Wire Detection using Synthetic Data and Dilated Convolutional Networks for Unmanned Aerial Vehicles

Ratnesh Madaan*, Daniel Maturana*, Sebastian Scherer*

***Abstract*— Wire detection is a key capability for safe navigation of autonomous aerial vehicles and is a challenging problem as wires are generally only a few pixels wide, can appear at any orientation and location, and are hard to distinguish from other similar looking lines and edges. We leverage the recent advances in deep learning by treating wire detection as a semantic segmentation task, and investigate the effectiveness of convolutional neural networks for the same. To find an optimal model in terms of detection accuracy and real time performance on a portable GPU, we perform a grid search over a finite space of architectures. Further, to combat the issue of unavailability of a large public dataset with annotations, we render synthetic wires using a ray tracing engine, and overlay them on 67K images from flight videos available on the internet. We use this synthetic dataset for pretraining our models before finetuning on real data, and show that synthetic data alone can lead to pretty accurate detections qualitatively as well. We also verify if providing explicit information about local evidence of wiry-ness in the form of edge and line detection results from a traditional computer vision method, as additional channels to the network input, makes the task easier or not. We evaluate our best models from the grid search on a publicly available dataset and show that they outperform previous work using traditional computer vision and various deep net baselines of FCNs, SegNet and E-Net, on both standard edge detection metrics and inference speed. Our top models run at more than 3Hz on the NVIDIA Jetson TX2 with input resolution of 480x640, with an Average Precision score of 0.73 on our test split of the USF dataset.**

I. INTRODUCTION

Thin wires and similar objects like power lines, cables, ropes and fences are one of the toughest obstacles to detect for autonomous flying vehicles, and are a cause of numerous accidents each year. They can be especially hard to detect in cases where the background is cluttered with similar looking edges, when the contrast is low, or when they are of barely visible thickness. Power line corridor inspection is another area of potentially widespread application of wire detection capabilities, and leveraging UAVs for this task can save a lot of money, time, and help avoid dangerous manual labor done by linemen. Apart from UAVs, there have been recent reports which document fatal injuries and deaths caused by hitting barbed wire fences while riding ATVs, and dirt and mountain bikes. Generally, one could use lidar, infrared, electromagnetic sensors or cameras to detect wires and power lines. Out of these, a monocular camera is the cheapest and most lightweight sensor, and considering the recent advances in deep learning on images, we use it as our perception sensor. Apart from a good detection rate, real time performance on a portable GPU like the NVIDIA Jetson TX2 with a decent resolution like 480x640 is critical to ensure that wires are still visible in the input image, and that the drone has enough time for potentially avoiding them.

Previous work uses strong priors on the nature of power lines - assuming they are straight lines, have the highest or lowest intensity, appear with a fixed number in an image, have a gaussian intensity distribution, are the longest line segments, are parallel to each other, or can be approximated by quadratic polynomials [1]–[6]. These works first gather local criteria for potential wire pixels by using an edge detection algorithm, filter them using the heuristics mentioned above, and then gather global criteria via variants of the Hough or Radon transform[1]–[3], clustering in space of orientations[4] and graph cut models[6]. These approaches demand a lot of parameter tuning, work well only in specific scenarios, and are prone to false positives and negatives since the distinction between wires and other line-like objects in the image does not always obey the appearance based assumptions and priors.

Another issue with wire detection is the unavailability of a sufficiently large public dataset with pixel wise annotations. The only existing dataset with a decent number of labeled images is provided by [2], which we refer to as the USF dataset. We use it for evaluation of our models, as done by [6] as well.

We address the former of the aforementioned issues by investigating the effectiveness of using convolutional neural networks (convnets) for our task. Recent advances in semantic segmentation and edge detection using convnets are an indication of an end to end wire detection system omitting hand-engineered features and parameter tuning, which can also potentially generalize to different weather and lighting conditions depending on the amounts of annotated data available [7]–[9]. To meet our objectives of real time performance on the TX2 and reasonable detection accuracy, we perform a grid search over 30 architectures chosen based on our intuition. We investigate the effect of filter rarefaction[10] by systematically adding dilated convolutional layers with increasing amounts of sparsity as shown in Table I and II. We show that the top 5 models from our grid search outperform various baselines like E-Net[11], FCN-8 and FCN-16s[7], SegNet[12], and non-deep baseline of [2] as shown in Fig. 4 and Table IV across multiple accuracy metrics and inference speeds on the Jetson TX2.

To address the latter issue of the unavailability of a large dataset with pixel wise annotations, we generate synthetic data by rendering wires in 3D as both straight lines and in their natural catenary curve shape, using the POV-Ray[13]

*All authors are with the Robotics Institute at Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. [ratneshm, dmaturan, basti]@andrew.cmu.edu

