Gmail François Decarie <frank.decarie@gmail.com>

# Patent Software Licenses - Implementation details

3 messages

**François Decarie** <frank.decarie@gmail.com> Thu, Feb 15, 2024 at 3:40 PM
To: charles.grant@cwgrant.ca

We clarify that:

Internal Use: The GPL-3.0 licensed software components are used strictly for internal purposes within MOCCUS MAXIMUS INC., and not distributed externally in any form. As such, we maintain that our use of these components does not trigger the copyleft provisions of the GPL-3.0 license that would require redistribution of our proprietary source code or the source code of the GPL-3.0 licensed components themselves.

Service Provision: Our paid services, which may utilize the GPL-3.0 licensed components as part of their operational infrastructure, do not constitute distribution of the software in accordance with the GPL-3.0 license terms. We provide access to these services without granting access to the underlying software, in compliance with the GPL-3.0 license.

Source Code Integrity: MOCCUS MAXIMUS INC. retains the integrity of our proprietary source code, which remains confidential and proprietary. We undertake measures to ensure that our use of GPL-3.0 licensed components does not necessitate the disclosure or distribution of our proprietary source code.

Compliance Measures: MOCCUS MAXIMUS INC. continuously monitors compliance with all software licenses, including the GPL-3.0, and takes proactive steps to ensure adherence to their terms. This includes regular audits, compliance training for our staff, and consultation with legal experts specializing in software licensing.

Future Changes: Should our use of GPL-3.0 licensed software components change in a manner that might affect our compliance with the license terms, we commit to promptly re-evaluating our use and distribution practices to ensure ongoing compliance.

This statement reflects MOCCUS MAXIMUS INC.'s current use and management of GPL-3.0 licensed software components and our dedication to upholding the legal and ethical standards of software use and distribution.

GNU General Public License v3.0 (GPL-3.0)
YOLOv7: A state-of-the-art, real-time object detection system, available under GPL-3.0, which allows for commercial use but with strict requirements, such as the obligation to disclose source code when distributing the software or derivatives thereof.
yolov7-pose-estimation, a YOLOv7 derivative also available under GPL-3.0.
DeepSORT: An algorithm for real-time multi-object tracking, also under GPL-3.0. Similar to YOLOv7, using DeepSORT in a product would necessitate compliance with the GPL-3.0 terms, including source code disclosure.

GNU General Public License v3.0
Permissions of this strong copyleft license are conditioned on making available complete source code of licensed works and modifications, which include larger works using a licensed work, under the same license. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights.

Apache License 2.0
TensorFlow: An open-source software library for machine learning, allowing for commercial use, modification, and distribution.

Meta SAM: Segment Anything Model, Facebook, Meta AI research: Apache 2.0, permitting commercial usage and distribution with few restrictions.
Keras: A high-level neural networks API, capable of running on top of TensorFlow, also covered under the Apache 2.0 license.

Creative Commons Attribution 4.0 International License (CC BY 4.0)
COCO Dataset: A large-scale object detection, segmentation, and captioning dataset. The CC BY 4.0 license allows for sharing and adaptation, even commercially, as long as appropriate credit is given, a link to the license is provided, and indicated if changes were made.

BSD License
OpenCV: A library of programming functions mainly aimed at real-time computer vision. The BSD license permits the use of OpenCV in commercial products and applications with minimal restrictions.
scikit-learn: A machine learning library for Python, licensed under a BSD license, allowing commercial use and distribution.
License Implications for Commercial Use

Apache 2.0: This license is very permissive, allowing for commercial use, modification, and distribution without the need to disclose source code. It requires a notice that includes the copyright and license disclaimer.

CC BY 4.0: This license allows for almost unrestricted use as long as attribution is given to the creators. It's quite permissive and suitable for both academic and commercial projects.

BSD License: Known for its minimal restrictions, allowing for the free use and redistribution of licensed software, even in proprietary software, without the need to disclose source code.

