REAL-TIME, FLIGHT-READY, NON-COOPERATIVE SPACECRAFT POSE ESTIMATION USING MONOCULAR IMAGERY

Kevin Black[*], Shrivu Shankar[*], Daniel Fonseka[*], Jacob Deutsch[*], Abhimanyu Dhir[*], and Maruthi R. Akella [†]

A key requirement for autonomous on-orbit proximity operations is the estimation of a target spacecraft's relative pose (position and orientation). It is desirable to employ monocular cameras for this problem due to their low cost, weight, and power requirements. This work presents a novel convolutional neural network (CNN)-based monocular pose estimation system that achieves state-of-the-art accuracy with low computational demand. In combination with a Blender-based synthetic data generation scheme, the system demonstrates the ability to generalize from purely synthetic training data to real in-space imagery of the Northrop Grumman Enhanced Cygnus spacecraft. Additionally, the system achieves real-time performance on low-power flight-like hardware.

INTRODUCTION

Frequent proximity operations between spacecraft in orbit are a key requirement for current and future activities such as formation flying, debris removal,[1] and on-orbit servicing.[2,3] Automation of these operations is crucial to their long-term value and viability. A fundamental requirement for autonomous proximity operations is the automatic detection of a target spacecraft's relative position and attitude, often referred to as pose estimation. Oftentimes, as in the case of debris removal, the target is non-cooperative: it does not employ any active communications or special markings to assist the sensing spacecraft. Furthermore, many of these missions utilize only small spacecraft with strict power and mass requirements. Monocular cameras — which are small in size, ubiquitous in nature, and consume little power compared to other sensors such as stereo cameras or LIDAR — are an ideal sensor for low-power non-cooperative pose estimation.

The problem of pose estimation from monocular imagery has been studied for many years. Early techniques focused on hand-engineered feature matching,[4,5] but have been known to exhibit to poor robustness and generalization capabilities. This is particularly true in the space environment, which presents unique challenges such as low signal-to-noise ratio, harsh and varied lighting conditions, and a dynamic Earth background. However, over the past decade, nearly all computer vision tasks such as monocular pose estimation have become increasingly dominated by deep convolutional neural networks (CNNs). Compared to prior techniques, CNNs have been shown to be more resilient to noise and better able to generalize to previously unseen scenarios. It is no surprise that CNNs are now being applied for in-space monocular pose estimation. The 2019 Satellite Pose Estimation

[*]Undergraduate Researcher, Texas Spacecraft Laboratory, The University of Texas at Austin
[†]Ashley H. Priddy Centennial Professorship in Engineering, Department of Aerospace Engineering and Engineering Mechanics, The University of Texas at Austin, AAS Fellow, Email: makella@mail.utexas.edu

Challenge (SPEC),[6] hosted by Stanford University and the European Space Agency (ESA), saw all of its top-performing submissions employ CNN-based deep learning models.

However, deep neural networks suffer from two major disadvantages: they require large amounts of labeled data to train and are computationally expensive to run. Due to the scarcity of labeled real images of spacecraft in orbit, most work on in-space monocular pose estimation has focused on synthetic or mockup-based imagery for both training and evaluation. For example, the dataset provided for the SPEC competition was the Spacecraft Pose Estimation Dataset (SPEED),[7,8] which consists of black-and-white synthetic images of the ESA Tango spacecraft as well as limited real images of a mockup. Models that perform well on a limited variety of artificial images will not necessarily transfer effectively to the real environment. There has also been relatively little discussion about the feasibility of running these models on flight-like hardware. Most CNNs require expensive computations that are optimized for a Graphics Processing Unit (GPU); however, the application of GPUs in the space domain is still a nascent field of study due to radiation exposure considerations.[9] While traditional processors are still capable of running CNNs, many spacecraft, especially smaller satellites such as CubeSats, only have low-power processors that are incapable of running large, state-of-the-art CNN models in real-time.

This work presents a monocular pose estimation system that addresses many of the challenges described above. The core of the system is a novel CNN-based pose estimation architecture that is computationally inexpensive, yet maintains state-of-the-art accuracy. Similar to some prior works,[10,11] this architecture is a 3-step process consisting of an object detection CNN, a keypoint regression CNN, and an off-the-shelf perspective-n-point (PnP) solver. It also includes a final error prediction step which identifies bad pose estimates and replaces them with "non-detections". Furthermore, the entire architecture is trained using a novel synthetic data generation scheme that produces photorealistic images with many of the degradations present in real space imagery. Synthetic images are generated using Cycles, a physically-based, ray-tracing, production rendering engine packaged with the open-source 3D graphics software Blender.[12]

Our aim is to present a nearly flight-ready system. As discussed above, real image performance, along with feasibility of using flight-like hardware, is essential for flight readiness. In contrast to most prior work, we analyze the ability of the pose estimation system to generalize to real images. These real images exhibit a wide range of poses, backgrounds, and the aforementioned degradations. In addition, we present a thorough benchmark of the system's performance on an Intel Joule 570x single-board computer, the same type which flew on the NASA Johnson Space Center (JSC) Seeker CubeSat mission in September 2019.[13]

We first provide a survey of related work. Then, we describe the details of our pose estimation architecture, as well as our synthetic data generation scheme using the Northrop Grumman Enhanced Cygnus vehicle as our target spacecraft. We next present results including the performance of our architecture on the SPEED dataset, the performance of our system on both synthetic and real images of Cygnus, and the hardware performance of our system on the Intel Joule 570x single-board computer.

RELATED WORK

There is much pre-existing work in the context of spacecraft monocular pose estimation, including a recently published overview of methods in the field by Cassinis et al. (2019).[14] Early methods of pose estimation relied on classical, non-learning methods such as Hough line detec-

tion[4] and Canny edge detection[5] for producing raw image features. However, pivotal papers such as Krizhevsky et al. (2012)[15] show that learned feature detection through the use of CNNs is both more accurate and robust for computer vision tasks.