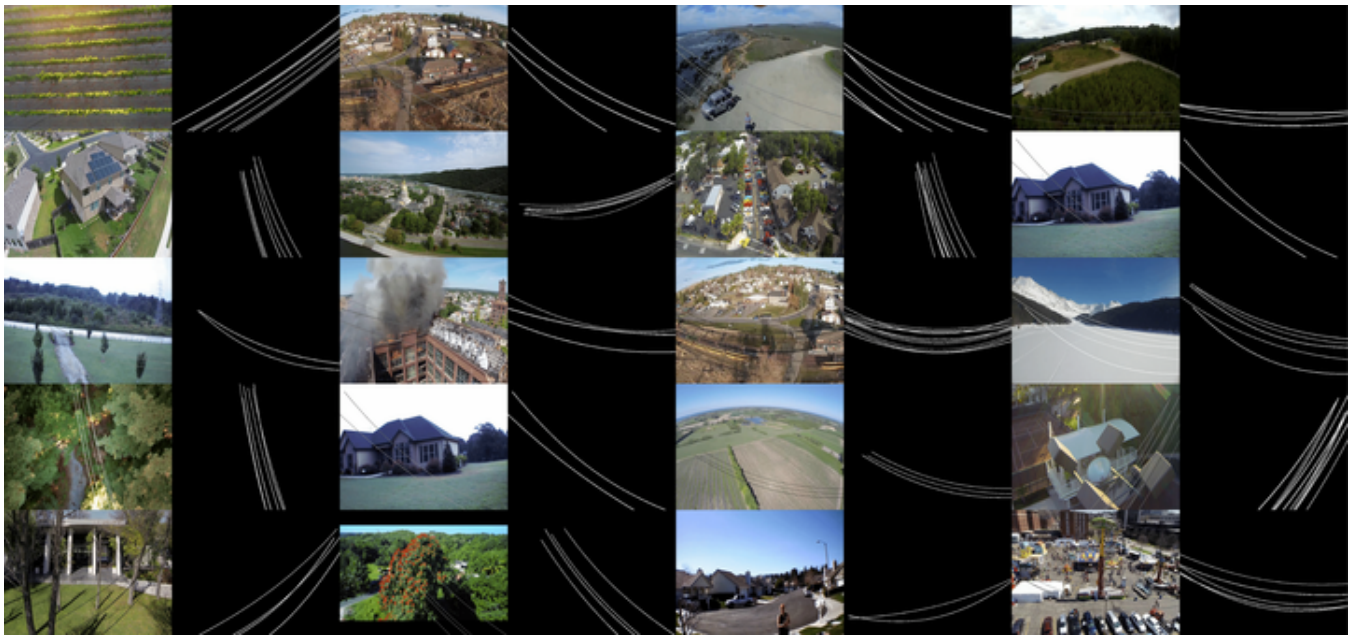

Fig. 1: A few samples from our synthetically generated dataset along with ground truth labels of wires. Video examples available at this link[1] and our project page[2].

ray-tracing engine with varying textures, numbers, orientations, positions, lengths, and light source, and superimpose them on 67702 frames obtained from 154 flight videos collected from the internet. A few samples from our dataset can be seen in Figure 1 and in the video here[3]. Although training on synthetic data alone is not enough to match the performance of models which are finetuned or trained from scratch on the USF dataset, we observe that synthetic data alone gave pretty accurate qualitative detections on a few unlabeled flight videos we tested on as shown in Figure 5. More results are available at our project page[4].

Further, for wire detection and avoidance, false negatives (undetected wires) are much more harmful than false positives. We adapt the line of thought of finding local evidence and then refining it via heuristics or global criteria used by previous works, to convnets by explicitly providing local evidence of 'wiryness' in the form of extra channels to the network input. One would expect that doing this would make the network's task easier as now it has to filter out the non-wire edges and line segments and return pixels that were actually wires. However, in the case of training from scratch, or finetuning on the USF dataset after pretraining on synthetic data, we find that the local primitive information does not have much effect in the performance of the network. On the other hand, we find out that adding this information of local criteria does help in the case of models trained on only synthetic data, as shown in Table II.

In summary, the contributions of this paper are:

- Systematic investigation of the effectiveness of convnets for wire detection by a grid search.
- Large scale generation of a synthetic dataset of wires and demonstrating its effectiveness.
- Investigation of the effect of explicitly providing local evidence of wiryness.
- Demonstrating real time performance on the NVIDIA Jetson TX2 while exceeding previous approaches both in terms of detection accuracy and inference speed.

Fig. 2: Close up of synthetic wires rendered using POV-Ray [13].

II. Related Work

A. Wire detection using traditional computer vision

One of the earliest works in wire detection is from Kasturi et al.[1], who extract an edgemap using Steger's algorithm [14], followed by a thresholded Hough transform which rejects lines of short length. However, due to unavailability of ground truth, they evaluate their approach only on synthetically generated wires superimposed on real images.

Candamo et al.[2] find edges using the Canny detector and then weigh them proportionally according to their estimated motion found using optical flow, followed by morphological filtering and the windowed Hough transform. The resulting parameter space is then tracked using a line motion model. They also introduce the USF dataset and show that their approach performs better than [1]. We modify their temporal algorithm to a per-frame algorithm as explained later, and use it as a baseline.

[3] https://www.youtube.com/watch?v=z6sPz-WPCWQ

[4] http://madratman.github.io/wire_detection_iros_2017

TABLE I: The parameter space over which we perform a grid search. Each context module from the 2nd column was stacked over each front-end module from the first column to get a total of 30 models. All layers use 3*3 convolutions. Each layer is separted by a '-' sign. 'k' refers to the number of channels in each layer. 'p' refers to pooling layers, 'd' refers to the dilation factor.

Front-end Modules		**Context Modules**	
Key	Architecture	Key	Architecture
f1	k64-k64-p2-k128-k128	c1	k2(none)
f2	k32-k32-k64-k64	c2	d1-d2-d1-k2
f3	k32-k32-k32-k32	c3	d1-d2-d4-d1-k2
f4	k32-k32-k64-k64-k64-k64	c4	d1-d2-d4-d8-d1-k2
f5	k32-k32-k32-k32-k32-k32	c5	d1-d2-d4-d8-d16-d1-k2
f6	k32-k32		

Song and Li [6] proposed a sequential local-to-global power line detection algorithm which can detect both straight and curved wires. In the local phase, a line segment pool is detected using Gaussian and first-order derivative of Gaussian filters, following which the line segments are grouped into whole lines using graph-cut models. They compare their method explicitly to [1] and their results indicate similar performance as [2], which is one of our baselines.

