

DevOps

If you are a teacher and feel bored please read Docs/DeepLearning.md where I documented the nightmares I went through during this project. Also look at the beautiful README I created, it looks nicer on GitLab

I will be using the 128x128 CNN model for my DevOps project. To make it easier to quickly glance the important information I have made this section.

Development Environment Setup

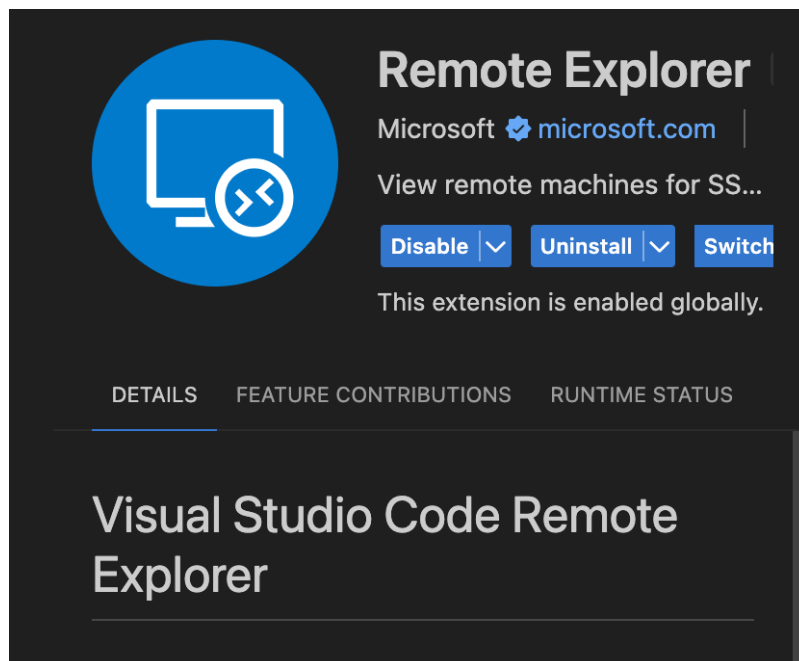
- Ensure that you have Docker installed, if not download it at this [website](#)
- Once you are done, run the following commands

```
docker pull python:3.9.18
docker run -it -v /var/run/docker.sock:/var/run/docker.sock --name
CNN_Server python:3.9.18 sh -c "apt-get update ; apt-get install docker.io -
y ; bash
```

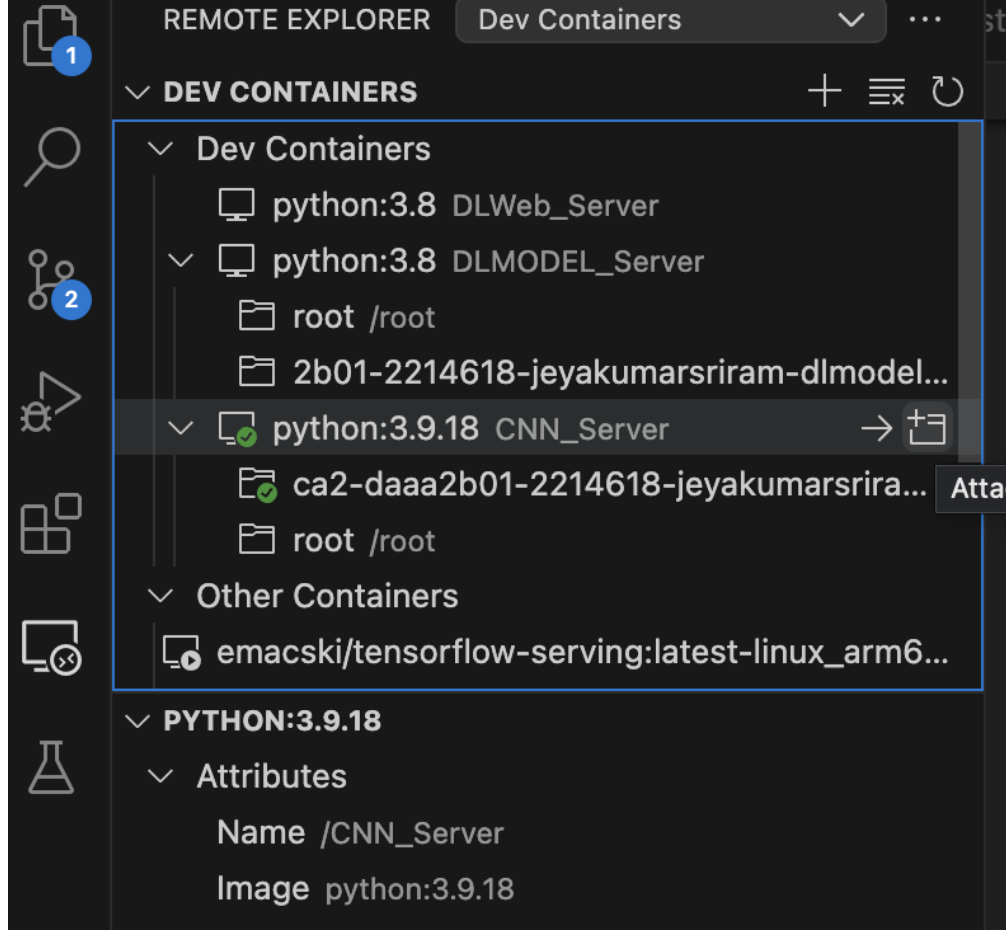
Note: If you are using Mac or Linux, these commands will work instantly, but if you are using Windows, you might have to enable Windows Subsystem for Linux before proceeding

