Abstract

Deep neural networks are known to be susceptible to adversarial perturbations – small perturbations that alter the output of the network and exist under strict norm limitations. Universal perturbation can be constructed to alter an arbitrary VO model's output on a set of inputs if the inputs match the distribution of the training set of the model. This has been shown in recent papers, specifically [4]. We build upon the latter model and modify it's loss function, adding terms aiming to facilitate the translation error of a VO model.

Specifically, the model we are trying to fool is Tartan VO [3]. A neural network based VO inference model, the input to which is a set of consecutive images and the output of which is the optical flow, translation and rotation between the frames.

In order to improve upon the results of the algorithm on a much smaller dataset, we try several methods. We try adding new terms to the original loss function in [4]. Aiming to target the early part of the VO model modifying the optical flow output.

We also try APGD[6] - a recent optimization method that builds on top of the standard PGD attack. We alter the algorithm's parameters so that it fits the optimization process we observe on the baseline method.

Finally, we add the two methods together to see if their effect adds up.

We run training and evaluation with no separate test set because of data and time scarcity.

We compare the baseline[4] performance vs. our additions in two parameters: optimization saturation and speed, based on the difference between accumulated losses of perturbed and unperturbed performances, and that distance relative to the clean error.

We find a better result of the APGD attack on all evaluation sets except one

Introduction

Adversarial attacks. Deep neural networks are expressive models that have achieved state of the art performance on speech and visual recognition tasks. They succeed mainly because of their expressiveness, though there are methods to cause a network to misclassify an image.

Universal adversarial attacks. DNNs are but one family of models discovered to be susceptible to adversarial perturbations – small norm bounded attacks which significantly alter the model's output. In addition, the specific nature of these perturbations is not a random artifact of learning: the same perturbation can cause a different network, that was trained on a different subset of the dataset, to misclassify the same input [2]. This type of adversarial attacks is termed "Universal" and is a more realistic setting for threats on DNN performance in real life applications where the specific model is not available to the attacker.

Monocular visual odometry (VO) is a set of models that aim to infer the relative camera motion (position and orientation) between two corresponding viewpoints. Recently, DNN-based VO models have outperformed traditional monocular VO methods. Specifically, the model suggested by Wang et al. (2020b)[3] shows a promising ability to generalize from simulated training data to real scenes. The model is trained in two stages – first, a network is trained to output optical flow from two input images. The flow serves as input along with scale characteristics to a second network, which outputs a relative translation and rotation (angle-axis) vectors.

Passive patch attacks of VO models. We study a special case of universal attacks in which a patch is planted in the environment in order to fool visual odometry systems. The norm of the attack is bounded only by the patch's physical size in the scene and the end goal of the attack is to maximize the inference error of the model. This type of attack has achieved significant performance in [4] fooling [3] on in sample data as well as generalization tasks.

Rotation and translation inferred from optical flow. Optical flow is a per pixel prediction which tries to estimate the motion of each pixel between two consecutive frames. Our odometry model [3] produces the flow as a DNN output, but there are several principles regarding motion inference that must hold true for all VO models, namely the additive dependence of optical flow on rotation and translation, and the structure of the OF fields that these two produce [7].

The PGD attack is a general-purpose optimization scheme in norm bound attacks based on taking the maximal allowed distance in the gradient direction (FSGM). In PGA we do FSGM on small $\alpha \ll \epsilon$ balls and iterate the process where ϵ is the norm's bound. The scheme is given fully in appendix A of [4].

In our work we use an elaboration of the PGD attack named APGD to check it's effect on the optimization process and eventual results.

We compare the models using the relative error distance of the perturbed model predictions from the clean ones given in Equation 2.

<u>Literary Review</u>

1. **Intriguing properties of neural networks by Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow and Rob Fergus, published at 19 Feb 2014**

   This paper's claim is that deep neural networks are highly expressive models, and this fact is what makes them be successful, but it can also cause them to learn uninterpretable solutions that could have counter-intuitive properties. In the article we get to know two such properties:

   a. Instead of the individual units, it is the space that contains the semantic information in the high layers of neural networks. This is because there is no distinction between individual high level units and random linear combinations of high level units.

b. We can cause the network to misclassify an image by applying a norm bound perturbation that is not perceptible to the human eye, which is found by maximizing the network's prediction error in the bounds of the said norm.

This paper has pioneered adversarial research in DNNs.

