Mediapipe 0.8.5 Installation

Stable Environment

- Ubuntu 20.04 LTS
- x86_64 Or aarch64
- Cuda 11.3
- Cudnn 8.6

Dependencies

- Python 3.8
- Bazel 4.0.0
- Protobuf-Compiler 3.19.1
- OpenCV 4.5.5.64
- OpenCV-Contrib 4.5.5.64

Install Dependencies

0. **Clone Mediapipe**
    - git clone -b v0.8.5 https://github.com/google/mediapipe
1. **Pre-Requirements**:

```
$sudo apt-get install pkg-config zip g++ zlib1g-dev unzip python3
$sudo apt-get update
$sudo apt install openjdk-11-jdk
```

2. **Bazel 4.0.0**：

```
- 下載壓縮檔：
https://github.com/bazelbuild/bazel/releases/download/4.0.0/bazel-
4.0.0-dist.zip
- 返回home目錄：$cd
- 創建Bazel資料夾：$mkdir bazel-4.0.0&&cd bazel-4.0.0
- 移動壓縮檔：$mv ~/Downloads/bazel-4.0.0-dist-zip bazel-4.0.0
- 解壓縮：$unzip bazel-4.0.0-dist.zip
- 編譯安裝：$bash ./compile.sh
- 設定環境：$sudo cp output/bazel /usr/local/bin
```

3. **Mediapipe Dependencies**

```
$sudo apt install -y python3-dev
$sudo apt install -y cmake
```

4. **Protobuf-Compiler**

```
$cd ~
$wget
https://github.com/protocolbuffers/protobuf/releases/download/v3.19.1/
protoc-3.19.1-linux-aarch_64.zip
$unzip protoc-3.19.1-linux-aarch_64.zip -d protoc3.19.1

將protobuf-compilerv3.19.1資料夾中的"bin"和include資料夾內的google複製至
mediapipe資料夾中
```

Mediapipe GPU Version

0. **(OPTIONAL) OpenCV 4.5.5.64 install**

```
#If you haven't install OpenCV4 yet, please install by using following
command.
#Reference:
https://developers.google.com/mediapipe/framework/getting_started/inst
all

(In mediapipe folder)
$sh setup_opencv.sh
```

1. **Config Cuda Path**

```
$cd mediapipe
$export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
$sudo ldconfig
(For x86_64)$export TF_CUDA_PATHS=/usr/local/cuda:/usr/lib/x86_64-
linux-gnu:/usr/include
(For aarch64)$export TF_CUDA_PATHS=/usr/local/cuda:/usr/lib/aarch64-
linux-gnu:/usr/include
```

2. **修改.bazelrc文件**

```
-於文件最後添加以下內容：

build:using_cuda --define=using_cuda=true
build:using_cuda --action_env TF_NEED_CUDA=1
build:using_cuda --
crosstool_top=@local_config_cuda//crosstool:toolchain
build --define=tensorflow_enable_mlir_generated_gpu_kernels=0
build:using_cuda --
define=tensorflow_enable_mlir_generated_gpu_kernels=1
```

```
build:cuda --config=using_cuda
build:cuda --define=using_cuda_nvcc=true
```

3. **配置Protobuf編譯器**

```
$sudo mv protoc3.19.1/bin/* /usr/local/bin/
$sudo mv protoc3.19.1/include/* /usr/local/include/
$sudo chown user /usr/local/bin/protoc
$sudo chown -R user /usr/local/include/google
```

4. **更改Requirements文件**

```
$sed -i -e "s/numpy/numpy==1.19.4/g" requirements.txt
$sed -i -e "s/opencv-contrib-python/opencv-python/g" requirements.txt
```

5. **依照Appendix進行Source Code修改**

6. **編譯**

```
$python3 setup.py gen_protos && python3 setup.py bdist_wheel
```

7. **安裝Mediapipe**

```
$python3 -m pip install cython
$python3 -m pip install numpy
$python3 -m pip install pillow
$python3 -m pip install mediapipe/dist/mediapipe-*-linux_aarch64.whl -
-force-reinstall opencv-contrib-python==4.5.5.64
```

8. **Appendix**

<setup.py>

```
Part1.
    Original:
        bazel_command = [
            'bazel',
            'build',
            '--compilation_mode=opt',
            '--define=MEDIAPIPE_DISABLE_GPU=1',
            '--action_env=PYTHON_BIN_PATH=' +
_normalize_path(sys.executable),
            os.path.join('mediapipe/modules/', graph_path),
            ]

    Modified:
        bazel_command = [
            'bazel',
            'build',
            '--compilation_mode=opt',
            '--config=cuda',
            '--spawn_strategy=local',
            '--define=no_gcp_support=true',
            '--define=no_aws_support=true',
            '--define=no_nccl_support=true',
            '--copt=-DMESA_EGL_NO_X11_HEADERS',
```

```
            '--copt=-DEGL_NO_X11',
            '--local_ram_resources=4096',
            '--local_cpu_resources=3',
            '--action_env=PYTHON_BIN_PATH=' +
_normalize_path(sys.executable),
            os.path.join('mediapipe/modules/', graph_path),
            ]
Part2.
    Original:
        bazel_command = [
            'bazel',
            'build',
            '--compilation_mode=opt',
            '--define=MEDIAPIPE_DISABLE_GPU=1',
            '--action_env=PYTHON_BIN_PATH=' +
_normalize_path(sys.executable),
            str(ext.bazel_target + '.so'),
            ]

    Modified:
        bazel_command = [
            'bazel',
            'build',
            '--compilation_mode=opt',
            '--config=cuda',
            '--spawn_strategy=local',
            '--define=no_gcp_support=true',
            '--define=no_aws_support=true',
            '--define=no_nccl_support=true',
            '--copt=-DMESA_EGL_NO_X11_HEADERS',
            '--copt=-DEGL_NO_X11',
            '--local_ram_resources=4096',
            '--local_cpu_resources=3',
            '--action_env=PYTHON_BIN_PATH=' +
_normalize_path(sys.executable),
            str(ext.bazel_target + '.so'),
            ]
Part3.
    Original:
        def run(self):
            _check_bazel()
            binary_graphs = [
            'face_detection/face_detection_front_cpu',
            'face_landmark/face_landmark_front_cpu',
            'hand_landmark/hand_landmark_tracking_cpu',
            'holistic_landmark/holistic_landmark_cpu',
'objectron/objectron_cpu',
            'pose_landmark/pose_landmark_cpu'
            ]
    Modified:
        def run(self):
            _check_bazel()
            binary_graphs = [
            'face_detection/face_detection_front_gpu',
```

```
            'face_landmark/face_landmark_front_gpu',
            'hand_landmark/hand_landmark_tracking_gpu',
            'holistic_landmark/holistic_landmark_gpu',
            'objectron/objectron_gpu',
            'pose_landmark/pose_landmark_gpu'
            ]
```