To make sure your docker works inside the development container, try running `docker info`

- Check your Docker Desktop to see if the CNN_Server container is running
- Open up Visual Studio Code and install "Remote Explorer" extension if you don't have it



- Navigate to the Explorer tab and click on the icon as shown below



Configuring Development Environment

- Run the sequence of commands below to download the project directory inside your development container and configure python with all the required packages.

```
git clone https://gitlab.com/4618-devops/ca2-daaa2b01-2214618-jeyakumarsriram-dl.git
python -m venv env
source ./env/bin/activate
cd /ca2-daaa2b01-2214618-jeyakumarsriram-dl
python -m pip install tensorflow
pip install matplotlib seaborn numpy pytest
```

- To train the models, you would need the dataset too. it is recommended to use the dataset from [this website](#)

Or alternatively if you have the Kaggle API acces you can run

```
kaggle datasets download -d moustacheman/vegetable-images
```

- After which, you should see a zip file, unzip it and place it into your project directory. Ensure it is named "Vegetable Images" if not change the `ROOT` variable below.

After all these steps, you should be ready to train some models!

Model Training

Run the following cells sequentially. Ensure that the kernel you selected is Venv

```
In [ ]: # Basics
import numpy as np
import pandas as pd
```

```

import matplotlib.pyplot as plt
import seaborn as sns
import os
import time

# ML
import tensorflow as tf
from tensorflow.keras.layers import *

# Miscellaneous
import warnings
warnings.filterwarnings("ignore")

# ML
import tensorflow as tf
import tensorflow.keras.backend as K

# Modelling
from tensorflow.keras.layers import *

tf.__version__

```

Modify the `ROOT` if your data directory is different.

```

In [ ]: ROOT = "./Vegetable Images"
        CLASSES = ['Broccoli', 'Capsicum', 'Bottle_Gourd', 'Radish', 'Tomato', 'Brinjal', 'Pumpkin']
        tfClasses = tf.constant(CLASSES)
        TRAIN = tf.data.Dataset.list_files(f"{ROOT}/train/*/*")
        TEST = tf.data.Dataset.list_files(f"{ROOT}/test/*/*")
        VAL = tf.data.Dataset.list_files(f"{ROOT}/validation/*/*")
        dataCats = ["train", "test", "validation"]

```

We will be performing some preprocessing on the images. If you are using a different dataset, edit this part as needed.

```

In [ ]: classCounts = []
        for cat in dataCats:
            class_distro = [len(tf.data.Dataset.list_files(f"{ROOT}/{cat}/{i}/*")) for i in CLASSES]
            classCounts.append(class_distro)

        train_distro = classCounts[0]

        targetSize = max(train_distro)
        additional_needed = [targetSize-i for i in train_distro]

        aug_train = []
        for i,v in enumerate(CLASSES):
            path = f'{ROOT}/train/{v}'
            imgNeeded = additional_needed[i]
            images = os.listdir(path)[:imgNeeded]
            aug_train.extend([path + "/" + i for i in images])
        train_data_aug = tf.data.Dataset.from_tensor_slices(aug_train)

        def createPreprocessor(imgSize):
            def processing(path):
                label = tf.strings.split(path, os.path.sep)
                one_hot = label[-2] == tfClasses
                label = tf.argmax(one_hot)
                label = tf.one_hot(label, depth=len(CLASSES))
                img = tf.io.read_file(path)
                img = tf.image.decode_jpeg(img, channels=3)
                img = tf.image.resize(img, [imgSize, imgSize])
                img = tf.image.rgb_to_grayscale(img)
                img = img / 255.0

