Deep Learning Methods for Video Prediction

Fall 2019 Project Course Report

Christopher Botica
Student
Northeastern University
botica.c@husky.neu.edu

Dr. Rose Yu
Instructor
Northeastern University
roseyu@northeastern.edu

1 Introduction and Motivation

Video synthesis and prediction is a topic of great interest in the Deep Learning community. It is particularly important in situations in which autonomous agents such as drones or vehicles require object detection, tracking, and motion prediction. Furthermore, whenever an agent such as a Google Maps application recommends a driving route, it relies on a high-resolution traffic prediction.

However, video prediction is a particularly difficult task because of its high nonlinearity and large size across both the spatial and temporal dimensions.

By applying our linear and non-linear methods for traffic prediction and also video translation models for synthesizing IR-RGB videos, we contribute to the advancement of this field with the following:

- Summary and limitations of linear, non-linear, and ensemble techniques for traffic prediction
- Novel technique for modeling local linear dynamics using the PyTorch deep learning framework
- Proof of concept for IR-RGB video synthesis and code for baseline model.

During this semester we focused on two problems: traffic prediction and IR-RGB video synthesis. We cover traffic prediction in Section 2 and IR-RGB video synthesis in Section 3.

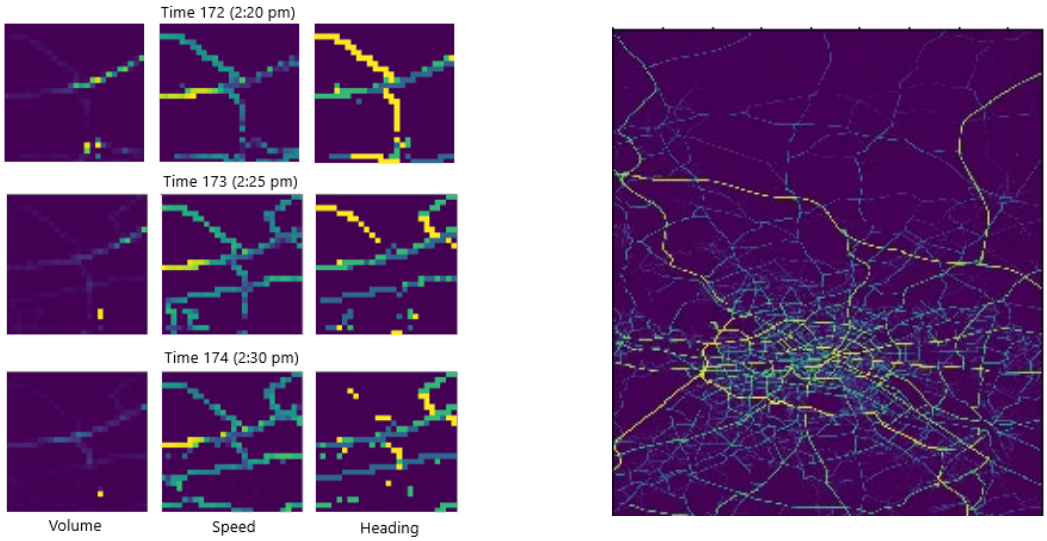
2 Traffic Prediction

With more cars and developing urban areas, cities are facing increasing mobility challenges due to traffic. Thus, improved traffic prediction models have great social and economic value. This problem becomes even more difficult for developing cities which do not have quality up to date road maps. To this end, the Traffic4Cast competition from IARAI [1] sets to improve such traffic prediction from purely image data, deliberately ignoring the city's underlying street network.

We joined this competition in a collaboration with USC. During this period, we employed many types of machine learning models with much computational effort. Even though we did not win, we gained valuable insights into the strengths and limitations of linear, non-linear, and ensemble techniques for traffic prediction. In doing so, we also developed innovative techniques for solving large linear systems for spatial dependencies using Pytorch deep learning framework, which we discuss in Section 2.2.2.

2.1 Traffic Data

The traffic data supplied by IARAI are high-resolution traffic movies: 288 images taken each day, at an interval of 5 minutes, for three cities: Berlin, Istanbul, and Moscow. Each image is of size 495×436 and has three channels: volume, speed, and heading. Example images are shown in Figure 1. The goal of the project is: given 12 past images (1-hour prior), predict the next 3 frames (15 minutes).