Attribution is hereby provided.



Portions of this software are based on the work of the OpenCV Foundation, TensorFlow, Keras, and scikit-learn contributors. OpenCV is licensed under a BSD license, TensorFlow and Keras are licensed under the Apache 2.0 License, and scikit-learn is licensed under a BSD license. For more details, see the respective licenses of these projects.

This software includes contributions from Francois Decarie.

Implementation Details:

Step 1: Input Video
Implementation: Use OpenCV, a popular open-source computer vision library, to capture video streams from various sources. OpenCV provides functions like cv2.VideoCapture to read video files or capture video streams from cameras.

Step 2: Object Detection
Algorithm: YOLOv7, an evolution of the YOLO (You Only Look Once) family, known for its speed and accuracy in real-time object detection.
Implementation: Utilize the pre-trained YOLOv7 model available on GitHub. Use OpenCV or PyTorch to load the model and apply it to each frame of the video stream to detect objects and obtain bounding boxes.

Initialization: The training begins with the YOLOv7 model initialized with weights pre-trained on the COCO dataset. These weights provide a solid foundation, as the COCO dataset includes a wide range of objects, allowing the model to have a good initial understanding of various features and patterns common in object detection tasks.

Advantages: Starting with COCO weights accelerates the training process for the custom dataset, as the model doesn't need to learn basic features from scratch. This approach often leads to better generalization and quicker convergence, especially when the custom dataset is relatively small or when the objects of interest are somewhat represented in the COCO dataset.

Preparing the Custom Dataset
Annotation: The custom dataset is meticulously annotated, possibly using tools like Meta SAM for precise and efficient labeling. The annotations must be in a format compatible with YOLOv7, typically involving bounding boxes and class labels for each object of interest.

Data Augmentation: To enhance the model's robustness and ability to generalize, the custom dataset is augmented with various transformations, such as scaling, cropping, and flipping. This step is crucial for improving the model's

performance, especially in diverse real-world conditions.

Custom Training Process
Configuration: The YOLOv7 model is configured for the custom training process. This includes adjusting hyperparameters such as the learning rate, batch size, and number of epochs, tailored to the size and complexity of the custom dataset.

Transfer Learning: The training process employs transfer learning, where the model initially trained on COCO weights is further trained (fine-tuned) on the custom dataset. This involves freezing the early layers of the model that capture basic features and only training the deeper layers to adapt to the new dataset.

Fine-Tuning: As the model trains on the custom dataset, it gradually learns to detect the custom objects with increasing accuracy. Fine-tuning allows the model to adjust its weights specifically to the features and patterns present in the custom dataset, improving its performance on these specific tasks.

Evaluation and Iteration: Throughout the training process, the model's performance is continuously evaluated using a validation set separate from the training data. This evaluation helps in tuning the hyperparameters and deciding when the model has sufficiently learned to detect the custom objects. The process may involve multiple iterations of training and evaluation to achieve optimal performance.

Deployment and Inference
Model Export: Once training is complete and the model performs satisfactorily on the custom dataset, the final model weights are saved. This model is now specifically tuned to detect objects from both the COCO dataset and the custom dataset with high accuracy.

Real-World Application: The custom-trained YOLOv7 model is deployed in the intended environment, where it processes video streams or images to detect and localize objects. Thanks to the initial COCO weights and subsequent fine-tuning on the custom dataset, the model is adept at handling a wide variety of objects and scenarios, tailored to the specific requirements of the application.

Step 3: Image Segmentation
Enhanced Dataset Preparation with Meta SAM
Annotation with Meta SAM: The Meta SAM tool is utilized to annotate the images in the training dataset. This involves marking the precise boundaries of objects within the images to create segmentation masks. Meta SAM's interface and features are designed to facilitate detailed and accurate annotations, even for complex images with multiple overlapping objects.