In some cases, this has gone as far as solving for pose as the direct output of a CNN. Su et al. (2016)[16] show that by discretizing the viewpoint space, a CNN can be used to classify an input image directly into a binned pose. Kehlet et. al. (2017)[17] expand on this to add refinement and verification methods after the pose classification. Rather than modeling the problem as a classification task, Mahendran et al. (2017)[18] show that directly regressing a quaternion representation of pose is a viable method where minimal, if any, post-processing is required. In this case, the task of pose estimation can be completely learned by an end-to-end CNN. Although more and more computer vision applications now rely on end-to-end CNNs, based on the results of the Satellite Pose Estimation Challenge,[6] the best performing pose estimation models in the space domain still use classical pose solvers such as POSIT[19] or Perspective-n-Point (PnP)[20] attached to the output of a CNN-based keypoint detector. Much recent work[10,11,14] has shown such a keypoints-based pipeline to be one of the most viable methods for the specific task of monocular satellite pose estimation. Furthermore, all of these recent works first use an object detection step to crop to the region of interest in order to provide scale invariance to the keypoint model.

To compensate for uncertainties in keypoint predictions, Cassinis et al. (2020)[21] show the problem can be modeled as predicting a heatmap (where peaks represent high-confidence locations of a given keypoint in the input image) for each keypoint as opposed to regressing coordinates directly. This provides some interpretability as each prediction can be visualized as a feature-heatmap when juxtaposed with the original image, and the 2D output allows for richer keypoint features to be fed to the accompanying pose solver. By instead encoding the heatmap as a pixel-level displacement field, pose estimation accuracy can further be improved as in Hou et. al.'s MobilePose.[22]

While these architectural advances are shown to improve pose accuracy in aggregate, deployed pose applications also require additional guarantees on worst-case predictions. For spacecraft, Kalman filters are often applied to a stream of pose model predictions and sensor data to validate the likelihood of the estimated poses.[14,23] In CullNet,[24] Gupta et al. also show that bad pose estimates can be culled with the use of an additional error-predicting CNN. They use each candidate pose estimate to produce a corresponding 2D mask of the target object, then feed this mask along with the original image into a CNN which attempts to predict the accuracy of the pose estimate.

As discussed above, another common challenge of extra-terrestrial pose estimation is the scarcity of real data. This drives many methods (especially those that utilize CNNs) to employ transfer learning.[14] In some cases, this requires initially training the keypoint model on a completely separate domain to learn key CNN filters that are still generalizable to the target domain of space.[25] Other work attempts to use visual simulations to generate artificial training data. Notably, this includes the OpenGL-rendered SPEED dataset,[7,8] as well as the Unreal Engine-based simulator presented in Proenca and Gao (2019).[26] Generative Adversarial Networks (GANs) have also been employed in similar contexts to improve the realism of artificial training data.[27]

METHODS

The proposed pose estimation architecture is composed of a 3-step process. An object detection network first determines the 2D bounding box of the target spacecraft, and then feeds a cropped region of interest (RoI) to a separate keypoint regression network. The keypoint regression network

regresses the 2D locations of predetermined 3D surface keypoints on the spacecraft model, from which the full pose is obtained using an off-the-shelf Perspective-n-Point (PnP) solver. This overall architecture was chosen because prior works,[6,10,11,14] as well as our own experiments, have found that such 3-step processes tend to outperform other techniques such as direct pose regression or classification. The final error prediction step predicts the approximate error of a given pose estimate. These error predictions can then be used to reject bad pose estimates and produce "non-detections" instead.

Keypoint Selection

We utilize surface keypoints for pose estimation rather than the corners of the 3D bounding box. The bounding box corners are often used in systems that need to handle multiple objects; however, as we only need to consider a single known object, we use surface keypoints due to their stronger correspondence with local image features.

To select keypoints, we exploit our access to the 3D mesh model of the Cygnus spacecraft that is used for synthetic image generation. A user-specified number of keypoints (n) are generated randomly using a sample elimination algorithm[28] that produces Poisson disk sample sets. The result is that the generated points are approximately "evenly spread out" in 3D space, serving well for the task of pose estimation for any given number of points. We find that using $n = 20$ keypoints produces good results, and use that value for all experiments.

For evaluation on the SPEED dataset, we use the same 11 surface keypoints as in Park et. al. (2019)[10] and Chen et. al. (2019).[11]

Object Detection

For the object detection network, we use off-the-shelf models from the TensorFlow Object Detection API.[29] These are easy to train and include access to weights pretrained on the COCO 2017 dataset.[30] We select the SSD MobileNetv2 architecture[31] with an input resolution of 320×320, as it provides adequate accuracy while being the least computationally expensive architecture with an available pretrained checkpoint.

We operate in the context of a guaranteed single spacecraft in every image, so we always take the top bounding box detection regardless of confidence. To produce the final RoI, the detected bounding box is made square by setting the length of its shorter side to that of its longer side and then expanded by 25% to ensure that the entire spacecraft lies inside.

Keypoint Regression

After bounding box detection, the image is cropped to the RoI and rescaled to a resolution of 224×224. The resulting image is then fed to the keypoint regression network.

The keypoint regression network (Figure 1) is based on a MobileNetv2[31] backbone and follows the general architecture of MobilePose.[22] Unlike MobilePose, we keep the original MobileNetv2 network and cut it off before its final convolutional layer, producing a $7 \times 7 \times 320$ feature map. The feature map is upsampled back to 14×14 with a transposed convolutional layer, followed by a skip connection via concatenation to the last layer in MobileNetv2 with the same 14×14 scale. This is followed by two more of MobileNetv2's inverted residual blocks, and then a final 1×1 convolution to produce an output tensor with dimension $14 \times 14 \times 2n$, where n is the number of keypoints.