<mediapipe/framework/tool/BUILD>

```
Orginal:
    cc_binary(
        name = "encode_as_c_string",
        srcs = ["encode_as_c_string.cc"],
        visibility = ["//visibility:public"],
        deps = [
        "@com_google_absl//absl/strings",
        ],
        )
Modified:
    cc_binary(
        name = "encode_as_c_string",
        srcs = ["encode_as_c_string.cc"],
        visibility = ["//visibility:public"],
        deps = [
        "@com_google_absl//absl/strings",
        ],
        linkopts = ["-lm"],
        )
```

<mediapipe/python/BUILD>

```
Original:
    cc_library(
        name = "builtin_calculators",
        deps = [
        "//mediapipe/calculators/core:gate_calculator",
        "//mediapipe/calculators/core:pass_through_calculator",

"//mediapipe/calculators/core:side_packet_to_stream_calculator",

"//mediapipe/calculators/core:split_normalized_landmark_list_calculato
r",
        "//mediapipe/calculators/core:string_to_int_calculator",

"//mediapipe/calculators/image:image_transformation_calculator",
        "//mediapipe/calculators/util:detection_unique_id_calculator",
        "//mediapipe/modules/face_detection:face_detection_front_cpu",
        "//mediapipe/modules/face_landmark:face_landmark_front_cpu",

"//mediapipe/modules/hand_landmark:hand_landmark_tracking_cpu",
        "//mediapipe/modules/holistic_landmark:holistic_landmark_cpu",
        "//mediapipe/modules/objectron:objectron_cpu",
        "//mediapipe/modules/palm_detection:palm_detection_cpu",
        "//mediapipe/modules/pose_detection:pose_detection_cpu",
        "//mediapipe/modules/pose_landmark:pose_landmark_by_roi_cpu",
        "//mediapipe/modules/pose_landmark:pose_landmark_cpu",

"//mediapipe/modules/selfie_segmentation:selfie_segmentation_cpu",
        ],
        )
Modified:
    cc_library(
        name = "builtin_calculators",
        deps = [
        "//mediapipe/calculators/core:gate_calculator",
        "//mediapipe/calculators/core:pass_through_calculator",

"//mediapipe/calculators/core:side_packet_to_stream_calculator",

"//mediapipe/calculators/core:split_normalized_landmark_list_calculato
r",
        "//mediapipe/calculators/core:string_to_int_calculator",

"//mediapipe/calculators/image:image_transformation_calculator",
        "//mediapipe/calculators/util:detection_unique_id_calculator",
        "//mediapipe/modules/face_detection:face_detection_front_cpu",
        "//mediapipe/modules/face_detection:face_detection_front_gpu",
        "//mediapipe/modules/face_landmark:face_landmark_front_cpu",
        "//mediapipe/modules/face_landmark:face_landmark_front_gpu",

"//mediapipe/modules/hand_landmark:hand_landmark_tracking_gpu",
```

```
        "//mediapipe/modules/holistic_landmark:holistic_landmark_cpu",
        "//mediapipe/modules/holistic_landmark:holistic_landmark_gpu",
        #"//mediapipe/modules/objectron:objectron_cpu",
        "//mediapipe/modules/objectron:objectron_gpu",
        "//mediapipe/modules/palm_detection:palm_detection_gpu",
        "//mediapipe/modules/pose_detection:pose_detection_gpu",
        "//mediapipe/modules/pose_landmark:pose_landmark_by_roi_gpu",
        "//mediapipe/modules/pose_landmark:pose_landmark_cpu",
        "//mediapipe/modules/pose_landmark:pose_landmark_gpu",

"//mediapipe/modules/selfie_segmentation:selfie_segmentation_cpu",

"//mediapipe/modules/selfie_segmentation:selfie_segmentation_gpu",
        "//mediapipe/gpu:image_frame_to_gpu_buffer_calculator",
        "//mediapipe/calculators/image:color_convert_calculator",
        ],
        )
```