```

```

        return img , label
    return processing

def augmentation(image,label):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.rot90(image)
    image = tf.image.random_brightness(image, max_delta=0.2)
    image = tf.image.random_contrast(image, lower=0.8, upper=1.2)
    return image , label

def configure_for_performance(ds):
    ds = ds.cache()
    ds = ds.shuffle(buffer_size=1000)
    ds = ds.batch(32)
    ds = ds.prefetch(buffer_size=tf.data.AUTOTUNE)
    return ds

train_ori_large = TRAIN.map(createPreprocessor(128))
train_aug_large = TRAIN.map(createPreprocessor(128)).map(augmentation)
train_large_ds = train_ori_large.concatenate(train_aug_large)

test_large_ds = TEST.map(createPreprocessor(128))
val_large_ds = VAL.map(createPreprocessor(128))

data = train_large_ds.concatenate(test_large_ds).concatenate(val_large_ds)
data = configure_for_performance(data)

train_ori_small = TRAIN.map(createPreprocessor(31))
train_aug_small = TRAIN.map(createPreprocessor(31)).map(augmentation)
train_small_ds = train_ori_small.concatenate(train_aug_small)

test_small_ds = TEST.map(createPreprocessor(31))
val_small_ds = VAL.map(createPreprocessor(31))

data_small = train_small_ds.concatenate(test_small_ds).concatenate(val_small_ds)
data_small = configure_for_performance(data_small)

```

We will be defining our 2 models here. One accepts a 128x128 size image input while the other takes a 31x31 input. Both will only use grayscale images as specified in the previous assignment brief.

```

In [ ]: reg_cnn_model = tf.keras.Sequential([
    Conv2D(16, 3, activation='relu', input_shape=(31, 31, 1)),
    BatchNormalization(),
    MaxPooling2D(),

    Conv2D(32, 3, activation='relu'),
    BatchNormalization(),
    MaxPooling2D(),

    Conv2D(64, 3, activation='relu'),
    BatchNormalization(),
    MaxPooling2D(),

    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(15, activation='softmax')
])

reg_cnn_model_large = tf.keras.Sequential([
    Conv2D(32, 3, activation='relu', input_shape=(128, 128, 1)),
    BatchNormalization(),
    MaxPooling2D(),

```

```

Conv2D(64, 3, activation='relu'),
BatchNormalization(),
MaxPooling2D(),

Conv2D(128, 3, activation='relu'),
BatchNormalization(),
MaxPooling2D(),

Conv2D(256, 3, activation='relu'),
BatchNormalization(),
MaxPooling2D(),

Flatten(),

Dense(256, activation='relu'),
BatchNormalization(),
Dropout(0.5),

Dense(128, activation='relu'),
BatchNormalization(),
Dropout(0.5),

Dense(15, activation='softmax')
])

```

This is a summary of the inputs of the various layers of the CNN models. The number of parameters in each is also mentioned. Note that models with higher parameters will take longer to train.

```

In [ ]: reg_cnn_model.summary()
        reg_cnn_model_large.summary()

```

We will be saving the models using the SavedModel format rather than h5 or .keras in order to ensure compatibility for Tensorflow Serving. The model training begins now, estimated wait time is 20 minutes. So have a nap or a small meal while it cooks!

```

In [ ]: es = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
        reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='loss', factor=0.5, patience=10)

        reg_cnn_model.compile(optimizer='adam',
                               loss='categorical_crossentropy',
                               metrics=['accuracy'])

        modelHist = reg_cnn_model.fit(
            data_small, epochs=25, callbacks=[es, reduce_lr]
        )

        reg_cnn_model.save("./cnn_small/1", save_format="tf")

        reg_cnn_model_large.compile(optimizer='adam',
                                      loss='categorical_crossentropy',
                                      metrics=['accuracy'])

        modelHist2 = reg_cnn_model_large.fit(
            data, epochs=15, callbacks=[es, reduce_lr]
        )

        reg_cnn_model_large.save("./cnn_large/1", save_format="tf")

```

For your convenience, I have written some code to see the difference in speed and accuracy between the two models, As you can see the larger model is slightly slower but more accurate.

```
In [ ]: # Measure prediction time for final_31
start_time = time.time()
prediction_31 = reg_cnn_model.evaluate(configure_for_performance(test_small_ds))
end_time = time.time()
time_taken_31 = end_time - start_time

# Measure prediction time for final_128
start_time = time.time()
prediction_128 = reg_cnn_model_large.evaluate(configure_for_performance(test_large_ds))
end_time = time.time()
time_taken_128 = end_time - start_time

# Print the prediction times
print(f"Prediction time for final_31: {time_taken_31} seconds")
print(f"Prediction time for final_128: {time_taken_128} seconds")
```

Model Serving

Ensure that you have `cnn_large/` and `cnn_small/` directories in your project's root folder.