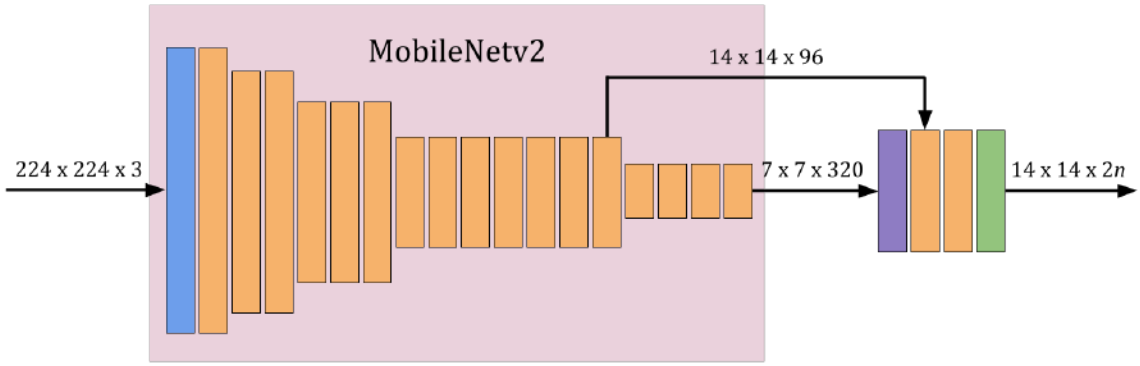


Figure 1: Layer diagram for the keypoint regression network. Blue rectangles are convolutional layers, purple are transposed convolutional layers, orange are inverted residual blocks, and green is the output 1×1 convolution.

Each of the 14×14 entries in the output tensor make a prediction for the 2D locations of all n keypoints, meaning each keypoint has 196 predictions. As in MobilePose, the network does not predict the keypoints' 2D locations directly, but rather each of the 14×14 output locations predict an offset from their corresponding location in the image to each of the keypoint's 2D locations. The network is trained with L1 (mean absolute error) loss as to be more lenient to outliers, since not every output location may be able to accurately predict the location of every keypoint.

The goal of the downsampling-upsampling architecture is to combine both global and local image features to obtain more accurate keypoint predictions. As with the object detection network, the depthwise separable convolutions of MobileNetv2 drastically reduce the number of computations compared to typical CNNs and allow real-time inference on low-power devices.

Pose Estimation

The keypoint regression network produces $14 \times 14 = 196$ predictions for each of n keypoints, resulting in $196n$ total correspondences between 3D keypoints and their 2D locations in the image. The spacecraft pose can then be obtained using off-the-shelf PnP solver software. However, the keypoint regression is deliberately designed to be lenient to outliers, so random sample consensus (RANSAC)[20] is also necessary to make the solution more robust. We use the *solvePnPRansac* function from the open-source OpenCV[32] library configured to use the EPnP[33] algorithm, which was found to provide the best balance of speed and accuracy.

Error Prediction

Similar to CullNet,[24] a 2D binary mask of the target spacecraft is first created using the estimated pose. This is done quickly using OpenGL. However, rather than concatenating the mask with the original image, the mask is instead used to "cut out" the shape of the target spacecraft; i.e., inside the mask, the original image is kept and outside the mask, every pixel is set to zero. All of these operations occur within the 224×224 cropped RoI. The resulting cutout is used as the input to the error prediction network.

Rather than trying to predict an error derived from the final pose solution, we found it more

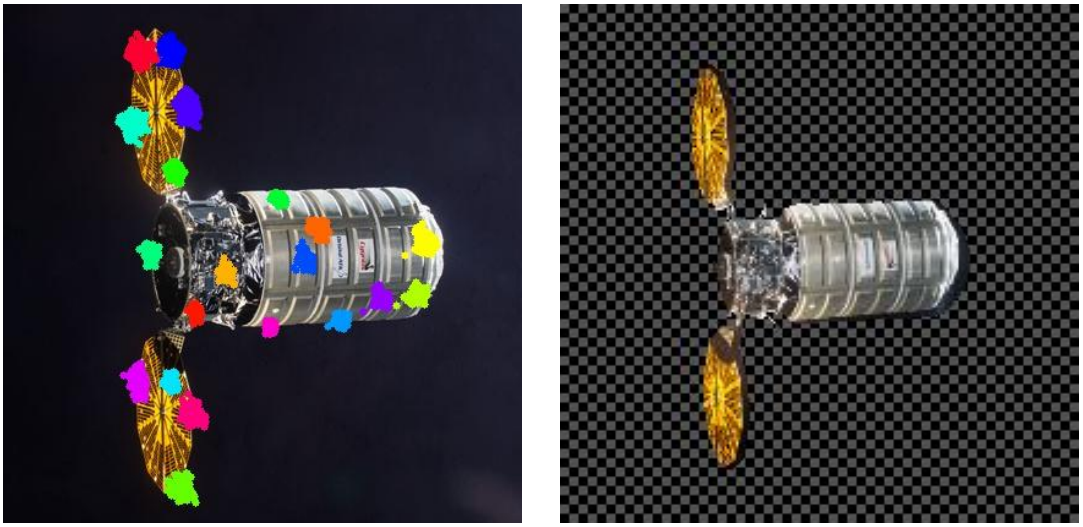


Figure 2: Left: an example output of the keypoint regression network. Each same-colored cluster of points is the 196 predictions for each keypoint. Right: an example of a mask cutout fed to the error prediction network. The background has been filled with a checkered pattern for illustration purposes; however, in practice those pixels would all be set to black.

accurate to instead predict an error derived from the output of the keypoint regression network. The error metric we use is the mean Euclidean distance between all 2D keypoint estimates and their corresponding ground truth locations:

$$E_k = \frac{1}{196n} \sum_{i=0}^{n} \sum_{j=0}^{196} ||\boldsymbol{k_i} - \hat{\boldsymbol{k}}_{\boldsymbol{i,j}}|| \tag{1}$$

where $\boldsymbol{k_i}$ is the ground truth location of the ith keypoint, and $\hat{\boldsymbol{k}}_{\boldsymbol{i,j}}$ is the jth estimate for the ith keypoint from the keypoint regression network. Keypoint locations are expressed as 2D pixel coordinates within the 224×224 cropped RoI. The task of the error prediction network is thus to take a cutout as the input and regress E_k.