<mediapipe/modules/holistic_landmark/holistic_landmark_gpu.pbtxt>

```
Part1.
    Original:
        #Predicts pose landmarks.
        node {
            calculator: "PoseLandmarkGpu"
            input_stream: "IMAGE:image"
            input_side_packet: "MODEL_COMPLEXITY:model_complexity"
            input_side_packet: "SMOOTH_LANDMARKS:smooth_landmarks"
            output_stream: "LANDMARKS:pose_landmarks"
            output_stream: "ROI_FROM_LANDMARKS:pose_landmarks_roi"
            output_stream: "DETECTION:pose_detection"
        }

    Modified:
        node: {
            calculator: "ColorConvertCalculator"
            input_stream: "RGB_IN:image"
            output_stream: "RGBA_OUT:image_rgba"
            }

        node: {
            calculator: "ImageFrameToGpuBufferCalculator"
            input_stream: "image_rgba"
            output_stream: "image_gpu"
            }
        #Predicts pose landmarks.
        node {
            calculator: "PoseLandmarkGpu"
            input_stream: "IMAGE:image"
            input_side_packet: "MODEL_COMPLEXITY:model_complexity"
            input_side_packet: "SMOOTH_LANDMARKS:smooth_landmarks"
            output_stream: "LANDMARKS:pose_landmarks"
            output_stream: "ROI_FROM_LANDMARKS:pose_landmarks_roi"
            output_stream: "DETECTION:pose_detection"
Part2.
    Original:
        #Predicts left and right hand landmarks based on the initial
pose landmarks.
        node {
            calculator: "HandLandmarksLeftAndRightGpu"
            input_stream: "IMAGE:image"
            input_stream: "POSE_LANDMARKS:pose_landmarks"
            output_stream: "LEFT_HAND_LANDMARKS:left_hand_landmarks"
            output_stream: "RIGHT_HAND_LANDMARKS:right_hand_landmarks"
            }
    Modified:
        #Predicts left and right hand landmarks based on the initial
pose landmarks.
        node {
            calculator: "HandLandmarksLeftAndRightGpu"
```

```
            input_stream: "IMAGE:image_gpu"
            input_stream: "POSE_LANDMARKS:pose_landmarks"
            output_stream: "LEFT_HAND_LANDMARKS:left_hand_landmarks"
            output_stream: "RIGHT_HAND_LANDMARKS:right_hand_landmarks"
            }
```

```
Original:
    #Predicts face landmarks based on the initial pose landmarks.
    node {
        calculator: "FaceLandmarksFromPoseGpu"
        input_stream: "IMAGE:image"
        input_stream:
"FACE_LANDMARKS_FROM_POSE:face_landmarks_from_pose"
        output_stream: "FACE_LANDMARKS:face_landmarks"
        }
Modified:
    #Predicts face landmarks based on the initial pose landmarks.
    node {
        calculator: "FaceLandmarksFromPoseGpu"
        input_stream: "IMAGE:image_gpu"
        input_stream:
"FACE_LANDMARKS_FROM_POSE:face_landmarks_from_pose"
        output_stream: "FACE_LANDMARKS:face_landmarks"
        }
```

<mediapipe/python/solutions/holistic.py>

```
BINARYPB_FILE_PATH =
'mediapipe/modules/holistic_landmark/holistic_landmark_cpu.binarypb'
↓
BINARYPB_FILE_PATH =
'mediapipe/modules/holistic_landmark/holistic_landmark_gpu.binarypb'
```

```
Original:

_download_oss_pose_landmark_model(model_complexity)
super().__init__(
    binary_graph_path=BINARYPB_FILE_PATH,
    side_inputs={
        'model_complexity': model_complexity,
        'smooth_landmarks': smooth_landmarks and not
static_image_mode,
    },
    calculator_params={
        'poselandmarkcpu__ConstantSidePacketCalculator.packet': [
            constant_side_packet_calculator_pb2
            .ConstantSidePacketCalculatorOptions.ConstantSidePacket(
                bool_value=not static_image_mode)
        ],

'poselandmarkcpu__posedetectioncpu__TensorsToDetectionsCalculator.min_
score_thresh':
            min_detection_confidence,

'poselandmarkcpu__poselandmarkbyroicpu__ThresholdingCalculator.thresho
ld':
            min_tracking_confidence,
    },
    outputs=[
        'pose_landmarks', 'left_hand_landmarks',
'right_hand_landmarks',
        'face_landmarks'
    ])

Modified:

_download_oss_pose_landmark_model(model_complexity)
super().__init__(
    binary_graph_path=BINARYPB_FILE_PATH,
    side_inputs={
        'model_complexity': model_complexity,
        'smooth_landmarks': smooth_landmarks and not
static_image_mode,
    },
    calculator_params={
```

```
        'poselandmarkgpu__ConstantSidePacketCalculator.packet': [
            constant_side_packet_calculator_pb2
            .ConstantSidePacketCalculatorOptions.ConstantSidePacket(
                bool_value=not static_image_mode)
        ],

'poselandmarkgpu__posedetectiongpu__TensorsToDetectionsCalculator.min_
score_thresh':
            min_detection_confidence,

'poselandmarkgpu__poselandmarkbyroigpu__ThresholdingCalculator.thresho
ld':
            min_tracking_confidence,
    },
    outputs=[
        'pose_landmarks', 'left_hand_landmarks',
'right_hand_landmarks',
        'face_landmarks'
    ])
```