- Run this command to pull the tensorflow serving docker image

```
docker pull tensorflow/serving
```

If you are using a M Series Macbook, use this command instead to ensure compatibility

```
docker pull emacski/tensorflow-serving:latest-linux_arm64
```

These are two tensorflow models that we will be serving using the same container. Since we are not serving just one model, we cannot use the standard command for tensorflow/serving.

- Create a `model_config.config` file in the root of your project directory.

For convenience, consider copying your `model_config.config`, `cnn_large/` and `cnn_small/` files to a easily accessible place like `C://production` in windows and `~/production` in Unix based systems.

- Add the following YAML code to your config

```
model_config_list:
  config:
    name: "large"
    base_path: "/models/cnn_large"
    model_platform: "tensorflow"
  config:
    name: "small"
    base_path: "/models/cnn_small"
    model_platform: "tensorflow"
```

- Run the following command to serve your models. Replace the string with your system's information before running the command

```
docker run --name cnn_models -p 8501:8501 \
  -v
"/Users/sriramjeyakumar/Production/model_config.config:/models/model_config.con
\
  -v "/Users/sriramjeyakumar/Production/cnn_large:/models/cnn_large" \
  -v "/Users/sriramjeyakumar/Production/cnn_small:/models/cnn_small" \
  -t emacski/tensorflow-serving:latest-linux_arm64 --
model_config_file=/models/model_config.config
```

Use the following command to change the tensorflow-serving image to latest-linux_arm64 to tensorflow/serving if you are not using a M Series Mac

Here is the breakdown of the command above: - We map the port 8501 to our localhost's port 8501 so we can make requests to it - We mount 3 different files: Our two model directories and the config file - We use the serving image to create the container - We specify to TF Serving the config file

You should expect to see something like this when it runs

```
(base) srirameyaykumar@Srirams-MacBook-Pro ca2-daa2b01-2214618-jeaykumarsriram-dl % docker run --name cnn_models -p 8501:8501 -v "/Users/srirameyaykumar/Production/model_config:config:/models/cnn_models:rw" tensorflow/serving:latest-linux_arm64
2024-01-19 03:48:55.888611 I external/tf_serving/tensorflow_serving/model_servers/server_core.cc:465 Adding/updating models.
2024-01-19 03:48:55.888839 I external/tf_serving/tensorflow_serving/model_servers/server_core.cc:591 (Re-)loading model: large
2024-01-19 03:48:55.888845 I external/tf_serving/tensorflow_serving/model_servers/server_core.cc:591 (Re-)loading model: small
2024-01-19 03:48:55.884389 I external/tf_serving/tensorflow_serving/core/basic_manager.cc:740 Successfully reserved resources to load servable (name: small version: 1)
2024-01-19 03:48:55.884393 I external/tf_serving/tensorflow_serving/core/loader_harness.cc:66 Approving load for servable version (name: small version: 1)
2024-01-19 03:48:55.884393 I external/tf_serving/tensorflow_serving/core/loader_harness.cc:741 Loading servable version (name: small version: 1)
2024-01-19 03:48:55.884409 I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:195 Reading SavedModel from: /models/cnn_small/1
2024-01-19 03:48:55.892455 I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:190 Reading meta graph with tags { serve }
2024-01-19 03:48:55.892571 I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:132 Reading SavedModel debug info (if present) from: /models/cnn_small/1
2024-01-19 03:48:55.893049 I external/org_tensorflow/tensorflow/core/common_runtime/process_util.cc:146 Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.
2024-01-19 03:48:55.893067 I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:211 Restoring SavedModel bundle.
2024-01-19 03:48:55.893269 W external/org_tensorflow/tensorflow/core/platform/profile_utils/cpu_utils.cc:97 Failed to get CPU frequency: -1
2024-01-19 03:48:55.896887 E external/org_tensorflow/tensorflow/core/framework/node_def_util.cc:629 NodeDef mentions attribute debug_name which is not in the op definition: Opname=VarHandleOp; signatures=> resource=resource; attrcontainer=string,default="" attrshared_name=string,default="" attrdtype=type; attrshape=shape; attrallowed_devices=List(string),default=[]; is_stateful=true This may be expected if your graph generating binary is newer than this binary. Unknown attributes will be ignored. NodeDef: (Node Adw/batch_normalization/beta/b)
2024-01-19 03:48:55.892633 I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:195 Running initialization op on SavedModel bundle at path: /models/cnn_small/1
2024-01-19 03:48:55.893219 I external/tf_serving/tensorflow_serving/core/basic_manager.cc:740 Successfully reserved resources to load servable (name: large version: 1)
2024-01-19 03:48:55.893264 I external/tf_serving/tensorflow_serving/core/loader_harness.cc:66 Approving load for servable version (name: large version: 1)
2024-01-19 03:48:55.893272 I external/tf_serving/tensorflow_serving/core/loader_harness.cc:741 Loading servable version (name: large version: 1)
2024-01-19 03:48:55.893351 I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:138 Reading savedModel from: /models/cnn_large/1
2024-01-19 03:48:55.898100 I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:190 Reading meta graph with tags { serve }
2024-01-19 03:48:55.898142 I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:132 Reading SavedModel debug info (if present) from: /models/cnn_large/1
2024-01-19 03:48:55.899288 I external/org_tensorflow/tensorflow/cc/saved_model/loader.cc:283 SavedModel load for tags { serve }; Status: success; 0s; Took 107859 microseconds.
```

