

Face Sentiment Detection

Yassine Maatougui

1. Project Inception

1.1. Framing the Business Idea as an ML Problem **(milestone 1, 5%)**

- **Business case description:**

The project aims to develop a Face Sentiment Detection system capable of identifying and analyzing facial expressions to detect emotions. This technology can be applied in various sectors such as customer service, security, and healthcare to enhance user interactions and emotional assessments.

- **Business value of using ML:**

By employing machine learning, we can automate the recognition of facial emotions with high accuracy, which is crucial for real-time applications. This reduces the need for manual interpretation and allows for quicker and more efficient decision-making processes in professional settings.

- **Data overview:**

The project utilizes a comprehensive dataset consisting of images labeled with eight distinct emotions. This dataset helps in training the model to accurately recognize and classify different facial sentiments.

- **Project archetype:**

This is a classification problem where the model predicts the type of emotion based on facial features extracted from images.

1.2. Feasibility analysis

- **Literature review:**

Reviewed various models from the TensorFlow 2 Detection Model Zoo and the yolov8-emotions model, which are renowned for their efficiency in object detection tasks. The selection was based on their performance metrics in similar tasks outlined in the model zoo documentation [TensorFlow 2 Detection Model Zoo](#).

- **Model choice/ specification of a baseline:**

Chose a robust model from the TensorFlow 2 Detection Model Zoo for initial benchmarking due to its pre-trained capabilities on similar image data, providing a solid baseline for performance comparison.

- **Metrics for business goal evaluation:**

The primary metrics for evaluating the business goals are accuracy, processing speed, and the model's adaptability across various environments and demographic groups.

2. ML Pipeline Development - From a Monolith to a Pipeline

2.1. Ensuring ML Pipeline Reproducibility (milestone 2, 15%)

- **Project structure definition and modularity:**

The ML pipeline is structured for modularity using TFX components to separate data handling, model training, and evaluation processes.

For installing TFX, I have used Anaconda:

- Installation of TFX using Miniconda:
 1. Firstly, I have installed miniconda3 from their official website.
 2. Then, I created an environment for my project using the command:

```
conda create --name tfx python=3.9
conda activate tfx
```
 3. After that I installed tfx: `pip install tfx==1.15.0 tensorflow==2.15.1`
 4. Now, I was able to use tfx library and it gives me this result:

```
tfxu.ipynb
[2]: print('TensorFlow version: {}'.format(tf.__version__))
print('TFX version: {}'.format(tfx.__version__))

TensorFlow version: 2.15.1
TFX version: 1.15.0
```

- However, I still had an issue when I wanted to use the Object Detection library. Therefore, I went to the tfx library, which I installed it using the previous commands and I changed the function ‘preprocessing_fn’. Under that function there is a function named ‘scale_input(inputs)’. On this function I have added these lines of code:

```
image = tf.image.decode_jpeg(inputs, channels=3)
image = tf.image.resize(image, [600, 600])
return image
```

For creating the Pipeline:

Command used: “`tfx pipeline create --pipeline-name=my_face_sentiment_detection_pipeline --pipeline-root={PIPELINE_ROOT} --endpoint={ENDPOINT}`”

- On my Github repository, you can find the script that contains the whole pipeline and it is named “`tfxfu.ipynb`” in the root folder.
- **Code versioning:**
Using Git hosted on GitHub for version control allows meticulous tracking of all changes throughout the development cycle, enhancing transparency and collaboration.

Git commands for versioning:

```
git init  
git add .  
git commit -m "Initial commit"
```

- **Data versioning:**
Data Version Control (DVC) manages dataset changes systematically, maintaining data integrity and ensuring reproducibility across different pipeline stages.

DVC commands used to track data versions:

```
dvc init  
dvc add data/dataset  
git add data.dvc .gitignore  
git commit -m "Add dataset with DVC"
```

- **Experiment tracking and model versioning:**
The Metadata Store in TFX is used for tracking experiments and managing model versions, providing detailed insights into model performance and version history.

- **Setting up a meta store for metadata:**
TFX's ML Metadata is used to store metadata about experiments, model training, and evaluation processes.
 - **Setting up the machine learning pipeline under an MLOps platform:**

The entire ML pipeline is configured and managed using TFX, facilitating continuous integration and deployment practices.

2.2. Pipeline Components (Milestone 3 and 4, 20%)

1. Setup of data pipeline within the larger ML pipeline/ MLOps Platform

o Data Validation and Verification:

TFX Example Validator ensures data quality by comparing data statistics against a schema that describes expectations about data.

ExampleGen

```
context = InteractiveContext()

WARNING:absl:InteractiveContext pipeline_root argument not provided: using temporary directory /tmp/tfx
WARNING:absl:InteractiveContext metadata_connection_config not provided: using SQLite ML Metadata database

example_gen = tfx.components.ImportExampleGen(input_base="my_tfx_project/data/processed/")
context.run(example_gen, enable_cache=True)

WARNING:apache_beam.runners.interactive.interactive_environment:Dependencies required for Interactive Beam
sary dependencies to enable all data visualization features.

WARNING:apache_beam.io.tfrecordio:Couldn't find python-snappy so the implementation of _TFRecordUtil.. m
▼ ExecutionResult at 0x7a1be426ee80

    .execution_id 1
    .component ►ImportExampleGen at 0x7a1be4568370
    .component.inputs {}
    .component.outputs
        ['examples'] ►Channel of type 'Examples' (1 artifact) at 0x7a1be4568f10

artifact = example_gen.outputs['examples'].get()[0]
print(artifact.split_names, artifact.uri)
["train", "eval"] /tmp/tfx-interactive-2024-05-12T12_47_30.457696-ip1zycrf/ImportExampleGen/examples/1
```

SchemaGen

```
schema_gen = tfx.components.SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    infer_feature_shape=False)
context.run(schema_gen, enable_cache=True)
```

▼ ExecutionResult at 0x7a1bdc5b7550

```
.execution_id 3
.component ►SchemaGen at 0x7a1be5608ee0
.component.inputs
  ['statistics'] ►Channel of type 'ExampleStatistics' (1 artifact) at 0x7a1be55f3d00
.component.outputs
  ['schema'] ►Channel of type 'Schema' (1 artifact) at 0x7a1bbc3aa970
```

```
context.show(schema_gen.outputs['schema'])
```

Artifact at /tmp/tfx-interactive-2024-05-12T12_47_30.457696-ip1zycrf/SchemaGen/schema/3

Feature name	Type	Presence	Valency	Domain
'image/encoded'	BYTES	required	single	-
'image/filename'	BYTES	required	single	-
'image/format'	STRING	required	single	'image/format'
'image/height'	INT	required	single	-
'image/object/bbox/xmax'	FLOAT	required	single	-
'image/object/bbox/xmin'	FLOAT	required	single	-
'image/object/bbox/ymax'	FLOAT	required	single	-
'image/object/bbox/ymin'	FLOAT	required	single	-
'image/object/class/label'	INT	required	single	-
'image/object/class/text'	STRING	required	single	'image/object/class/text'
'image/source_id'	BYTES	required	single	-
'image/width'	INT	required	single	-

Values

Domain	Values
'image/format'	'jpeg'
'image/object/class/text'	'anger', 'contempt', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise'

```

example_validator = tfx.components.ExampleValidator(
    statistics=statistics_gen.outputs['statistics'],
    schema=schema_gen.outputs['schema'])
context.run(example_validator, enable_cache=True)

▼ ExecutionResult at 0x7a1be565ad00
  .execution_id 4
  .component ►ExampleValidator at 0x7a1bdc6df8b0
    .component.inputs
      ['statistics'] ►Channel of type 'ExampleStatistics' (1 artifact) at 0x7a1be55f3d00
      ['schema'] ►Channel of type 'Schema' (1 artifact) at 0x7a1bbc3aa970
    .component.outputs
      ['anomalies'] ►Channel of type 'ExampleAnomalies' (1 artifact) at 0x7a1bdc6df640

from cassandra.cluster import Cluster
cluster = Cluster(['127.0.0.1'])
session = cluster.connect('emotion_detection')
rows = session.execute('SELECT * FROM image_labels')

```

ExampleValidator

```

: context.show(example_validator.outputs['anomalies'])

Artifact at /tmp/tfx-interactive-2024-05-12T12_47_30.457696-ip1zycrf/ExampleValidator/anomalies/4

'train' split:

No anomalies found.

'eval' split:

No anomalies found.