The cutout method allows the error prediction network to reuse nearly the full architecture and weights of the keypoint regression network, with only the final 1×1 convolution replaced with a fully connected layer. We found the ability to initialize the error prediction network with the weights of the keypoint regression network to significantly aid training.

Once the network produces a prediction for the keypoint error, $\hat{E}_k$, a threshold must be chosen above which pose estimates are rejected. Using the synthetic test set, we choose 20 pixels as a good balance between rejecting bad estimates and keeping good ones.

Synthetic Data Generation

Due to the scarcity of labeled real images, we generate synthetic images for our experiments with the Northrop Grumman Enhanced Cygnus spacecraft. Images are generated using Blender,[12] an open-source 3D modeling suite, and its Cycles* rendering engine. Cycles is a physically-based, ray-tracing, production-quality rendering engine. In comparison to prior work which has used OpenGL

*https://www.cycles-renderer.org/

(a) Cygnus with no augmentations in front of a real Earth background

(b) Cygnus in front of a randomized background

(c) Cygnus with a blur effect

(d) Cygnus with a lens flare effect

(e) Cygnus with a "fog glow" effect

(f) Cygnus with a "simple star" effect

Figure 3: Example synthetic images with and without augmentations. The bottom row all falls under the category of "glare and lens flares".

shaders[7] or Unreal Engine,[26] we use Cycles to better mimic real camera parameters and produce more photorealistic images. We obtain our model of the Cygnus spacecraft from an open-source GitHub repository*.

Blender's powerful Python API also allows easy configuration of the camera position, lighting angle, and spacecraft pose. We use a custom-built library† to leverage this API and enable flexible, automated generation of labeled synthetic images.

To better simulate images taken in the true space environment, we additionally use Blender to add custom augmentations to the synthetic data. These augmentations include various types of randomized glare, lens flares, blur, and background images. Two types of background images are used. The first is real satellite photos of Earth, intended to further improve realism. The second is completely randomized non-realistic background images, intended to reduce overfitting during training and improve generalization to the variance present in real images. These augmentations to the training data have led to a marked increase in the performance of the pose estimation system on real images.

*https://github.com/brickmack/Blender-Spaceflight-Models
†https://autognc-starfish.readthedocs.io

Training Details

The object detection network is initialized with weights pre-trained on the COCO 2017 dataset.[30] It is trained for 40,000 steps using the Adam optimizer with a batch size of 10 and an initial learning rate of 0.001 that decays to 0.0001 at 20,000 steps and 0.00001 at 30,000 steps. Random flips, 90° rotations, and crops are applied.

The MobileNetv2 portion of the keypoint regression network is initialized with weights pre-trained on the ImageNet dataset.[34] The entire network is then trained for 300 epochs using the Adam optimizer with a batch size of 60 and an initial learning rate of 0.001 that decays to 0.0001 at 100 epochs and begins decaying exponentially after 200 epochs as $0.001e^{-0.05(\text{epoch}-200)}$. The RoIs used for training are derived from the ground-truth bounding box labels with random translations and expansions applied. Random flips, 90° rotations, brightness, contrast, and saturation augmentations are also applied.

The error prediction network is initialized with weights from the fully trained keypoint regression network. The training data is a holdout test set consisting of synthetic images never seen by the object detection or keypoint regression networks. The network is trained for 50 epochs using the Adam optimizer with a batch size of 64 and a learning rate of 0.001.

RESULTS

Evaluation Metrics

For evaluating the object detection network individually, we use the standard intersection-over-union (IoU) of the detected and ground-truth bounding boxes. We also report an additional metric we call RoI accuracy, which is the proportion of images in which the final region of interest (the detected bounding box squared and expanded by 25%) contains the entire ground-truth bounding box.

Pose is represented as a pair $(\boldsymbol{q}, \boldsymbol{t})$ where $\boldsymbol{q}$ is a rotation represented as a quaternion and $\boldsymbol{t}$ is a 3-dimensional translation vector. Applied together, $(\boldsymbol{q}, \boldsymbol{t})$ align the camera reference frame with the target reference frame. 4 metrics will be used to measure the error between a ground-truth pose $(\boldsymbol{q}, \boldsymbol{t})$ and predicted pose $(\hat{\boldsymbol{q}}, \hat{\boldsymbol{t}})$. The first metric measures the rotation error, computed as:

$$E_R = 2 \arccos |\boldsymbol{q} \cdot \hat{\boldsymbol{q}}| \tag{2}$$

where $\cdot$ is the vector dot product. E_R corresponds to the angle of the smallest rotation that aligns $\hat{\boldsymbol{q}}$ and $\boldsymbol{q}$. Another two metrics measure the translation error:

$$E_T = ||\boldsymbol{t} - \hat{\boldsymbol{t}}|| \tag{3}$$

$$E_{TN} = \frac{||\boldsymbol{t} - \hat{\boldsymbol{t}}||}{||\boldsymbol{t}||} \tag{4}$$

E_T is the distance between the ground truth and predicted translations in a real unit, such as meters; E_{TN} is this distance normalized by the ground truth distance between the target and the camera. A final metric combines the two errors:

$$E_C = E_R + E_{TN} \tag{5}$$

For experiments with Cygnus, these pose metrics will be reported twice: once for the full dataset, and once with rejected pose estimates removed using the error prediction network. The proportion of pose estimates that were rejected will also be reported.