Also run the following commands to connect your development container to the Tensorflow Server so you have call its api from within your development environment.

```
apt-get install iputils-ping
docker network create dl_network
docker network connect dl_network cnn_models
docker network connect dl_network CNN_Server
Try ping cnn_models and you should see something like this
```

```
2172621098a1778c508bcb62000351771f0162753e090d0f1e2e88c01a147d0
(env) root@d22794546508:~# docker network connect dl_network cnn_models
(env) root@d22794546508:~# docker network connect dl_network CNN_Server
(env) root@d22794546508:~# ping cnn_models
PING cnn_models (172.19.0.2) 56(84) bytes of data:
64 bytes from cnn_models.dl_network (172.19.0.2): icmp_seq=1 ttl=64 time=0.250 ms
64 bytes from cnn_models.dl_network (172.19.0.2): icmp_seq=2 ttl=64 time=0.141 ms
64 bytes from cnn_models.dl_network (172.19.0.2): icmp_seq=3 ttl=64 time=0.190 ms
64 bytes from cnn_models.dl_network (172.19.0.2): icmp_seq=4 ttl=64 time=0.290 ms
64 bytes from cnn_models.dl_network (172.19.0.2): icmp_seq=5 ttl=64 time=0.112 ms
64 bytes from cnn_models.dl_network (172.19.0.2): icmp_seq=6 ttl=64 time=0.090 ms
64 bytes from cnn_models.dl_network (172.19.0.2): icmp_seq=7 ttl=64 time=1.28 ms
64 bytes from cnn_models.dl_network (172.19.0.2): icmp_seq=8 ttl=64 time=0.032 ms
64 bytes from cnn_models.dl_network (172.19.0.2): icmp_seq=9 ttl=64 time=0.302 ms
```

Checking if TensorFlow Serving is Running

1. Large Model:

- URL: <http://localhost:8501/v1/models/large>
- This endpoint should provide information about the 128x128 TensorFlow model, including its status and configuration.

2. Small Model:

- URL: <http://localhost:8501/v1/models/small>
- This endpoint should provide information about the 31x31 TensorFlow model, including its status and configuration.

Open these URLs in your web browser or use tools like `curl` or `wget` in the command line to make HTTP requests. If TensorFlow Serving is running and the models are successfully loaded, you should see something like this

```
localhost:8501/v1/models/large

{
  "model_version_status": [
    {
      "version": "1",
      "state": "AVAILABLE",
      "status": {
        "error_code": "OK",
        "error_message": ""
      }
    }
  ]
}
```

```
localhost:8501/v1/models/small

{
  "model_version_status": [
    {
      "version": "1",
      "state": "AVAILABLE",
      "status": {
        "error_code": "OK",
        "error_message": ""
      }
    }
  ]
}
```

Also run this command that will use pytest to check various parts of our model. It will test the basic functionalities, test the range of the model as well as its consistency.

```
python -m pytest
```

You should expect to see something like this.

```
(env) root@d22794546588:~/ca2-daaa2b01-2214618-jeyakumarsriram-dl# python -m pytest
platform linux -- Python 3.9.18, pytest-7.4.4, pluggy-1.3.0
rootdir: /root/ca2-daaa2b01-2214618-jeyakumarsriram-dl
collected 12 items

tests/test_consistency.py .. [ 16%]
tests/test_expected.py xxxx [ 50%]
tests/test_range.py xxxx [ 83%]
tests/test_unexpected.py .. [100%]

===== 4 passed, 8 xfailed in 2.67s =====
(env) root@d22794546588:~/ca2-daaa2b01-2214618-jeyakumarsriram-dl#
```

Congrats! 🥳 You have successfully served your models.