(a) (b)

Figure 1: (a) Sample zoom-in image data from Berlin. (b) Full image example, with heading values

The volume and speed channels are continuous, with values in [0, 255], whereas the heading channel is categorical. For this channel, the raw heading values are mapped to four categories [NW, NE, SW, SE] corresponding to the distinct numerical values [1, 85, 170, 255].

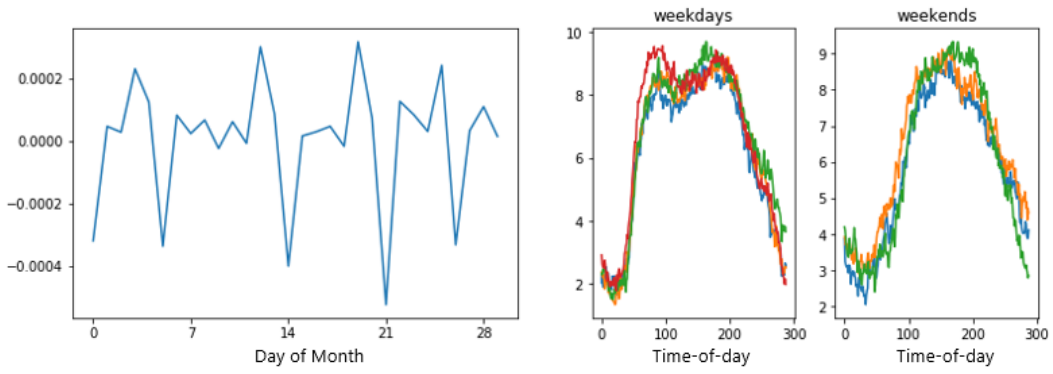
Due to thresholds on the GPS sensors used by IARAI, the image data is very sparse, with many points considered as "none" or missing. This missing data is denoted by 0's in every channel. Thus, it is important to keep true zero in images. Further, not every point is stationary: in some frames, a given spatial point might have a reading (resulting in a non-zero pixel output), and in other frames, it does not (resulting in a zero pixel value across all channels). The competition criteria for optimal predictions is the mean squared error (MSE) between the true and prediction image. Note that even though the heading channel is categorical, they employ this criteria across all channels for consistency.

Out of the 364 days of images, $285 + 7$ are used for training and validation and 72 for testing. The days for testing were sampled at random throughout the year and are kept by IARAI to evaluate the quality of the images. Thus, the training set does not contain all contiguous days of training.

2.2 Methods

2.2.1 Historical Averages

During our image analysis phase, we identified two overlapping seasons in the data: the day-of-week season (1 out of 7 days) and the time-of-day season (288 intervals of 5 minutes each). This results in a total of $7 \times 288 = 2016$ seasons. These seasons are illustrated in Figure 2.



(a) (b)

Figure 2: (a) Average value of normalized images for each day in a month, showing the day-of-week season. Note that there are missing images from the training set, reserved for competition testing, which is why the season period appears irregular. (b) Average value of unnormalized images for each day, multiple days a week, showing the time-of-day season.

In order to take advantage of this seasonal aspect of our data, we create our seasonal biases, or historical averages (HA): for each pixel value, take the averages across all 2016 seasons. Thus, we have one HA for each for each day-of-week and time-of-day. An example of an historical average is shown in Figure 3.

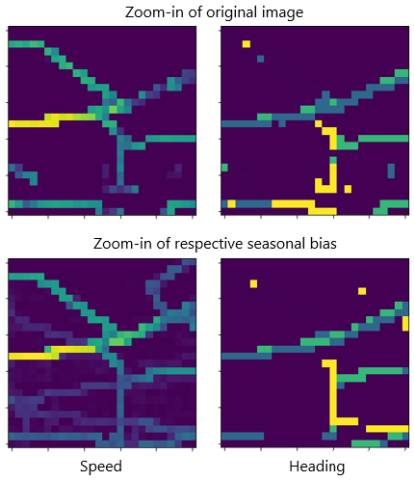


Figure 3: Example of an historical average (bottom) and respective true image (top).

2.2.2 Global Auto-Regressive Models

Due to the highly temporal and seasonal aspect of our data, we analyzed the prediction ability of auto-regressive (AR) models and, further, seasonal auto-regressive and moving average (SARIMA) models. However, our data is multidimensional and such models are limited to one-dimensional temporal data (excluding vector auto-regressive models). Thus, for each channel, we compress our 495×436 image to one datapoint by taking the average value of the image. This output serves as the entire image bias, hence the term global AR.