B. Semantic segmentation and edge detection using deep learning

Fully Convolutional Networks(FCNs)[7] proposed learned upsampling and skip layers for the task of semantic segmentation, while SegNet[12] proposed an encoder-decoder modeled using pooling indices. For thin wires, FCNs and SegNet are intuitively suboptimal as crucial information is lost in pooling layers which becomes difficult to localize in the upsampling layers. E-Net [11] develop a novel architecture by combining tricks from multiple works for real time semantic segmentation performance, but one of their guiding principles is aggresive downsampling again.

III. Approach

A. The search for the right architecture

For a wire detection network running on a platform like the NVIDIA TX2 atop a small UAV, we desire low memory consumption and a fast inference time. Further, as wires are barely a few pixels wide, we do not want to lose relevant information by the encoder-decoder approaches [7,11,12] in which one first downsamples to aggregate global evidence and then upsamples back up to localize the target object.

Dilated convolutional layers [10] are a simple and effective way to gather context without reducing feature map size. Each layer is defined by a dilation factor of d, which correspond of $d-1$ number of alternating zeros between the learnt elements, as visualized in [10,15]. [10] proposed appending a 'context module' comprising of 7 layers with increasing dilation factors (d for each layer following the series $\{1,1,2,4,8,16,1\}$) to existing segmentation networks (front-end moduleS), which boosted their performance significantly.

In order to investigate the effect of dilation on our task, we run a grid search over a finite set of front-end and context modules as summarized in Table I. We now introduce a simple naming scheme to refer our models with. Each layer is separated by a '-' the number succeeding 'k' is equal to the number of channels, 'p' refers to a pooling layer and 'd' to a dilation layer. For all layers, we use 3*3 convolutions.

For front-end modules, we begin with trimming down VGG[16] to its first two blocks, which is the model f1 in Table I. Next, we halve the number of channels across f1 and remove the pooling layer to get f2. Then, we halve the number of channels in the last two layers of f2 of this model to get f3. Further, to investigate the effect of depth we append 2 layers each to the f2 and f3, while maintaining the number of channels to get f4 and f5. We stop at 6 layers as previous work [9] found it to be enough to detect edges. Finally to check how well a simple two layer front-end performs, we add f6 to our list. The second column shows the range of the context modules we evaluated, starting with an empty context module (c1) and building up to c5 as in [10]. Our first model is d1+d2+d1 due to the fact that a dilation factor of 1 is equivalent to standard convolution without holes. Note that there is always a k2 in the end as in the last layer, we do a softmax operation over two output classes (wire and not-wire).

By choosing a front-end (f1-f6) and appending it with a context module (c1-c5), we do a grid search over 30 architectures and do multiple experiments - training each model on synthetic data, real data, finetuning models trained on synthetic data, and finally evaluation on real data as shown in Table II. We use a 50-50 train-test split of the USF dataset for the grid search. We then choose the top 5 models based on the accuracy and inference speed on the NVIDIA TX2 (bold text in Table III) for comparison with various baselines.

B. Generating synthetic data

Due to the unavailaibility of a large dataset with annotations of wires, we render synthetic wires using the POV-Ray ray tracing engine [13], and superimpose them on 67702 frames sampled from 154 flight videos collected from the internet. Figure 1 and this video shows some examples from our synthetic dataset.

Our dataset consists of wires occuring as both straight lines and catenary curves (the natural shape of a wire hanged at its ends under uniform gravity). We vary the wire sag, material properties, light source location, reflection parameters, camera angle, number of wires and the distance between them across the dataset to obtain a wide variety of configurations. A close up of the rendered wires can be seen in Figure 2.

C. Class Balancing Loss Function

As wire pixels occupy only a minisicule fraction of the total number of pixels in an image, it is important to account for class imbalance in the loss function, as well as in the evaluation metrics used. To that end, we use the image-level, class-balanced cross entropy loss function defined in [9]. For the USF dataset, we find that wire pixels account for only 4.9% of the total number of pixels, and the rest 95.1% are background pixels, and weigh the loss accordingly.

TABLE II: Results of our grid search experiments. The strings in bold, blue text represent the front end architecture as explained in text. Each front end is appended with five different context modules as shown in the Col. 1. We conduct six experiments, all of which are evaluated on our test split of the USF dataset. Each experiment category is grouped with the same hue (Col. 2-3, 4-5, 6-7 respectively). Darker color implies better performance. Choice of colors is arbitrary. Col. 8-11 list the performance speeds on the NVIDIA TitanX Pascal and the Jetson TX2, with batch size 1 and input resolution of 480x640.