Now they are only accessible in your laptop. If you want to make it accessible everywhere, consider using a cloud solution like [Render](#) to host your model

Deployment

Before you do any deployment, you need your changes to be pushed to the cloud. But before you commit your changes so far into git, we have to address how you are storing your models.

Tensorflow models are not small and are more like static object which are not supposed to be tracked by git. So it is not the best idea to push your entire model onto github using the conventional way.

While there are probably better industry-standard ways to store your models, most of them are PAID 😞. Since you my friend, have no cash to spare, I will give you a simpler solution that solves this problem. Git Large File Storage.

Git LFS basically allows git to take in your files but not track all the changes. When you push your repo to the platform like GitHub or GitLab, items marked with LFS won't be stored in the same place as the rest of your code. Instead it will be stored in a LFS server and a pointer to it will be stored in your main repo. Don't worry, when you push and pull, Git will automatically get your LFS items for you.

LFS Setup

For Windows, skip this section as it is installed by default for you, if it isn't for some reason, good luck!

For Macbook, you can run `brew install git-lfs`

But if you are working on this inside a Development container, you are in a linux environment. You will have to run more disgusting commands

```
curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.deb.sh | bash
apt-get install git-lfs
```

We omitted the use of sudo here as it's not needed for docker containers

Confirm installation using the command `git lfs`

Now, which files are so huge that we need Git LFS to track them? Well you might think that it is the entire directories of `cnn_large/` and `cnn_small/` but NO. If you snoop around into these directories, most of the files basically contain metadata, like your model structure and whatnot. The real heavy files are those under `variables` subdirectory. So we will only use LFS on them. Now you should run the following commands.

```
git lfs install
git lfs track "*/*/variables/*"
git add .gitattributes
```

And that's it! You have setup LFS, now you just have to add and commit as usual. Whenever you create new models, remember to run the track command again.

If you are unsure if you have done this correctly, run

```
git lfs status
```

which will show the files that are tracked.

One more thing, under your GitLab ensure that the LFS option is selected like this

View and edit files in this project.

☒

Merge requests

Submit changes to be merged upstream.

☒

Forks

Users can copy the repository to a new project.

☒

Git Large File Storage (LFS)

Manages large files such as audio, video, and graphics files. [Learn more.](#)

☒





CI/CD

Build, test, and deploy your changes.

☒

Container registry

When you deployed to gitlab you will see a Tag like this to signify that it is under the LFS server

<div> Used git LFS Moustache Man authored just now</div> <div>4110b11e </div>		
Name	Last commit	Last update
..		
 variables.data-00000-of-00001 LFS	Used git LFS	just now
 variables.index LFS	Used git LFS	just now

Deploying to Render

Ensure that you have committed all your changes to GitLab. You either create a new branch called `releases` like me or you can just use the main branch for deployment, whatever works for you. I'm treating my `main` branch more like a development branch while `releases` is the final branch for me. But again, whatever works for you.

I would also suggest that you make `releases` a protected branch so it can't just be deleted liek this

Protected branches 1

By default, protected branches restrict who can modify the branch. [Learn more.](#)

Protect a branch

Branch:

releases

Wildcards such as `**stable` or `production/*` are supported. Branch names are case-sensitive.

Allowed to merge:

Developers + Maintainers

Allowed to push and merge:

Developers + Maintainers

Allowed to force push:

×

Allow all users with push access to [force push](#).

Protect

Cancel

Branch	Allowed to merge	Allowed to push and merge	Allowed to force push
main default	Maintainers	Maintainers	<div>×</div> <div>Unprotect</div>

To deploy,

- Create an account on render and log in.
- After going to your dashboard, click "New" and select "Web Service" as the type of project you will be deploying
- Connect your GitLab to Render for easier deployment
- Fill in the name and make sure the Environment is Docker, set the render to track `releases` branch or whatever you want
- Click deploy and wait for the logs to say your service is live

You should see something like this if you model deployment was successful.

WEB SERVICE

VegetableCNN

Docker

Free

Connect

Manual Deploy

4618-devops / ca2-daaa2b01-2214618-jeyakumarsriram-dl p releases

<https://vegetablecnn-053e.onrender.com>

Events

Logs

Disks

Environment

Shell

Previews

Jobs

Metrics

Scaling

Settings

Free instance types will spin down with inactivity. [Upgrade to a paid instance type](#) to prevent this behavior. [Learn more.](#)

January 24, 2024 at 11:18 PM

Live

dtel250c Related to #5, fixed tf image compatibility

All logs

Search

Live tail

GMT+8

↑

⌵

Jan 24 11:21:17 PM

2024-01-24 15:21:17.578766: I tensorflow_serving/core/loader_harness.cc:95] Successfully loaded servable version {name: large version: 1}