```

o Preprocessing and Feature Engineering:

TFX Transform component is used for data preprocessing and feature engineering, ensuring that data is in the correct format for training.

```

def create_tf_example(image_path, annotations, class_int):
    # Load the image
    image = Image.open(image_path)
    image = image.resize((640, 640)) # Resize if not already 640x640

    # Encode the image to JPEG
    img_byte_arr = io.BytesIO()
    image.save(img_byte_arr, format='JPEG')
    image_bytes = img_byte_arr.getvalue()

    # Normalize bounding box coordinates
    xmin = [anno['xmin'] / image.width for anno in annotations]
    xmax = [anno['xmax'] / image.width for anno in annotations]
    ymin = [anno['ymin'] / image.height for anno in annotations]
    ymax = [anno['ymax'] / image.height for anno in annotations]
    filename = image_path.split('/')[-1].encode('utf-8')
    classes_text = [anno['class'].encode('utf-8') for anno in annotations]
    classes = [class_id for class_id in class_int]

    data = {
        'image/height': dataset_util.int64_feature(image.height),
        'image/width': dataset_util.int64_feature(image.width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(image_bytes),
        'image/format': dataset_util.bytes_feature(b'jpeg'),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmin),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmax),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymin),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymax),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }

    return tf.train.Example(features=tf.train.Features(feature=data))

```

```

_transform_module_file = '_transform_component.py'

%%writefile {_transform_module_file}
import tensorflow_transform as tft
import tensorflow as tf

def preprocessing_fn(inputs):
    """Preprocess input columns into transformed columns."""
    outputs = inputs

    return outputs

Overwriting _transform_component.py

transform = tfx.components.Transform(
    examples=example_gen.outputs['examples'],
    schema=schema_gen.outputs['schema'],
    module_file=os.path.abspath(_transform_module_file))
context.run(transform, enable_cache=True)

```

StatisticsGen

```
statistics_gen = tfx.components.StatisticsGen(
    examples=example_gen.outputs['examples'])
context.run(statistics_gen, enable_cache=True)
```

▼ ExecutionResult at 0x7a0f8c236dc0

.execution_id 2

.component ►StatisticsGen at 0x7a0eb5d830a0

.component.inputs

['examples'] ►Channel of type 'Examples' (1 artifact) at 0x7a0ebdb5a90

.component.outputs

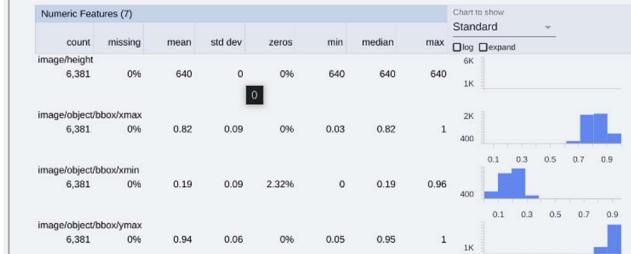
['statistics'] ►Channel of type 'ExampleStatistics' (1 artifact) at 0x7a0eb5d83160

```
context.show(statistics_gen.outputs['statistics'])
```

Artifact at /tmp/tfx-interactive-2024-05-13T22_13_58-400683-egr8fug/StatisticsGen/statistics/2

'train' split:

Sort by Feature order ▾ Reverse order Feature search (regex enabled)
Features: int(3) float(4) string(5)



Sort by Feature order ▾ Reverse order Feature search (regex enabled)
Features: int(3) float(4) string(5)

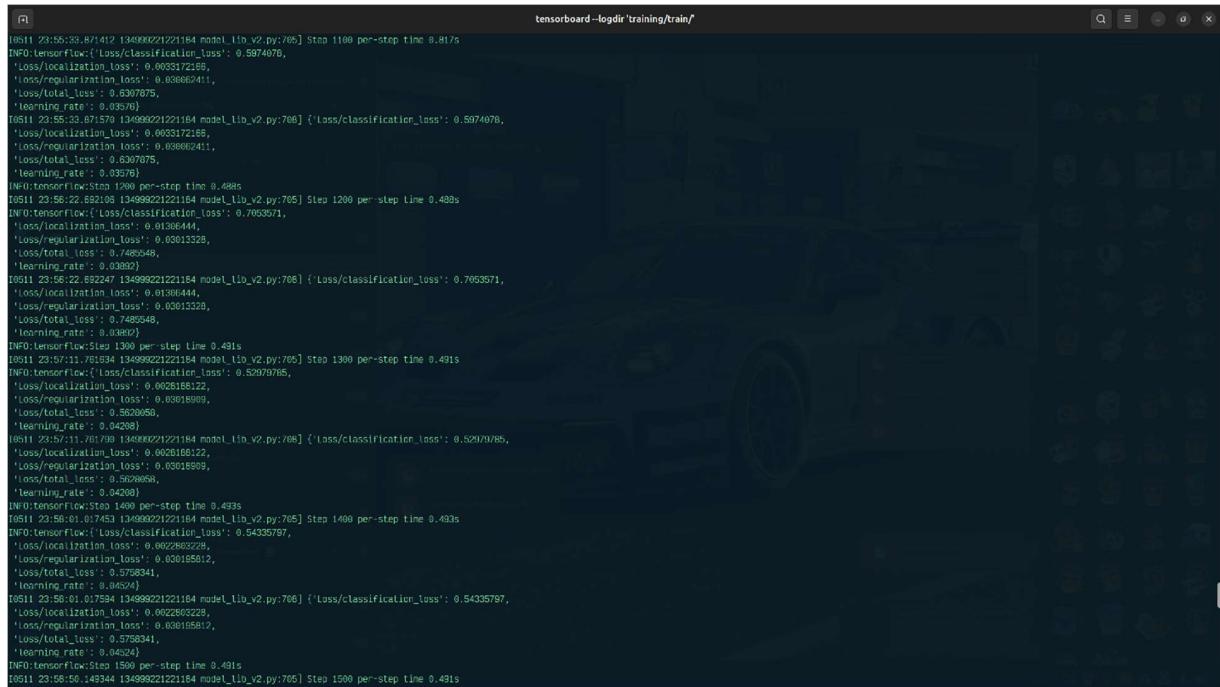


2. Integration of model training and offline evaluation into the ML pipeline / MLOps Platform

TFX Trainer and Evaluator components are integrated for model training and performance evaluation, ensuring the model is trained and assessed rigorously before deployment.

```
trainer = tfx.components.Trainer(  
    module_file=os.path.abspath(_trainer_module),  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    schema=schema_gen.outputs['schema'],)  
context.run(trainer, enable_cache=True)
```

Result:



The screenshot shows the TensorBoard interface with the path "tensorboard --logdir 'training/train'" selected. The main area displays a graph of training metrics over time. The x-axis represents the number of steps, ranging from 0 to 1500. The y-axis represents the value of the metric. There are four primary data series: 'Loss/classification_loss' (blue line), 'Loss/localization_loss' (orange line), 'Loss/regularization_loss' (green line), and 'Loss/total_loss' (red line). All metrics show a general downward trend as the number of steps increases. The 'Loss/total_loss' metric starts at approximately 0.5974 and ends at approximately 0.4915. The 'Loss/classification_loss' metric starts at approximately 0.5974 and ends at approximately 0.4915. The 'Loss/localization_loss' metric starts at approximately 0.0039 and ends at approximately 0.0226. The 'Loss/regularization_loss' metric starts at approximately 0.0039 and ends at approximately 0.0039. A legend on the right side of the graph identifies these series. The overall plot shows a clear learning process where the total loss is decreasing while individual components like classification and localization losses remain relatively stable or slightly increase.

```

nightstalker@night-stalker:~/Projects/yolov8-emotions
```

```

bashjupyter.sh          nightstalker@night-stalker:~/Projects/yolov8-emotions
Class Images Instances Box(P R mAP50 mAP50-95) 59/59 [00:18:00:00, 3.11it/s]
all 1887 1918 0.624 0.717 0.733 0.67

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
46/50 7.07G 0.3422 0.7397 1.049 12 640: 100% [02:28:00:00, 2.76it/s]
Class Images Instances Box(P R mAP50 mAP50-95) 100% 59/59 [00:18:00:00, 3.13it/s]
all 1887 1918 0.632 0.694 0.724 0.659

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
47/50 7.07G 0.3438 0.7355 1.047 12 640: 100% [02:28:00:00, 2.75it/s]
Class Images Instances Box(P R mAP50 mAP50-95) 100% 59/59 [00:18:00:00, 3.11it/s]
all 1887 1918 0.67 0.802 0.735 0.669

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
48/50 7.08G 0.3558 0.7162 1.045 12 640: 100% [02:29:00:00, 2.75it/s]
Class Images Instances Box(P R mAP50 mAP50-95) 100% 59/59 [00:18:00:00, 3.11it/s]
all 1887 1918 0.668 0.696 0.736 0.671

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
49/50 7.07G 0.3312 0.711 1.042 12 640: 100% [02:29:00:00, 2.75it/s]
Class Images Instances Box(P R mAP50 mAP50-95) 100% 59/59 [00:18:00:00, 3.11it/s]
all 1887 1918 0.658 0.71 0.742 0.676

Epoch GPU mem box_loss cls_loss dfl_loss Instances Size
50/50 7.08G 0.3369 0.8772 1.037 12 640: 100% [02:30:00:00, 2.75it/s]
Class Images Instances Box(P R mAP50 mAP50-95) 100% 59/59 [00:18:00:00, 2.92it/s]
all 1887 1918 0.676 0.888 0.744 0.68

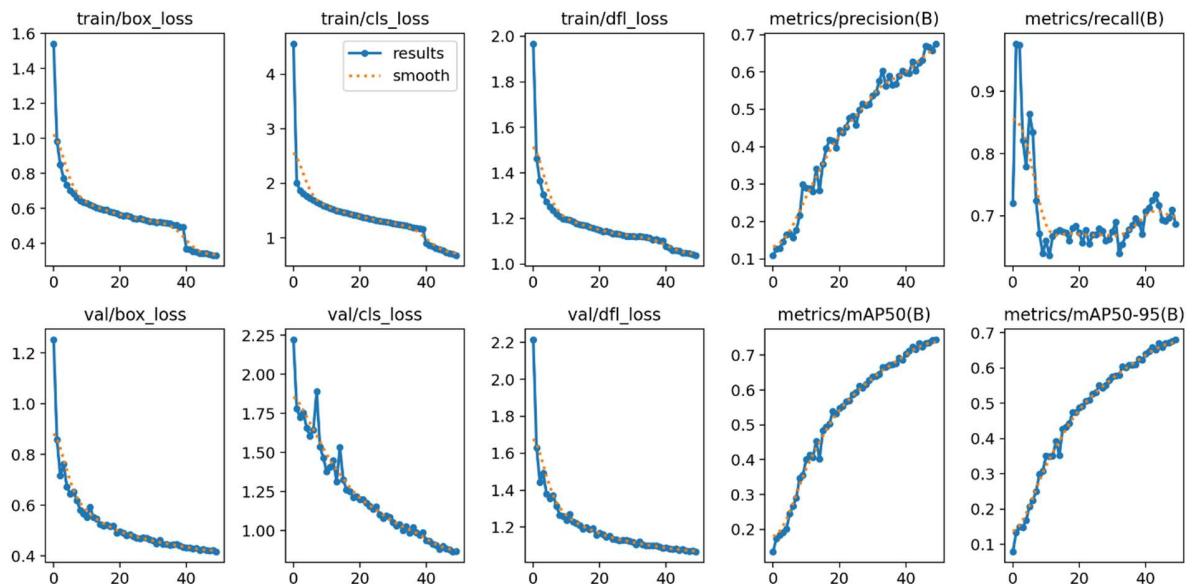
50 epochs completed in 2,360 hours.
Optimizer stripped from runs/detect/train3/weights/best.pt, 52.9MB
Optimizer stripped from runs/detect/trains/weights/best.pt, 52.9MB

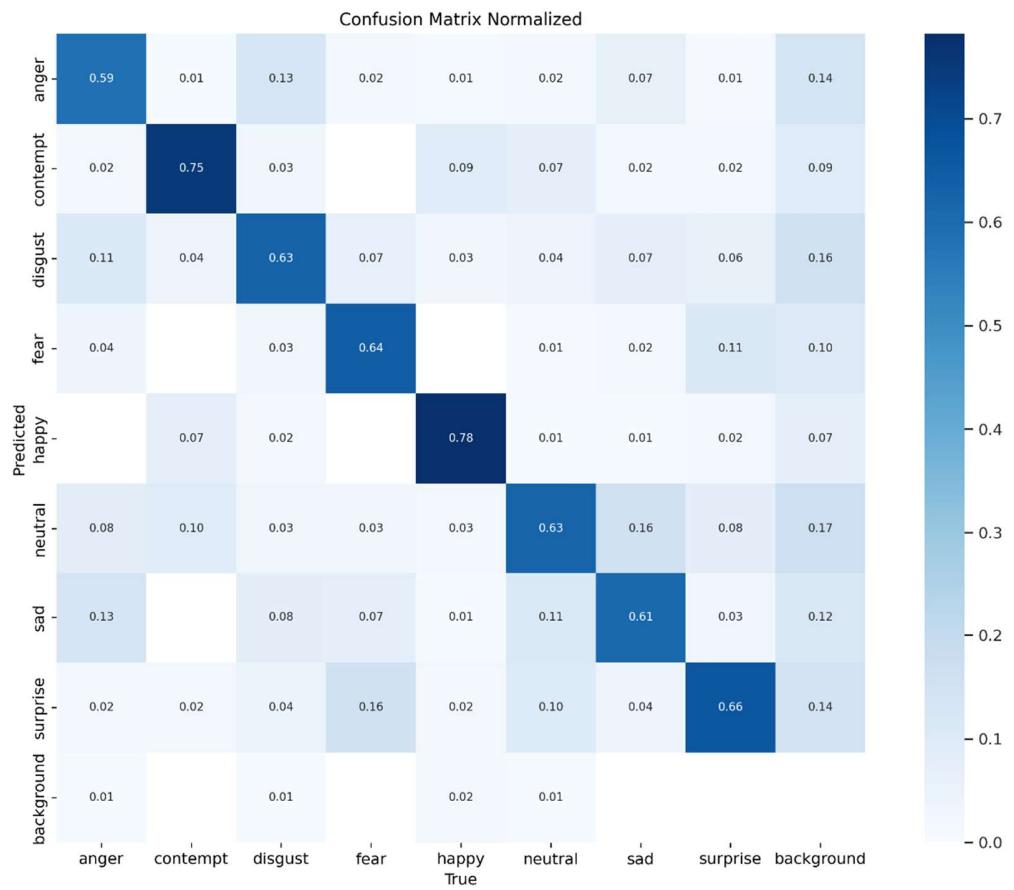
Validating runs/detect/train/weights/best.pt...
 ultralytics YOLOv8.0.12g - Python-3.8.16 torch-2.0.1+cu117 CUDA-8 (NVIDIA GeForce RTX 3070 Laptop GPU, 7967MHz)
Model summary (fused): 218 layers, 2584432 parameters, 0 gradients
Class Images Instances Box(P R mAP50 mAP50-95) 59/59 [00:19:00:00, 2.9it/s]
all 1887 1918 0.674 0.688 0.744 0.68
anger 1887 242 0.686 0.807 0.69 0.597
contempt 1887 241 0.737 0.78 0.822 0.771
disgust 1887 256 0.619 0.854 0.699 0.844
fear 1887 215 0.725 0.888 0.765 0.893
happy 1887 228 0.8 0.791 0.872 0.81
neutral 1887 251 0.576 0.701 0.694 0.531
sad 1887 235 0.604 0.809 0.668 0.804
surprised 1887 253 0.646 0.897 0.745 0.897

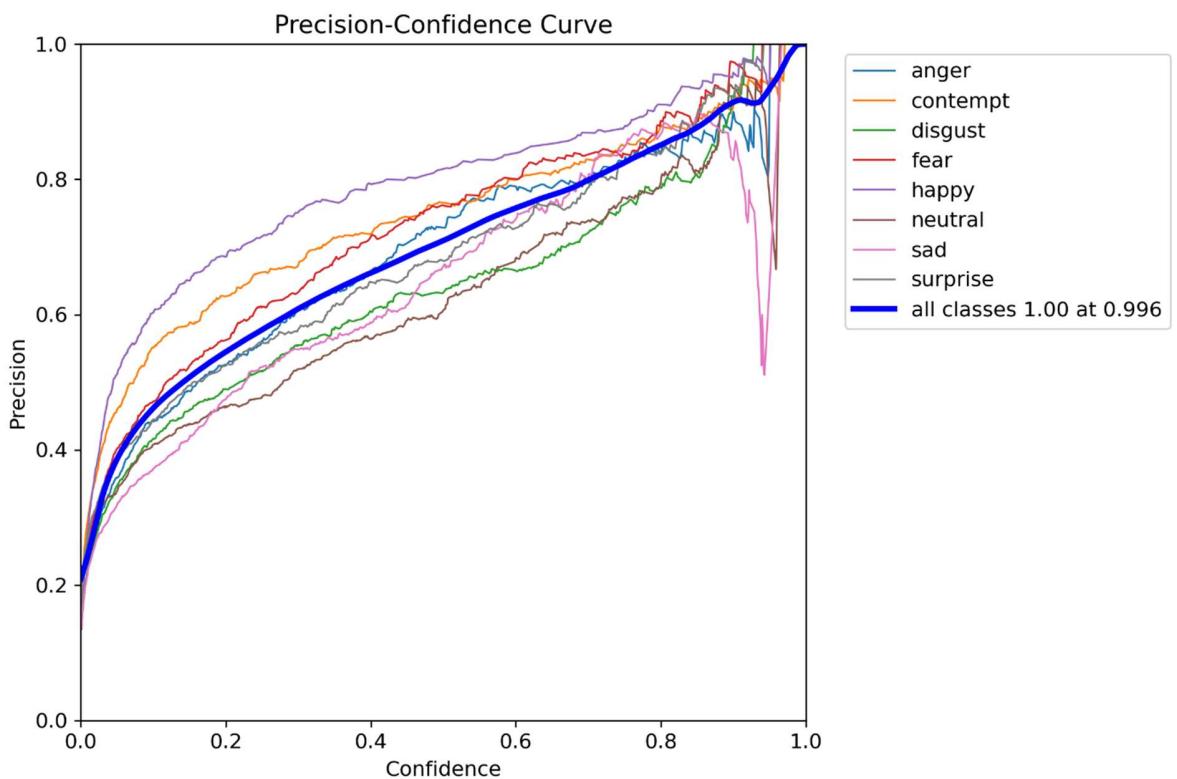
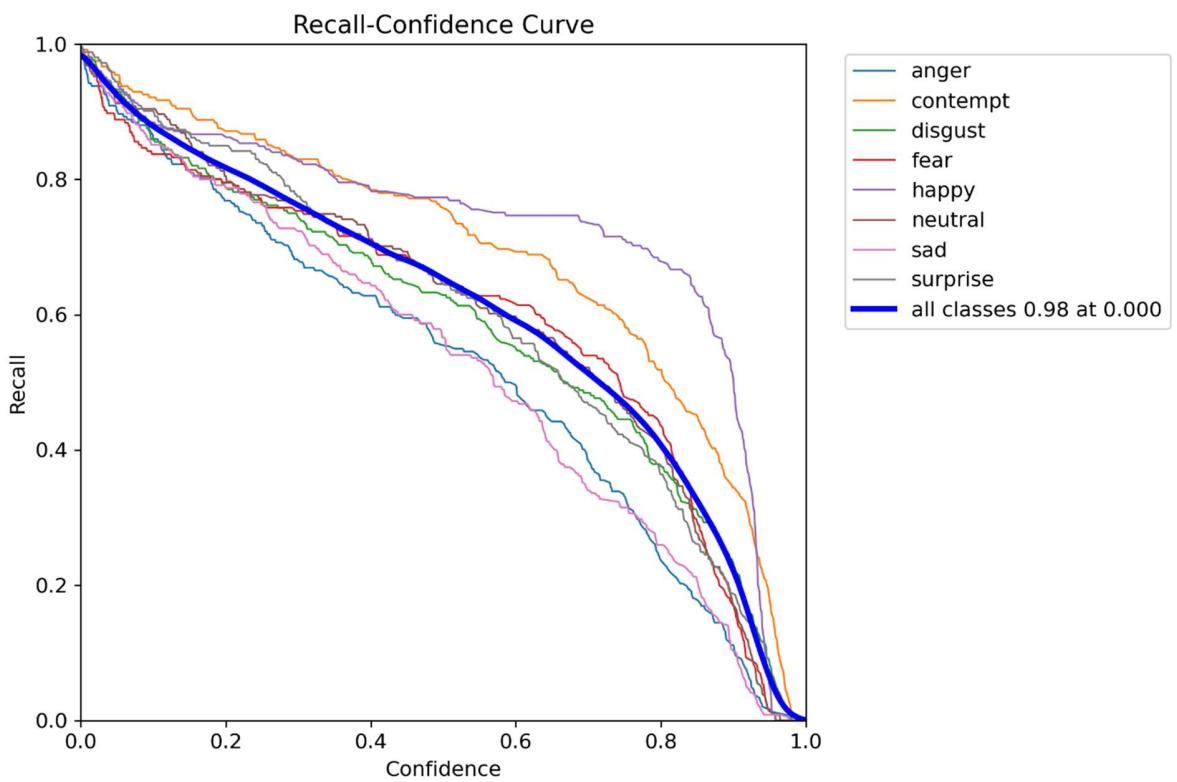
Speed: 0.1ms preprocess, 7.7ms inference, 0.0ms loss, 0.3ms postprocess per image
Results saved to runs/detect/train
```

