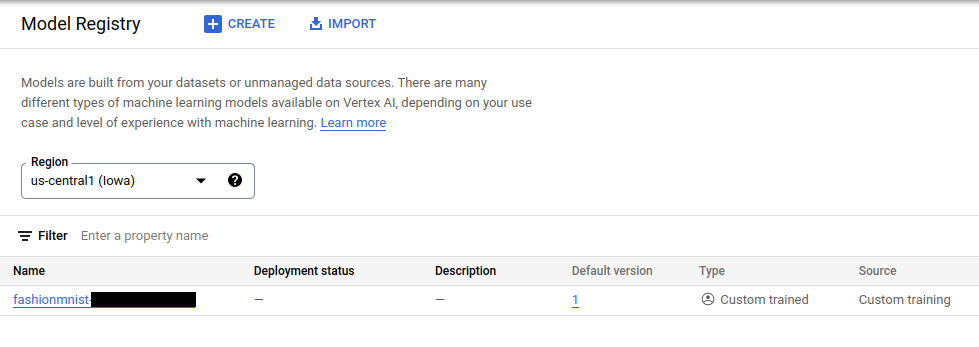
You can see the status of the current training pipeline as seen below

Graphical user interface, text, application

Description automatically generated

Once the model has been successfully trained, you can see a custom trained model if you head to **Vertex AI** ➞ **Model Registry**





**Check progress**

Go back to lab manual and Click *Check my progress* under **Train the Model on Vertex AI** section to verify that the pipeline ran successfully.

**Testing**

Get the test dataset and load the images/labels.

Set the batch size to -1 to load the entire dataset.

[15]:



**import** tensorflow\_datasets **as** tfds

**import** numpy **as** np

​

tfds.disable\_progress\_bar()

[16]:



datasets, info **=** tfds.load('horses\_or\_humans', batch\_size**=-**1, with\_info**=True**, as\_supervised**=True**)

​

test\_dataset **=** datasets['test']

Downloading and preparing dataset 153.59 MiB (download: 153.59 MiB, generated: Unknown size, total: 153.59 MiB) to /home/jupyter/tensorflow\_datasets/horses\_or\_humans/3.0.0...

Dataset horses\_or\_humans downloaded and prepared to /home/jupyter/tensorflow\_datasets/horses\_or\_humans/3.0.0. Subsequent calls will reuse this data.

2023-03-18 13:45:11.734788: W tensorflow/stream\_executor/platform/default/dso\_loader.cc:64] Could not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory; LD\_LIBRARY\_PATH: /usr/local/cuda/lib64:/usr/local/nccl2/lib:/usr/local/cuda/extras/CUPTI/lib64

2023-03-18 13:45:11.735018: W tensorflow/stream\_executor/cuda/cuda\_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (303)

2023-03-18 13:45:11.735059: I tensorflow/stream\_executor/cuda/cuda\_diagnostics.cc:156] kernel driver does not appear to be running on this host (cloudlearningservices): /proc/driver/nvidia/version does not exist

2023-03-18 13:45:11.739414: I tensorflow/core/platform/cpu\_feature\_guard.cc:151] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 FMA

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2023-03-18 13:45:17.177719: W tensorflow/core/framework/cpu\_allocator\_impl.cc:82] Allocation of 277290000 exceeds 10% of free system memory.

Load the TensorFlow Dataset as NumPy arrays (images, labels)

[17]:



x\_test, y\_test **=** tfds.as\_numpy(test\_dataset)

​

*# Normalize (rescale) the pixel data by dividing each pixel by 255.*

x\_test **=** x\_test.astype('float32') **/** 255.