<mediapipe/modules/pose_landmark/pose_landmark_gpu.pbtxt>

```
Original:

    #Calculates size of the image.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:image"
        output_stream: "SIZE:image_size"
        }

Modified:

    node: {
        calculator: "ColorConvertCalculator"
        input_stream: "RGB_IN:image"
        output_stream: "RGBA_OUT:image_rgba"
        }
    node: {
        calculator: "ImageFrameToGpuBufferCalculator"
        input_stream: "image_rgba"
        output_stream: "image_gpu"
        }
    #Calculates size of the image.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:image_gpu"
        output_stream: "SIZE:image_size"
        }
```

```
Original:

    #round of pose detection.
    node {
        calculator: "GateCalculator"
        input_stream: "image"
        input_stream: "image_size"
        input_stream:
"DISALLOW:prev_pose_rect_from_landmarks_is_present"
        output_stream: "image_for_pose_detection"

Modified:
    #round of pose detection.
    node {
        calculator: "GateCalculator"
        input_stream: "image_gpu"
        input_stream: "image_size"
        input_stream:
"DISALLOW:prev_pose_rect_from_landmarks_is_present"
        output_stream: "image_for_pose_detection"
```

```
Original:

    node {
        calculator: "PoseLandmarkByRoiGpu"
        input_side_packet: "MODEL_COMPLEXITY:model_complexity"
        input_stream: "IMAGE:image"
        input_stream: "ROI:pose_rect"
        output_stream: "LANDMARKS:unfiltered_pose_landmarks"
        output_stream:
"AUXILIARY_LANDMARKS:unfiltered_auxiliary_landmarks

Modified:

    node {
        calculator: "PoseLandmarkByRoiGpu"
        input_side_packet: "MODEL_COMPLEXITY:model_complexity"
        input_stream: "IMAGE:image_gpu"
        input_stream: "ROI:pose_rect"
        output_stream: "LANDMARKS:unfiltered_pose_landmarks"
        output_stream:
"AUXILIARY_LANDMARKS:unfiltered_auxiliary_landmarks"
```

```
Original:

    #timestamp bound update occurs to jump start the feedback loop.
    node {
        calculator: "PreviousLoopbackCalculator"
        input_stream: "MAIN:image"
        input_stream: "LOOP:pose_rect_from_landmarks"
        input_stream_info: {
        tag_index: "LOOP"

Modified:
    #timestamp bound update occurs to jump start the feedback loop.
    node {
        calculator: "PreviousLoopbackCalculator"
        input_stream: "MAIN:image_gpu"
        input_stream: "LOOP:pose_rect_from_landmarks"
        input_stream_info: {
        tag_index: "LOOP"
```

<mediapipe/python/solutions/pose.py>

```
BINARYPB_FILE_PATH =
'mediapipe/modules/pose_landmark/pose_landmark_cpu.binarypb'
↓
BINARYPB_FILE_PATH =
'mediapipe/modules/pose_landmark/pose_landmark_gpu.binarypb'
```

```
Original:

    class Pose(SolutionBase):
        .ConstantSidePacketCalculatorOptions.ConstantSidePacket(
        bool_value=not static_image_mode)
        ],

'poselandmarkcpu__posedetectioncpu__TensorsToDetectionsCalculator.min_
score_thresh':
        min_detection_confidence,

'poselandmarkcpu__poselandmarkbyroicpu__ThresholdingCalculator.thresho
ld':
        min_tracking_confidence,
        },
        outputs=['pose_landmarks'])

Modified :

    class Pose(SolutionBase):
        .ConstantSidePacketCalculatorOptions.ConstantSidePacket(
        bool_value=not static_image_mode)
        ],

'poselandmarkgpu__posedetectiongpu__TensorsToDetectionsCalculator.min_
score_thresh':
        min_detection_confidence,

'poselandmarkgpu__poselandmarkbyroigpu__ThresholdingCalculator.thresho
ld':
        min_tracking_confidence,
        },
        outputs=['pose_landmarks'])
```

<mediapipe/modules/hand_landmark/hand_landmark_tracking_gpu.pbtxt>

```
Original:

    #Drops the incoming image if enough hands have already been
identified from the
    #previous image. Otherwise, passes the incoming image through to
trigger a new
    #round of palm detection.
    node {
        calculator: "GateCalculator"
        input_stream: "image"
        input_stream: "DISALLOW:prev_has_enough_hands"
        output_stream: "palm_detection_image"
        options: {
            [mediapipe.GateCalculatorOptions.ext] {
            empty_packets_as_allow: true
            }
        }
    }

Modified:

    node: {
        calculator: "ColorConvertCalculator"
        input_stream: "RGB_IN:image"
        output_stream: "RGBA_OUT:image_rgba"
        }
    node: {
        calculator: "ImageFrameToGpuBufferCalculator"
        input_stream: "image_rgba"
        output_stream: "image_gpu"
        }
    #Drops the incoming image if enough hands have already been
identified from the
    #previous image. Otherwise, passes the incoming image through to
trigger a new
    #round of palm detection.
    node {
        calculator: "GateCalculator"
        input_stream: "image_gpu"
        input_stream: "DISALLOW:prev_has_enough_hands"
        output_stream: "palm_detection_image"
        options: {
            [mediapipe.GateCalculatorOptions.ext] {
            empty_packets_as_allow: true
            }
        }
    }
```

```
Original:

    #Extracts image size.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:image"
        output_stream: "SIZE:image_size"
        }

Modified:

    #Extracts image size.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:image_gpu"
        output_stream: "SIZE:image_size"
        }
```

```
Original:

    node {
        calculator: "BeginLoopNormalizedRectCalculator"
        input_stream: "ITERABLE:hand_rects"
        input_stream: "CLONE:0:image"
        input_stream: "CLONE:1:image_size"
        output_stream: "ITEM:single_hand_rect"
        output_stream: "CLONE:0:image_for_landmarks"
        output_stream: "CLONE:1:image_size_for_landmarks"
        output_stream: "BATCH_END:hand_rects_timestamp"
        }
Modified:

    node {
        calculator: "BeginLoopNormalizedRectCalculator"
        input_stream: "ITERABLE:hand_rects"
        input_stream: "CLONE:0:image_gpu"
        input_stream: "CLONE:1:image_size"
        output_stream: "ITEM:single_hand_rect"
        output_stream: "CLONE:0:image_for_landmarks"
        output_stream: "CLONE:1:image_size_for_landmarks"
        output_stream: "BATCH_END:hand_rects_timestamp"
        }
```

```
Original:
    node {
        calculator: "PreviousLoopbackCalculator"
        input_stream: "MAIN:image"
        input_stream: "LOOP:hand_rects_from_landmarks"
```

```
        input_stream_info: {
            tag_index: "LOOP"
            back_edge: true
            }
        output_stream: "PREV_LOOP:prev_hand_rects_from_landmarks"
        }

Modified:

    node {
        calculator: "PreviousLoopbackCalculator"
        input_stream: "MAIN:image_gpu"
        input_stream: "LOOP:hand_rects_from_landmarks"
        input_stream_info: {
            tag_index: "LOOP"
            back_edge: true
            }
        output_stream: "PREV_LOOP:prev_hand_rects_from_landmarks"
        }
```

