CS230

Runway Recognition

Michael Hardt
Stanford University
mwhardt@stanford.edu

Carlos Querejeta
Stanford University
carlosq@stanford.edu

Grzegorz Kawiecki
Stanford University
kawiecki@stanford.edu

Abstract

The objective of the presented project is to develop a functionality that takes as input aerial images captured from a camera mounted on an aircraft in a forward-looking direction, then detects the presence of a runway in the image, and finally estimates its size and orientation. State-of-the-art deep learning approaches such as Yolo and semantic segmentation have been implemented and tested to achieve the objective. Training and configuration procedures are described, and experimental results are presented.

1 Introduction

The onset of autonomous aircraft requires new solutions to gain in redundancy and robustness. Accurate navigation is a critical aspect of flight. Current aircraft, in particular small platforms, cannot rely exclusively upon GPS signals, and image information can serve as an additional sensor for localization. In particular, this work concentrates upon the landing process when the aircraft's current position is not well known. The general direction of the runway is known, but not with precision. The developed functionality in this work identifies whether the runway is in view and estimates its size and orientation within the image. Then using known runway ground coordinates and characteristics (i.e., runway orientation and elevation), a flight control system can calculate accurately its distance and relative angle with respect to the runway, thereby serving as a landing aid system under nominal or emergency conditions.

2 Related Work

This functionality has been already addressed by multiple studies, e.g., [Tripathi18] or [Nazir18]. The latter has been performed using images from nose-mounted camera, processed using Sobel operator for edge detection, spectral residual saliency map (SRS), binary gradient mask and dilation. Runway position with respect to the aircraft was found using the pixel approximation technique. They have reported the accuracy of runway detection at 94%, with a processing time of 0.3sec.

3 Dataset and Features

The dataset consists of 1186 color images of runways from 16 different single-runway airports, taken from different distances and perspective angles, and with a resolution of 1024x768 pixels, later scaled and padded to 416x416. The dataset has been split as 80% training set, 10% dev set, and 10% test set. All runways were labeled with bounding boxes and pixel masks. As no labeled runway image dataset was publicly available we generated labeled images using two different sources:

1. Google Earth. We developed a small application to generate camera placemarks in different positions around the runways of the considered airports, and then saved and labeled such images manually. 371 images of the dataset come from this distribution.
2. The remaining 815 images were generated using a customized geospatial visualization environment built on top of osgEarth, with the capability of transforming images from online satellite orthoimagery databases, e.g. http://services.arcgisonline.com, to an image taken from a virtual camera at any given position and orientation. A random sampling of camera positions is taken within 2 and 8 km distance to the runway, and elevation angles between 10 and 45 degrees. With each runway 50 snapshots were taken plus another 10 snapshots in wrong directions to accumulate negative samples without runways.

Figure 3.1: Runway image (left) and ground truth mask and bounding box overlay (right)

4 Methods

Two different image recognition methods were tested for the purpose of runway recognition: Yolo and semantic segmentation. Yolo calculates bounding boxes around recognized objects in an image. Segmentation, on the other hand, estimates on a pixel level the object class. It is expected that more semantic information may be extracted from the latter approach, though it is supposedly harder to train and achieve good results. Both approaches represent alternatives for solving the problem. They can also be complementary in that the output of the first approach can be used to initialize the second.

4.1 Object Detection

The Yolo architecture (v3) [Redmon16, Redmon18] is selected accompanied by an existing pretrained network. Yolo was designed by formulating the object detection problem as a regression problem, estimating each object's bounding box coordinates and associated class probabilities, based upon a convolutional network architecture. The newest revision of Yolo (v3) uses a backbone called Darknet-53 composed of ConvNet layers and shortcut paths across layers, though no pooling layers nor fully connected layers are used. It incorporates the simultaneous detection at three different scales. It's based on a squared error loss for the bounding box coordinates and makes use of a multilabel classification rather than a softmax with cross-entropy error terms for the object confidence and class predictions, thus permitting multiple labels in a given box which is not possible with a softmax. Since we're interested in only one object, a runway, with simple features, we also consider Yolo's little brother, tiny-Yolo, based upon what is known as the Tiny Darknet backbone. This architecture has only 22 layers and uses a softmax for classification. In both cases, the native model resolution of 416x416 is used which provides a good balance between performance and time between iterations. We have also analyzed Yolo's performance for a reduced number of layers.