Let $x(t)$ represent the image average for a given channel at time t. The AR model seeks to learn the parameters $\varphi_1, \varphi_2, \ldots, \varphi_k$, where

$$x(t) = \sum_{i=1}^{k} \varphi_i x(t-i),$$

and k represents the order of the model (i.e., the number of previous frames).

Next, we set to combine the temporal aspect of our image data with its seasonality. One method we attempt is extending AR by using the SARIMA model, which incorporates auto-regression, moving average of the error terms, and the season by lagging each term in the equation.

Alternatively, we set to de-season our data by subtracting with the historical average, and learn the new AR parameters on the residuals. Now, our HA-AR equation becomes:

$$r_{x(t+1)} = \sum_{i=1}^{k} \varphi_i r_{x(t-i)},$$

where the residual is defined as $r_{x(t)} = x(t) - bias_{day-of-week(t)} - bias_{time-of-day(t)}$.

Thus, in our HA-AR model, we aim to infer the parameters $\varphi_1, \varphi_2, \ldots, \varphi_k$ to then form accurate predictions on the residuals. Once we predict the residual, we add it to its respective bias to find the true image value:

$$x(t+1) = r_{x(t+1)} + bias_{day-of-week(t+1)} + bias_{time-of-day(t+1)}$$

The key to this approach is its implementation. There exist many built-in libraries with AR and SARIMA models. However, this task becomes particularly difficult due to the large memory requirements. Because of the large image sizes, only samples of up to ~ 100 pixels per images were feasible, which made the Vector AR model very weak. Furthermore, the training data is not contiguous, since test days are randomly sampled throughout the year. The packages for SARIMA don't handle missing values.

To overcome these deficiencies, we implemented a custom AR model, based on the PyTorch deep learning framework. We used a 3D Convolutional layer with a $d \times 1 \times 1$ kernel, where d is the depth of frames. This layer is illustrated in Figure 4. This has the added benefit of incorporating every pixel and increased efficiency by using the deep learning GPU framework

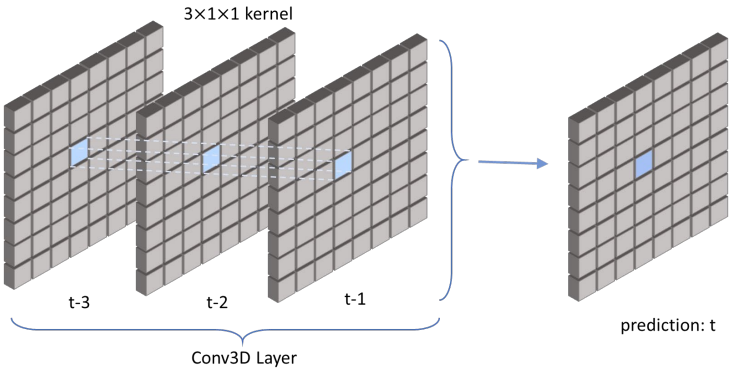


Figure 4: custom AR model implementation for depth = 3

This residual approach becomes problematic when applied to the heading channel. Since heading is a categorical variable, this yields the question 'What does the difference between two heading values mean?' On one hand, we can treat heading as a continuous variable and proceed as with volume or speed. This, surprisingly, results in the better MSE. However, another approach is to treat heading as an additive angle instead of categorical variable. For example, if a car switches direction from NE (90°) to SE (180°), we can argue that the car added 90° to its current heading. Figure 5 illustrates this.

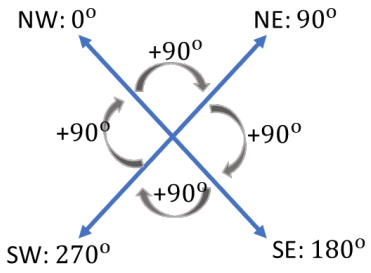


Figure 5: Treating heading as an additive variable with NW as 0°

Thus, we can now treat the residual differences for heading as additive operations modulo 360. Unfortunately, this did not work well in practice since we used the median heading value as its seasonal bias, which, for this case, turned out to be weaker than the average value (treating it as continuous).

2.2.2 Local Auto-Regressive Model

The main disadvantage of our global AR models is that they don't take advantage of local dynamics. Now, we extend the global models to local AR models. The assumption here is that each pixel is a linear combination of previous neighboring pixels, in a backwards growing window.