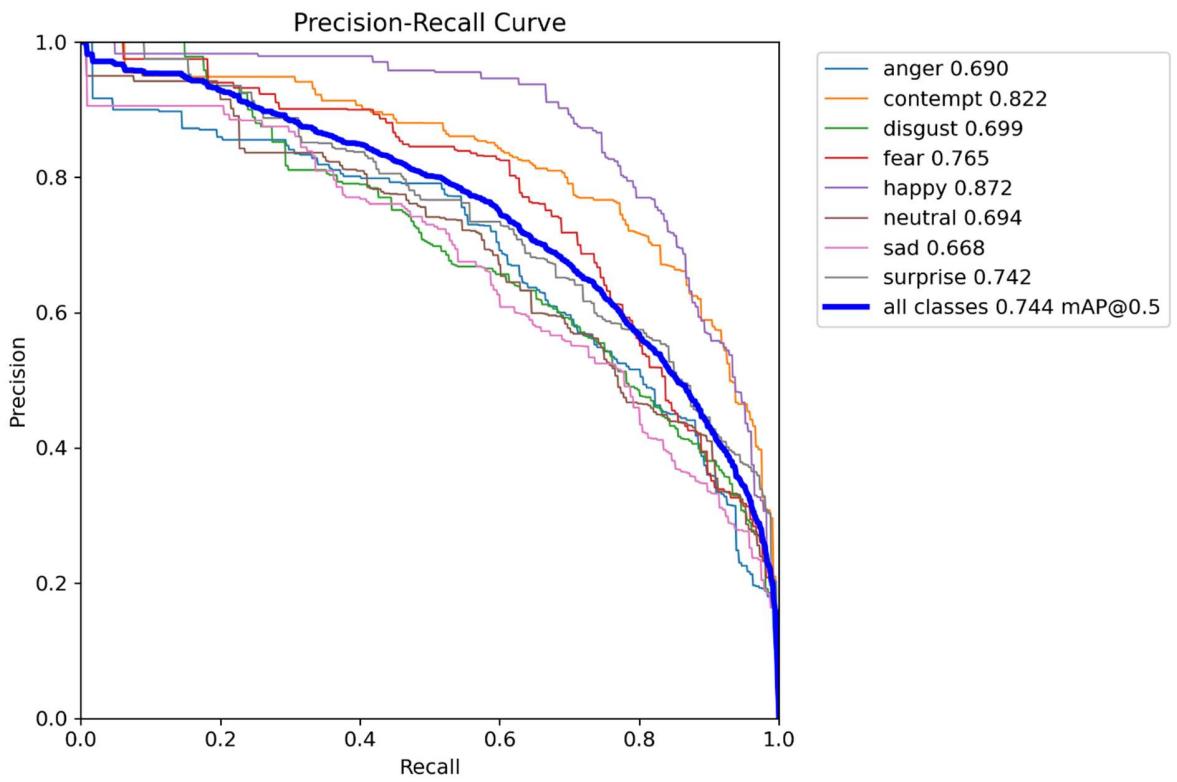
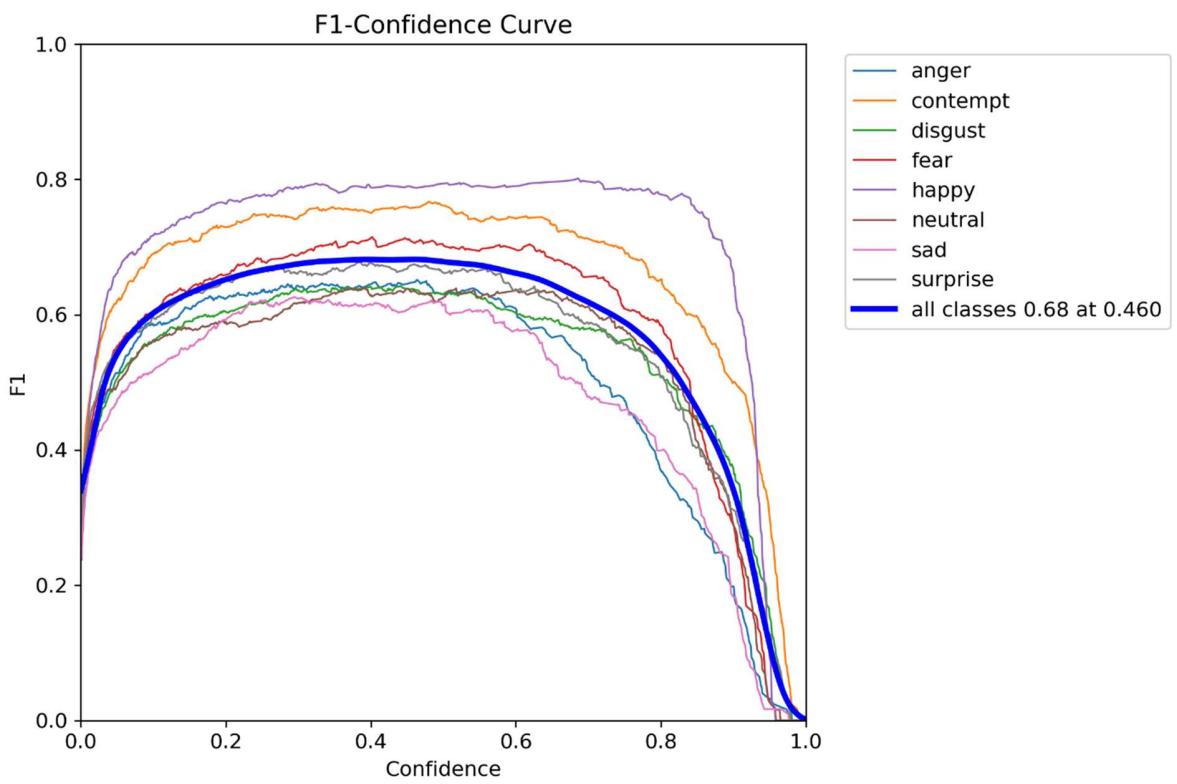
3. Development of model behavioral tests