4.2 Segmentation

Segmentation estimates individually for each pixel whether it belongs to a given object. In the case of runways, potentially more precise information can be determined with regards to its size and orientation. The Deeplab v3+ architecture has been selected [Chen18] along with its most common backbone architecture, ResNet-101. It consists of an encoder-decoder architecture characteristic typical of the latest deep convolutional networks as well as spatial pyramid pooling.

5 Experiments / Results / Discussion

5.1 Object Detection

The runway detection experiments have been divided into two successive phases.

1. Train both Yolo and Tiny-Yolo.
2. Hyperparameter model search with Yolo and based upon transfer learning approach.

Both experiments have used as a starting point a Yolo model pre-trained for the COCO dataset.

5.1.1 Yolo vs Tiny-Yolo

As mentioned in section 4.1, it is likely that because of the simplicity of most runway configurations, the simpler Tiny-Yolo could suffice. To check this, both Yolo and Tiny-Yolo are trained using the same

strategy and dataset, and their performance is analyzed. The common training strategy involves two different stages whose parameters are detailed in Table 5.1. It has to be noted that the initial learning rate is the same for both models, and it is reduced by half every time the model shows no dev loss reduction greater than 0.1 for 8 consecutive epochs.

Stage	Trainable Layers	Batch Size	Learning Rate	Optimizer
Initial	3 deepest layers	32	10^{-3}	Adam
Final	All	8	10^{-4}	

Table 5.1. Yolo vs Tiny-Yolo, common training strategy

Model	epoch	Dev loss	Test mAP_{50}
Yolo v3	298	5.27	0.94
Tiny-Yolo v3	192	8.24	0.72

Table 5.2. Yolo vs Tiny-Yolo, training results

Figures 5.1 and 5.2 show the training results for both models. The graph on the left shows the model loss for the train and dev sets, while the graph on the right shows a common metric used to measure performance of object detection models: mAP_{50}. The end of the initial training stage happens at epoch 106 for the full Yolo model and at epoch 89 for the Tiny-Yolo. The performance of the models at this stage is poor with the test mAP below 3% in both cases. It is not until all layers are unfrozen that the performance improves substantially. There's a more detailed discussion of this effect in section 5.1.2, but essentially, the complex image features learned by Yolo in the COCO dataset have little to do with those present in the runway images. We conclude that only lower level features of the model are useful in the transfer learning approach which implies that more layers need to be freed for training from the beginning.