Since weight-sharing proved to be inefficient for the previous AR models, we build the local AR model such that it does not share any weights. In contrast to a convolutional layer, which has the same kernel weights, each pixel now has its own unique parameters. Similar to the global AR models, we implement this model in Pytorch. Furthermore, instead of using a constant window size in past images, we implemented decreasing filters from past timesteps (larger filters for older frames, smaller filters for more recent). This is illustrated in Figure 6.

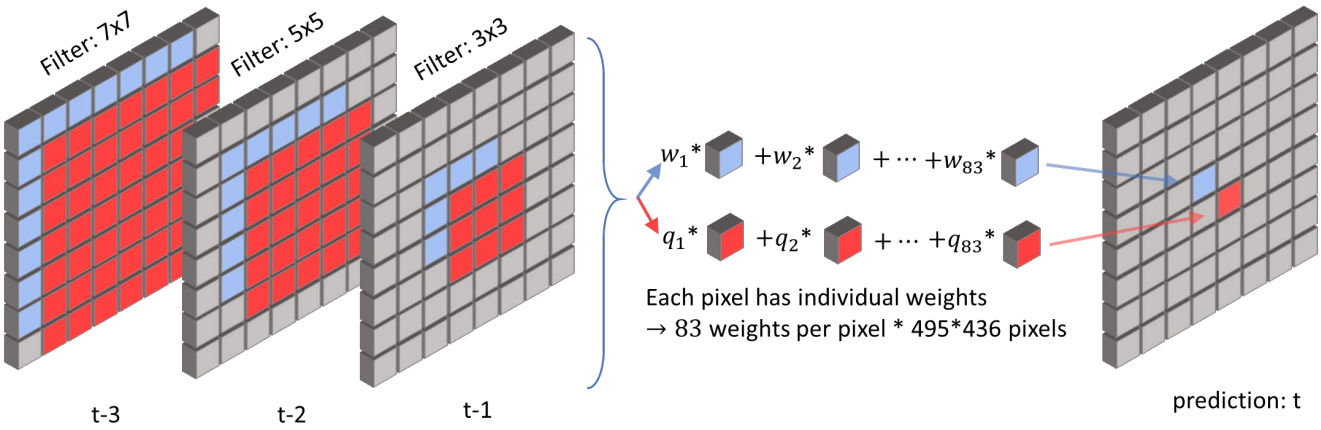


Figure 6: The two different filters, blue and red, have different weights in the Local-AR model: $w_1, w_2, \dots, w_{83}$ and $q_1, q_2, \dots q_{83}$ respectively

The motivation for this method stems from the fact that traffic propagates at a rate smaller (or at least comparable rate) than the frame rate of our video. This belief can be summarized as: 'in the immediate time, far-away traffic does not affect current traffic, but, given enough time, traffic will ultimately propagate to the current location'. Thus, in a recent frame, traffic values in distant pixels will not affect traffic in our location, but values from distant traffic and further away frames would have had an effect, so we include only those in our model. This also introduces attention: our model only focuses on the provided surrounding

pixels. Furthermore, this approach reduces the number of parameters occupy CUDA memory.

The key to this approach is the implementation. Our main implementation bottleneck here was combining the parameters for each pixel with a window with unique weights. Using standard Conv2d or Conv3d in Pytorch did not work since it would have shared weights across the kernels. Instead, we created tensor parameters and used the Pytorch nn.Unfold() function to extract sliding blocks from the image of shape 3x3, then 5x5, 7x7 and so on, multiply the input block by the its respective parameters, and then use nn.Fold() to add the results back into the image format.

This is a novel implementation of local linear dynamics, and easily be extended to modeling more nonlinear dependencies by replacing the linear combination step with additional layers (such as nonlinear activations and convolutional layers).

2.2.3 Other Models

Our team used the following additional methods to model traffic:

- Residual neural network (ResNet) [2]
- Convolutional neural network for image segmentation (U-Net) [3]
- XGBoost regressor [4]
- K-nearest neighbor (KNN) algorithm [5]

Two variants of KNN were employed: same-time KNN and warping-time KNN. The intuition behind same-time KNN is to find the day and time when the traffic in previous timeslot is most similar. Thus, the features are all readings at the timeslot before each predicting index, and the labels are the readings at predicting timeslots. Here the searching space is restricted to the times before predicting index.