	Average Precision Scores (evaluation on USF test split)						**Performance**			
Trained On →	**Synthetic Data**		**Scratch on USF**		**Finetuned on USF**		**TX2**		**TitanX**	
Context Module ↓	**RGB**	**RGBLE**	**RGB**	**RGBLE**	**RGB**	**RGBLE**	**Time (ms)**	**Speed (fps)**	**Time (ms)**	**Speed (fps)**
Column 1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Front End Module →	**k32-k32**									
k2	0.23	0.30	0.45	0.45	0.50	0.47	37.7	26.6	3.2	309.6
d1-d2-d1-k2	0.33	0.35	0.63	0.58	0.61	0.59	115.7	8.7	8.0	124.7
d1-d2-d4-d1-k2	0.28	0.42	0.65	0.61	0.64	0.62	159.3	6.3	10.2	98.0
d1-d2-d4-d8-d1-k2	0.35	0.45	0.65	0.62	0.66	0.65	203.7	4.9	12.4	80.6
d1-d2-d4-d8-d16-d1-k2	0.36	0.49	0.70	0.64	0.70	0.69	249.4	4.0	14.6	68.4
k32-k32-k32-k32										
k2	0.24	0.32	0.57	0.50	0.57	0.54	70.7	14.2	5.7	176.1
d1-d2-d1-k2	0.25	0.39	0.62	0.60	0.64	0.59	148.5	6.7	10.1	99.1
d1-d2-d4-d1-k2	0.28	0.40	0.63	0.62	0.66	0.63	192.7	5.2	11.9	83.9
d1-d2-d4-d8-d1-k2	0.33	0.42	0.69	0.62	0.68	0.66	236.6	4.2	13.8	72.6
d1-d2-d4-d8-d16-d1-k2	0.43	0.48	0.70	0.64	0.72	0.64	282.6	3.5	15.7	63.9
k32-k32-k32-k32-k32-k32										
k2	0.18	0.35	0.63	0.53	0.61	0.57	104.0	9.6	8.2	121.7
d1-d2-d1-k2	0.30	0.42	0.61	0.58	0.64	0.59	181.5	5.5	13.2	76.1
d1-d2-d4-d1-k2	0.25	0.43	0.62	0.59	0.67	0.62	225.4	4.4	15.3	65.2
d1-d2-d4-d8-d1-k2	0.32	0.47	0.66	0.64	0.70	0.65	270.1	3.7	17.6	57.0
d1-d2-d4-d8-d16-d1-k2	0.03	0.47	0.68	0.66	0.47	0.65	315.0	3.2	19.8	50.6
k32-k32-k64-k64										
k2	0.20	0.28	0.59	0.50	0.60	0.55	118.7	8.4	8.3	120.9
d1-d2-d1-k2	0.25	0.45	0.62	0.60	0.62	0.60	350.7	2.9	19.9	50.4
d1-d2-d4-d1-k2	0.29	0.38	0.65	0.62	0.68	0.62	496.2	2.0	26.3	38.1
d1-d2-d4-d8-d1-k2	0.28	0.45	0.66	0.64	0.66	0.64	641.7	1.6	32.8	30.5
d1-d2-d4-d8-d16-d1-k2	0.40	0.48	0.71	0.62	0.71	0.67	787.6	1.3	38.4	26.1
k32-k32-k64-k64-k64-k64										
k2	0.27	0.39	0.63	0.57	0.61	0.58	206.6	4.8	13.4	74.8
d1-d2-d1-k2	0.29	0.42	0.62	0.60	0.67	0.59	439.2	2.3	24.7	40.5
d1-d2-d4-d1-k2	0.28	0.38	0.64	0.60	0.65	0.62	582.7	1.7	31.0	32.3
d1-d2-d4-d8-d1-k2	0.37	0.45	0.68	0.62	0.67	0.63	729.8	1.4	37.2	26.9
d1-d2-d4-d8-d16-d1-k2	0.42	0.49	0.68	0.63	0.70	0.63	877.7	1.1	43.9	22.8
k64-k64-p2-k128-k128										
k2	0.26	0.35	0.65	0.60	0.66	0.60	136.0	7.4	8.1	124.1
d1-d2-d1-k2	0.30	0.34	0.66	0.63	0.72	0.66	279.8	3.6	17.1	58.5
d1-d2-d4-d1-k2	0.41	0.45	0.70	0.67	0.73	0.67	350.8	2.9	21.8	45.8
d1-d2-d4-d8-d1-k2	0.34	0.49	0.71	0.66	0.73	0.70	421.8	2.4	26.5	37.7
d1-d2-d4-d8-d16-d1-k2	0.36	0.47	0.72	0.64	0.75	0.68	493.2	2.0	31.2	32.0

D. Explicitly providing local evidence of wires

Previous works extract local evidence of wiryness via edge maps obtained from standard edge detection algorithms, which is followed by filtering using domain knowledge, and finally gathering global evidence. We adapt this line of thought to convnets by appending the results of line segment and edge detection using the recently proposed CannyLines and CannyPF detectors [17] in the form of two additional channels to the network's input.

IV. Experiments and Results

A. Evaluation Metrics

We first define and justify the metrics used. Average Precision(AP) and Optimal Dataset Scale(ODS) F1 score are standard metrics used for edge detection tasks[18]. AP is defined as the mean precision taken over all recall values, and is equal to the area under the Precision-Recall(PR) curve. ODS F-score is the optimal F1 score obtained by choosing the threshold of a probabilistic classifier which gives the best performance across the training dataset. Previous works on wire detection use the Receiver Operating Characteristics(ROC) at a *per-wire* basis. This entails fitting a line or a curve to thresholded predictions and then counting instances of the same, which leads to ambiguity due to issues like minimum length to register a detection, choosing polynomials to fit the thresholded predictions for curved wires, and coming up with tolerances which relax the detection criteria. Therefore, we calculate both ROC and PRC on a *per-pixel* basis and report Area under the Curve of the ROC diagram(AUC), which like AP, summarizes the curve with a single number.

It is also worth mentioning that for the case of severely class-imbalanced problems such as ours, AP is a better performance metric than ROC. In ROC analysis. false detections are evaluated via False Positive Rate, which compares false positives to true negatives (i.e. background pixels which account for roughly 95% of the USF dataset). Whereas in PR curves, the measure of false detections is done by Precision, which compares false positives to true positives. Therefore, in problems where the negative class outnumbers the positive class by a huge margin, AP is a better measure for classifier evaluation. We refer the reader to [19] for more details.