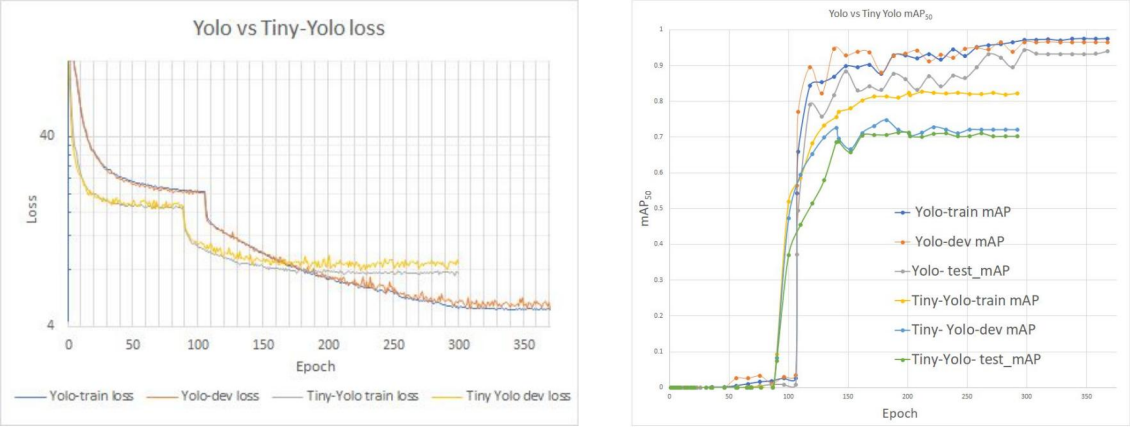

Figure 5.1. Yolo vs Tiny-Yolo training results. Loss (left), mAP (right).

The final performance for the Tiny-Yolo model achieves a good test mAP of 0.71. In this model it can be observed that the train mAP (0.81) is considerably higher than the dev mAP (0.72), unveiling some difficulties of this model to generalize to unseen data. Thus, besides some bias, also variance issues are present. On the other hand, the full Yolo model's results shows outstanding results (mAP=0.94), despite taking many more epochs to reach a steady state, and the graphs show no overfitting or bias issues. Comparing with other datasets, it should be noted that Yolo achieves a mAP of 0.55 in the COCO dataset, while the best performing models barely reach 0.60. This project takes advantage that the capabilities of the Yolo model are applied only to a single class, runways. The much better performance of the full Yolo model can be explained by two factors: i) over triple the layers than Tiny-Yolo which adds capacity to capture subtle details in the runway surroundings helping in its detection, and ii) more anchors (9 in Yolo vs 6 in Tiny-Yolo) providing more flexibility in finding an appropriate size bounding box and which impacts directly the IoU between predicted and ground truth bounding boxes and, hence, the mAP metric.
Figure 5.2a shows an image of a 2-runway airport which was wrongly selected as single runway airport which didn't escape Yolo. On the other hand, Tiny-Yolo failed to identify either. Fig. 5.2b illustrates a correct detection. Fig. 5.2c shows a case in which Tiny-Yolo identifies a road as a runway.

Figure 5.2 (a) Yolo detecting multiple runways. (b) Yolo correct detection, (c) Tiny-Yolo wrongly detecting additional runway.

5.1.2 Transfer Learning and Hyperparameter Search

A comparative study has been performed for 3, 10 and 30 trainable layers with batch sizes of 8, 16 and 32 and a learning rate of 0.001. Fig. 5.3 shows the mAP metric for different numbers of trained layers. The model with 30 trainable layers is much better, most likely due to the lack of applicability of the learned features from the COCO database. Table 5.2 shows other representative results. The batch size of 8 assured a better performance that the batch size of 32. This finding is illustrated in Fig. 5.4 in more detail.

	3 trained layers, batch size=8	30 trained layers, batch size=8	3 trained layers, batch size=32	30 trained layers, batch size=32
Train accuracy	0.12	0.65	0.04	0.46
Dev accuracy	0.08	0.68	0.08	0.48
Test accuracy	0.09	0.58	0.008	0.39

Table 5.2 Key accuracy parameters obtained during the transfer learning study

From this initial hyperparameter search, the following parameters were set as 30 trainable layers, batch size of 8 and a learning rate of 0.001. These were then used to train larger parts of Yolo considering 64, 128 and then all layers "unfrozen." The erratic behavior of the full net, see Fig. 5.5, has been attributed to an excessive value of Learning Rate (LR). Indeed, the following investigation of the effect of LR on training results has shown that LR equal to 0.0001 assures much better performance than that of 0.001 (see Fig. 5.6) and that of 0.00001 (not shown). Table 5.3 shows final accuracy parameters obtained for the "half-net" and "full net" trained with the selected parameters: learning rate of 0.0001 and batch size of 8. We can see that these two versions of Yolo achieve good bias and variance values. However, the test accuracy of the "half-net" falls slightly below 90%.