On the other hand, warping-time KNN allows the prediction to be similar to (a) same time on a recent day, or (b) close time prior to prediction. Thus, we include samples of prior timeslots in addition to the same-time KNN. Both cases utilize k = 40.

Our implementation code is available at our GitHub repo: https://github.com/Rose-STL-Lab/traffic4cast

2.4 Experiments

During the month and a half of the competition, we ran large-scale experiments. Our main experiments and results are summarized in Table 1.

The errors differed across cities and channels. The volume contributed nearly nothing to the overall MSE, whereas heading contributed the most. The ratios of contribution are approximately 0.1: 1: 8. Across different cities, we found that Moscow has the highest errors, approximately twice those of Berlin.

2.4.2 Historical Averages

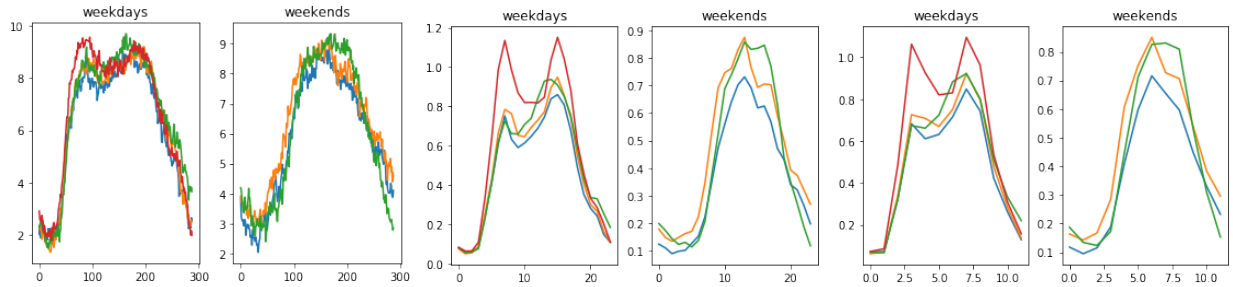
Our HA model result in unexpected good predictions. Since traffic readings always exhibit strong daily patterns (such as morning and evening rush hours) and weekly patterns (such as Monday morning rush), we generate historical averages for each season: time-of-day and day-of-week, totaling 2016 seasons. Because of the power of large averaging base (all training data), the HA model resulted in good predictions, with MSE = 9.83e-03, even though it is blind to the input pixels.

Having different seasonal biases for each 5-minute time-block introduces additional noise in the seasonal bias model. For example, if we use a moving average on the residuals (HA of that timestep - actual value), we introduce noise due to the difference between seasonal biases from one timestep to another. At first it seemed as if these differences would be negligible for volume and speed (since they are somewhat continuous), but when we compute the MSE for the HA-AR model, we found that it is worse than the blind seasonal biases. This suggests that there is too much variation from the seasonal bias of one timestep to another. To overcome this, we expanded the time-of-day season to intervals of 60-minutes and 120-minutes,

Model	MSE
Ensemble: HA, ResNet, KNN-40	9.46e-03
Ensemble: HA, KNN-40	9.69e-03
ResNet	9.76e-03
HA	9.83e-03
KNN-40	9.86e-03
U-Net	9.92e-03
XGBoost	1.005e-02
Sliding Average (using 6 frames)	1.097e-02
Local AR (using 6 frames)	1.170e-02
Sliding Average (using 3 frames)	1.278e-02
HA + AR (using 3 frames)	1.389e-02
Seasonal Bias + AR	1.389e-02
All zeros	2.169e-02
AR (using 3 frames)	3.00e-02

Table 1: Experiments Results

so that there is less variation among our test data and predictions. Figure 7 illustrates the difference between these time-of-day interval lengths.



(a) (b) (c)

Figure 7: (a) average image value in a day using 5-minute intervals: large variation from one interval to the other, (b) average value using 60-minute intervals: smoother, less variation, (c) average value using 120-minute interval: too smooth

However, expanding the time-of-day seasons from 5-minute intervals to 60, 120-minute intervals removes the benefit of computing AR on the residuals. For example, if we are trying to predict 15 minutes of traffic based on the past 30 minutes, then using the 60-minute seasonal bias to calculate the residuals of the past will be of no effect, since the residuals of the future 15 minutes still depend on the same seasonal bias.