Quality and Consistency: The use of Meta SAM ensures that the annotations are of high quality and consistency across the dataset. This is crucial for segmentation tasks, where the model's ability to precisely segment objects depends heavily on the accuracy and consistency of the training data annotations.

Efficiency in Annotation: Meta SAM's features, such as semi-automatic annotation tools and the ability to manage large datasets, speed up the annotation process. This efficiency is particularly beneficial when preparing large datasets required for training deep learning models like YOLOv7_segmentation.

Exporting Annotations: Once the images are annotated, Meta SAM allows for the export of segmentation masks in a format compatible with the training process. These masks serve as the ground truth for training the segmentation model, guiding the model to learn the precise contours and shapes of the objects.

Training the Custom YOLOv7_segmentation Model
With the dataset annotated using Meta SAM, the custom YOLOv7_segmentation model is trained on this accurately labeled data. The high-quality segmentation masks produced by Meta SAM play a critical role in this phase, providing the model with clear, precise examples of how objects should be segmented from the background.

Impact on Model Performance
The use of Meta SAM for dataset annotation directly impacts the performance of the custom-trained YOLOv7_segmentation model. The precision and consistency of the annotations contribute to the model's ability to accurately segment objects in new images, reflecting the effectiveness of Meta SAM in the dataset preparation process.

By integrating Meta SAM into the workflow for preparing the training dataset, the overall segmentation system benefits from enhanced data quality, leading to more accurate and reliable segmentation outcomes from the custom-

trained YOLOv7_segmentation model. This precision is essential for the subsequent steps in the machine vision system, where the segmented objects are further analyzed for feature extraction, object tracking, and other advanced processing tasks.

Step 4: Feature Extraction
Selection of a Pre-Trained CNN
A pre-trained CNN, ResNet, is chosen due to its proven effectiveness in capturing rich feature representations from images. ResNet is particularly favored for its deep residual learning framework, which facilitates the training of networks that are substantially deeper than those used previously.

Preparation of Segmented Objects
The segmented objects, obtained from the previous segmentation step, are prepared for feature extraction. This preparation typically involves resizing the segmented images to the input size expected by the pre-trained CNN and normalizing the pixel values.

Feature Extraction Pipeline
The pre-trained ResNet model is loaded without its top layer, making it act as a feature extractor rather than a classifier. This configuration allows the network to output a feature map instead of a class prediction.

The segmented objects are passed through the ResNet model. As the images propagate through the network, each layer captures increasingly complex features. Early layers might capture basic patterns such as edges and textures, while deeper layers capture high-level features that are more abstract and object-specific.

The output at a certain layer is extracted as the feature vector for each segmented object. The choice of layer from which to extract features depends on the level of abstraction required for the regression task. For instance, features from the final convolutional layers contain very high-level information and are typically used for tasks requiring understanding of complex structures within the images.

Integration with Regression Model
The extracted feature vectors serve as the input to a regression model. This model is designed to map the high-dimensional feature vectors to a continuous output space, corresponding to the pose or any other quantitative attribute of the object being analyzed.

The regression model could be a simple linear regressor, a more complex neural network, or any other suitable machine learning model, depending on the complexity of the task and the nature of the output variables.

Training and Fine-Tuning
In cases where the pre-trained features are not sufficiently discriminative for the specific regression task, fine-tuning can be employed. This involves unfreezing some of the top layers of the pre-trained CNN and continuing the training process on the new dataset, allowing the network to adjust its weights to better suit the specific task.

The regression model is trained using a dataset of segmented objects with known output values. The model learns to predict the output values from the feature vectors extracted by the CNN.

This approach leverages the powerful feature extraction capabilities of pre-trained CNNs, such as ResNet, to obtain meaningful representations of segmented objects, which are then used to perform regression tasks. By utilizing deep features extracted from a well-established architecture, the system benefits from the extensive knowledge captured by these networks during their training on large, diverse image datasets.

Step 5: Feature Normalization
Implementation: Use scikit-learn, a machine learning library in Python, to apply normalization techniques such as Min-Max scaling or Z-score normalization to the extracted features.