SPEED Dataset

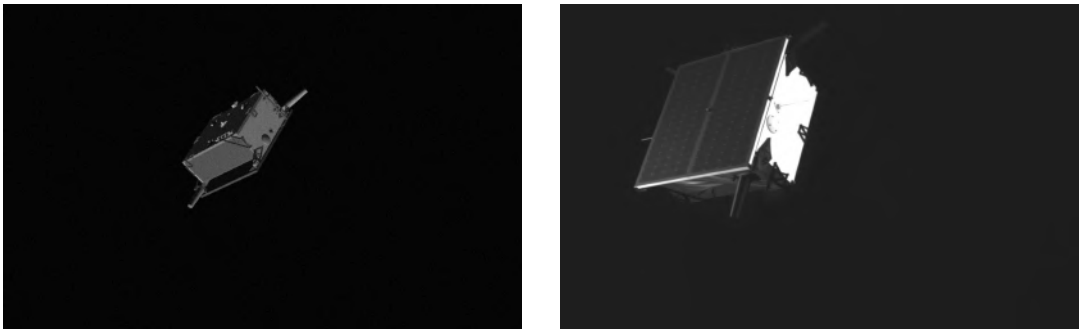


Figure 4: A synthetic image (left) and real image of a mockup (right) from the SPEED dataset

First, our pose estimation system is evaluated on the SPEED dataset[8] in order to provide a comparison to the top submissions in the Satellite Pose Estimation Challenge (SPEC).[6] The SPEED dataset consists of 12,000 training images and 2,998 test images of the ESA Tango spacecraft, all grayscale and synthetic. 300 real grayscale images of a Tango mockup are also included. The images have a resolution of 1900×1200 and a horizontal field of view of 35.1°. The distance of Tango from the camera ranges from 3 to 40.5 meters, with bounding box side length ranging from 27 pixels to 1432 pixels with a mean of 371 pixels.

Table 1: Satellite Pose Estimation Challenge Comparison

SPEC Rank	Name	Mean E_C (synthetic)	Mean E_C (real)	# parameters (millions)
1	`UniAdelaide`[11]	0.0094	0.3752	60+
2	`EPFL_cvlab`*	0.0215	0.1140	60+
	Ours	**0.0409**	**0.2918**	**6.9**
3	`pedro_fairspace`[26]	0.0570	0.1555	60+
4	`stanford_slab`[10]	0.0626	0.3951	11.2

Table 1 shows our system ranked alongside the top 4 submissions to the SPEC competition. The labels for the SPEED test set have not been released publicly, so we are only able to report the E_C scores obtained from the SPEC post-mortem scoring server†. We are also not able to test the error prediction step, since a pose estimate must be returned for every image. The competition ranks submissions based on the synthetic mean E_C score only. The parameter counts are obtained from the publications corresponding to each of the submissions, and roughly correspond to a network's computational needs.

Our pose estimation system achieves competitive accuracy, placing 3rd in both synthetic and real image score while having the fewest parameters. The only other top submission with a comparable network size is that of `stanford_slab`.

*`https://indico.esa.int/event/319/attachments/3561/4754/pose_gerard_segmentation.pdf`

†`https://kelvins.esa.int/pose-estimation-challenge-post-mortem/`

Cygnus: Synthetic Data

Table 2: Dataset Breakdown

Description	Number of Images
No augmentations	3,000
Glare and lens flares	3,000
Blur	3,000
Spacecraft partially out-of-frame	3,000
No augmentations, real Earth background	2,000
Glare and lens flares, real Earth background	2,000
Blur, real Earth background	2,000
No augmentations, randomized background	2,000
Total	**20,000**

Table 2 shows the composition of the synthetic dataset we use for all experiments with the Cygnus spacecraft. All images have resolution 1024×1024 with a field of view of 39.6°. Position, orientation, and lighting angle are all randomized. Target distance from the camera is randomized from 35 to 75 meters. Not accounting for images with Cygnus partially out of frame, bounding box side length ranges from 76 to 459 pixels, with a mean of 232 pixels. The dataset is split into 64% training, 16% validation, and a 20% holdout test images.

Table 3: Synthetic Dataset Performance

	Metric	Median	Mean
Object Detection Metrics	RoI Accuracy	-	0.98
	IoU	0.95	0.92
Pose Metrics	E_R (deg)	3.22	25.96
	E_T (meters)	0.93	85.49
	E_{TN}	0.017	1.565
	E_C	0.074	2.018
Rejected Estimates ($\hat{E}_k > 20$) Removed	Proportion Rejected	-	0.18
	E_R (deg)	2.64	6.45
	E_T (meters)	0.71	1.08
	E_{TN}	0.013	0.019
	E_C	0.062	0.132

Table 3 shows the performance of the pose estimation system on the aforementioned holdout set. The object detection network performs very well: the resulting RoI includes the entire spacecraft in 98% of images. The pose errors, on the other hand, are characterized by extreme outliers: due to the various augmentations, the system produces a few "bad estimates" on the most difficult images. (This includes, of course, the 2% of images where the initial RoI is inaccurate.) In particular, the translation errors are unbounded, and so a few extreme outliers can severely inflate the mean. However, the error prediction network easily detects these extreme outliers. As mentioned before, the $\hat{E}_k$ threshold is chosen using this synthetic dataset to provide a good balance between rejecting bad estimates and keeping good ones. The chosen threshold of 20 pixels results in 18% of estimates being rejected while bringing down the mean errors to reasonable values.

Overall, as with the SPEED dataset, it is expected that the system will perform well on the same type of synthetic data it is trained on. The synthetic image error provides a rough lower bound on the achievable real image error.