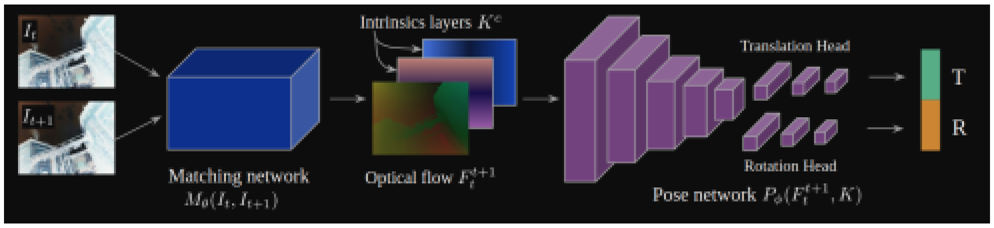
2. **Explaining and harnessing adversarial examples by Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy In 2015.**

   It argues that the primary cause of neural networks' vulnerability to adversarial perturbation is their piecewise linear nature when mapping the input space to the output space (as opposed to the extreme **non**-linear nature of the mapping from parameter space to the output space), which makes the search space for adversarial examples relatively straight-forward (literally). This ease of search in most of the input space makes it easy for adversarial examples to **generalize across different models**. This also includes different loss functions as is shown in [4] as well as in the results section.

   The linear search space described for other neural network input spaces might not be relevant in our case since we work in a much smaller subspace - around 7% of the FOV [4] which translates to 7% of the dimensions of the perturbed images. Even though the albedo bounded dynamic range is much larger than the norm bound in classical adversarial optimization, the search space grows exponentially in the dimensionality and only linearly on the dynamic range and we assume that the search space hasn't grown by that much.

   This still doesn't mean that we cannot get universal results, but that they are probably much harder to find.

3. **Wang, Wenshan, Yaoyu Hu, and Sebastian Scherer. "Tartanvo: A generalizable learning-based vo."** ***arXiv preprint arXiv:2011.00359*** **(2020).**
   The paper presents the model on which we train and test our adversarial attacks. The model is a two-part neural network. The input to the first layer is two images and it outputs optical flow. The flow along with the intrinsic properties of the camera are fed as input to the second network which outputs a translation and a Rodrigues rotation vector.



   The model conducts it's inference scale free and is fed the data collection intrinsic properties at runtime.

4. Nemcovsky, Yaniv, et al. "Physical Passive Patch Adversarial Attacks on Visual Odometry Systems." arXiv preprint arXiv:2207.05729 (2022).

The paper investigates the planting of adversarial images into a real-life scene (e.g. a billboard, t-shirt, fighting drone) in order to fool DNN based visual odometry models, and ultimately navigation and self-driving systems.

The paper presents an optimization scheme and results on several simulated and real-world scenes physically perturbed by adversarial images aimed at maximizing training loss, and the performance of the scheme on in and out of sample data with a simpler loss function than the one used in training. Showing the ability of the training to generalize and weaken state of the art models of visual odometry.

The losses and PGD attack we use for the baseline results in our study are taken from this article, and are elaborated upon in the methods section.

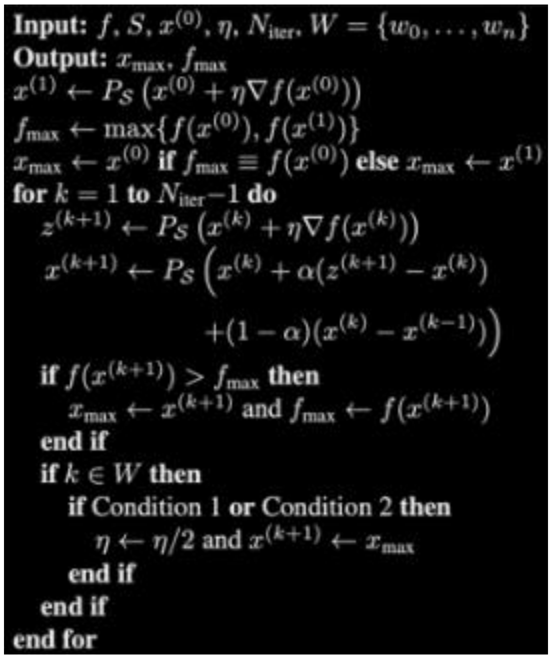
5. **Madry, Aleksander, et al. "Towards deep learning models resistant to adversarial attacks." arXiv preprint arXiv:1706.06083 (2017).**

The paper introduces the PGD attack which we use for the initial perturbation optimization.

The algorithm of the PGD attack we use in our study is elaborated in appendix A of 4 under "Algorithm 2: Universal PGD Attack"

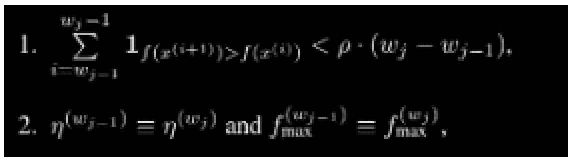
6. Reliable Evaluation of Adversarial Robustness with an Ensemble of Diverse Parameter-free Attacks by Francesco Croce and Matthias Hein, published at 4 Aug 2020.