	128 layers, LR=0.001	128 layers, LR=0.0001	Full net, LR=0.001	Full net, LR=0.0001
Train accuracy	0.94	0.94	0.87	0.97
Dev accuracy	0.93	0.94	0.89	0.96
Test accuracy	0.87	0.89	0.84	0.93

Table 5.3 Final accuracy parameters

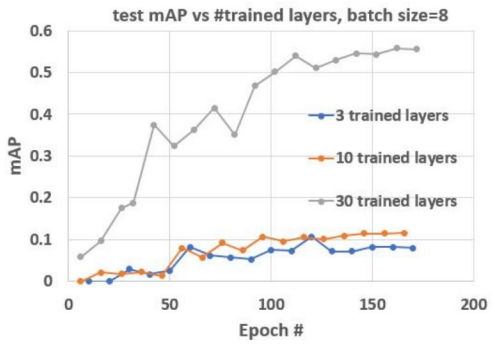

Fig. 5.3 The effect of trainable layers number on performance. The learning rate of 0.001.

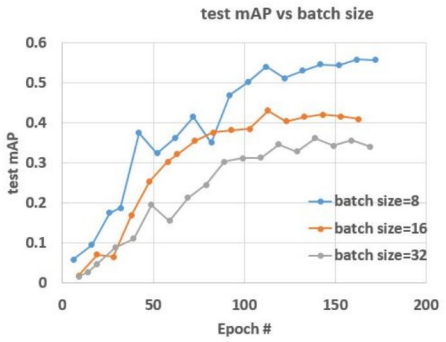

Fig. 5.4 The effect of batch size on performance. 30 trainable layers, The learning rate of 0.001.

To summarize, the full network achieves a test accuracy of 93% for the learning rate of 10^{-4}. It is interesting to note that the "half-network" achieves a respectable 89% accuracy at approximately half the cost. Also, the mAP performance of Yolo with 64 trainable layers appears to converge to 0.8 (see Fig. 5.5), while the equivalent result for Tiny-Yolo is just 0.72.

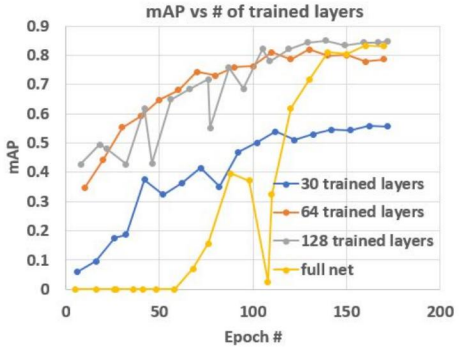

Fig. 5.5: mAP vs trainable layers.

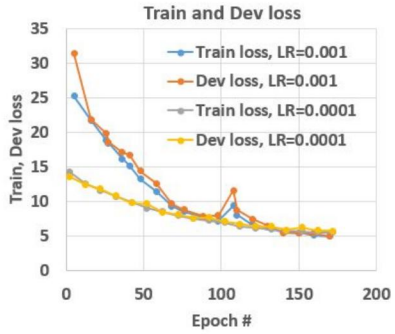

Fig. 5.6 Losses for the full net trained with different learning rates.

5.2 Segmentation

No conclusive results were obtained from segmentation. A total of 879 images have been divided into the training, dev and test set with 70%, 15%, and 15% proportions respectively. The pretrained ResNet-101 backbone is downloaded previous to training which is used to initialize the weights. An end-to-end training of the network resulted in a quick decrease of the loss function with very few epochs. A revision of the results, however, indicated that the segmentation was selecting the background class for all pixels and never the runway class. This result was understandable due to the relatively small frequency of runway pixels within the training set images. An implemented function which calculates balanced weights within the loss function for the two categories, background and runway, calculated weights of 1 and 50 respectively. No improvement was obtained so these weights were artificially raised to 1 and 2000, and again, no correct identification was achieved. Thus, this line of investigation was left to future work.