Behavioral tests are designed to verify the model's functionality and performance under various conditions, ensuring robustness and reliability. I've used the library pytest for this step.













4. Energy efficiency measurement

Monitors and measures the model's energy consumption during training and inference to assess and optimize the system's energy efficiency.

```

nightstalker@night-stalker:~/Projects/yolov8-emotions
bash jupyter.sh
nightstalker@night-stalker:~/Projects/yolov8-emotions
nightstalker@night-stalker:~/Projects/yolov8-emotions

50 epochs completed in 2.069 hours.
Optimizer scripted from runs/detect/train/weights/last.pt, 52.0MB
optimizer scripted from runs/detect/train/weights/best.pt, 52.0MB

Validating runs/detect/train/weights/hest.pt...
ultralytics YOLOv8.1.0+ Python 3.8.16 torch-2.0.1+cu114 CUDA-8 (NVIDIA GeForce RTX 3070 Laptop GPU, 7GB/MIB)
Model summary (fused): 2.8 layers, 25644392 parameters, 0 gradients
  Class   Images Instances   Box(F1)   AP@50   AP@50-95%   mAP@50-95%
    all     1833      0.518   0.860   0.860   0.860
    anger   1887      0.729   0.868   0.867   0.868
    disgust  1887      0.741   0.737   0.738   0.737
    fear    1887      0.753   0.819   0.854   0.859
    disgust  1887      0.753   0.725   0.698   0.705   0.698
    fear    1887      0.753   0.725   0.698   0.705   0.698
    happy   1887      0.225      0.8   0.701   0.672   0.61
    neutral 1887      0.261   0.378   0.701   0.694   0.631
    sad    1887      0.233   0.364   0.899   0.898   0.894
    surprise 1887      0.233   0.549   0.867   0.742   0.687
Specs: 0.1ms preprocess, 7.7ms inference, 0.0ms postprocess per image
Results saved to runs/detect/train

yolo@nightstalker:~/Projects/yolov8-emotions
yolo@nightstalker:~/Projects/yolov8-emotions
Mon May 13 22:57:44 2024
|
| NVIDIA-SMI 500.64.15    Driver Version: 500.64.15    CUDA Version: 12.4 |
|=====================================================================|
| GPU  Name        Persistence-M | Bus-Id   Disp.A  Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr-Usage/Cap| Memory-Usage | GPU-Util  Compute M.  MIG M. |
|=====================================================================|
|  0  NVIDIA GeForce RTX 3070...  Off       00000000:01:00.0  N/A |
|  N/A  38C  P8    12W / 80W  6432MiB / 8192MiB  0%  Default  N/A |
|                                         MIG M. |
| Processes:
| GPU  ID  CI   PID  Process name         GPU Memory Usage
|  0   N/A  N/A  3263  /usr/lib/xorg/xorg          4MiB
|  0   N/A  N/A  58399  /opt/nightstalker/miniconda3/envs/tfx/bin/python  6416MiB
|
yolo@nightstalker:~/Projects/yolov8-emotions
yolo@nightstalker:~/Projects/yolov8-emotions

```

3. Model Deployment (Milestones 5-6, 35%)

3.1 ML System Architecture

- Drawing with architecture highlights:



3.2 Application development

- **Model service development:**

```
docker pull tensorflow/serving
docker run -p 8501:8501 --name=my_model_serving -v "/models/model/yolo-v8" -e MODEL_NAME=model tensorflow/serving
```

- **Front-end client development:**

```
function App() {
  const webcamRef = useRef(null);
  const canvasRef = useRef(null);

  // Main function
  const runCoco = async () => {
    const net = await tf.loadGraphModel('https://directionstfod.s3.au-syd.cloud-object-storage.appdomain.cloud/model.json')

    // Loop and detect hands
    setInterval(() => {
      detect(net);
    }, 16.7);
  };
}
```

3.3 Integration and Deployment

- **Packaging and containerization:**

```
docker build -t my-model-service .
docker run -d -p 8501:8501 my-model-service
```

- **Integration with a CI/CD Pipeline:**

The CI/CD framework, orchestrated with GitHub Actions, is in the deepest of this structure. Running at every single update in data or change in the code base, this automated pipeline will trigger all the related workflows. For orchestration of the various tasks, rebuilding of Docker containers automatically, in case of changes in dependencies, pushing in a registry of the said containers, and deployment in production, the GitHub Actions will be utilized. This integration is key to ensure that the least possible downtime is incurred and that the model benefits from continuous updates with as little human intervention as possible.