   In this paper the APGD algorithm is presented. We learn about two extensions of the PGD-attack overcoming failures due to fixed step size and problems of the objective function. The algorithm also adds a momentum term which the original method was lacking. In the "improvement methods" section, we show our use of the algorithm and our modifications to it. The original algorithm is shown in Figure 1.



```
Input: $f, S, x^{(0)}, \eta, N_{\text{iter}}, W = \{w_0, \dots, w_n\}$
Output: $x_{\max}, f_{\max}$
$x^{(1)} \leftarrow P_S\left(x^{(0)} + \eta\nabla f(x^{(0)})\right)$
$f_{\max} \leftarrow \max\{f(x^{(0)}), f(x^{(1)})\}$
$x_{\max} \leftarrow x^{(0)}$ if $f_{\max} \equiv f(x^{(0)})$ else $x_{\max} \leftarrow x^{(1)}$
for $k = 1$ to $N_{\text{iter}}-1$ do
  $z^{(k+1)} \leftarrow P_S\left(x^{(k)} + \eta\nabla f(x^{(k)})\right)$
  $x^{(k+1)} \leftarrow P_S\Big(x^{(k)} + \alpha(z^{(k+1)} - x^{(k)})$
              $+(1-\alpha)(x^{(k)} - x^{(k-1)})\Big)$
  if $f(x^{(k+1)}) > f_{\max}$ then
    $x_{\max} \leftarrow x^{(k+1)}$ and $f_{\max} \leftarrow f(x^{(k+1)})$
  end if
  if $k \in W$ then
    if Condition 1 or Condition 2 then
      $\eta \leftarrow \eta/2$ and $x^{(k+1)} \leftarrow x_{\max}$
    end if
  end if
end for
```

Conditions:



1. $\sum_{i=w_{j-1}}^{w_j-1} \mathbf{1}_{f(x^{(i+1)})>f(x^{(i)})} < \rho \cdot (w_j - w_{j-1})$,
2. $\eta^{(w_{j-1})} \equiv \eta^{(w_j)}$ and $f_{\max}^{(w_{j-1})} \equiv f_{\max}^{(w_j)}$,

Figure 1: APGD algorithm

7. https://www.coursera.org/lecture/robotics-perception/3d-velocities-from-optical-flow-DgSNW
   This lecture explains the way camera translation and rotation affect the eventual optical flow. And a formula is given for the optical flow between two frames based on these values:

   Equation 1: optical flow from translation and rotation

$$\dot{p} = \frac{1}{Z}\begin{bmatrix} -f & 0 & x \\ 0 & -f & y \end{bmatrix}\vec{T} + \begin{bmatrix} xy & -(1+x)^2 & y \\ (1+y)^2 & -xy & -x \end{bmatrix}\vec{\Omega}$$

   As we can see, the flow depends additively on the translation and rotation.

   We can draw the flow lines (or believe the video) and see that the translational field flow lines meet at a single point named the focus of expansion or FOE, and that flow gets bigger the farther we go from that point. The translation flow can be seen in Figure 2: optical flow for pure translation

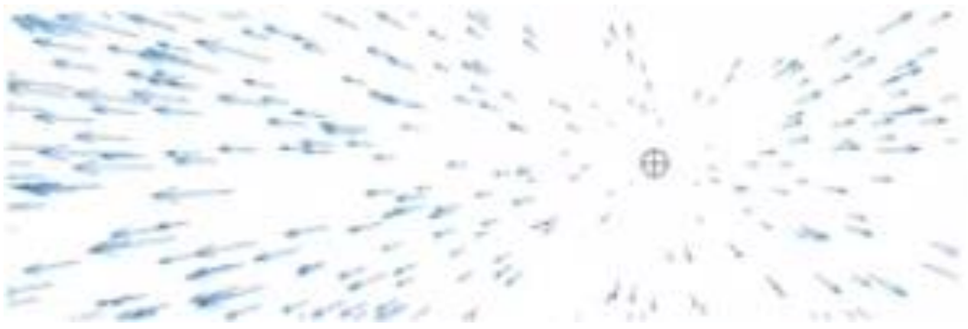


Figure 2: optical flow for pure translation

We use the equation and translation flow field property when creating the "Flow Divergence" loss term.

8. **K. Kanatani, "Transformation of optical flow by camera rotation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 10, no. 2, pp. 131-143, March 1988, doi: 10.1109/34.3879.**
   The paper derives the rotation component of optical flow in **Equation 1** analytically.

Methods

Original methods

We base the research on the code and theory provided in the home exercise, all of which can be found in [4].

In short, we define the perturbed image:

$$I^P = A(I,P) = \big(H(P)\big) * (I^1 - I^0) + I^0$$

Where I^0, I^1 are the dark and bright albedo images, and H is the homography transforming P into the appropriate plane and location in I.