Furthermore, we found that computing the median of the heading channel was worse than simple averaging. The median of the previous n frames is equivalent to applying "majority vote" on the previous n frames. However, the result is worse than the simple "mean" of the 3 frames since MSE penalizes more on discrete predictions than smoother ones.

2.4.1 Global Auto-regressive Models

Due to the missing data problem SARIMA did not converge. Furthermore, it cannot handle competing seasons (more than one). Similarly, the AR model yields bad results. Scaling the results back from the 1D case to the 2D image proved to be ineffective, since this number did not capture local dynamics. As a result, the AR model MSE was very bad (0.03).

Furthermore, the AR model was unable to find an optimal solution due to the different seasons. For example, we obtained better results using a sliding average model, which is AR with constant weights of 1.

2.4.2 Local AR model

The local AR model obtained decent results, but not extraordinary. The main reason is that the dependencies between the pixel values are highly nonlinear. Even so, it was still able converge and to capture some of the local and temporal dependencies.

In order to enforce the categorical nature of the heading channel, we implemented one hot encoding of the variable. This process is illustrated in Figure 8.

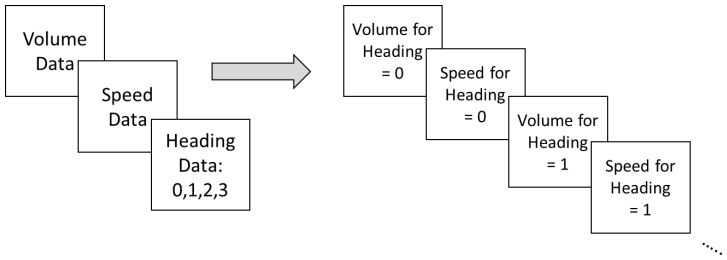


Figure 8: One-hot encoding of heading channel

During the early phases of training, the volume channel would often collapse to zero, causing vanishing gradients. To solve this, we a applied a log-transform of the volume channel. This data was extremely skewed to 0, even after normalization, and resulted in skewed prediction of the AR net. Applying a log-transform fixed this problem. This is illustrated in Figure 9.

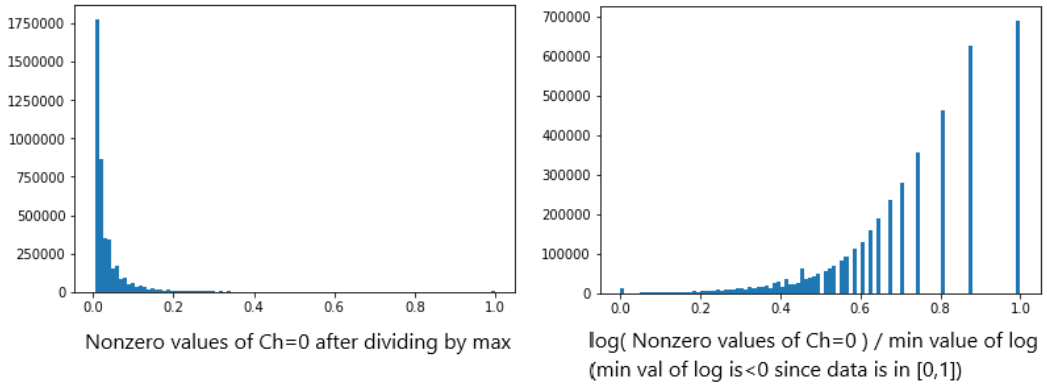


Figure 9: Volume data is skewed towards 0 prior to log-transform

Furthermore, custom PyTorch initialization methods hampered the inability to converge. These were not able to take into account the number of variables in each linear equation and normalize accordingly. Furthermore, negative initializations often resulted in negative outputs. To solve this, we created a custom initialization: we sample the initial weights for each layer from a uniform $U(0, w_{param})$, where w_{param} is

a hyperparameter. Furthermore, to enforce positive outputs, we add a ReLU activation layer.

We found that training one model for a single prediction, using that to find the next prediction, and so on... is suboptimal, since any errors made in the prediction at time t will automatically propagate in the predictions t+1 and t+2. Therefore, we trained 3 separate local AR models for each prediction step. Furthermore, training a separate model for each prediction channel output improved results since it allowed the model to focus on each specific output (continuous output, MSE loss for volume and speed and binary output, BCE loss for the one-hot heading variable).