Ensure the shapes are correct here

[18]:



x\_test.shape, y\_test.shape

[18]:

((256, 300, 300, 3), (256,))

[19]:



*#@title Pick the number of test images*

NUM\_TEST\_IMAGES **=** 20 *#@param {type:"slider", min:1, max:20, step:1}*

x\_test, y\_test **=** x\_test[:NUM\_TEST\_IMAGES], y\_test[:NUM\_TEST\_IMAGES]

[20]:



*# Convert to python list*

list\_x\_test **=** x\_test.tolist()

**Batch prediction request**

Since the test data is large, you will send a batch prediction request to your deployed model instead of an online prediction. While starting a batch prediction job you will specify a cloud storage bucket and the output of the batch prediction will be saved in that bucket.

**Precomputed batch prediction files**

Batch prediction requests will take 25 to 30 mins to finish computing on Vertex AI. For ease, we ran batch prediction on this model and saved its prediction files in a public GCS bucket(gs://spls). You will use these files for prediction. After finishing this section you can run the Batch prediction request and check it for yourselves.

**Retrieve batch prediction results from public gcs bucket**

You'll now retrieve the predictions and store them in a list. The predictions will be in a JSONL format, which you indicate when you create the batch prediction job. The predictions are located in a subdirectory starting with the name prediction. Within that directory, there is a file named prediction.results-xxxx-of-xxxx.

The following code displays the contents. You will get a row for each prediction.

[21]:



**import** json

​

PUBLIC\_BUCKET\_NAME **=** 'gs://spls'

PUBLIC\_DESTINATION\_FOLDER **=** 'gsp634'

​

RESULTS\_DIRECTORY **=** "prediction\_results"

​

*# Create destination directory for downloaded results.*

os.makedirs(RESULTS\_DIRECTORY, exist\_ok**=True**)

​

BATCH\_PREDICTION\_GCS\_DEST\_PREFIX **=** PUBLIC\_BUCKET\_NAME **+** "/" **+** PUBLIC\_DESTINATION\_FOLDER

​

*# Download the results.*

**!** gsutil **-**m cp **-**r $BATCH\_PREDICTION\_GCS\_DEST\_PREFIX $RESULTS\_DIRECTORY

​

*# Store the downloaded prediction file paths into a list*

results\_files **=** []

**for** dirpath, subdirs, files **in** os.walk(RESULTS\_DIRECTORY):

**for** file **in** files:

**if** file.startswith("prediction.results"):

results\_files.append(os.path.join(dirpath, file))

​

**def** consolidate\_results():

*# Consolidate all the results into a list*

results **=** []

**for** results\_file **in** results\_files:

*# Download each result*

**with** open(results\_file, "r") **as** file:

results.extend([json.loads(line) **for** line **in** file.readlines()])

**return** results

​

results **=** consolidate\_results()

Copying gs://spls/gsp634/prediction.results-00000-of-00003...

Copying gs://spls/gsp634/prediction.results-00001-of-00003...

Copying gs://spls/gsp634/prediction.results-00002-of-00003...

- [3/4 files][ 51.5 MiB/ 51.5 MiB] 99% Done

**Check progress**

Go back to lab manual and Click *Check my progress* under **Copy the Batch Prediction Results** section to verify that the deployment was a success.

**Rearrange the Batch prediction result**

Batch prediction results are always delivered in random order. So we will compare the result with the original x\_test value and rearrange the results according to that. This is necessary for evaluating the results.

[22]:



**def** rearrange\_results():

*# Empty array to hold the new ordered result*

new\_results **=** [**None**]**\***20

​

**for** result **in** results:

instance\_img **=** np.array(result["instance"])

**for** i **in** range(len(x\_test)):

*# If the result image and x\_test image matches,*

*# then get the index of the image in x\_test.*

*# Insert the result into that index*

*# in the new\_results array*

**if** (instance\_img **==** x\_test[i]).all():

new\_results[i] **=** result

**return** new\_results

new\_results **=** rearrange\_results()

**Evaluate results**

You can then run a quick evaluation on the prediction results:

1. Compare the predicted labels to the actual labels
2. Calculate accuracy as correct/total

[23]:



**def** evaluate\_results():

*# If the prediction value of result is less than 0.5 then it means model predicted this image to belong to class 0, i.e, horse,*

*# else the model predicted the image to belong to class 1, i.e, human.*

y\_predicted **=** [0 **if** result["prediction"][0] **<=** 0.5 **else** 1 **for** result **in** new\_results]

​

correct\_ **=** sum(y\_predicted **==** y\_test)

accuracy **=** len(y\_predicted)

print(

f"Correct predictions = {correct\_}, Total predictions = {accuracy}, Accuracy = {correct\_**/**accuracy}"

)

**return** y\_predicted

y\_predicted **=** evaluate\_results()

Correct predictions = 19, Total predictions = 20, Accuracy = 0.95

**Plot the result**

Here you will plot each image with its predicted label as the title.

If you go through the images, you can see the ones that were wrongly predicted by the model.

[24]:



**import** matplotlib.pyplot **as** plt

​

label **=** {0: 'horse', 1: 'human'}

​

**def** display\_instance\_images(rows**=**64, cols**=**4):

"""Display given images and their labels in a grid."""

fig **=** plt.figure()

fig.set\_size\_inches(cols **\*** 3, rows **\*** 3)

count **=** 0

i **=** 0

**for** result **in** new\_results:

plt.subplot(rows, cols, i **+** 1)

plt.axis('off')

plt.imshow(result['instance'])

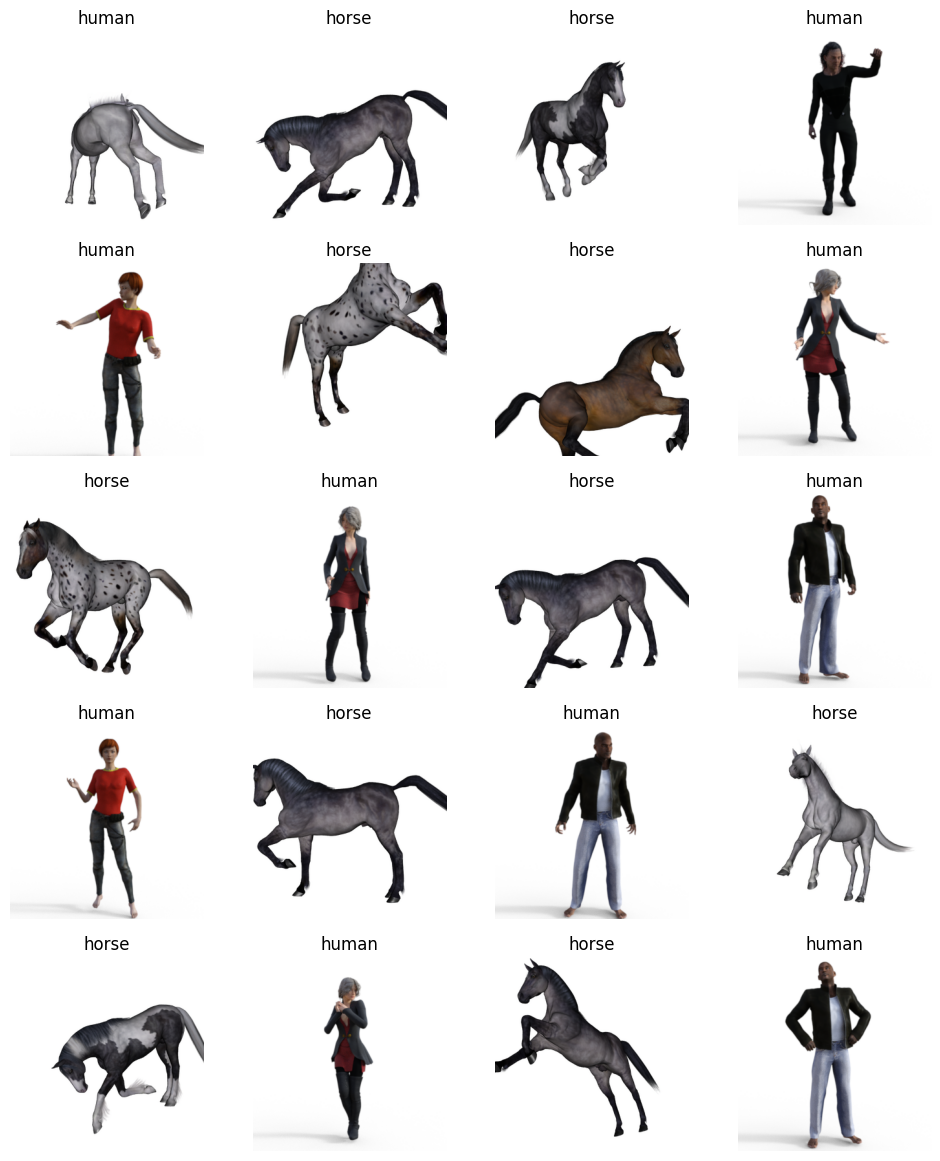
plt.title(label[y\_predicted[count]])