<mediapipe/python/solutions/hands.py>

```
BINARYPB_FILE_PATH =
'mediapipe/modules/hand_landmark/hand_landmark_tracking_cpu.binarypb'
↓
BINARYPB_FILE_PATH =
'mediapipe/modules/hand_landmark/hand_landmark_tracking_gpu.binarypb'
```

```
Original:

    calculator_params={
        'ConstantSidePacketCalculator.packet': [
        constant_side_packet_calculator_pb2
        .ConstantSidePacketCalculatorOptions.ConstantSidePacket(
        bool_value=not static_image_mode)
        ],

'palmdetectioncpu__TensorsToDetectionsCalculator.min_score_thresh':
        min_detection_confidence,
        'handlandmarkcpu__ThresholdingCalculator.threshold':
        min_tracking_confidence,
        },
        outputs=['multi_hand_landmarks', 'multi_handedness'])

Modified:

    calculator_params={
        'ConstantSidePacketCalculator.packet': [
        constant_side_packet_calculator_pb2
        .ConstantSidePacketCalculatorOptions.ConstantSidePacket(
        bool_value=not static_image_mode)
        ],

'palmdetectiongpu__TensorsToDetectionsCalculator.min_score_thresh':
        min_detection_confidence,
        'handlandmarkgpu__ThresholdingCalculator.threshold':
        min_tracking_confidence,
        },
        outputs=['multi_hand_landmarks', 'multi_handedness'])
```

<mediapipe/modules/selfie_segmentation/selfie_segmentation_gpu.pbtxt>

```
Original:

    #Resizes the input image into a tensor with a dimension desired by
the model.
    node {
        calculator: "SwitchContainer"
        input_side_packet: "SELECT:model_selection"
        input_stream: "IMAGE_GPU:image"
        output_stream: "TENSORS:input_tensors"
        options: {
        [mediapipe.SwitchContainerOptions.ext] {

Modified:

    node: {
        calculator: "ColorConvertCalculator"
        input_stream: "RGB_IN:image"
        output_stream: "RGBA_OUT:image_rgba"
        }
    node: {
        calculator: "ImageFrameToGpuBufferCalculator"
        input_stream: "image_rgba"
        output_stream: "image_gpu"
        }
    #Resizes the input image into a tensor with a dimension desired by
the model.
    node {
        calculator: "SwitchContainer"
        input_side_packet: "SELECT:model_selection"
        input_stream: "IMAGE_GPU:image_gpu"
        output_stream: "TENSORS:input_tensors"
        options: {
        [mediapipe.SwitchContainerOptions.ext] {
```

```
Original:

    #Retrieves the size of the input image.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:image"
        output_stream: "SIZE:input_size"
        }

Modified:

    #Retrieves the size of the input image.
    node {
        calculator: "ImagePropertiesCalculator"
```

```
        input_stream: "IMAGE_GPU:image_gpu"
        output_stream: "SIZE:input_size"
        }
```

```
Original:

    #Processes the output tensors into a segmentation mask that has
the same size
    #as the input image into the graph.
    node {
        calculator: "TensorsToSegmentationCalculator"
        input_stream: "TENSORS:output_tensors"
        input_stream: "OUTPUT_SIZE:input_size"
        output_stream: "MASK:mask_image"
        options: {
            [mediapipe.TensorsToSegmentationCalculatorOptions.ext] {
            activation: NONE
            gpu_origin: TOP_LEFT
            }
        }
    }
    #Converts the incoming Image into the corresponding GpuBuffer
type.
    node: {
        calculator: "FromImageCalculator"
        input_stream: "IMAGE:mask_image"
        output_stream: "IMAGE_GPU:segmentation_mask"
        }

Modified:

    #Processes the output tensors into a segmentation mask that has
the same size
    #as the input image into the graph.
    node {
        calculator: "TensorsToSegmentationCalculator"
        input_stream: "TENSORS:output_tensors"
        input_stream: "OUTPUT_SIZE:input_size"
        output_stream: "MASK:mask_image"
        options: {
            [mediapipe.TensorsToSegmentationCalculatorOptions.ext] {
            activation: NONE
            gpu_origin: TOP_LEFT
            }
        }
    }
    #Converts the incoming Image into the corresponding GpuBuffer
type.
    node: {
        calculator: "FromImageCalculator"
        input_stream: "IMAGE:mask_image"
```

```
        output_stream: "IMAGE_CPU:segmentation_mask"
        }
```

<mediapipe/python/solutions/selfie_segmentation.py>

```
BINARYPB_FILE_PATH =
'mediapipe/modules/selfie_segmentation/selfie_segmentation_cpu.binaryp
b'
↓
BINARYPB_FILE_PATH =
'mediapipe/modules/selfie_segmentation/selfie_segmentation_gpu.binaryp
b'
```