Step 6: Image Regression
Designing the CNN Architecture
Define the Model: Start by defining a sequential or functional model in Keras. The choice between a sequential and functional API depends on the complexity of your network. For more complex networks with multiple inputs or outputs, the functional API is preferred.

Layer Configuration: Add convolutional layers (Conv2D) with activation functions, typically ReLU, to extract features from the input images. Pooling layers (MaxPooling2D) are used to reduce the spatial dimensions of the feature maps, and dropout layers (Dropout) can be added to prevent overfitting.

Regression Layer: The final layers of the network should be designed for regression. Flatten the output from the convolutional layers using a Flatten layer, followed by one or more dense (Dense) layers. The last Dense layer should have a number of units corresponding to the dimensions of the pose you're estimating (e.g., if predicting x, y coordinates for keypoints, you'd need 2 units per keypoint) and should use a linear activation function since it's a regression task.

Model Compilation: Compile the model using an optimizer like Adam, and choose a loss function suitable for regression, such as mean squared error (MSE). Metrics like MAE (Mean Absolute Error) can also be tracked to monitor performance.

Preparing the Dataset
Dataset Collection: Gather a dataset containing images of the objects along with their annotated poses. The annotations could be in the form of coordinates representing the pose of the object in the image.

Pre-processing: Normalize the image data and the pose annotations to ensure smooth training. Image pixel values are typically normalized to the range [0, 1] by dividing by 255, and pose annotations might need scaling based on the model's expected output range.

Data Augmentation: Apply data augmentation techniques to increase the diversity of the training data and help the model generalize better. Keras' ImageDataGenerator can be used for real-time data augmentation.

Training the Model
Define Callbacks: Use Keras callbacks like ModelCheckpoint to save the best model during training and EarlyStopping to prevent overfitting by stopping the training when the validation loss stops improving.

Train the Model: Train the model on the prepared dataset, using a validation split to monitor the model's performance on unseen data. Adjust the number of epochs and batch size based on your dataset size and the complexity of the model.

Pose Estimation
Model Inference: Use the trained model to predict poses on new images. Pre-process these images in the same way as the training images before feeding them into the model.

Post-processing: The predicted poses might need scaling back to the original image dimensions if they were normalized or scaled during the pre-processing step.

Tools and Libraries
Keras: Used for designing, compiling, and training the CNN model. It provides a high-level API that is user-friendly and integrates seamlessly with TensorFlow.
TensorFlow: The backend for Keras, providing a comprehensive ecosystem for machine learning and deep learning tasks.
Numpy: Essential for data manipulation, especially for pre-processing the dataset and post-processing the model predictions.
Matplotlib/OpenCV: Useful for visualizing the predictions by overlaying the estimated poses on the images for qualitative analysis.

Step 7: Counting
Basic Counting Method
Bounding Box Detection: The first step involves using an object detection algorithm to identify objects in each frame of the video stream. This results in a set of bounding boxes, each corresponding to a detected object.

Counting Logic: In Python, the counting is straightforward once bounding boxes are detected. For each frame processed by the object detection algorithm, the number of bounding boxes is counted, which directly corresponds to the number of objects detected in that frame. This can be achieved using a simple length function in Python, for example, len(bounding_boxes), where bounding_boxes is a list containing all detected boxes in a frame.

Advanced Counting with Tracking
Integration of Tracking Algorithm: To account for object movements and occlusions, a tracking algorithm like DeepSORT is integrated. DeepSORT combines motion and appearance information to robustly track objects across frames, which is particularly useful in crowded or dynamic scenes.

Tracking Initialization: After detecting objects and their bounding boxes in the initial frame using an object detection model, these detections are fed into the tracking algorithm to initialize the tracking process.

Object ID Assignment: DeepSORT assigns a unique ID to each tracked object. This ID persists across frames as long as the object is detected or until the tracker determines the object has left the scene.