TFX is the project's machine learning backbone, from data ingestion with ExampleGen down to model validation and serving with components such as Evaluator and Pusher. Each of the TFX components is designed within data transformation, model training, and deployment in a clear, reproducible way across environments, which is enforced by containerization in Docker.

In addition, Docker makes TFX even more robust as it wraps each component in a separate container, hence managing dependencies and configurations with simplicity. Furthermore, this scaling in deployment will also lead to isolation of processes, hence increasing safety and stability in the machine-learning pipeline. In essence, orchestration between TFX, Docker, and GitHub Actions is a best practice in MLOps that presents the way forward for the realization of a resilient, scalable, and sustainable infrastructure toward pushing machine learning models. It's a strategic way to achieve model adaptation and improvements in productions while keeping up with the principles of agility and excellence in machine learning deployments.

- **Hosting the application:**

I have hosted the application on Digital Ocean:

The screenshot shows the IBM Cloud Cloud Object Storage interface. On the left, there's a sidebar with options like Instances, Integrations, Endpoints, Documentation, and Billing. The main area shows a bucket named 'directionstfod'. A warning message says 'All objects in this bucket have public view access.' Below it, a message says 'If you're seeing more usage than expected, versions count towards your usage or you may have incomplete uploads [Learn more](#)'. There are four objects listed: 'group1-shard1of3.bin' (4.0 MB, last modified 2024-05-02 12:15 AM), 'group1-shard2of3.bin' (4.0 MB, last modified 2024-05-02 12:15 AM), 'group1-shard3of3.bin' (3.2 MB, last modified 2024-05-02 12:15 AM), and 'model.json' (556.7 KB, last modified 2024-05-02 12:15 AM). At the bottom, there's a 'Drag and drop files (objects) here or click to upload' button.

3.4. Model Serving and online testing

- **Model serving runtime:**

The model is served using TensorFlow Serving, which is optimized for both high performance and high availability. TensorFlow Serving provides out-of-the-box support for serving ML models, handling requests in real-time with low latency. It's set up to automatically load new versions of the model as they are trained and pushed to the serving directory, ensuring that the most accurate and up-to-date model is always in use.

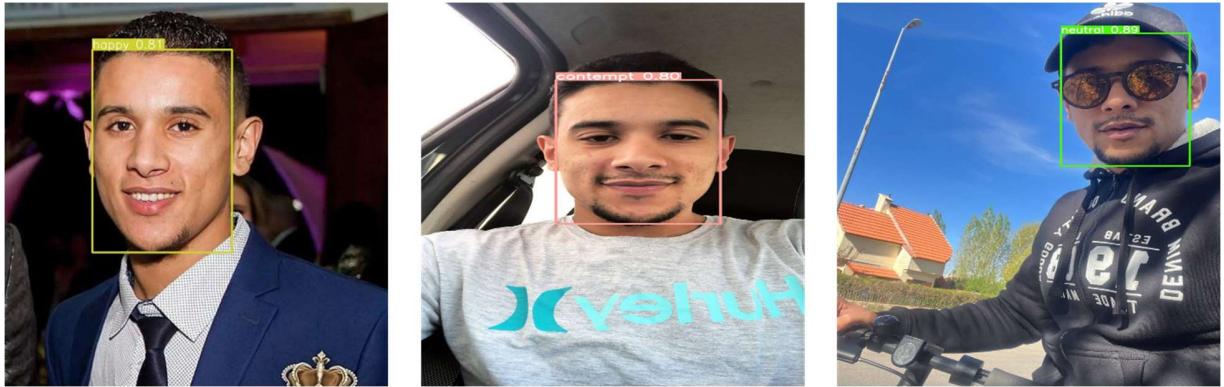
```
pusher = tfx.components.Pusher(  
    model=trainer.outputs['model'],  
    model_blessing=evaluator.outputs['blessing'],  
    push_destination=tfx.proto.PushDestination(  
        filesystem=tfx.proto.PushDestination.Filesystem(  
            base_directory=_serving_model_dir)))  
context.run(pusher, enable_cache=True)
```

- Serving mode (batch, on demand to a human, on demand to a machine):

```
def serve_real_time(image):  
    """Handle real-time requests."""  
    response = requests.post("http://localhost:8501/v1/models/model:predict", json={  
        return response.json()
```

- Online testing (A/B Testing, Bandit):





1. Detection Quality:

- **TensorFlow Object Detection API:** While it provides good detection within its targeted areas (upper face), it may generate multiple bounding boxes that can complicate the output, especially in scenarios needing streamlined and minimal error processing. This could impact its performance negatively if the goal is clear and singular detection per face.
- **YOLOv8 Detection:** It consistently delivers high-quality detections with a single bounding box per face, closely matching the ground truth. This reliability makes it preferable in applications where accurate full-face detection is critical, such as in security systems or comprehensive emotion analysis.

2. Model Robustness:

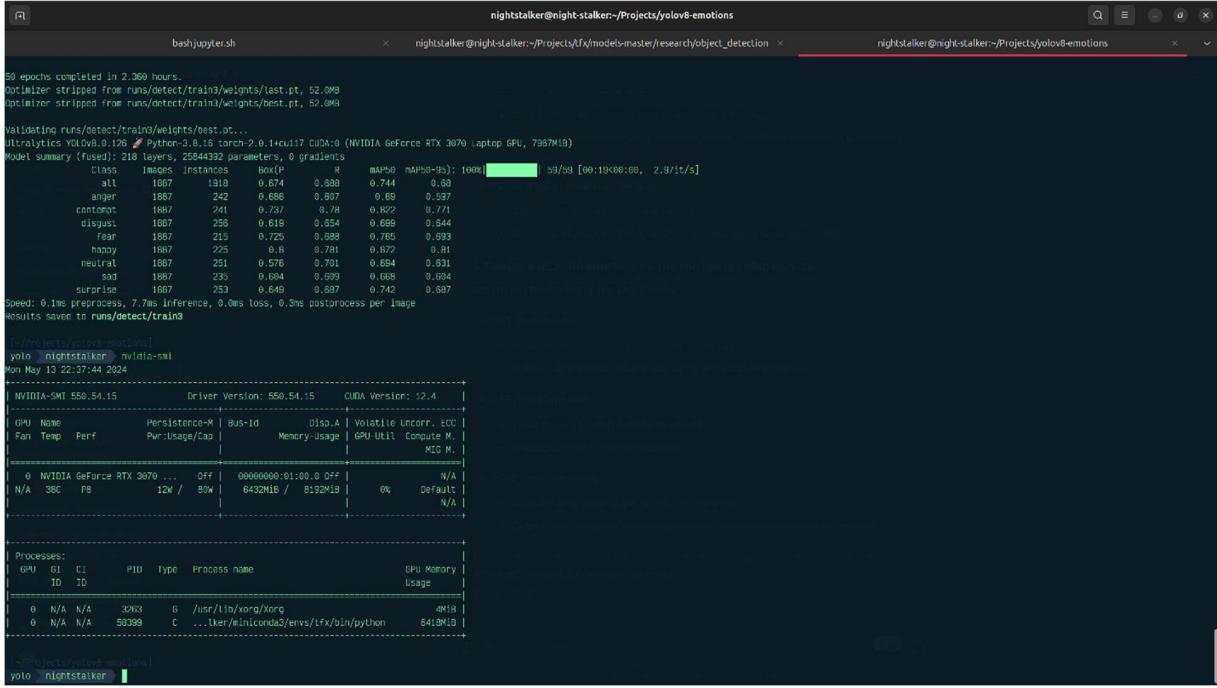
- Since YOLOv8 provides close-to-ground truth detection without multiple bounding box issues, it suggests greater robustness in varied real-world conditions compared to TensorFlow, which might struggle with overlapping or closely situated faces.