<mediapipe/modules/objectron/objectron_gpu.pbtxt>

```
Original:

    #Input/Output streams and input side packets.
    #Note that the input image is assumed to have aspect ratio 3:4
(width:height).
    input_stream: "IMAGE_GPU:image"
    #Allowed category labels, e.g. Footwear, Coffee cup, Mug, Chair,
Camera

Modified:

    #Input/Output streams and input side packets.
    #Note that the input image is assumed to have aspect ratio 3:4
(width:height).
    input_stream: "IMAGE_GPU:image"
    #Path to TfLite model for 3D bounding box landmark prediction
    input_side_packet: "MODEL_PATH:box_landmark_model_path"
    #Allowed category labels, e.g. Footwear, Coffee cup, Mug, Chair,
Camera
```

```
Original:

    output_stream: "FRAME_ANNOTATION:detected_objects"
    #Defines whether landmarks from the previous video frame should be
used to help

Modified:

    output_stream: "FRAME_ANNOTATION:detected_objects"
    #Collection of box landmarks. (NormalizedLandmarkList)
    output_stream: "MULTI_LANDMARKS:multi_box_landmarks"
    #Crop rectangles derived from bounding box landmarks.
    output_stream: "NORM_RECTS:multi_box_rects"
    #Defines whether landmarks from the previous video frame should be
used to help
```

```
Original:

    #Defines whether landmarks from the previous video frame should be
used to help

Modified:

    #Loads the file in the specified path into a blob.
    node {
```

```
        calculator: "LocalFileContentsCalculator"
        input_side_packet: "FILE_PATH:0:box_landmark_model_path"
        output_side_packet: "CONTENTS:0:box_landmark_model_blob"
        }
    #Converts the input blob into a TF Lite model.
    node {
        calculator: "TfLiteModelCalculator"
        input_side_packet: "MODEL_BLOB:box_landmark_model_blob"
        output_side_packet: "MODEL:box_landmark_model"
        }
    #Defines whether landmarks from the previous video frame should be
used to help
```

```
Original:

    #Drops the incoming image if BoxLandmarkSubgraph was able to
identify box
    #presence in the previous image. Otherwise, passes the incoming
image through
    #to trigger a new round of box detection in
ObjectDetectionOidV4Subgraph.
    node {
        calculator: "GateCalculator"
        input_stream: "image"
        input_stream: "DISALLOW:prev_has_enough_objects"
        output_stream: "detection_image"
        options: {
            [mediapipe.GateCalculatorOptions.ext] {
            empty_packets_as_allow: true
            }
        }
    }

Modified:

    node: {
        calculator: "ColorConvertCalculator"
        input_stream: "RGB_IN:image"
        output_stream: "RGBA_OUT:image_rgba"
        }
    node: {
        calculator: "ImageFrameToGpuBufferCalculator"
        input_stream: "image_rgba"
        output_stream: "image_gpu"
        }
    #Drops the incoming image if BoxLandmarkSubgraph was able to
identify box
    #presence in the previous image. Otherwise, passes the incoming
image through
    #to trigger a new round of box detection in
ObjectDetectionOidV4Subgraph.
```

```
    node {
        calculator: "GateCalculator"
        input_stream: "image_gpu"
        input_stream: "DISALLOW:prev_has_enough_objects"
        output_stream: "detection_image"
        options: {
            [mediapipe.GateCalculatorOptions.ext] {
            empty_packets_as_allow: true
            }
        }
    }
```

```
Original:

    #Extracts image size from the input images.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:image"
        output_stream: "SIZE:image_size"
        }

Modified:
    #Extracts image size from the input images.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:image_gpu"
        output_stream: "SIZE:image_size"
        }
```

```
Original:

    node {
        calculator: "BeginLoopNormalizedRectCalculator"
        input_stream: "ITERABLE:box_rects"
        input_stream: "CLONE:image"
        output_stream: "ITEM:single_box_rect"
        output_stream: "CLONE:landmarks_image"
        output_stream: "BATCH_END:box_rects_timestamp"
        }

Modified:

    node {
        calculator: "BeginLoopNormalizedRectCalculator"
        input_stream: "ITERABLE:box_rects"
        input_stream: "CLONE:image_gpu"
        output_stream: "ITEM:single_box_rect"
        output_stream: "CLONE:landmarks_image"
```

```
        output_stream: "BATCH_END:box_rects_timestamp"
        }
```

```
Original:

    node {
        calculator: "PreviousLoopbackCalculator"
        input_stream: "MAIN:image"
        input_stream: "LOOP:box_rects_from_landmarks"
        input_stream_info: {
            tag_index: "LOOP"
            back_edge: true
            }
        output_stream: "PREV_LOOP:prev_box_rects_from_landmarks"
        }

Modified:

    node {
        calculator: "PreviousLoopbackCalculator"
        input_stream: "MAIN:image_gpu"
        input_stream: "LOOP:box_rects_from_landmarks"
        input_stream_info: {
            tag_index: "LOOP"
            back_edge: true
            }
        output_stream: "PREV_LOOP:prev_box_rects_from_landmarks"
        }
```

```
Original:

    #Subgraph that localizes box landmarks.
    node {
        calculator: "BoxLandmarkSubgraph"
        input_stream: "IMAGE:landmarks_image"
        input_stream: "NORM_RECT:single_box_rect"
        output_stream: "NORM_LANDMARKS:single_box_landmarks"
        }

Modified:

    #Subgraph that localizes box landmarks.
    node {
        calculator: "BoxLandmarkSubgraph"
        input_stream: "IMAGE:landmarks_image"
        input_side_packet: "MODEL:box_landmark_model"
        input_stream: "NORM_RECT:single_box_rect"
        output_stream: "NORM_LANDMARKS:single_box_landmarks"
        }
```

```
Original:

    #Performs association between NormalizedRect vector elements from
previous
    #image and rects based on object detections from the current
image. This
    #calculator ensures that the output box_rects vector doesn't
contain
    #overlapping regions based on the specified
min_similarity_threshold.
    node {
        calculator: "AssociationNormRectCalculator"
        input_stream: "box_rects_from_detections"
        input_stream: "gated_prev_box_rects_from_landmarks"
        output_stream: "box_rects"
        options: {
            [mediapipe.AssociationCalculatorOptions.ext] {
            min_similarity_threshold: 0.2
            }
        }
    }
    #Outputs each element of box_rects at a fake timestamp for the
rest of the
    #graph to process. Clones image and image size packets for each
    #single_box_rect at the fake timestamp. At the end of the loop,
outputs the
    #BATCH_END timestamp for downstream calculators to inform them
that all
    #elements in the vector have been processed.
    node {
        calculator: "BeginLoopNormalizedRectCalculator"
        input_stream: "ITERABLE:box_rects"
        input_stream: "CLONE:image"
        output_stream: "ITEM:single_box_rect"
        output_stream: "CLONE:landmarks_image"
        output_stream: "BATCH_END:box_rects_timestamp"
        }

Modified:

    #Performs association between NormalizedRect vector elements from
previous
    #image and rects based on object detections from the current
image. This
    #calculator ensures that the output box_rects vector doesn't
contain
    #overlapping regions based on the specified
min_similarity_threshold.
    node {
        calculator: "AssociationNormRectCalculator"
        input_stream: "box_rects_from_detections"
```

```
        input_stream: "gated_prev_box_rects_from_landmarks"
        output_stream: "multi_box_rects"
        options: {
            [mediapipe.AssociationCalculatorOptions.ext] {
            min_similarity_threshold: 0.2
            }
        }
    }
    #Outputs each element of box_rects at a fake timestamp for the
rest of the
    #graph to process. Clones image and image size packets for each
    #single_box_rect at the fake timestamp. At the end of the loop,
outputs the
    #BATCH_END timestamp for downstream calculators to inform them
that all
    #elements in the vector have been processed.
    node {
        calculator: "BeginLoopNormalizedRectCalculator"
        input_stream: "ITERABLE:multi_box_rects"
        input_stream: "CLONE:image"
        output_stream: "ITEM:single_box_rect"
        output_stream: "CLONE:landmarks_image"
        output_stream: "BATCH_END:box_rects_timestamp"
        }
```