6 Conclusions and Future Work

The object detection approach works remarkably well using the full Yolo v3 model, and also fairly well with the Tiny-Yolo variant. Nevertheless, the dataset that was used is small, and there are doubts the results generalize well for actual camera images taken in a wide range of conditions. Certainly, in order to mature towards a real application, more real camera data would have to be used for training, further complemented by data augmentation techniques. The estimated bounding box tends to surround very accurately the runway, and in almost all cases, the runway fits into one of diagonals of its bounding box. Thus, one achieves a direct estimation for the runway length in pixels, and two options are provided for the runway orientation. Assuming a previous maximum directional error of the camera, one may determine the correct diagonal a posteriori and, hence, the runway orientation. Nevertheless, we plan to modify the Yolo architecture and incorporate minimally the estimation of the correct bounding box diagonal for the runway. Finally, note that the achieved runway detection rate of 94% is very close to that presented by [Nazir18]. Future work will also determine the relationship between the Yolo output and the resulting aircraft's positioning accuracy.

7 Contributions

All three members of the team contributed equally to the project. The data was collected using Google Earth by Carlos Querejeta. The osgEarth geovisualization framework was automated and setup to acquire images by Michael Hardt. Carlos Querejeta was primarily responsible for setting up the Yolo and TinyYolo calculation environments. Grzegorz Kawiecki performed hyperparameter searches in order to tune the Yolo environment. Michael Hardt worked on setting up the semantic segmentation environment. The results were collected and analyzed by all participants.

The code has been stored in the following repository: https://github.com/txoritxo/DroneWaves.git
Furthermore, this repository contains the following branches:

- master: keras Yolo v3 implementation for the project milestone **(deprecated)**

- https://github.com/txoritxo/DroneWaves/tree/trainFinal: final keras Yolo v3 and tiny Yolo implementation with detailed documentation on its use and results
- https://github.com/txoritxo/DroneWaves/tree/Branch_segmentation: pytorch DeepLab v3+ implementation

Access has been granted to Ishan Patil (github user iapatil).

8 References

[Anonymous2019] Anonymous, 2019, "Image Segmentation," scikit-image, image processing in python, http://scikit-image.org/docs/dev/user_guide/tutorial_segmentation.html

[Chen18] L.C. Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, 2018, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," https://arxiv.org/abs/1802.02611

[leimao19], 2019, "Google DeepLab v3 for Image Semantic Segmentation," https://github.com/leimao/DeepLab_v3

[Nazir18] Nazir S., Aziz S., Khaliq Y. and Adnan S. M., 2018, "Vision Based Autonomous Runway Identification and Position Estimation for UAV Landing," International Conference on Artificial Intelligence and Data Processing (IDAP), https://ieeexplore.ieee.org/document/8620824

[qqwwee18], 2018, "A Keras implementation of YOLOv3 (Tensorflow backend)," https://github.com/qqwweee/keras-yolo3

[rpadilla18], Rafael Padilla, 2018, "Metrics for Object Detection", https://github.com/rafaelpadilla/Object-Detection-Metrics

[Redmon16] Redmon J., Divvala S., Girshick R. and Farhadi A., 2016, "You only look once: Unified, Real-Time Object Detection," IEEE CVPR Conference, pp. 779-788. https://pjreddie.com/media/files/papers/yolo.pdf

[Redmon18] Redmon J. and Farhadi A., 2016, "YOLOv3: An Incremental Improvement," Technical Report. https://pjreddie.com/publications/

[Tripathi10] Tripathi A. K., Patel V. V. and Padhi R., 2018, "Vision Based Automatic Landing with Runway Identification and Tracking," 15th International Conference on Control, Automation, Robotics and Vision (ICARCV). https://ieeexplore.ieee.org/document/8581208