Handling Occlusions and Movements: The tracking algorithm updates object positions in each frame based on detections and predictions. When occlusions occur, the tracker uses motion prediction to estimate the occluded object's position. The appearance model within DeepSORT helps re-identify objects after they become visible again, ensuring accurate counting even in challenging scenarios.

Counting with Tracking: The actual counting in this approach involves keeping a tally of unique object IDs seen over time. A cumulative count can be maintained to track the total number of unique objects seen throughout the video. Alternatively, a count per frame can be maintained, which requires considering objects entering or leaving the scene and only counting visible objects in each frame.

Implementing in Python: This advanced counting logic is implemented in Python by interfacing with the tracking algorithm's output. After each frame is processed, the list of active tracks (each with a unique ID) is updated. The count of objects can be derived from the number of active tracks, and adjustments can be made based on the specific counting requirements (e.g., counting all objects ever seen vs. counting only currently visible objects).

Tools and Libraries
Object Detection and Tracking Libraries: Libraries such as OpenCV for object detection and specialized tracking libraries that can implement or interface with DeepSORT are essential. PyTorch or TensorFlow might also be used depending on the implementation details of the chosen object detection and tracking algorithms.
Data Structures for Tracking: Efficient data structures and algorithms are crucial for managing and updating the list of tracked objects, especially in real-time applications or when processing high-resolution videos.

Step 8: Keypoint and Signature Extraction with Custom-Trained yolov7-pose-estimation

Data Preparation
A dataset is compiled, consisting of images or video frames that contain the objects and scenarios relevant to the application. This dataset is annotated meticulously to mark the keypoints of interest on each object.
The data undergoes pre-processing to ensure uniformity in image sizes and pixel values. Data augmentation techniques may be applied to enhance the dataset's diversity and robustness.

Customization of yolov7-pose-estimation
The architecture of the yolov7-pose-estimation network is adjusted to suit the specific requirements of the keypoints being detected. This might involve modifications to the network's layers to capture relevant features effectively.
A training pipeline is established, incorporating a suitable loss function, often a combination of confidence map losses for individual keypoints and part affinity fields for connections between keypoints. The choice of optimizer and regularization techniques is also crucial in this phase.

Model Training
The custom yolov7-pose-estimation model is trained using the prepared dataset. Training involves careful monitoring to avoid overfitting and adjustments to the model based on its performance.
The model's effectiveness is validated using a separate set of data, ensuring it generalizes well to new, unseen images or video frames.
Integration for Real-Time Inference
The trained model is optimized for real-time inference, which may include techniques such as model quantization, pruning, or conversion to a format suitable for the deployment environment.
Post-processing steps are implemented to refine the keypoints detected by the model. This can include smoothing the keypoints over time to ensure consistency across frames.

Evaluation and Refinement
The performance of the keypoint detection module is evaluated within the broader context of the machine vision system. Adjustments are made based on the system's requirements and the module's performance in real-world scenarios.
An ongoing process of improvement is maintained, with new data collected, especially from cases where the system did not perform as expected, to retrain and refine the model.
Throughout these phases, tools and libraries such as Caffe, TensorFlow, or PyTorch are utilized for model development and training. OpenCV is employed for video frame pre-processing and post-processing tasks. Annotation tools play a

crucial role in the initial dataset preparation phase, ensuring accurate and consistent marking of keypoints.

Step 9: Re-identification with Siamese Networks
Siamese Network Architecture: A Siamese network consists of two identical subnetworks joined at their outputs. The purpose is to take in pairs of input (in this case, feature signatures extracted from objects in images) and compute a similarity score between them.