B. Grid Search

We do extensive evaluation of the models listed in Table I on the USF dataset. First, we randomly select half of the USF videos (21 out of 42 available) for the training set and kept the rest for testing, leading to 2035 frames for training and 1421 frames for testing. Meanwhile, we also have a synthetic dataset consisting of 67702 frames from 154 publicly available flight videos as mentioned before. For all cases, the input resolution used is 480x640. For each model from Table I, we conduct three kind of experiments:

- **Synthetic**: Train on our synthetic dataset
- **Scratch**: Train on train split of USF dataset
- **Finetune**: Finetune on the USF train split after training on synthetic dataset

For each category above, we conduct runs with two different inputs to investigate the benefits of explicitly providing local evidence:

- **RGB**: training on monocular images only
- **RGBLE**: training on monocular images, concatenated with lines(L) and edges(E) detected by [17].

Table II shows the results of all the six experiments on each model, along with the AP scores and performance on the NVIDIA Titan X Pascal and the TX2. Each experiment category is assigned an arbitrary hue, and darker color emphasizes better performance. Same is true for the last 4 columns which depict the inference performance with a batch size of 1, and input resolution of 480x640. Each class of models with the front-end is demarked by the bold blue text, while context modules of increasing amount of dilation are added to them in 5 consecutive rows. From Table II, we can draw the following conclusions:

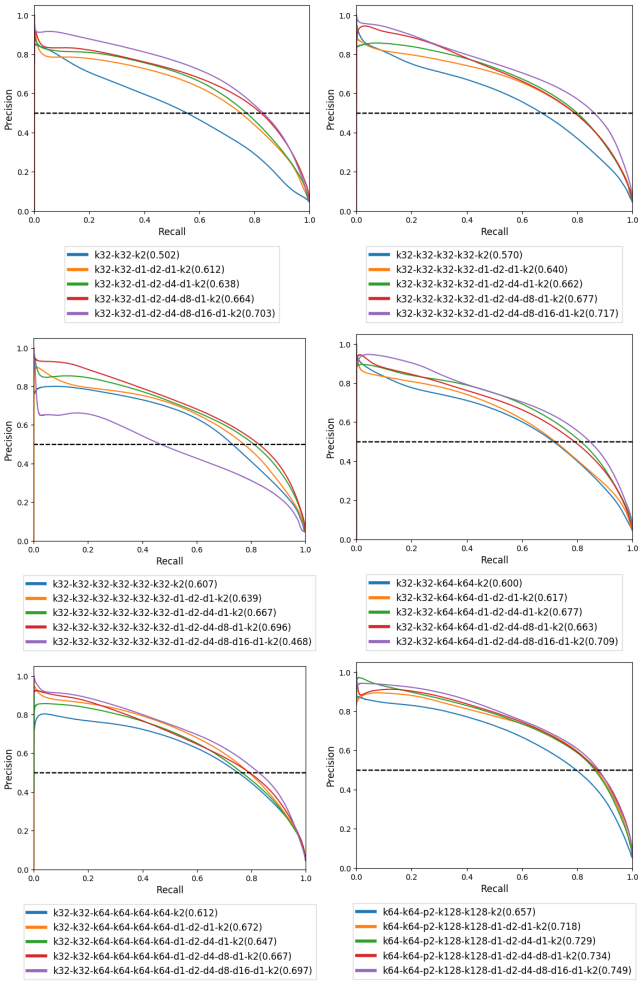

Fig. 3: Precision recall curves obtained by evaluating our models on USF test split with RGB input, after finetuning our models on USF train split after pretraining on synthetid data. Each subfigure has the same front end module, while the context module changes as shown in the legends. Numbers in paranthesis are the Average Precision scores for the respective model. In all cases we can see that adding increasing dilation increases the precision of the classifier at various recall levels (except for one case in k32-k32-k32-k32-k32-k32's series where the model with the maximum dilation didn't converge.

- *Dilated kernels with increasing sparsity help*:
  This trend can be seen throughout in Col 2-7 for each front-end. For each column, for the same frontend module, we can observe increasing AP scores as we add more dilation in the context modules. As the AP numbers can be close and hard to judge, we verify the same with Precision Recall curves for each front-end with varying context modules as shown in Figure 3. At the same time, the run-time performance of the models decrease as we can keep on adding more layers (Col 8-11).

TABLE III: Top models from our grid search. All models were trained on synthetic data first, then finetuned on our USF dataset test split using RGB input. Darker color in each colum implies higher accuracy, higher speed (frames per second) on the TX2, and smaller number of model parameters. Hues for each column are chosen arbitrarily.