Cygnus: Real Data

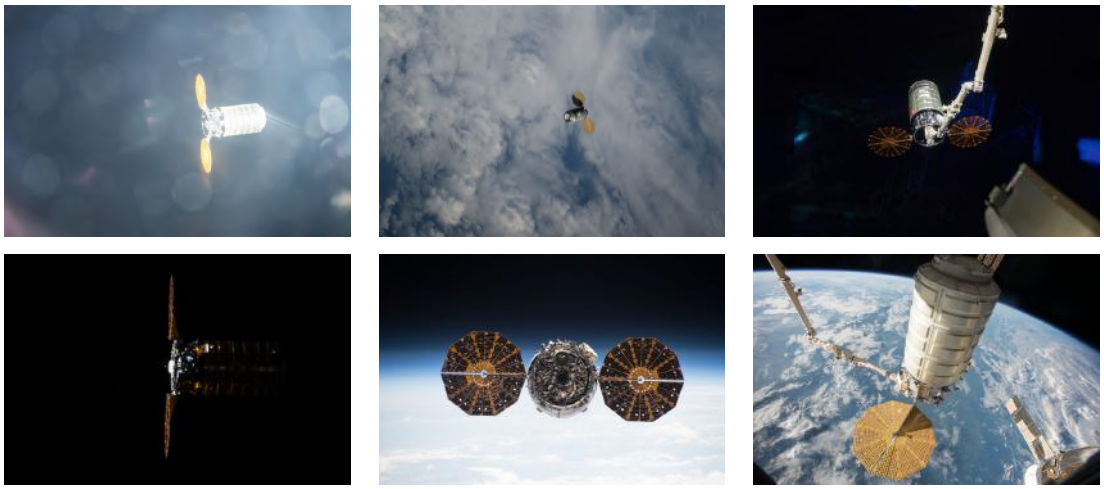


Figure 5: Examples of real images

The real dataset consists of 540 photos of the Cygnus spacecraft taken in orbit. The photos are obtained from NASA*. They cover a variety of poses and lighting conditions, and include degradations such as glare and motion blur. The field of view varies from 1.8° to 86.9°, and bounding box side length varies from 54 pixels to 744 pixels with a mean of 231 pixels.

Table 4: Human Labeling Error

Metric	Median	Mean
E_R (deg)	1.71	2.11
E_T (meters)	1.91	3.66
E_{TN}	0.011	0.020
E_C	0.045	0.056

The images are hand-labeled with pose information using a custom-built tool† that overlays a 3D model of the spacecraft on top of each image and allows the user to line it up with the real spacecraft. We also estimate the human error associated with this labeling tool by creating a synthetic dataset with the same number of images and similar poses as the real image dataset. This gives access to known ground-truth poses so that the human labeling error can be measured. The results are presented in Table 4. We find that the human labeling error is small enough to still conduct a robust analysis of the pose estimation system's error.

*https://images.nasa.gov/

†https://github.com/autognc/opat-js

Table 5: Real Dataset Performance

	Metric	Median	Mean
Object Detection Metrics	RoI Accuracy	-	0.95
	IoU	0.90	0.85
Pose Metrics	E_R (deg)	7.33	35.26
	E_T (meters)	7.36	112.83
	E_{TN}	0.032	0.317
	E_C	0.170	0.932
Rejected Estimates ($\hat{E}_k > 20$) Removed	Proportion Rejected	-	0.16
	E_R (deg)	6.48	29.88
	E_T (meters)	6.19	10.94
	E_{TN}	0.026	0.040
	E_C	0.156	0.561

Table 5 shows the performance of the pose estimation system on the real images. The object detection network still performs quite well. The pose errors once again contain some "bad estimates" on the more difficult images. Keeping the same $\hat{E}_k$ threshold of 20 pixels, the error prediction network rejects 16% of the estimates and easily recognizes all of the most extreme outliers. Overall, considering the relative difficulty of the real image test set, the system generalizes well.

However, the error prediction network notably does not help very much with the mean rotation error compared to the mean translation error. This is because the Cygnus spacecraft has a major symmetry: namely, a 180° rotation around the barrel is difficult to distinguish, indicated primarily by the presence or absence of two logos. This poses an especially challenging obstacle to the keypoint regression network when generalizing to real data. In some images, the network misses the subtle features necessary to determine the correct orientation, due to the fact that it was trained using an imperfect synthetic model. This particular type of mistake produces a confident estimate that is unlikely to be rejected by the error prediction network, yet has near 180° rotation error (see Figure 7 (d) and (e) for examples).

Table 6: Rotation Error Allowing Symmetrical Orientations

	Metric	Median	Mean
Pose Metrics	E_R (deg)	6.20	11.97
	E_C	0.154	0.526
Rejected Estimates ($\hat{E}_k > 20$) Removed	E_R (deg)	5.38	8.37
	E_C	0.136	0.186

Table 6 presents updated rotation and combined error values if the issue of symmetry is ignored; i.e., the rotation error is computed as the smallest angle rotation that aligns the estimated pose with either of the two ambiguous orientations. These values correspond to the feasible mission scenario where the orientation of Cygnus about this axis of symmetry is not important, only the orientation of its overall shape and major features. Without the 180° outliers, the pose estimation system achieves a rotation error within 10° and a combined error lower than it does on the much easier SPEED real image test set.

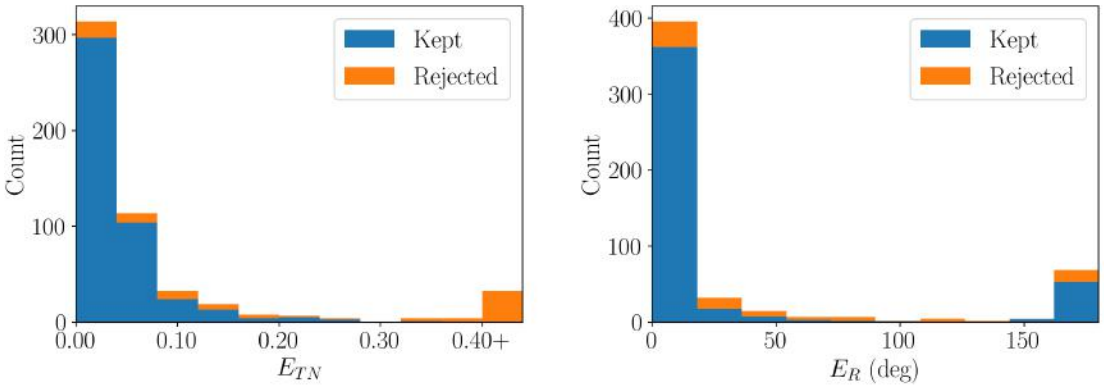
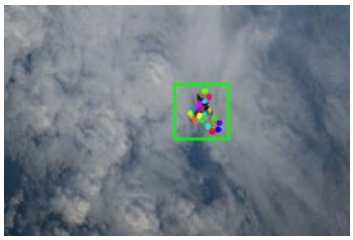
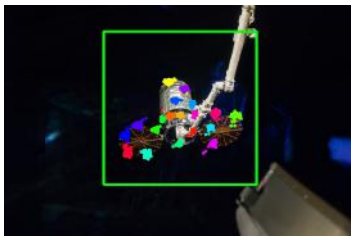


Figure 6: Histogram of translation error (left) and rotation error (right) that also shows images rejected by the error prediction network. The error prediction step helps immensely with translation error, rejecting nearly all of the outliers. However, it is much less effective for rotation error, due to the rotational symmetry. All of the outliers near 180° are otherwise "good estimates" with the incorrect orientation (e.g. Figure 7 (d) and (e)).