<mediapipe/modules/objectron/box_landmark_gpu.pbtxt>

```
Original:

    input_stream: "IMAGE:image"
    input_stream: "NORM_RECT:box_rect"
    output_stream: "NORM_LANDMARKS:box_landmarks"

Modified:

    input_stream: "IMAGE:image"
    input_stream: "NORM_RECT:box_rect"
    input_side_packet: "MODEL:model"
    output_stream: "NORM_LANDMARKS:box_landmarks"
```

```
Original:

    #Runs a TensorFlow Lite model on GPU that takes an image tensor
and outputs a
    #vector of tensors representing, for instance, detection
boxes/keypoints and
    #scores.
    node {
        calculator: "InferenceCalculator"
        input_stream: "TENSORS:image_tensor"
        output_stream: "TENSORS:output_tensors"
        options: {
            [mediapipe.InferenceCalculatorOptions.ext] {
            model_path: "object_detection_3d.tflite"
            delegate { gpu {} }
            }
        }
    }

Modified:

    #Runs a TensorFlow Lite model on GPU that takes an image tensor
and outputs a
    #vector of tensors representing, for instance, detection
boxes/keypoints and
    #scores.
    node {
        calculator: "InferenceCalculator"
        input_stream: "TENSORS:image_tensor"
        input_side_packet: "MODEL:model"
        output_stream: "TENSORS:output_tensors"
        options: {
            [mediapipe.InferenceCalculatorOptions.ext] {
            model_path: "object_detection_3d.tflite"
            delegate { gpu {} }
```

```
            }
        }
    }
```

<mediapipe/python/solutions/objectron.py>

```
BINARYPB_FILE_PATH =
'mediapipe/modules/objectron/objectron_cpu.binarypb'
↓
BINARYPB_FILE_PATH =
'mediapipe/modules/objectron/objectron_gpu.binarypb'
```

<mediapipe/python/solutions/face_mesh.py>

```
Original:

    super().init(
        binary_graph_path=BINARYPB_FILE_PATH,
        side_inputs={
            'num_faces': max_num_faces,
            },
        calculator_params={
            'ConstantSidePacketCalculator.packet': [
            constant_side_packet_calculator_pb2
            .ConstantSidePacketCalculatorOptions.ConstantSidePacket(
            bool_value=not static_image_mode)
            ],

'facedetectionfrontcpu__TensorsToDetectionsCalculator.min_score_thresh
':
            min_detection_confidence,
            'facelandmarkcpu__ThresholdingCalculator.threshold':
            min_tracking_confidence,
            },
        outputs=['multi_face_landmarks'])

Modified:

    super().init(
        binary_graph_path=BINARYPB_FILE_PATH,
        side_inputs={
            'num_faces': max_num_faces,
            },
        calculator_params={
            'ConstantSidePacketCalculator.packet': [
            constant_side_packet_calculator_pb2
            .ConstantSidePacketCalculatorOptions.ConstantSidePacket(
            bool_value=not static_image_mode)
            ],

'facedetectionfrontgpu__TensorsToDetectionsCalculator.min_score_thresh
':
            min_detection_confidence,
            'facelandmarkgpu__ThresholdingCalculator.threshold':
            min_tracking_confidence,
            },
        outputs=['multi_face_landmarks'])
```