The VO function:

$$VO(\{I_t\}_{t=0}^{L-1}) = \left\{\widehat{\delta}_t^{\,t+1}\right\}_{t=0}^{L}$$

Where $\{I_t\}_{t=0}^{L-1}$ is a sequence of images forming a trajectory and $\left\{\widehat{\delta}_t^{\,t+1}\right\}_{t=0}^{L}$ is a sequence of matrices in the form $\begin{bmatrix}\overleftrightarrow{R} & \vec{T} \\ \vec{0} & 1\end{bmatrix}$ each representing the camera motion between two consecutive frames, with $\overleftrightarrow{R}$ being a rotation matrix and $\vec{T}$ being a translation vector.

We are solving the optimization problem:

$$P_{ua} = arg \max_{P\in\Omega} \sum_{i=0}^{N-1} \ell\big(VO(\{I_t^P\}_i), \{\delta_t^{t+1}\}_{t=0_i}^{L}\big)$$

Where i counts a set containing different trajectories. This is a universal adversarial setting since we are trying to fit a perturbation to as large a scenario set as we can.

The task criterion:

$$\ell_{VO}\left(VO\{I_t^p\}_{t=0}^{L}, \{\delta_t^{t+1}\}_{t=0}^{L}\right) = \left| q(\prod_t VO(I_t^P, I_{t+1}^P)) - q\left(\prod_t \delta_t^{t+1}\right)\right|_2$$

Where $q(R,T) = T$ is the final translation in the coordinate system moving with the camera.

The training loss is different, and is defined to be:

$$\ell_{train} \triangleq \ell_{MPRMS}(VO(\{I^P\}), \{\delta_t^{t+1}\}_{t=0}^{L}) = \sum_{l=1}^{L} \frac{1}{L-l+1} \sum_{i=0}^{L-1} \ell_{VO}\left(VO\{I_t^p\}_{t=0}^{i+l-1}, \{\delta_t^{t+1}\}_{t=0}^{i+l-1}\right)$$

We were given a dataset composed of five folders each containing ten trajectories starting at the same point in space, each trajectory containing eight frames. The trajectories are created with Blender and all simulate the same camera characteristics.

Improvement Methods

Added loss terms

In addition to using the MPRMS loss on the perturbed trajectories' translation for ℓ_{train}, we tried different additional loss terms that would facilitate an increase in translation error.

Optical Flow Parallel Punishment. We denote the optical flow to be $\dot{p}$. The difference in optical flow between clean and perturbed images is the basis for the misprediction we are looking for, since it is the input to the pose network. We aim to affect the optical flow predicted from the frame change in a way that will facilitate a larger translation error.

In our practical implementation, we have added a loss term which rewards perpendicular clean and adversarial flows, i.e.:

$$\ell_{\parallel} = \quad -\dot{p}_{clean} \cdot \dot{p}_{adv}$$

This loss term rewards an adversarial flow field that goes in the direction perpendicular to the original flow field. The model does not interpret this disturbance to the flow field as noise thanks to the first loss term L_t, which breaks the clockwise-counterclockwise symmetry of the dot product. The less aligned the flow vectors, the better, supposedly.

In code:

```
--attack_flow_crit proj --attack_flow_factor 1
```

Optical Flow Divergence Reward. As shown in [7], Equation 1 describes the optical flow generated by the rotation and translation. Taking the curl of the RHS shows that the translation field is non-rotational. Therefore, maximizing the divergence of the optical flow along with L_T could provide a more translation-like flow field. Since there is only one FOE, we expect the divergence maximization to align the flow vectors to the FOE rather than create new sources:

$$\ell_{div} = \mathbb{E}_{\vec{p}}\left(\left(\nabla \cdot \dot{p}(\vec{p})\right)^2\right)$$

The divergence is squared so that it doesn't depend on the sign of the $\hat{z}$ translation component.

In code: to add this term to the loss function simply call:

```
--attack_flow_crit div --attack_flow_factor 10
```

For us, the more intuitive loss would have been to minimize the curl (which would mean reducing only the rotational part of the flow as the translation term is curl free), but our experiments showed no improvement to the baseline, also when dividing the divergence by the curl.

In general, all attempts to add punishments as extra loss terms were not successful, as will be shown in the results section, and discussed in the discussion section.

Data

Train-eval split. The data we received is split into 5 folders. To train our perturbation for generalization on unseen data, we split the data into 4 training and one evaluation folder as it is common practice. We do not do the full train-eval-test split because of a lack in time. We evaluate on the evaluation set using L_{VO} as demanded.

In code: the flags –test-folder and –eval folder followed by the wanted folder number will also make the rest of the available folders (0-4) training folders. The flags default to -1 which will not make the test/eval folder.