3. Potential for False Positives/Negatives:

- The TensorFlow model, due to its tendency to create multiple bounding boxes, might have a higher rate of false positives. This is important in scenarios where precision is more crucial than recall.
- YOLOv8, with its single, accurate bounding box approach, likely exhibits fewer false positives and negatives, providing more reliable results overall.

Monitoring and Continual Learning (milestone 7, 25%)

4.1. (2 pts) Resource Monitoring



The terminal window displays three tabs: bash/jupyter.sh, nightstalker@night-stalker:~/Projects/yolov8-emotions, and nightstalker@night-stalker:~/Projects/yolov8-emotions. The main tab shows the results of a 50 epoch training run, including a model summary table and performance metrics. The bottom tab shows GPU usage information for an NVIDIA GeForce RTX 3070.

```
50 epochs completed in 2.369 hours.
Optimizer stripped from runs/detect/train3/weights/last.pt, 52.0MB
Optimizer stripped from runs/detect/train3/weights/best.pt, 52.0MB

Validating runs/detect/train/weights/best.pt...
ultralytics YOLOv8.0.12b Python-3.8.15 torch-2.0.1cu117 CUDA-0 (NVIDIA GeForce RTX 3070 Laptop GPU, 7967MiB)
Model Summary (fused): 210 layers, 2594432 parameters, 0 gradients
COCO   Images  Instances  BoxIoU  mAP50  mAP50-95%  100%  Speed
all    1867    1318    0.674  0.888  0.744  0.88  2.9f/s
anger   1867    242    0.686  0.897  0.88  0.937
contempt  1867    241    0.737  0.78  0.822  0.771
disgust   1867    291    0.619  0.854  0.698  0.844
fear     1867    215    0.725  0.888  0.785  0.833
happy    1867    225    0.8  0.791  0.872  0.81
neutral   1867    251    0.576  0.791  0.694  0.831
sad      1867    235    0.604  0.899  0.668  0.894
surprise  1867    253    0.649  0.887  0.742  0.897

Speed: 0.0ms preprocess, 7.7ms inference, 0.0ms loss, 0.0ms postprocess per image
Results saved to runs/detect/train

yolo@night-stalker:~/Projects/yolov8-emotions$ nvidia-smi
Mon May 13 22:37:44 2024
+-----+
| NVIDIA-SMI 550.54.15    Driver Version: 550.54.15    CUDA Version: 12.4 |
| GPU  Name        Persistence-M  Bus-Id     Disp.A | Volatile Uncorr. ECC | |
| Fan  Temp  Perf  Pwr/Usage/Cap | Memory-Usage | GPU-Util Compute M. |
|          % /   /       /   / |      / /       / |      / /       / |
+-----+
| 0  NVIDIA GeForce RTX 3070 ... Off  00000000:01:00.0 Off | N/A |
| N/A 38C P8    12W / 80W | 6432MiB / 6192MiB | 0% Default N/A |
+-----+
Processes:
GPU GI CI PID Type Process name          GPU Memory Usage
ID ID
0 N/A N/A 3263 B /usr/lib/xorg/Xorg 4MiB
0 N/A N/A 58399 C ...taker@night-stalker:~/Projects/yolov8-emotions/lfx/bin/python 6418MiB
+-----+
```

- To calculate the energy consumed:

Since power is typically calculated in watts (Joules per second), we need to convert the time from hours to seconds. Then, we multiply the power usage by the time in seconds to get the total energy in Joules.

Given:

$$Power (P) = 80 \text{ watts}$$

$$Time (T) = 2.36 \text{ hours}$$

Conversion:

$$\text{Time in seconds} = (2.36 \times 3600) \text{ seconds/hour} = 8496 \text{ seconds}$$

Energy Consumption:

$$\text{Total energy}(E) = Power \times Time = (80 \text{ watts} \times 8496 \text{ seconds})$$

Calculate the Total Energy:

$$E = 80 \text{ W} \times 8496 \text{ s} = 679,680 \text{ Joules}$$

This is the energy consumed in Joules. If we wish to convert this into more common energy billing units like kilowatt-hours (kWh), you can use the *conversion*($1 \text{ kWh} = 3,600,000 \text{ J}$).

Conversion to kWh:

$$\text{kWh} = \frac{679,680 \text{ J}}{3,600,000 \text{ J/kWh}} \approx 0.189 \text{ kWh}$$

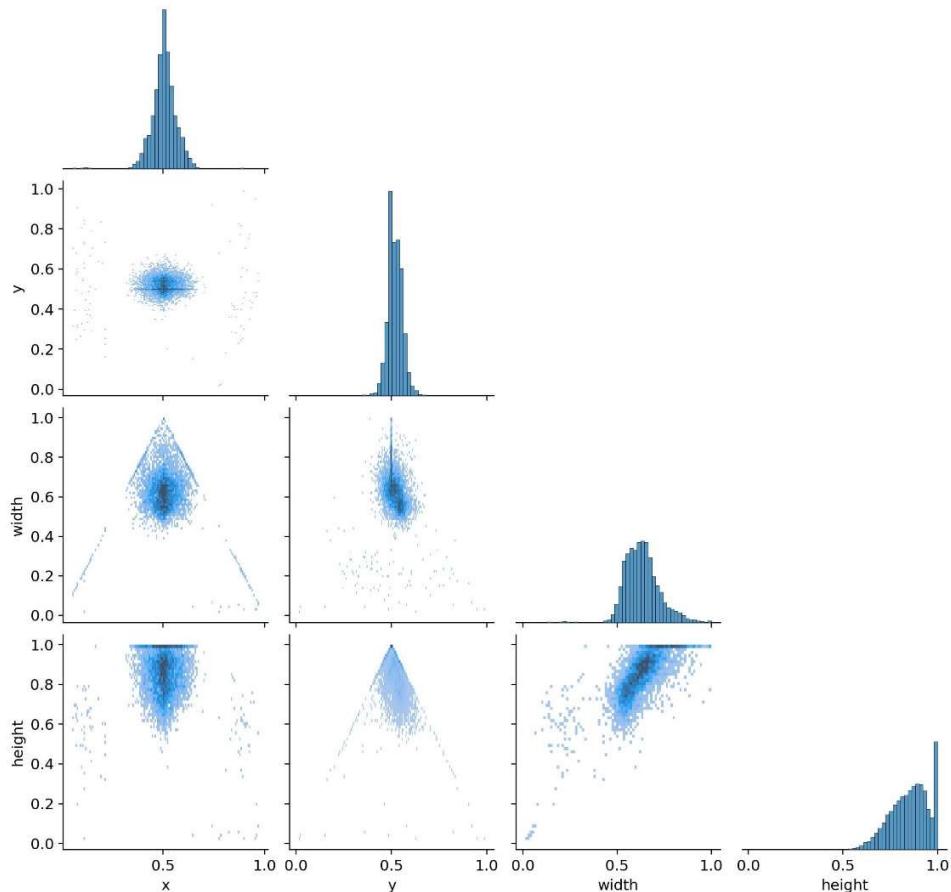
Thus, we consumed approximately 0.189 kWh of energy during the training session.