Model	AP Score (Finetune RGB)	TX2 (fps)	Number of params
k64-k64-p2-k128-k128-d1-d2-d4-d8-d16-d1-k2	0.75	2.03	1,147,970
k64-k64-p2-k128-k128-d1-d2-d4-d1-k2	**0.73**	**2.85**	**852,802**
k64-k64-p2-k128-k128-d1-d2-d4-d8-d1-k2	0.73	2.37	1,000,386
k32-k32-k32-k32-d1-d2-d4-d8-d16-d1-k2	**0.72**	**3.54**	**84,706**
k64-k64-p2-k128-k128-d1-d2-d1-k2	0.72	3.57	705,218
k32-k32-k64-k64-d1-d2-d4-d8-d16-d1-k2	0.71	1.27	288,290
k32-k32-d1-d2-d4-d8-d16-d1-k2	**0.70**	**4.01**	**66,210**
k32-k32-k32-k32-k32-k32-d1-d2-d4-d8-d1-k2	**0.70**	**3.7**	**93,954**
k32-k32-k64-k64-k64-k64-d1-d2-d4-d8-d16-d1-k2	0.70	1.14	362,146
k32-k32-k32-k32-d1-d2-d4-d8-d1-k2	0.68	4.23	75,458
k32-k32-k64-k64-d1-d2-d4-d1-k2	0.68	2.02	214,434
k32-k32-k32-k32-k32-k32-d1-d2-d4-d1-k2	**0.67**	**4.44**	**84,706**
k32-k32-k64-k64-k64-k64-d1-d2-d1-k2	0.67	2.28	251,362
k32-k32-k64-k64-k64-k64-d1-d2-d4-d8-d1-k2	0.67	1.37	325,218

- *Explicitly providing local information only helped in the case of synthetic data*:
  This can be seen by comparing the RGB and RGBLE columns in Table II under each category. In case of training (Col 4-5) or finetuning on real data (Col 6-7), we actually observe slightly poorer performance by providing results of line and edge detection. However, for the case of training on synthetic data (Col 2-3), providing local evidence helps to boost the networks' performance on real data by fairly big margins. We believe this is due to the nature of the synthetic dataset which doesn't have the same distribution as the USF test split, and hence, giving the networks all lines and edges explicitly helps to improve their performance. This also suggests that an RGBLE input might generalize better to a test set having a significantly different distribution from the training set, but more experimentation is needed to ascertain that.
- *Pretraining on synthetic data helps slightly*:
  This can be seen by comparing Col 4 with Col 6 (RGB inputs). We believe this is another artifact of the USF dataset, which is small in size and a lot of images are relatively simplistic. The previous statement is supported with evidence of results presented in Fig. 5. Here, we train only on synthetic data with 720x1280 resolution, and tested on a few publicly available videos. In this case, the value of synthetic data is clearly evident.

For our use case, we desire fast inference speed on the TX2 and decent precision at high recalls for wires. To pick the best of the lot, the top 15 models from our grid search experiments on finetuning on the USF dataset with RGB input can be seen in Table III, along with inference speed with batch size 1 and resolution of 480x640, and number of parameters (although number of parameters isn't that much of a burden at inference time due to enough memory on the TX2). We cherry pick the 5 models in bold text by looking at these three metrics, the precision recall scores (Fig. 4), and qualitative comparisons (Fig. 6). Table IV compares these top-5 models with various baselines.

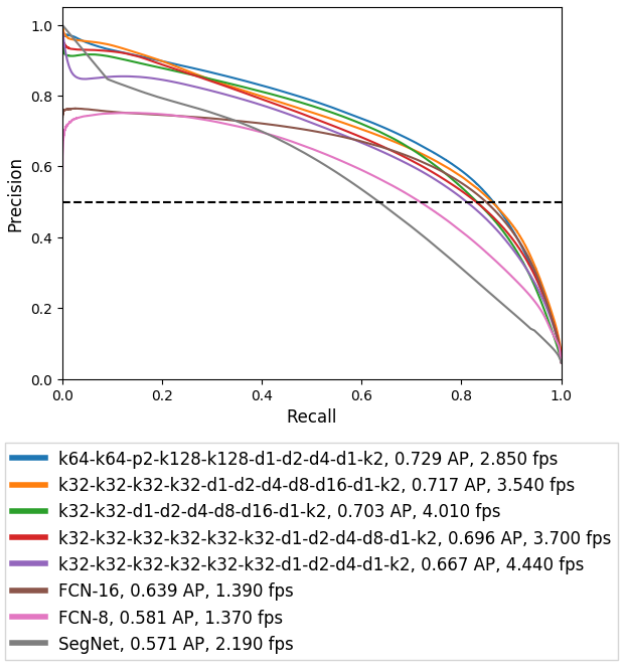

Fig. 4: Precision Recall curves of the top models from our grid search experiment (in bold text in Table III). Legend shows Average Precision scores on the USF test split, and speed on the NVIDIA TX2 in frames per second.

TABLE IV: Our top models compared with various baselines. The metrics used are Average Precision(AP), Area under the Curve of ROC diagram(AUC), Optimal Dataset Scale F1-Score(ODS F-Score)

Model	AP	AUC	ODS F-Score	TX2 (fps)
FCN-8 [7]	0.581	0.960	0.598	1.37
FCN-16 [7]	0.639	0.972	0.663	1.39
Segnet [12]	0.571	0.939	0.567	2.19
Candamo [2]	0.408	-	0.382	-
E-Net [11]	0.580	0.945	0.595	-
k64-k64-p2-k128-k128-d1-d2-d4-d1-k2	**0.729**	0.972	**0.688**	2.85
k32-k32-k32-k32-d1-d2-d4-d8-d16-d1-k2	0.717	**0.976**	0.678	3.54
k32-k32-d1-d2-d4-d8-d16-d1-k2	0.703	0.969	0.673	4.01
k32-k32-k32-k32-k32-k32-d1-d2-d4-d8-d1-k2	0.696	0.973	0.656	3.70
k32-k32-k32-k32-k32-k32-d1-d2-d4-d1-k2	0.667	0.970	0.647	**4.44**

Implementation details: For training on synthetic data, we use a minibatch size of 4 and train for 2000 iterations, while for the USF dataset, we train for 1000 iterations. All input images are of 480x640 resolution. We use AdaDelta [20] we found that stochastic gradient descent with momentum lead to slower convergence and a few architectures getting stuck in local minimas. We implement our models in both Lasagne [21] and Pytorch, as we found that the latter is faster and consumes less memory on the TX2.