E.g., if we want to evaluate on folder 4 and train on the rest, call run_attack.py with:

```
--test-folder -1 --eval-folder 4
```

APGD Optimizer

As an optimizer, we experimented with upgrading the vanilla PGD attack. We test the APGD or Auto-PGD attack [2] for improving the optimization.

APGD adds a momentum term in the perturbation update, interpolating the current and previous PGD updates. Also, the iterations are divided in a user determined way, and the step size is reduced between divisions if improvement conditions are met.

What drove us to use this algorithm besides the prevalence of momentum terms in neural network optimization is the fact that at some point in the optimization, the loss would hardly go up and would be oscillating between higher and lower values, climbing slowly until reaching saturation. We hypothesized that a step size too large was driving the perturbation image in and out of high loss regions.

The APGD algorithm is shown in Figure 1: APGD algorithm. Since APGD was first conceived to work on classical adversarial optimization, we made a few adjustments to it.

1. In the original paper, the checkpoints are set according to a fixed ratio. Since we saw similar places in all the datasets in which the loss grows more slowly and oscillates, we changed the checkpoints to be at fixed iterations: 40, 75, 90, 130, 150, 175, 190. If the iteration number is smaller than any of the checkpoints, they and the ones after them are not counted.
   In code: this is done automatically and cannot be configured.
2. Momentum was changed from 0.75 (in favor of the latest update) to 0.9 for the baseline loss function, and to 0.95 for the . This is because too high of a momentum could make a losing update affect the updates that come after (and indeed it does as will be shown in the results section).
3. The fraction for of update iterations out of the total iterations between checkpoints under which step size is reduced is changed to 0.5 since we saw that the optimization process can recover sometimes from an oscillation phase.
4. Technical: in the original paper, the momentum value describes the weight of the current optimization step in the interpolation. We changed it to be the old update's weight. So, in our implementation we changed $\alpha = 0.25$ to 0.1 for the baseline loss function and 0.05 for the baseline + flow losses.

In code:

```
--attack apgd --momentum 0.1
```

Experiments:

We run each experiment five times, for each of which we use a different folder for evaluation and the rest for training. We use the recommended step size of 0.05 (initial step size for APGD), and take 200 iterations so that the optimization process reaches saturation.

This setup is far from optimal. The standard optimization method demands we take 3 folder for training, one for evaluation and one for testing. Because of a lack in time, and a feeling that the data is too little, we ran the optimization the way we did.

The hypotheses we test are:

1. Baseline: the loss is L_t and the attack is PGD
   Command to run:
   ```
   -eval-folder <folder number> -attack-pgd -alpha 0.05
   ```
2. $L_{train} = L_T + L_{\dot{p}}$ with PGD attack
   Command to run:
   ```
   eval-folder <folder number> --attack pgd -attack_flow_crit <div, proj>
   -attack_flow_factor <div: 10, proj:1 >
   ```
   --
3. $L_{train} = L_T$ with APGD attack
   Command to run:
   ```
   --eval-folder <folder number> --attack apgd -momentum 0.1
   ```
4. $L_{train} = L_T + L_{\dot{p}}$ with APGD attack
   Command to run:
   ```
   --eval-folder <folder number> --attack apgd -attack_flow_factor
   <proj/div> --attack_flow_factor <div: 10, proj: 1>
   ```

We examine the results in terms of eventual summed loss difference between the model's outputs on the perturbed and unperturbed trajectories on the evaluation sets, relative to the clean trajectory calculated. namely:

Equation 2: Performance parameter of the optimization process

$$\frac{\sum_{i\in E} L_{adv}^{i} - \sum_{i\in E} L_{clean}^{i}}{\sum_{i\in E} L_{clean}^{i}}$$

Where:

$$L_c^i = L_{VO}(\{I_t\}^i, \{\delta_{t,t+1}\}_{t=0\ldots L}^{i})$$

$$L_p^i = L_{VO}(A(\{I_t\}_{t=0\ldots L}^{i}, P), \{\delta_{t,t+1}\}_{t=0\ldots L}^{i})$$

This gives us an estimation of the factor by which the model's error increased relative to it's unperturbed error.

Results and discussion

Note: the results slightly change from run to tun when using the same parameters even when initializing the perturbation to be constant. We assume that this has to do with some random nature of the VO model that we are using **Error! Reference source not found.**. We did not run statistics on the confidence range for every parameter and the baseline because of a lack of time. In the following we will call changes “significant” by referring to values that are much larger or smaller than the ones achieved in relation to the baseline values.

Dataset number 2:

This dataset achieves lower performance values than all other datasets with a saturation value that is smaller by a factor of 3 than the second smallest evaluation saturation (folder 3) and by a factor of 10 than the largest saturation (folder 4).