i **+=** 1

count **+=** 1

​

display\_instance\_images()



**[Optional] Make a batch prediction request**

Now let's run a batch prediction request for your model.

**Prepare data for batch prediction**

Before you can run the data through batch prediction, you need to save the data into one of a few possible formats.

For this tutorial, use JSONL as it's compatible with the 3-dimensional list that each image is currently represented in. To do this:

1. In a file, write each instance as JSON on its own line.
2. Upload this file to Cloud Storage.

For more details on batch prediction input formats: <https://cloud.google.com/vertex-ai/docs/predictions/batch-predictions#batch_request_input>

[25]:



**if** BUCKET\_NAME **==** "" **or** BUCKET\_NAME **is** **None** **or** BUCKET\_NAME **==** "gs://[your-bucket-name]":

BUCKET\_NAME **=** "gs://" **+** PROJECT\_ID

[26]:



**import** json

​

BATCH\_PREDICTION\_INSTANCES\_FILE **=** "batch\_prediction\_instances.jsonl"

​

BATCH\_PREDICTION\_GCS\_SOURCE **=** (

BUCKET\_NAME **+** "/batch\_prediction\_instances/" **+** BATCH\_PREDICTION\_INSTANCES\_FILE

)

​

*# Write instances at JSONL*

**with** open(BATCH\_PREDICTION\_INSTANCES\_FILE, "w") **as** f:

**for** x **in** list\_x\_test:

f.write(json.dumps(x) **+** "\n")

​

*# Upload to Cloud Storage bucket*

**!** gsutil cp $BATCH\_PREDICTION\_INSTANCES\_FILE $BATCH\_PREDICTION\_GCS\_SOURCE

​

print("Uploaded instances to: ", BATCH\_PREDICTION\_GCS\_SOURCE)

Copying [file://batch\_prediction\_instances.jsonl](file:///\\batch_prediction_instances.jsonl\) [Content-Type=application/octet-stream]...

- [1 files][ 51.5 MiB/ 51.5 MiB]

Operation completed over 1 objects/51.5 MiB.

Uploaded instances to: gs://qwiklabs-gcp-02-bd07dc32b9c1/batch\_prediction\_instances/batch\_prediction\_instances.jsonl

**Send the prediction request**

To make a batch prediction request, call the model object's batch\_predict method with the following parameters:

* instances\_format: The format of the batch prediction request file: "jsonl", "csv", "bigquery", "tf-record", "tf-record-gzip" or "file-list"
* prediction\_format: The format of the batch prediction response file: "jsonl", "csv", "bigquery", "tf-record", "tf-record-gzip" or "file-list"
* job\_display\_name: The human readable name for the prediction job.
* gcs\_source: A list of one or more Cloud Storage paths to your batch prediction requests.
* gcs\_destination\_prefix: The Cloud Storage path that the service will write the predictions to.
* model\_parameters: Additional filtering parameters for serving prediction results.
* machine\_type: The type of machine to use for training.
* accelerator\_type: The hardware accelerator type.
* accelerator\_count: The number of accelerators to attach to a worker replica.
* starting\_replica\_count: The number of compute instances to initially provision.
* max\_replica\_count: The maximum number of compute instances to scale to. In this tutorial, only one instance is provisioned.