C. Baselines

We consider four baselines to compare our approach with - Candamo et al.[2], FCNs[7], SegNet[12] and E-Net[11]. We adapt the temporal approach of [2] described in the related work section to a per-frame method: first we perform edge detection using [17], followed by morphological filtering of 8-connected components with connectivity of less than 30 pixels, followed by a windowed Hough transform by breaking up the edgemap image into 16 (4*4) subwindows.

Table IV and Figure 4 shows the performance of our top 5 models along with the baselines. We report AP, AUC, ODS-F1 scores, inference speed for a 480x640 image with batch size 1 on the NVIDIA TX2, and the PR curves. For [2], as the output is a binary prediction (detected lines via windowed Hough transform) and not a confidence value as in the case of CNNs, we report the F1 score and the Precision value in the ODS F-score and the AP fields respectively. We don't report AUC as that is applicable to probabilistic classifiers only. To calculate the metrics in this case, we rasterize the results of the Hough transform with varying thickness of the plotted lines from 1 to 5 pixels and report the best F1 and Precision scores obtained for a fair comparison.

V. Conclusion and Future Work

In this paper, we presented a method to detect wires using dilated convnets to facilitate autonomous UAVs. We generate a large synthetic dataset by rendering wires and superimposing them on frames of publicly available flight videos, and demonstrate its effectiveness by showing qualitative and quantitative results. Our experiments systematically find the best architectures over a finite space of model parameters and outperform various baselines across multiple detection metrics and inference speeds on the NVIDIA TX2.

We are currently working on multi-view stereo methods to find distance of the wires and coming up with methods to avoid them robustly at high speeds. For improving the perception pipeline, we are looking into eliminating false positives and considering temporal information into account for consistent detection of wires.

References

[1] R. Kasturi and O. I. Camps, "Wire detection algorithms for navigation," *NASA Technical Report*, 2002.

[2] J. Candamo, R. Kasturi, D. Goldgof, and S. Sarkar, "Detection of thin lines using low-quality video from low-altitude aircraft in urban settings," *IEEE Transactions on aerospace and electronic systems*, vol. 45, no. 3, 2009.

[3] G. Yan, C. Li, G. Zhou, W. Zhang, and X. Li, "Automatic extraction of power lines from aerial images," *IEEE Geoscience and Remote Sensing Letters*, vol. 4, no. 3, pp. 387–391, 2007.

[4] Z. Li, Y. Liu, R. Walker, R. Hayward, and J. Zhang, "Towards automatic power line detection for a uav surveillance system using pulse coupled neural filter and an improved hough transform," *Machine Vision and Applications*, vol. 21, no. 5, pp. 677–686, 2010.

[5] J. N. Sanders-Reed, D. J. Yelton, C. C. Witt, and R. R. Galetti, "Passive obstacle detection system (pods) for wire detection," in *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics, 2009, pp. 732 804–732 804.

[6] B. Song and X. Li, "Power line detection from optical images," *Neurocomputing*, vol. 129, pp. 350–361, 2014.

[7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[8] G. Bertasius, J. Shi, and L. Torresani, "Deepedge: A multi-scale bifurcated deep network for top-down contour detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4380–4389.

[9] S. Xie and Z. Tu, "Holistically-nested edge detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1395–1403.

[10] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[11] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*, 2016.

[12] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for scene segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[13] "Persistence of vision raytracer (version 3.7)," 2004. [Online]. Available: http://www.povray.org/download/

[14] C. Steger, "An unbiased detector of curvilinear structures," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 2, pp. 113–125, 1998.

[15] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR abs/1609.03499*, 2016.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] X. Lu, J. Yao, K. Li, and L. Li, "Cannylines: A parameter-free line segment detector," in *Image Processing (ICIP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 507–511.

[18] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 898–916, 2011.

[19] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 233–240.

[20] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[21] S. Dieleman, J. Schlter, C. Raffel, E. Olson, S. K. Snderby, D. Nouri, *et al.*, "Lasagne: First release." Aug. 2015. [Online]. Available: http://dx.doi.org/10.5281/zenodo.27878

