Computer Vision - Fire detection

Hi! PARIS x Pyronear

François Soulier

Contents

1 Approaches

1.1 You Only Look Once (YOLO)

1.1.1 YOLOv5

The YOLOv5 model is a cutting-edge object detection model known for its high accuracy in identifying objects in images. It it built upon the YOLO (You Only Look Once) architecture, a single-stage detection model designed for real-time object detection. It is a state-of-the-art model that is widely used in the computer vision community, and it was pre-trained on the COCO dataset.

Consequently, this model is well-suited for the detection of forest fires in landscape images. In a real-world scenario, it is crucial to detect fires as quickly as possible to prevent them from spreading and causing massive damage. In addition, the YOLOv5 model offers high accuracy in locating object boundaries in images, which is essential to reduce false alarms (false positive predictions) and to ensure the early detection of fires.

1.1.2 Data selection

To run the YOLOv5 model training loop within a reasonable amount of time and using one T4 NVIDIA GPU on Google Colab, the training dataset was reduced to a subset containing only 10% of the original images.

1.1.3 Results on fire detection

The YOLOv5 model was trained on a fragment of the fire detection dataset.

Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
all	152	153	0.301	0.245	0.219	0.0941

Table 1: Performance metrics of the model

As shown in the **Table 1**, the trained model did not perform well on the fire detection task. The test was performed on 152 images containing 153 instances of fire.

1.2 Multi-Head detector

1.2.1 Description

This approach uses a model that is lighter than the YOLO architecture, and is composed of a ResNet-50 backbone with a multi-head detector. The model comprises a ResNet-50 backbone, followed by a regression head (based on LSTM layers) to predict object bounding boxes and a classification head (based on linear layers) to predict the presence of fire in each possible bounding box. For each sample, the output is a list of bounding boxes and an associated binary value indicating the presence of fire in each bounding box.

One main advantage of this model is that it can be trained on smaller computational resources and fine-tuned on larger datasets to improve performance. However, it only supports a fixed, limited number of bounding boxes per image, which can be a limitation in certain cases.

The loss function used for training is a combination of the cross-entropy loss for the classification and the mean square error (MSE) for the regression task.

1.2.2 Data preprocessing and augmentation

The images were processed according to the following steps:

- Reduce noise by applying smoothing and blurring filters (Gaussian, Median, Bilateral).
- Resizing to 224x224 pixels.
- Normalize the pixel values to the range [0, 1].
- Apply random color modifications (brightness, contrast, saturation).

1.2.3 Performance (ResNet-50 & LSTM)

- **ResNet-50 Backbone**: The ResNet-50 model, pre-trained on the ImageNet dataset, is a state-of-the-art convolutional encoder that efficiently extracts visual features. As a relatively lightweight model, it is suitable for fast, low-resource training and can be used as a backbone in a transfer learning approach.
- **LSTM Regression Head**: The LSTM (Long Short-Term Memory) layers are used to predict the bounding boxes of objects in the image. LSTM layers are ideal for this task because they capture the "semantic" information of the feature maps extracted by the encoder, predicting bounding boxes with a global visual context.

1.2.4 Results on fire detection

The results are discussed directly in the notebook *fire_detection.ipynb*, with visualizations of the model predictions.

2 Improvements

To further improve model performance and enhance efficiency, the following improvements can be considered in one month of development time:

- **Hyperparameter tuning**: Optimize chosen hyperparameters using techniques such as grid search or random search.
- **Monitoring and logging**: Implement a monitoring and logging system to track model performance during training and deployment. The MLFlow library can be used to log all metrics, parameters, and artifacts.
- **Experiment tracking**: Use a tool such as Weights & Biases to track the experiments and visualize the results interactively.
- **Model ensembling**: Combine the predictions of multiple models to improve the overall performance and prediction reliability. For instance, this can be done by averaging the predictions of the models.
- **Large scale training**: Train the large YOLOv5 model on the entire dataset, potentially using distributed training on multiple GPUs or cloud-based clusters (e.g., AWS, GCP, Azure).
- **Explainability**: Visualize model predictions and feature maps to understand the model's decision-making process. The Grad-CAM can be used for visualization purposes.
- **Video processing**: Extend the algorithm to video processing, using an object tracking algorithm (e.g., Kalman filters) to detect fire in a sequence of frames.
- **Coninuous integration and deployment**: Implement CI/CD pipelines to automate the model training, evaluation, and deployment processes. They could be integrated with GitHub Actions or Gitlab CI/CD, running test suites.