**Compute instance scaling**

You can specify a single instance (or node) to process your batch prediction request. This tutorial uses a single node, so the variables MIN\_NODES and MAX\_NODES are both set to 1.

If you want to use multiple nodes to process your batch prediction request, set MAX\_NODES to the maximum number of nodes you want to use. Vertex AI autoscales the number of nodes used to serve your predictions, up to the maximum number you set. Refer to the [pricing page](https://cloud.google.com/vertex-ai/pricing#prediction-prices) to understand the costs of autoscaling with multiple nodes.

[27]:



MIN\_NODES **=** 1

MAX\_NODES **=** 1

​

*# The name of the job*

BATCH\_PREDICTION\_JOB\_NAME **=** "horsesorhuman\_batch-" **+** TIMESTAMP

​

*# Folder in the bucket to write results to*

DESTINATION\_FOLDER **=** "batch\_prediction\_results"

​

*# The Cloud Storage bucket to upload results to*

BATCH\_PREDICTION\_GCS\_DEST\_PREFIX **=** BUCKET\_NAME **+** "/" **+** DESTINATION\_FOLDER

​

*# Make SDK batch\_predict method call*

batch\_prediction\_job **=** model.batch\_predict(

instances\_format**=**"jsonl",

predictions\_format**=**"jsonl",

job\_display\_name**=**BATCH\_PREDICTION\_JOB\_NAME,

gcs\_source**=**BATCH\_PREDICTION\_GCS\_SOURCE,

gcs\_destination\_prefix**=**BATCH\_PREDICTION\_GCS\_DEST\_PREFIX,

model\_parameters**=None**,

machine\_type**=**DEPLOY\_COMPUTE,

accelerator\_type**=None**,

accelerator\_count**=**0,

starting\_replica\_count**=**MIN\_NODES,

max\_replica\_count**=**MAX\_NODES,

sync**=True**,

)

Creating BatchPredictionJob

INFO:google.cloud.aiplatform.jobs:Creating BatchPredictionJob

BatchPredictionJob created. Resource name: projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob created. Resource name: projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432

To use this BatchPredictionJob in another session:

INFO:google.cloud.aiplatform.jobs:To use this BatchPredictionJob in another session:

bpj = aiplatform.BatchPredictionJob('projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432')

INFO:google.cloud.aiplatform.jobs:bpj = aiplatform.BatchPredictionJob('projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432')

View Batch Prediction Job:

<https://console.cloud.google.com/ai/platform/locations/us-central1/batch-predictions/8030995244679954432?project=117287216777>

INFO:google.cloud.aiplatform.jobs:View Batch Prediction Job:

<https://console.cloud.google.com/ai/platform/locations/us-central1/batch-predictions/8030995244679954432?project=117287216777>

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_PENDING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_PENDING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_PENDING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_PENDING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_PENDING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_PENDING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_RUNNING

BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

JobState.JOB\_STATE\_SUCCEEDED

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432 current state:

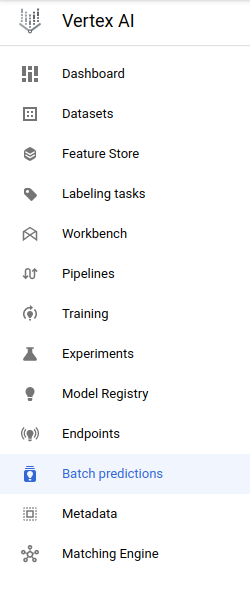
JobState.JOB\_STATE\_SUCCEEDED

BatchPredictionJob run completed. Resource name: projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432

INFO:google.cloud.aiplatform.jobs:BatchPredictionJob run completed. Resource name: projects/117287216777/locations/us-central1/batchPredictionJobs/8030995244679954432

The Batch prediction job will take an average of **23 to 25 minutes** to finish.