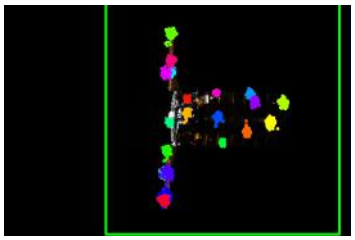
(a) $E_R = 4.22°$, $E_{TN} = 0.013$ Noisy keypoint estimates can still produce a good pose estimate by using RANSAC.



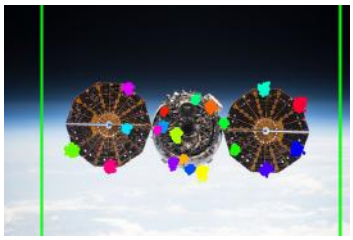
(b) $E_R = 2.72°$, $E_{TN} = 0.034$ The object detection network performs well at multiple scales.



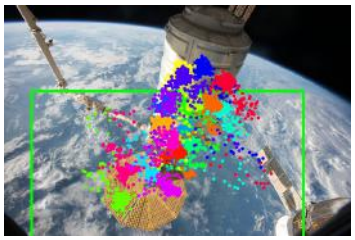
(c) $E_R = 6.34°$, $E_{TN} = 0.045$ The system performs well even with an object slightly occluding Cygnus, which does not appear at all in the training data.



(d) $E_R = 177.6°$, $E_{TN} = 0.025$ In dark lighting, the network cannot see the logos and guesses the incorrect orientation around the barrel.



(e) $E_R = 179.5°$, $E_{TN} = 0.009$ This viewing angle obscures the logos, again making it very difficult for the network to determine the correct orientation.



(f) $E_R = 17.50°$, $E_{TN} = 0.105$ This is an obvious failure case, which is rejected easily by the error prediction network ($\hat{E}_k = 86$).

Figure 7: The pose estimation system applied to the 6 real images from Figure 5. The green box is the RoI from the object detection network, and the colored dots are the output from the keypoint detection network same as in Figure 2.

Hardware Performance

To benchmark the pose estimation model, we use the Intel Joule 570x single-board computer. It has been used previously in a CNN-based visual navigation system onboard a 3U CubeSat,[13] establishing its viability as flight-ready commercial off-the-shelf (COTS) hardware. The computational capabilities of the Joule are severely limited in comparison to typical ground-based hardware such as a laptop or smartphone. Achieving real-time performance on the device demonstrates the suitability of our system for nearly any realistic mission hardware.

The Joule contains a 1.7GHz Intel Atom processor, 4 GB of RAM, and no dedicated graphics card. With these specifications, the pose estimation model runs at approximately 0.56 Hz using TensorFlow.[35] However, a default TensorFlow installation is not optimized for inference on low-power devices. Proper optimization for low-power devices is known to increase speed by 10-20 times. The most common toolkit for such optimization is TensorFlow Lite;[35] however, TensorFlow Lite is optimized for the ARM architecture that is dominant among mobile devices. The analogous toolkit for x86 Intel processors is OpenVINO*. By converting the pose estimation system into the OpenVINO format, inference speed increased to 6.6 Hz without a reduction in model accuracy.

Table 7: Inference Speed Comparison (ms)

Tool	Object Detection	Keypoint Regression	PnP Solver	Error Prediction	**Combined**
TensorFlow	710	434	10	621	**1777**
OpenVINO	68	31	10	41	**152**

Table 7 presents inference times for each component of the system. Timing data was collected by measuring the average inference time over over a period of 10 minutes, allowing the Joule reach thermal equilibrium. Once at thermal equilibrium, the power consumption for all models averaged 3.7 Watts.

CONCLUSION

The presented pose estimation system achieves state-of-the-art accuracy on the Spacecraft Pose Estimation Dataset (SPEED). At the same time, it runs at 6.6 Hz — fast enough to be considered real-time for most applications — on low-power commercial-off-the-shelf (COTS) hardware suitable for small satellites. Most importantly, given a complete lack of real training data, the system is still able to perform accurately on real images with difficult and diverse conditions. In the case of particularly difficult images or random failures, the error prediction network is able to automatically filter out bad estimates.

As such, the presented work comprises a nearly flight-ready pose estimation system. It is suitable to run in real-time on a small satellite with limited power requirements and a single monocular camera. The synthetic image generation and training techniques are easily applicable to any spacecraft. Overall, the system's real image performance is good enough to be useful for many applications in autonomous proximity operations; examples include formation flying, debris removal, and on-orbit inspection or servicing.

Future work will focus on integrating the pose estimation system with the other components of a fully flight-ready system. Primarily, this includes applying a dynamics-aware filtering algorithm

*https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html

(such as a Kalman filter) to the produced pose estimates. However, properly evaluating such a fully integrated system requires labeled time-series data from real proximity operations, which is even more difficult to obtain than ordinary real images. Another avenue of future work is to further address the issue of rotational symmetry, which is the biggest weakness of the presented pose estimation system. There exists prior work that aims to solve the specific problem of rotational symmetry in pose estimation, which could possibly be applied to this work.

ACKNOWLEDGEMENTS

We would like to thank Siddarth Kaki for his support and feedback on a variety of research topics. We also thank Evan Wilde for his aid with Blender modeling. Finally, we thank the NASA Johnson Space Center, in particular Sam Pedrotty, which provided funding that contributed to this work.

REFERENCES

[1] J. Forshaw, G. Aglietti, N. Navarathinam, H. Kadhem, T. Salmon, A. Pisseloup, E. Joffre, T. Chabot, I. Retat, R. Axthelm, S. Barraclough, A. Ratcliffe, C. Bernal, F. Chaumette, A. Pollini, and W. Steyn, "RemoveDEBRIS: An in-orbit active debris removal demonstration mission," *Acta Astronautica*, Vol. 127, 06 2016, 10.1016/j.actaastro.2016.06.018.