2.4.2 Other Models

ResNet obtain the best result as a stand-alone model. However, our results drastically improved once we used an ensemble approach, indicating that each model captures a different aspect of the solution. The ensemble of ResNet, KNN-40, and HA gave us the best score. The key insight of this ensemble approach was to use good methods with different learning spaces.

On the other hand, Resnet produced many small predictions such as 1 on 0 cells. This is because such prediction sacrifices errors on small values to reduce the risk of making errors on larger values. For example, if the true value is 0, and ResNet allows small predictions of 1, then the losses increases by $1^2 = 1$. However, if the true value is 170, and the model is 1 off, the loss increases by $170^2 - 169^2 = 339$.

2.5 Lessons Learned

Traffic video prediction is a particularly difficult problem due to the high number of dimensions and parameters needed. We found that taking advantage the seasonal aspect of the data was crucial. Even blind seasonal predictions obtain good results. Unfortunately, we did not have enough time to fully integrate seasonality with the other models.

On the other hand, we found that the linear models performed very poorly. This is, in large, due to the fact that traffic is highly nonlinear. While it remains a good idea in theory, residual learning did not perform in practice due to the heading variable. Furthermore, the MSE distribution in the heading channel forces each model to penalizes extreme values. Thus, 'smoother' values like 50 balance the errors from frequent 0s and large values like 255. Averaging across more image yields better accuracy because of this smoothing.

Our best results, by far, were obtained using ensemble models. This suggests that each individual model captured a unique aspect of the solution.

This was a very computationally heavy competition. We learned to work with and set up AWS and share the GPU resources with our teammates. Further, the local AR model benefited greatly from additional fine-tuning and improved initialization. Adding additional convolutional layers to the model to enable nonlinear approximations would improve it for additional applications.

3 IR-RGB Video Translation

Video translation is particularly useful in situations in which autonomous agent such as drones or autonomous vehicles require object detection or mapping. However, the inability for systems used to RGB images to coordinate with IR images, and vice versa, renders these tasks useless.

IR-RGB image translation is particularly difficult because of the different information captured by IR and visible light. RGB images represent the manifestation of energy that is largely reflected off objects, whereas IR images are obtained through energy that is both reflected and emitted (e.g. gas emissions). Therefore, a direct translation, without any a priori knowledge of the scene, is impossible.

In this project, we aim to construct a sufficient a priori knowledge of natural scenes as to enable the

translation from IR frames to RGB frames.

This is particularly useful for drones in enabling them to 'see' at night. Current commercial drones are equipped with RGB cameras that fail in the dark. However, equipping the same drones with IR cameras also fails because of the incompatibility between RGB and IR systems. For example, if the drone attempts to locate itself in a pre-built RGB map of the world using just IR image inputs, it often fails to do so.

For this project, we collaborate with the team from Northeastern Mech/EE department to enable drones to, ultimately, perform patch similarity.

3.1 IR-RGB Image Data

We use the FLIR [6] and KAIST [7] IR-RGB video datasets. For the FLIR dataset, the data was acquired on a vehicle with an IR and RGB cameras side by side. Figure 10 shows an example IR and corresponding RGB image from the FLIR dataset.

(a) (b)

Figure 10: (a) example IR image from the FLIR dataset (b) corresponding RGB image

The KAIST dataset, on the other hand contains a significantly larger number of paired images (50,184 total pairs) and used custom developed hardware with a color camera, a thermal camera and a beam splitter to avoid misalignment.

3.2 Related Work

In 2016, DeTone et al. trained a CNN for deep homography estimation between image pairs. Even though their model works on RGB-RGB images, their technique outperforms traditional homography techniques. Since the Mech/EE department are ultimately interested in patch similarity and IR-RGB homography, one possible extension to DeTone's work would be to retrain their model on IR-RGB images pairs and employ the Mech/EE task.

In 2018, Zhang et al. retrained the pix2pix GAN to synthesize IR images from RGB inputs. Even though they obtain good results in terms of image to image MSE, their model does not take into the temporal aspect and, thus, is temporally incoherent. This model is a good baseline for our vid2vid approach.

3.3 Methods

During our data analysis phase, we discovered that the IR and RGB images from the FLIR dataset are spatially misaligned. They were taken from two camera at different angles and fields-of-view. Figure 11 shows the misalignment for an example image.

In order to correct for this misalignment, we propose constructing a custom homography to realign the training IR and RGB images. The main assumption to this approach is that the cameras used to capture the images are stationary across each run and across different runs.