In order to view your batch prediction job, you can head over to **Vertex AI** ➞ **Batch predictions**

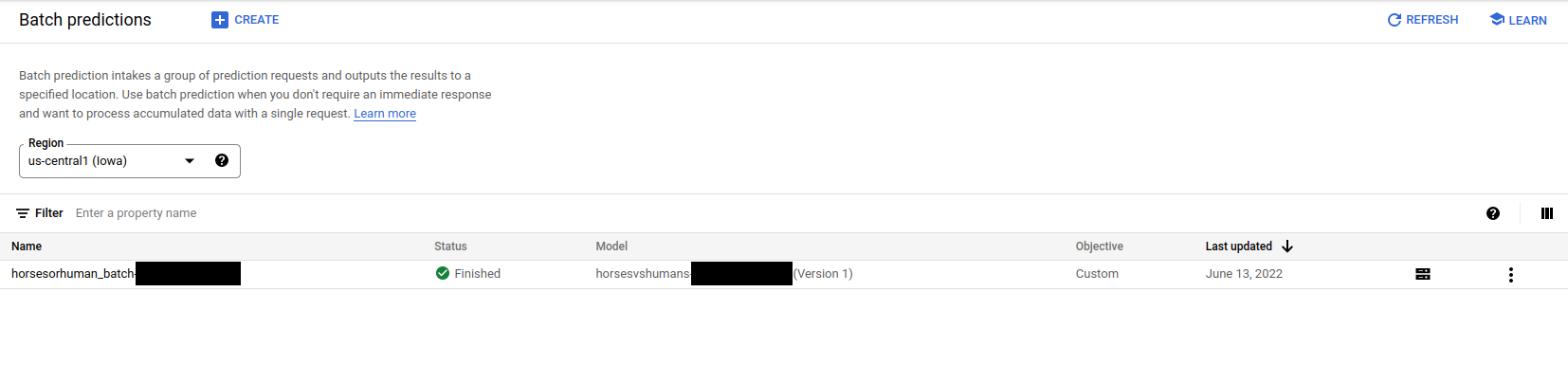


You can check if your batch prediction job is currently in the list of **Batch predictions** list.

Graphical user interface, text, application, chat or text message

Description automatically generated

Once the batch prediction is successfull, you should be able to see the status change to Finished in the above screenshot.



**Retrieve batch prediction results**

When the batch prediction is done processing, you can finally view the predictions stored at the Cloud Storage path you set as output. The predictions will be in a JSONL format, which you indicated when you created the batch prediction job. The predictions are located in a subdirectory starting with the name prediction. Within that directory, there is a file named prediction.results-xxxx-of-xxxx.

[28]:



RESULTS\_DIRECTORY **=** "prediction\_results"

RESULTS\_DIRECTORY\_FULL **=** RESULTS\_DIRECTORY **+** "/" **+** DESTINATION\_FOLDER

​

*# Create missing directories*

os.makedirs(RESULTS\_DIRECTORY, exist\_ok**=True**)

​

*# Get the Cloud Storage paths for each result*

**!** gsutil **-**m cp **-**r $BATCH\_PREDICTION\_GCS\_DEST\_PREFIX $RESULTS\_DIRECTORY

​

*# Get most recently modified directory*

latest\_directory **=** max(

[

os.path.join(RESULTS\_DIRECTORY\_FULL, d)