<mediapipe/modules/face_detection/face_detection_front_gpu.pbtxt>

```
Original:

    #Converts the input GPU image (GpuBuffer) to the multi-backend
image type
    #(Image).
    node: {
        calculator: "ToImageCalculator"
        input_stream: "IMAGE_GPU:image"
        output_stream: "IMAGE:multi_backend_image"
        }

Modified:
    node: {
        calculator: "ColorConvertCalculator"
        input_stream: "RGB_IN:image"
        output_stream: "RGBA_OUT:image_rgba"
        }
    node: {
        calculator: "ImageFrameToGpuBufferCalculator"
        input_stream: "image_rgba"
        output_stream: "image_gpu"
        }
    #Converts the input GPU image (GpuBuffer) to the multi-backend
image type
    #(Image).
    node: {
        calculator: "ToImageCalculator"
        input_stream: "IMAGE_GPU:image_gpu"
        output_stream: "IMAGE:multi_backend_image"
        }
```

<mediapipe/modules/face_landmark/face_landmark_front_gpu.pbtxt>

```
Original:

    #Drops the incoming image if enough faces have already been
identified from the
    #previous image. Otherwise, passes the incoming image through to
trigger a new
    #round of face detection.
    node {
        calculator: "GateCalculator"
        input_stream: "image"
        input_stream: "DISALLOW:prev_has_enough_faces"
        output_stream: "gated_image"
        options: {
            [mediapipe.GateCalculatorOptions.ext] {
            empty_packets_as_allow: true
            }
        }
    }

Modified:

    node: {
        calculator: "ColorConvertCalculator"
        input_stream: "RGB_IN:image"
        output_stream: "RGBA_OUT:image_rgba"
        }
    node: {
        calculator: "ImageFrameToGpuBufferCalculator"
        input_stream: "image_rgba"
        output_stream: "image_gpu"
        }
    #Drops the incoming image if enough faces have already been
identified from the
    #previous image. Otherwise, passes the incoming image through to
trigger a new
    #round of face detection.
    node {
        calculator: "GateCalculator"
        input_stream: "image_gpu"
        input_stream: "DISALLOW:prev_has_enough_faces"
        output_stream: "gated_image_gpu"
        options: {
            [mediapipe.GateCalculatorOptions.ext] {
            empty_packets_as_allow: true
            }
        }
    }
    node {
        calculator: "GateCalculator"
        input_stream: "image"
```

```
        input_stream: "DISALLOW:prev_has_enough_faces"
        output_stream: "gated_image_cpu"
        options: {
            [mediapipe.GateCalculatorOptions.ext] {
            empty_packets_as_allow: true
            }
        }
    }
```

```
Original:

    #Detects faces.
    node {
        calculator: "FaceDetectionFrontGpu"
        input_stream: "IMAGE:gated_image"
        output_stream: "DETECTIONS:all_face_detections"
        }

Modified:

    #Detects faces.
    node {
        calculator: "FaceDetectionFrontGpu"
        input_stream: "IMAGE:gated_image_cpu"
        output_stream: "DETECTIONS:all_face_detections"
        }
```

```
Original:

    #Calculate size of the image.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:gated_image"
        output_stream: "SIZE:gated_image_size"
        }

Modified:

    #Calculate size of the image.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:gated_image_gpu"
        output_stream: "SIZE:gated_image_size"
        }
```

```
Original:
```

```
    #Calculate size of the image.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:image"
        output_stream: "SIZE:image_size"
        }

Modified:

    #Calculate size of the image.
    node {
        calculator: "ImagePropertiesCalculator"
        input_stream: "IMAGE_GPU:image_gpu"
        output_stream: "SIZE:image_size"
        }
```

```
Original:

    node {
        calculator: "BeginLoopNormalizedRectCalculator"
        input_stream: "ITERABLE:face_rects"
        input_stream: "CLONE:0:image"
        input_stream: "CLONE:1:image_size"
        output_stream: "ITEM:face_rect"
        output_stream: "CLONE:0:landmarks_loop_image"
        output_stream: "CLONE:1:landmarks_loop_image_size"
        output_stream: "BATCH_END:landmarks_loop_end_timestamp"
        }

Modified:

    node {
        calculator: "BeginLoopNormalizedRectCalculator"
        input_stream: "ITERABLE:face_rects"
        input_stream: "CLONE:0:image_gpu"
        input_stream: "CLONE:1:image_size"
        output_stream: "ITEM:face_rect"
        output_stream: "CLONE:0:landmarks_loop_image"
        output_stream: "CLONE:1:landmarks_loop_image_size"
        output_stream: "BATCH_END:landmarks_loop_end_timestamp"
        }
```

```
Original:

    node {
        calculator: "PreviousLoopbackCalculator"
        input_stream: "MAIN:image"
        input_stream: "LOOP:face_rects_from_landmarks"
        input_stream_info: {
```

```
            tag_index: "LOOP"
            back_edge: true
            }
        output_stream: "PREV_LOOP:prev_face_rects_from_landmarks"
        }

Modified:

    node {
        calculator: "PreviousLoopbackCalculator"
        input_stream: "MAIN:image_gpu"
        input_stream: "LOOP:face_rects_from_landmarks"
        input_stream_info: {
            tag_index: "LOOP"
            back_edge: true
            }
        output_stream: "PREV_LOOP:prev_face_rects_from_landmarks"
        }
```