Jan 24 11:21:17 PM

2024-01-24 15:21:17.579841: I tensorflow_serving/model_servers/server_core.cc:488] Finished adding/updating models

Jan 24 11:21:17 PM

2024-01-24 15:21:17.579885: I tensorflow_serving/model_servers/server.cc:118] Using InsecureServerCredentials

Jan 24 11:21:17 PM

2024-01-24 15:21:17.579898: I tensorflow_serving/model_servers/server.cc:383] Profiler service is enabled

Jan 24 11:21:17 PM

2024-01-24 15:21:17.589568: I tensorflow_serving/model_servers/server.cc:409] Running gRPC ModelServer at 0.0.0.0:8500 ...

Jan 24 11:21:17 PM

2024-01-24 15:21:17.592382: I tensorflow_serving/model_servers/server.cc:438] Exporting HTTP/REST API at: localhost:8501 ...

Jan 24 11:21:17 PM

[evhttp_server.cc : 245] NET_LOG: Entering the event loop ...

Jan 24 11:21:20 PM

Your service is live

Now, edit the `pytest/conftest.py` file and replace the `cnv_model:8501` with the url of your render application and rerun PyTest. You should expect to see something like this.

```
(env) root@22794546588:~/ca2-daa2b01-2214618-jevakumarsriram-d1# python -m pytest
platform linux -- Python 3.9.18, pytest-7.4.4, pluggy-1.3.0
rootdir: /root/ca2-daa2b01-2214618-jevakumarsriram-d1
collected 12 items

tests/test_consistency.py .. [ 16%]
tests/test_expected.py xxxx [ 50%]
tests/test_range.py xxxx [ 83%]
tests/test_unexpected.py .. [100%]

===== 4 passed, 8 xfailed in 4.50s =====
(env) root@22794546588:~/ca2-daa2b01-2214618-jevakumarsriram-d1#
```

Setting Up CI/CD Pipeline

This is to basically make you like easier. Whenever you commit into the Releases branch, you want to make sure your code works before it deploys to the internet. Therefore I create a pipeline to test the code before deploying to the production server.

There are many ways to test a Tensorflow Serving app. I unsuccessfully tried running docker inside the CICD pipeline and test. Therefore, once again, I will show you a less fancy but crud way of solving this problem. Having two deployments: one for testing, one for production.

Let me explain the process to you. When a commit happens in the `releases` branch, the following will happen.

1. GitLab will trigger the redeployment of your code in a Test Server hosted on Render (separate from the one you made)
2. GitLab will run PyTest on that test server to confirm if it works as intended
3. If the PyTest returns a success code of 0, Gitlab will trigger the redeployment of your main production server

The configuration for the Test and Production server are exactly the same so you can test your code without messing up the main server. For this you would need to do a couple of things.

- Setup another render deployment called `Test_Server`. Follow the exactl same steps you followed just now but different name
- Go to the Settings of both Render deployments, scroll down and disable automatic redeploy. This is to make Render wait for you call before deploying something.

assets to a CDN.

Auto-Deploy

Automatic deploy on every push to your repository or changes to your service? Select "No" to handle your deploys manually.

No

Cancel

Save changes

Deploy Hook

Your private URL to trigger a deploy for this server. Remember to keep this a secret.











<https://api.render.com/deploy/srv-cmt>



Regenerate Hook

- Copy the Deploy Hook URL into a safe place

- In your gitlab project, go to Settings/CICD and expand the Variables tab. Under which you should add two variables using the two deploy hooks as shown below.

CI/CD Variables </> 2		Reveal values	Add variable
↑ Key	Value	Environments	Actions
PROD_DEPLOY  Prod Deploy Hook Protected Expanded	***** 	All (default) 	 
TEST_DEPLOY  Test Deploy Hook Protected Expanded	***** 	All (default) 	 

Group variables (inherited)

- Now edit your `tests/conftest` file and change the url to your Test Server's url - Create a `.gitlab-ci.yml` file in your project's root folder and put the following things in there.

stages:

- test_deploy
- test
- deploy

test_deployment:

```
stage: test_deploy
script:
  - curl -s "$TEST_DEPLOY"
  - sleep 1m
only:
  - releases
```

pytest:

```
stage: test
image: python:3.9.18
script:
  - pip install -r requirements.txt
  - python -m pytest --junitxml=report.xml
  - TEST_CODE=$?
  - |
    if [ $TEST_CODE -eq 0 ]; then
      echo "Tests are all passed!"
    else
      echo "Tests failed"
      sleep 1m
      exit 1
    fi
artifacts:
  when: always
  reports:
    junit: report.xml
retry:
  max: 2
  when:
    - always
only:
  - releases
dependencies:
  - test_deployment
```

deployment:

```
stage: deploy
script:
  - curl -s "$PROD_DEPLOY"
only:
  - releases
dependencies:
  - pytest
```