**for** d **in** os.listdir(RESULTS\_DIRECTORY\_FULL)

],

key**=**os.path.getmtime,

)

​

*# Get downloaded results in directory*

results\_files **=** []

**for** dirpath, subdirs, files **in** os.walk(latest\_directory):

**for** file **in** files:

**if** file.startswith("prediction.results"):

results\_files.append(os.path.join(dirpath, file))

​

*# Consolidate all the results into a list*

**def** consolidate\_results():

*# Consolidate all the results into a list*

results **=** []

**for** results\_file **in** results\_files:

*# Download each result*

**with** open(results\_file, "r") **as** file:

results.extend([json.loads(line) **for** line **in** file.readlines()])

**return** results

results **=** consolidate\_results()

Copying gs://qwiklabs-gcp-02-bd07dc32b9c1/batch\_prediction\_results/prediction-horsesvshumans-20230318133218-2023\_03\_18T06\_46\_56\_434Z/prediction.results-00000-of-00004...

Copying gs://qwiklabs-gcp-02-bd07dc32b9c1/batch\_prediction\_results/prediction-horsesvshumans-20230318133218-2023\_03\_18T06\_46\_56\_434Z/prediction.results-00002-of-00004...

Copying gs://qwiklabs-gcp-02-bd07dc32b9c1/batch\_prediction\_results/prediction-horsesvshumans-20230318133218-2023\_03\_18T06\_46\_56\_434Z/prediction.errors\_stats-00000-of-00001...

Copying gs://qwiklabs-gcp-02-bd07dc32b9c1/batch\_prediction\_results/prediction-horsesvshumans-20230318133218-2023\_03\_18T06\_46\_56\_434Z/prediction.results-00003-of-00004...

Copying gs://qwiklabs-gcp-02-bd07dc32b9c1/batch\_prediction\_results/prediction-horsesvshumans-20230318133218-2023\_03\_18T06\_46\_56\_434Z/prediction.results-00001-of-00004...

\ [5/5 files][ 51.5 MiB/ 51.5 MiB] 100% Done

Operation completed over 5 objects/51.5 MiB.

**Rearrange the Batch prediction result**

[29]:



*# Rearrange results*

**def** rearrange\_results():

*# Empty array to hold the new ordered result*

new\_results **=** [**None**]**\***20

​

**for** result **in** results:

instance\_img **=** np.array(result["instance"])

**for** i **in** range(len(x\_test)):

*# If the result image and x\_test image matches,*

*# then get the index of the image in x\_test.*

*# Insert the result into that index*

*# in the new\_results array*

**if** (instance\_img **==** x\_test[i]).all():

new\_results[i] **=** result

**return** new\_results

new\_results **=** rearrange\_results()

**Evaluate results**

[30]:



*# Evaluate results*

**def** evaluate\_results():

*# If the prediction value of result is less than 0.5 then it means model predicted this image to belong to class 0, i.e, horse,*

*# else the model predicted the image to belong to class 1, i.e, human.*

y\_predicted **=** [0 **if** result["prediction"][0] **<=** 0.5 **else** 1 **for** result **in** new\_results]

​

correct\_ **=** sum(y\_predicted **==** y\_test)

accuracy **=** len(y\_predicted)

print(

f"Correct predictions = {correct\_}, Total predictions = {accuracy}, Accuracy = {correct\_**/**accuracy}"

)

**return** y\_predicted

y\_predicted **=** evaluate\_results()

Correct predictions = 18, Total predictions = 20, Accuracy = 0.9

**Plot the result**

[31]:



**import** matplotlib.pyplot **as** plt

​

label **=** {0: 'horse', 1: 'human'}

​

**def** display\_instance\_images(rows**=**64, cols**=**4):

"""Display given images and their labels in a grid."""

fig **=** plt.figure()

fig.set\_size\_inches(cols **\*** 3, rows **\*** 3)

count **=** 0

i **=** 0

**for** result **in** new\_results:

plt.subplot(rows, cols, i **+** 1)

plt.axis('off')

plt.imshow(result['instance'])