The trajectories in the dataset all contain a large tree in the field of view which the VO model is probably good at recognizing but our perturbation is not trained enough to handle. Indeed, when comparing flow difference maps between the clean and adversarial flows inferred by the model of dataset 2 to flow differences from other datasets we see a qualitative difference between them, as shown in Figure 3. We can see that the perturbation combined with the plant creates some uncontrollable difference between the clean and adversarial images in the location of the plant. As opposed to being centralized around the patch location. We assume that this is the cause of the reduced saturation value we achieved.

We highly suspect that this folder is an outlier, indeed, running the optimization without it is shown in Figure 4. We see that saturation values have not significantly improved, but that the oscillations of the optimization process do become lower.

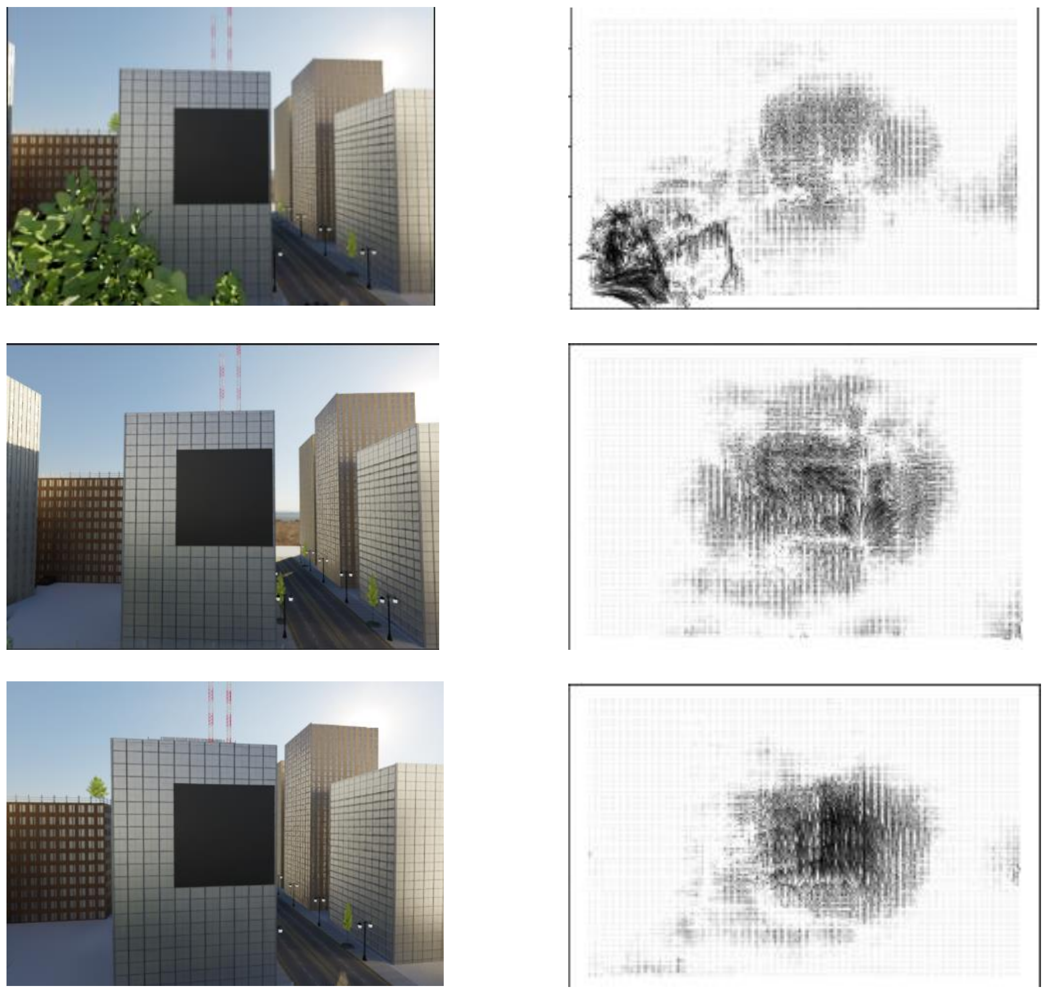


Figure 3: comparison of optical flow difference between the adversarial and clean model runs. Left: original image, Right: flow difference. Top to bottom: datasets 2, 3, 1