[2] B. Sullivan, D. Barnhart, L. Hill, P. Oppenheimer, B. L. Benedict, G. V. Ommering, L. Chappell, J. Ratti, and P. Will, *DARPA Phoenix Payload Orbital Delivery System (PODs): "FedEx to GEO"*, 10.2514/6.2013-5484.

[3] B. B. Reed, R. C. Smith, B. J. Naasz, J. F. Pellegrino, and C. E. Bacon, *The Restore-L Servicing Mission*, 10.2514/6.2016-5478.

[4] A. Cropp, *Pose estimation and relative orbit determination of a nearby target microsatellite using passive imagery.* PhD thesis, University of Surrey (United Kingdom)., 2001.

[5] X. Du, B. Liang, W. Xu, and Y. Qiu, "Pose measurement of large non-cooperative satellite based on collaborative cameras," *Acta Astronautica*, Vol. 68, No. 11, 2011, pp. 2047 – 2065, https://doi.org/10.1016/j.actaastro.2010.10.021.

[6] M. Kisantal, S. Sharma, T. H. Park, D. Izzo, M. Märtens, and S. D'Amico, "Satellite Pose Estimation Challenge: Dataset, Competition Design, and Results," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 56, No. 5, 2020, pp. 4083–4098, 10.1109/TAES.2020.2989063.

[7] S. Sharma, C. Beierle, and S. D'Amico, "Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks," *2018 IEEE Aerospace Conference*, 2018, pp. 1–12.

[8] S. Sharma, T. H. Park, and S. D'Amico, "Spacecraft Pose Estimation Dataset (SPEED)," Stanford Digital Repository. Available at: https://doi.org/10.25740/dz692fn7184, 2019.

[9] L. Kosmidis, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, "GPU4S: Embedded GPUs in Space," *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 399–405.

[10] T. H. Park, S. Sharma, and S. D'Amico, "Towards Robust Learning-Based Pose Estimation of Noncooperative Spacecraft," *2019 AAS/AIAA Astrodynamics Specialist Conference, Portland, Maine*, August 11-15 2019.

[11] B. Chen, J. Cao, A. Parra, and T. Chin, "Satellite Pose Estimation with Deep Landmark Regression and Nonlinear Pose Refinement," *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 2816–2824.

[12] B. O. Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2020.

[13] N. Dhamani, G. Martin, C. Schubert, P. Singh, N. Hatten, and M. R. Akella, *Applications of Machine Learning and Monocular Vision for Autonomous On-Orbit Proximity Operations*, 10.2514/6.2020-1376.

[14] L. Pasqualetto Cassinis, R. Fonod, and E. Gill, "Review of the robustness and applicability of monocular pose estimation systems for relative navigation with an uncooperative spacecraft," *Progress in Aerospace Sciences*, Vol. 110, 2019, p. 100548, https://doi.org/10.1016/j.paerosci.2019.05.008.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[16] H. Su, C. Ruizhongtai Qi, Y. Li, and L. Guibas, "Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views," 05 2015, 10.1109/ICCV.2015.308.

[17] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again," *CoRR*, Vol. abs/1711.10006, 2017.

[18] S. Mahendran, H. Ali, and R. Vidal, "3D Pose Regression Using Convolutional Neural Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 494–495, 10.1109/CVPRW.2017.73.

[19] D. DeMenthon and L. Davis, "Model-Based Object Pose in 25 Lines of Code," Vol. 15, 02 1998, 10.1007/3-540-55426-2_38.

[20] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Readings in Computer Vision* (M. A. Fischler and O. Firschein, eds.), pp. 726 – 740, San Francisco (CA): Morgan Kaufmann, 1987, https://doi.org/10.1016/B978-0-08-051581-6.50070-2.

[21] L. P. Cassinis, R. Fonod, E. Gill, I. Ahrns, and J. G. Fernandez, *CNN-Based Pose Estimation System for Close-Proximity Operations Around Uncooperative Spacecraft*, 10.2514/6.2020-1457.

[22] T. Hou, A. Ahmadyan, L. Zhang, J. Wei, and M. Grundmann, "MobilePose: Real-Time Pose Estimation for Unseen Objects with Weak Shape Supervision," 2020.

[23] L. Zhang, H. Yang, H. Cai, and S. Qian, "Kalman Filtering for Relative Spacecraft Attitude and Position Estimation: A Revisit," *Journal of Guidance, Control, and Dynamics*, Vol. 37, 09 2014, pp. 1706–1711, 10.2514/1.G000204.

[24] K. Gupta, L. Petersson, and R. Hartley, "CullNet: Calibrated and Pose Aware Confidence Scores for Object Pose Estimation," *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 2758–2766.

[25] J.-F. Shi, S. Ulrich, and S. Ruel, "CubeSat Simulation and Detection using Monocular Camera Images and Convolutional Neural Networks," 01 2018, 10.2514/6.2018-1604.

[26] P. F. Proença and Y. Gao, "Deep Learning for Spacecraft Pose Estimation from Photorealistic Rendering," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 6007–6013.

[27] B. Joshi, M. Modasshir, T. Manderson, H. Damron, M. Xanthidis, A. Quattrini Li, I. Rekleitis, and G. Dudek, "DeepURL: Deep Pose Estimation Framework for Underwater Relative Localization," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, 2020, p. accepted.

[28] C. Yuksel, "Sample Elimination for Generating Poisson Disk Sample Sets," *Comput. Graph. Forum*, Vol. 34, May 2015, p. 25–32, 10.1111/cgf.12538.

[29] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3296–3297.

[30] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," *European conference on computer vision*, Springer, 2014, pp. 740–755.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[32] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[33] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate O(n) solution to the PnP problem," *International Journal of Computer Vision*, Vol. 81, 02 2009, 10.1007/s11263-008-0152-6.

[34] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[35] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. Software available from tensorflow.org.