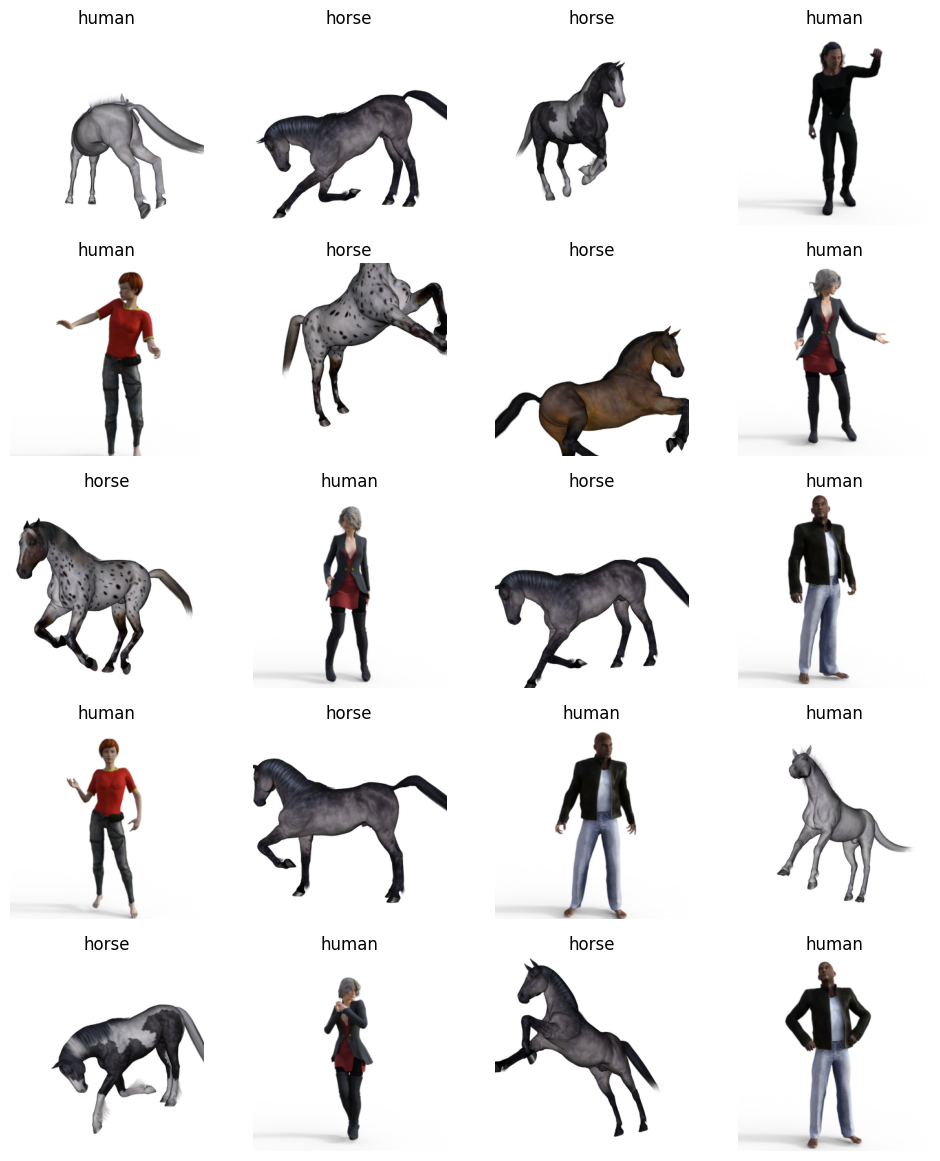
plt.title(label[y\_predicted[count]])

i **+=** 1

count **+=** 1

​

display\_instance\_images()



[ ]:



​