Figure 11: Example IR image (Green) overlapped with RGB image (Red) from FLIR dataset, displaying misalignment in camera angle and field-of-view

A different approach would be to use the KAIST dataset instead for training. This is the more favorable approach, since the IR-RGB images from KAIST are spatially aligned (with minor temporal misalignments due to the beam splitter). Furthermore, the KAIST dataset it the largest open source IR-RGB video dataset, with 50,184 image pairs, that captures a many different scenes.

For our model, we proposed using the vid2vid GAN [10] for ir-rgb translation. This is a generative adversarial network, optimized for high resolution video synthesis. Furthermore, the vid2vid model implements a temporal loss and, thus, is temporally coherent. The custom vid2vid model takes label maps and edge maps as input and outputs photorealistic RGB videos.

3.4 Experiments

We use the pre-built vid2vid model for our video synthesis. Setting up the model required changing the CUDA and GCC dependencies. We changed the inputs from label maps to 3-channel IR images and the output to 3-channel RGB images. Furthermore, we normalize the image inputs and outputs to a mean of 0.5 and standard deviation of 0.5 per channel and cropped them to a size of 512×256.

We then trained the altered vid2vid model on a subset of the FLIR and KAIST. According to Figure 12, the discriminator was able to overfit on the training set. We trained using the custom vid2vid model parameters and did not have enough time to get results from finetuning.

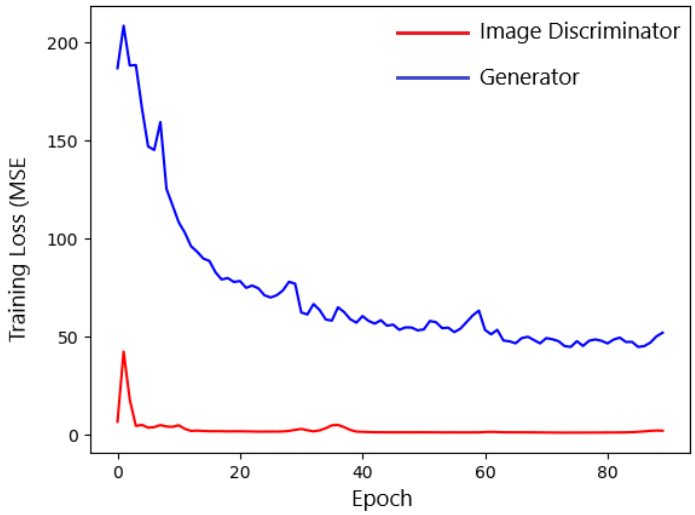


Figure 12: Generator and Discriminator Loss (MSE)

According to Figure 13, even after training on a sample dataset of 1000 images and using the custom vid2vid model hyperparameters, the GAN was still able to produce reasonable outputs.

(a) (b) (c)

Figure 13: (a) Example testing IR image input, (b) corresponding testing RGB image output, (c) model RGB image prediction

A complete implementation of this model is available at our github repo: https://github.com/Rose-STL-Lab/ir2rgb

3.5 Lessons Learned

This project served to lay the groundwork and as a proof-of-concept for IR-RGB video translation. Even from the first model implementation, the results are very promising. However, more training data, training time, and fine-tuning is needed. The GAN is able to overfit the training data but it needs to be robust against various scenes and images. With more training data and additional fine-tuning, the model will be able to a construct sufficient a priori knowledge of natural scenes as to enable the IR-RGB video translation.

This project was very heavy on implementation. Making the model compatible with IR image inputs was a significant task that required additional learning. For example, the we learned about CUDA dependency scripts, GCC scripts, and recompiling snapshots of models.

References

[1] IARAI Traffic4Cast competition: https://www.iarai.ac.at/traffic4cast/

[2] ResNet paper https://arxiv.org/abs/1512.03385

[3] U-Net paper https://arxiv.org/abs/1505.04597

[4] XGBoost paper https://arxiv.org/abs/1603.02754

[5] KNN book https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879

[6] FLIR dataset https://www.flir.com/oem/adas/adas-dataset-form/

[7] KAIST dataset https://sites.google.com/site/pedestrianbenchmark/

[8] pix2pix for IR-RGB paper https://arxiv.org/pdf/1806.01013.pdf

[9] Deep homography estimation: https://arxiv.org/abs/1606.03798

[10] vid2vid model https://github.com/NVIDIA/vid2vid