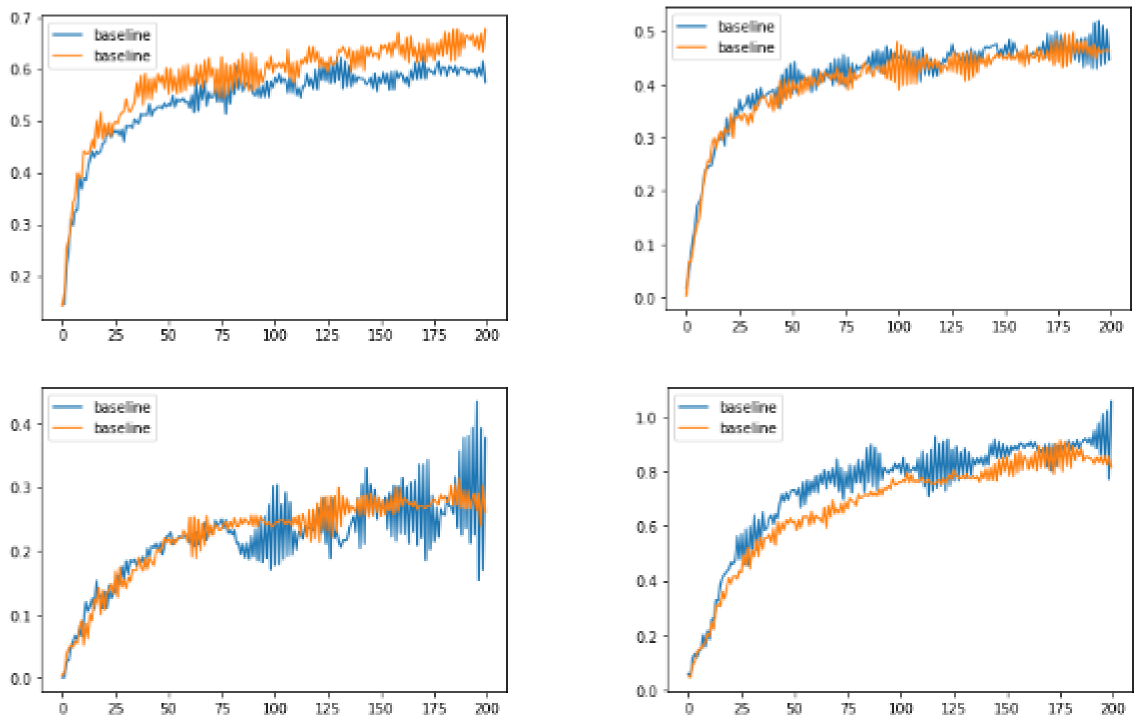


Figure 4: evaluation results when optimization is run without folder 2. top to bottom, left to right folders 0,1,3,4

In code: to run without folder 2, simply call

```
--no-2
```

Main Results

The main results are presented in Figure 5: experiment results for the 5 evaluation sets. Top to bottom, Left to right: folders 0,1,2,3,4Figure 5, each showing the optimization process of the four algorithms suggested.