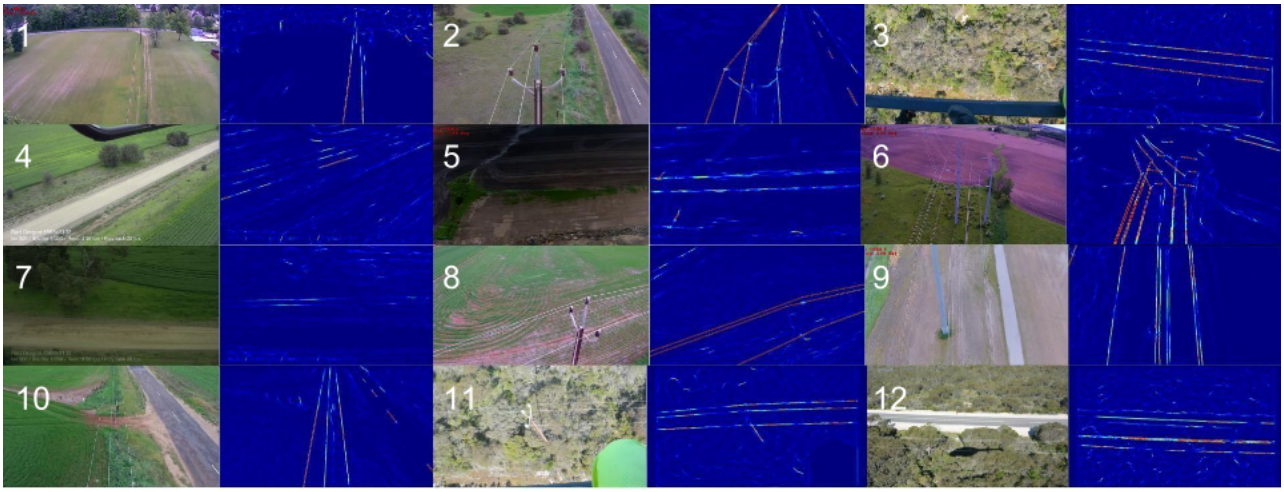

Fig. 5: Qualitative results on real images with model k32-k32-k32-k32-d1-d2-d4-d8-d16-d1-k2 trained on our synthetic dataset with input resolution of 720x1280. High resolution helps the network pick far off (Image 6), very thin wires (Images 3 and 4), those in clutter (Image 11) and taken when in high speed (Images 5 and 7). Video available at this link and our project page.

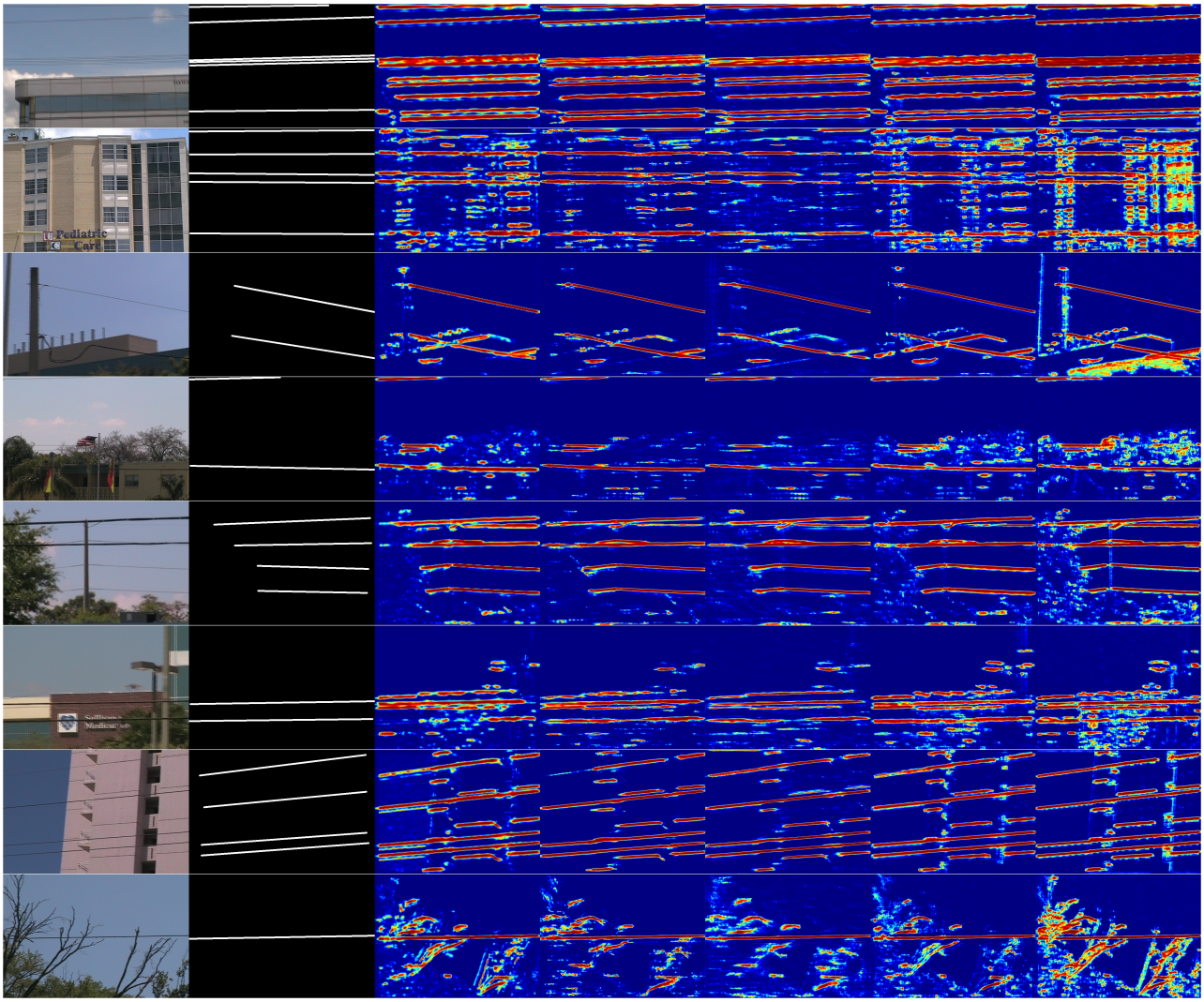

Fig. 6: Qualitative results of our top 5 models after finetuning on the USF dataset with RGB input. Left to right : Input image, Ground Truth, predictions with architecture k64-k64-p2-k128-k128-d1-d2-d4-d1-k2, k32-k32-k32-k32-d1-d2-d4-d8-d16-d1-k2, k32-k32-d1-d2-d4-d8-d16-d1-k2, k32-k32-k32-k32-k32-k32-d1-d2-d4-d8-d1-k2, k32-k32-k32-k32-k32-k32-d1-d2-d4-d1-k2.