This pipeline basically does what I said. To point out a few things, the testing stage will retry twice before quitting and with a 1 minute delay inbetween. This delay is included because something Render takes very long to deploy and a delay in render deployment shouldn't cause the pipeline to fail. I will also save the PyTest in a report.xml file so that you can view it later. To trigger the redeployment, all you have to do is send a GET request to the Deploy Hook you copied. This can be done using curl. To assess the success of PyTest, we will be utilising the status code. 0 means all tests have passed. Only in that scenario we will deploy the application, if not we will just stop the pipeline using `exit 1`.

Now you can try to make a commit in the `releases` branch. To see your pipeline running go to Build/Pipelines in your GitLab interface. You should see something like this

Related to #7 fixed file paths

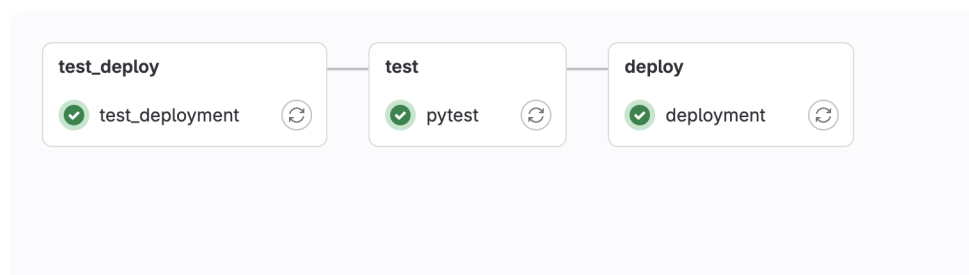
✓ Passed **Moustache Man** created pipeline for commit `0dc5d036` finished 4 minutes ago

Delete

1 related merge request: [!5 Bring back changes made in releases to main](#)

latest 3 Jobs 3.67 3 minutes 40 seconds, queued for 3 seconds

Pipeline Needs Jobs 3 Tests 12




And if you click on the test stage, you can view the results like this

```

183 [notice] A new release of pip is available: 23.0.1 -> 23.3.2
184 [notice] To update, run: pip install --upgrade pip
185 $ python -m pytest --junitxml=report.xml
186 ===== test session starts =====
187 platform linux -- Python 3.9.18, pytest-7.4.4, pluggy-1.3.0
188 rootdir: /builds/4618-devops/ca2-daaa2b01-2214618-jeyakumarsriram-dl
189 collected 12 items
190 tests/test_consistency.py .. [ 16%]
191 tests/test_expected.py xxxx [ 50%]
192 tests/test_range.py xxxx [ 83%]
193 tests/test_unexpected.py .. [100%]
194 - generated xml file: /builds/4618-devops/ca2-daaa2b01-2214618-jeyakumarsriram-dl/report.xml -
195 ===== 4 passed, 8 xfailed in 7.47s =====
196 $ TEST_CODE=$?
197 $ if [ $TEST_CODE -eq 0 ]; then # collapsed multi-line command
198 Tests are all passed!
199 Uploading artifacts for successful job 00:02
200 Uploading artifacts...
201 report.xml: found 1 matching artifact files and directories
202 WARNING: Upload request redirected location=https://gitlab.com/api/v4/jobs/6071108398/artifacts?artifact_format=gzip&artifact_type=junit new-url=https://gitlab.com
203 WARNING: Retrying... context=artifacts-uploader error=request redirected
204 Uploading artifacts as "junit" to coordinator... 201 Created id=6071108398 responseS tatus=201 Created token=glcvt-65
205 Cleaning up project directory and file based variables 00:01
206 Job succeeded



```


If you move over to render you will see the logs mentioned that it was redeployed like this

 WEB SERVICE

VegetableCNN

Docker
Free
[Upgrade your instance →](#)

 4618-devops / ca2-daaa2b01-2214618-jeyakumarsriram-dl  releases

<https://vegetablecnn-053e.onrender.com> 

Connect ▾

Manual Deploy ▾

Events

Logs


Disks


Environment

Shell

Previews


Jobs

 Your free instance will spin down with inactivity, which can delay requests by 50 seconds or more. [Upgrade now.](#)



Deploy live for Odc5d036: Related ...

February 1, 2024 at 6:36 PM



Deploy started for Odc5d036: Related ...

Triggered via Deploy Hook

February 1, 2024 at 6:35 PM

That's it! You are done!