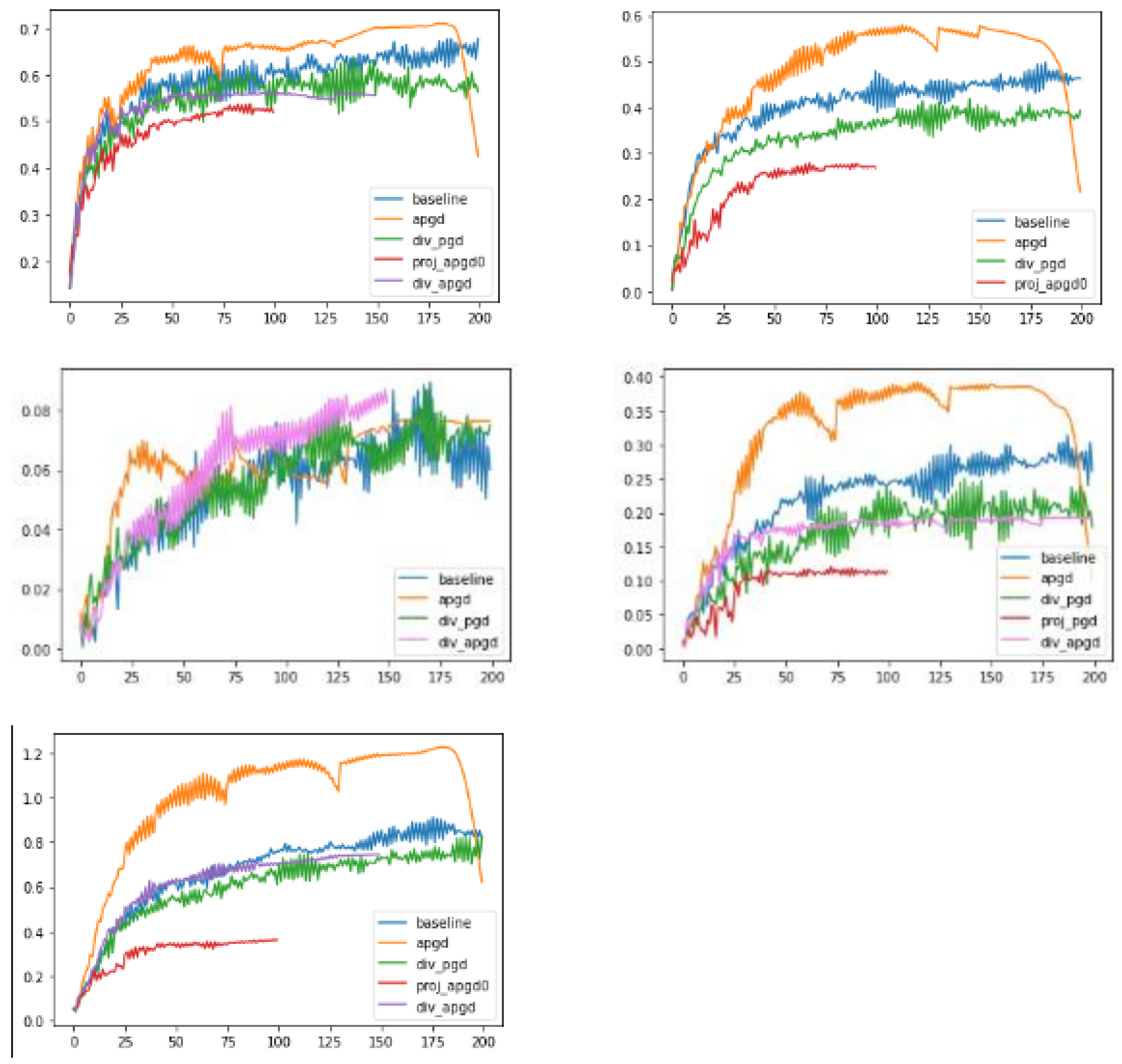


Figure 5: experiment results for the 5 evaluation sets. Top to bottom, Left to right: folders 0,1,2,3,4.

APGD

We can see that the APGD attack had the only effect on the out of sample set in every experiment. It reaches a higher saturation in a shorter amount of time than all of the other methods we tried. We can see the performance is characterized by “dips” where the loss consistently goes down until the checkpoint at which the step size is halved and the best performing perturbation so far is taken, which stops the descent. We believe that this dip is caused by a too high of a momentum factor, causing a bad performance inertia. We improved this in the divergence + APGD experiments on eval sets 0 and 4, by using a momentum value of 0.05, Although that didn't help the divergence loss overcome the baseline.

Divergence and Projection Loss Terms

We can see that both techniques performed worse than the baseline in all datasets apart from dataset number 2, in which it showed improvement to the other algorithms in terms of stability. Unfortunately, we did not run the optimization for a full 200 iterations for a lack of time, but it seems that even after stably achieving the best results of the other two methods, it could improve even more. This implies that the divergence term along with the APGD attack are more robust than the other methods for out of sample errors.

In most cases, the divergence loss achieved slightly lower saturation and convergence speed than the baseline. This also happened when combined with the APGD but it might have been caused from taking the checkpoints too dense in the later parts of the training which may have caused smaller improvements due to too small step sizes.

Since the performance is worse on all datasets except 2, we assume that it's worse when evaluated on data more probable compared with the adversarial training data.

The projection loss came much lower than all the algorithms with a clearly low saturation value.

The addition of the successful APGD optimization slightly improved the divergence loss but not by much but did good in removing the noticeable oscillations as expected.

Penalties and Rewards

Another conclusion of our research is that penalty terms in the loss function one and all only reduce the efficiency of the attack. We have tried several other penalties besides the projection e.g. the norm of the difference in flow (which includes the projection loss implicitly), a penalty on curl (which only punishes the rotation part of the flow) and divergence divided by curl (which becomes larger when curl is reduced). All of these losses caused the eventual translation loss to reduce.

This might imply to the sharpness of the loss function where any term penalizing the flow will dominate at some point of the optimization. This corresponds to the observation we made in [3], where we claim that the problem doesn't have a linear input space as opposed to full image adversarial attacks.

The Selected Patch:

We have selected the patch optimized with only APGD on dataset 4 since it seems as the best improvement in terms of Equation 2.

Summary

We have trained a perturbative patch to improve on the results of [4] in increasing the prediction error made by the Tartan VO model[3].

We have suggested two methods for improvement – APGD and an additional loss term on the divergence of the optical flow output of the model.

The APGD scheme performed the best on most of the datasets available to us except for one less probable dataset on which it achieved the best and most stable results combined with the divergence loss term.

Future Work:

1. Thailand

2. Optimize the hyper-parameters of the APGD and the weighting between the different loss terms
3. Run a standard optimization process consisting of 3 training folders, one evaluation folder and one testing folder in order to understand the real effect of the adversarial training.
4. Reduce the momentum in the APGD algorithm along with step size to try and reduce the dipping effect at high iterations.