Feature Extraction for Re-Identification:
Utilize the feature vectors extracted from objects detected in previous steps as input to the Siamese network. These features can be derived from the segmentation or keypoint extraction steps.
Ensure that the features are normalized and prepared in a way that is compatible with the Siamese network input requirements.
Training the Siamese Network:

Collect or create a training dataset consisting of pairs of feature vectors with labels indicating whether the pair is the same object (positive pair) or different objects (negative pair).
Train the Siamese network on this dataset. The network learns to minimize the distance between feature vectors of the same object and maximize the distance between feature vectors of different objects. Contrastive loss or triplet loss functions are commonly used for this purpose.

Implementing the Re-Identification Logic:
For each object detected in the current frame, extract its feature vector and compare it with feature vectors of objects from previous frames using the Siamese network. The comparison yields a similarity score.
Set a threshold for the similarity score to determine whether two feature vectors are considered the same object. Objects with similarity scores above this threshold are treated as re-identifications of the same object.
Integration with Tracking:

Integrate the Siamese network-based re-identification process with an object tracking algorithm to maintain identity consistency across frames. This integration helps in cases where objects leave the frame and re-enter or when occlusions occur.
Optimization and Evaluation:

Continuously evaluate and optimize the Siamese network's performance using metrics such as accuracy, precision, and recall, specifically focusing on the re-identification task.
Adjust the training dataset, network architecture, and hyperparameters based on performance on a validation set to improve re-identification accuracy.

Step 10: Database with Blockchain

System Overview
Each pig in the system is assigned a unique identifier (pigID), which is linked to a sophisticated image-based signature profile. This profile is created using cutting-edge image ReID technology, capturing distinct visual features of each pig to ensure accurate and reliable identification over time.

Signature Profile and Growth Data
Image-Based ReID: The signature profile for each pig is derived from high-resolution images, capturing unique physical characteristics. Advanced algorithms analyze these images to create a distinctive digital signature for each pig, ensuring accurate identification through visual features alone.

Growth Curve Data: The database meticulously records the growth data of each pig, including weight measurements at specific timestamps. This data is encrypted and stored alongside the pigID and signature profile, providing a comprehensive growth history for each animal.

Security and Data Integrity
Encryption: All data within the system, including signature profiles and growth data, is encrypted using state-of-the-art encryption techniques. This ensures the confidentiality and security of sensitive information.
Blockchain Integration: The system leverages blockchain technology to store each pig's data as a unique digital token. This approach guarantees the non-fungibility and immutability of the records, with each data modification securely logged and verifiable.
Access Control: Robust access control mechanisms are in place, ensuring that only authorized personnel can access the database. This is further reinforced by detailed audit logs that track all interactions with the data.

Operational Highlights
Real-Time Monitoring: The system allows for real-time monitoring of each pig's growth progress, with automated alerts for significant milestones or deviations from expected growth patterns.
Data Analytics: Advanced analytics tools are integrated into the system, offering insights into growth trends, health indicators, and operational efficiencies.
User Interface: A user-friendly interface provides easy access to the database, enabling quick searches, data retrieval, and the generation of detailed reports based on the pigID, timestamped events, and growth data.

Compliance and Ethics
The system is designed with a strong commitment to compliance with data protection regulations and ethical standards, particularly concerning the use of image-based identification technologies.

This encrypted database represents a significant advancement in agricultural and livestock management, offering unparalleled accuracy, security, and reliability in tracking the identity and growth of pigs. Its implementation streamlines operations, enhances data-driven decision-making, and sets a new standard for ethical and responsible livestock management.

//Signed//
Francois Decarie,CD,B.Sc,M.Sc
CTO MOCCUS MAXIMUS
15 Feb 2024

---

**François Decarie** <frank.decarie@gmail.com> Wed, Feb 21, 2024 at 12:16 PM
To: Johnorange@sympatico.ca, charles.grant@cwgrant.ca

[Quoted text hidden]

---

**johnorange@sympatico.ca** <johnorange@sympatico.ca> Wed, Feb 21, 2024 at 12:18 PM
To: François Decarie <frank.decarie@gmail.com>, charles.grant@cwgrant.ca

Got it thanks

[Quoted text hidden]