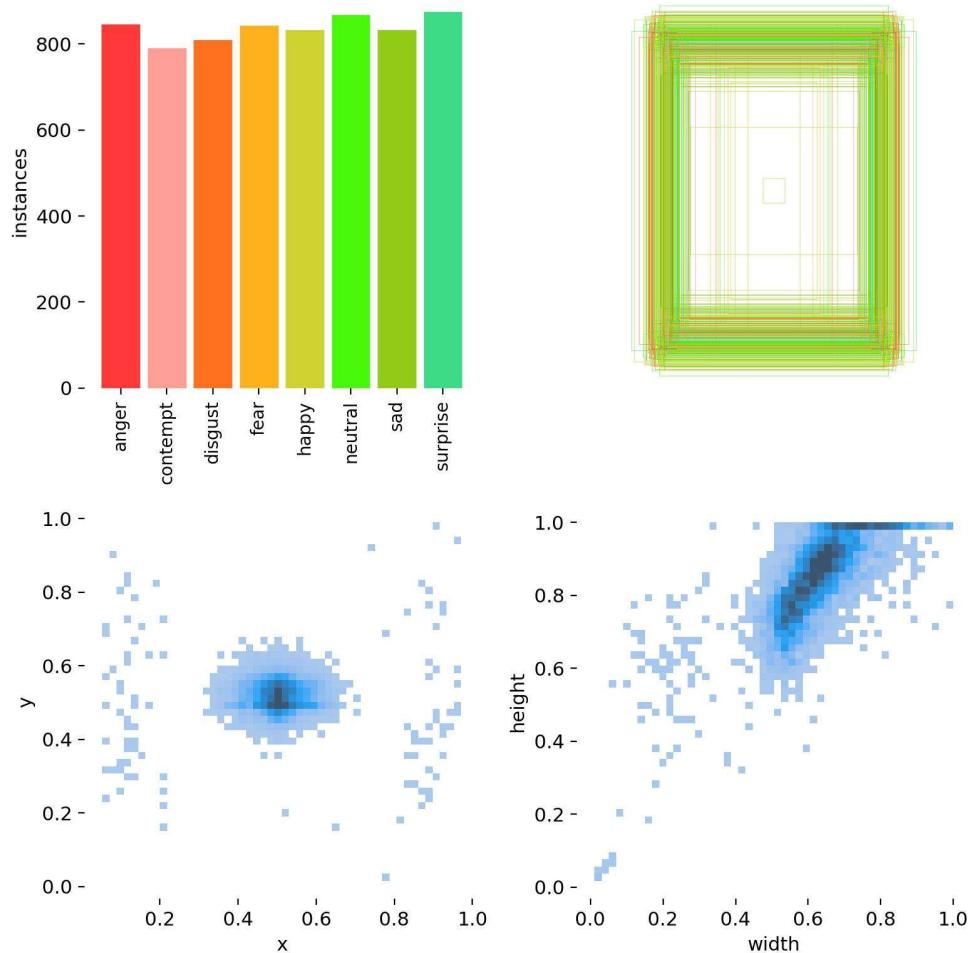
Cost Calculation:

If you want to know how much this cost, you'd multiply the kWh by the cost per kWh from your local utility provider. For example, if the cost is \$0.12 per kWh:

$$\text{Cost} = 0.189 \text{ kWh} \times 0.12 \text{ per kWh} = 0.023$$

4.2. (10 pts) Model Performance Monitoring or data distribution drift monitoring





To ensure sustained model performance, continuous monitoring for model degradation and data drift is implemented. We utilize Evidently AI to detect significant shifts in data distribution and monitor predictive performance, helping to maintain the model's accuracy over time.

```
# dashboard for monitoring data drift
from evidently.dashboard import Dashboard
from evidently.dashboard.tabs import DataDriftTab

data_drift_dashboard = Dashboard(tabs=[DataDriftTab()])
data_drift_dashboard.calculate(reference_data, production_data, column_mapping=None)
data_drift_dashboard.save('/save/data_drift/dashboard.html')
```

4.3. (10 pts) Continual Learning: CT/CD pipeline

The CT/CD pipeline ensures the model is continuously updated with new data, automatically retraining, and deploying without manual intervention. I used GitHub Actions

to automate the retraining process whenever new data is available or periodically based on a schedule.

```
ct.yaml
1  name: CI/CD Pipeline for YOLOv8 Model Retraining
2
3  on:
4    push:
5      branches:
6        - main
7      paths:
8        - "data/**"
9
10 jobs:
11   train-yolov8:
12     runs-on: ubuntu-latest
13     container:
14       image: ultralytics/yolov8:latest # Ensure you are using the correct Docker image
15
16     steps:
17       - name: Checkout repository
18         uses: actions/checkout@v2
19
20       - name: Set up Python environment
21         uses: actions/setup-python@v2
22         with:
23           python-version: "3.9"
24
25       - name: Install dependencies
26         run: |
27           pip install -r requirements.txt
28
29       - name: Sync data
30         run: |
31           echo "Syncing dataset..."
32           rsync -avz your-data-storage:/path/to/data ./data
33
34       - name: Train YOLOv8 model
35         run: |
36           echo "Running YOLOv8 training script..."
37           python train.py --img 640 --batch 16 --epochs 50 --data ./data/dataset.yaml --weights best.pt --cache
38
39       - name: Save trained model
40         run: |
41           echo "Saving trained model..."
42           cp runs/train/exp/weights/best.pt ./models/
43
44       - name: Deploy model
45         if: ${{ github.ref == 'refs/heads/main' }}
46         run: |
47           echo "Deploying model..."
48           scp ./models/best.pt deploy@nightstalker:/path/to/models/
49           ssh deploy@nightstalker 'bash deploy_new_model.sh'
```

4.4. (3 pts) Pipeline orchestration

Apache Airflow orchestrates the workflow of the ML pipeline, managing tasks like data ingestion, preprocessing, model training, evaluation, and deployment. This coordination ensures that ML pipeline operates smoothly and efficiently.

```

    default_args = {
        'owner': 'ml_team',
        'depends_on_past': False,
        'start_date': datetime(2024, 05, 01),
        'email': ['y.maatougui@auai.ma'],
        'email_on_failure': False,
        'email_on_retry': False,
        'retries': 1,
        'retry_delay': timedelta(minutes=5),
    }

    dag = DAG('ml_workflow',
               default_args=default_args,
               description='Machine Learning workflow',
               schedule_interval=timedelta(days=1))

    t1 = PythonOperator(task_id='train_model',
                        python_callable=train_model,
                        dag=dag)

    t2 = PythonOperator(task_id='deploy_model',
                        python_callable=deploy_model,
                        dag=dag)

    t1 >> t2

```

5. Responsible AI (milestone 8-optional, for later, 15% bonus)

5.1 Evaluation Beyond Accuracy

- **Audit Model for Bias:**
- **Model Explainability and Interpretability:**

6. Conclusion

- **Summary of Achievements:**
Developed a sophisticated ML system for face sentiment detection, leveraging advanced models and a robust pipeline, with high accuracy in detecting emotional states.

- **Lessons Learned:**
Emphasized the critical role of high-quality data and the impact of efficient data pipeline management in achieving high model performance.
- **Future Directions :**
Plan to enhance the model's capability to recognize a broader range of emotions and subtle expressions, integrate newer and more efficient ML models, and expand the application areas of the system.

References

5. TensorFlow Documentation

TensorFlow is extensively used for model training and serving in your project. The official TensorFlow documentation provides comprehensive insights into its APIs, functionalities, and best practices.

- Website: [TensorFlow](#)

6. TFX User Guide

TFX (TensorFlow Extended) is critical for implementing the full ML pipeline in your project. This guide covers all components of TFX, including setup, usage, and examples.

- Website: TFX User Guide

7. Prometheus Documentation

Used for monitoring system metrics within my project, Prometheus's documentation offers details on setup, configuration, and query execution.

- Website: Prometheus

8. Evidently AI

For monitoring model performance and data distribution drift, evidently AI's tools are used. Their documentation provides guidelines on how to integrate their tools with existing machine learning pipelines.

- Website: [Evidently AI](#)

9. Apache Airflow Documentation

As the orchestration tool in your project, understanding Airflow is crucial. This documentation provides information on how to define, execute, and monitor DAGs.

- Website: [Apache Airflow](#)

10. GitHub Actions Documentation

For implementing CI/CD pipelines that facilitate continual learning and deployment, GitHub Actions is a core component. This resource offers comprehensive guides on how to set up and manage workflows.

- Website: [GitHub Actions](#)

11. Docker Documentation

Docker is used for creating, deploying, and running applications by using containers. The documentation provides a solid understanding of Docker commands and best practices.

- Website: Docker

12. Cassandra Documentation

Since Cassandra is used for storing features, the official documentation can help understand its data model, query language (CQL), and setup procedures.

- Website: [Apache Cassandra](#)

13. TensorFlow Serving with Docker

For deploying models into production, TensorFlow Serving with Docker provides a robust and scalable framework. This article offers a deep dive into configuration and optimization.

- Website: [Serving TensorFlow Models](#)