Week 8
- Autopilot

2019.07.06

Solaris

(http://solarisailab.com)

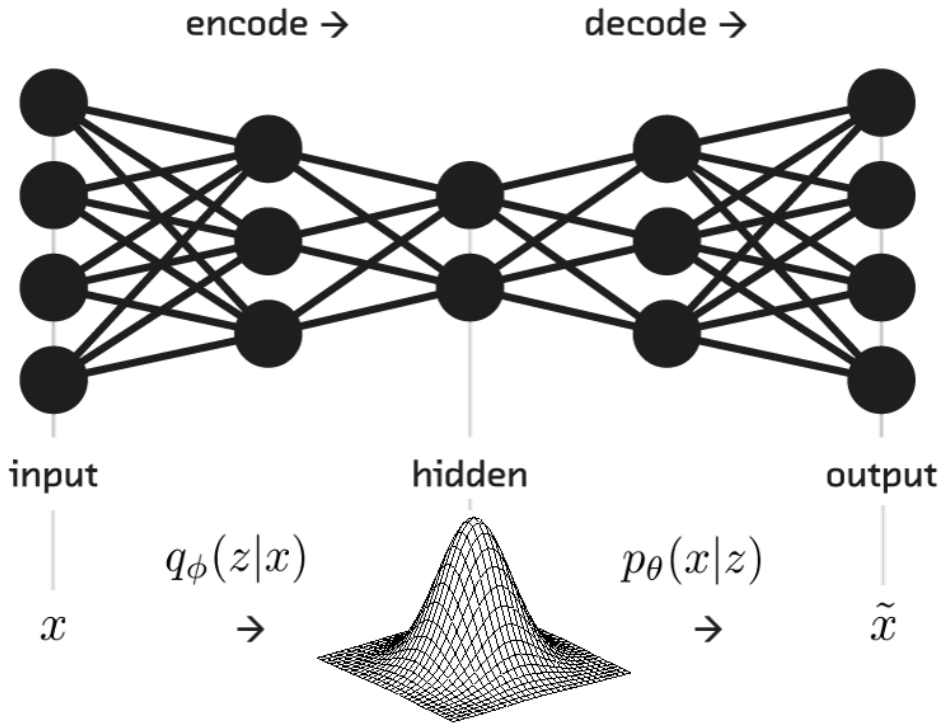
Week7 복습 - Week7의 학습목표

- **7강의 학습목표 :**

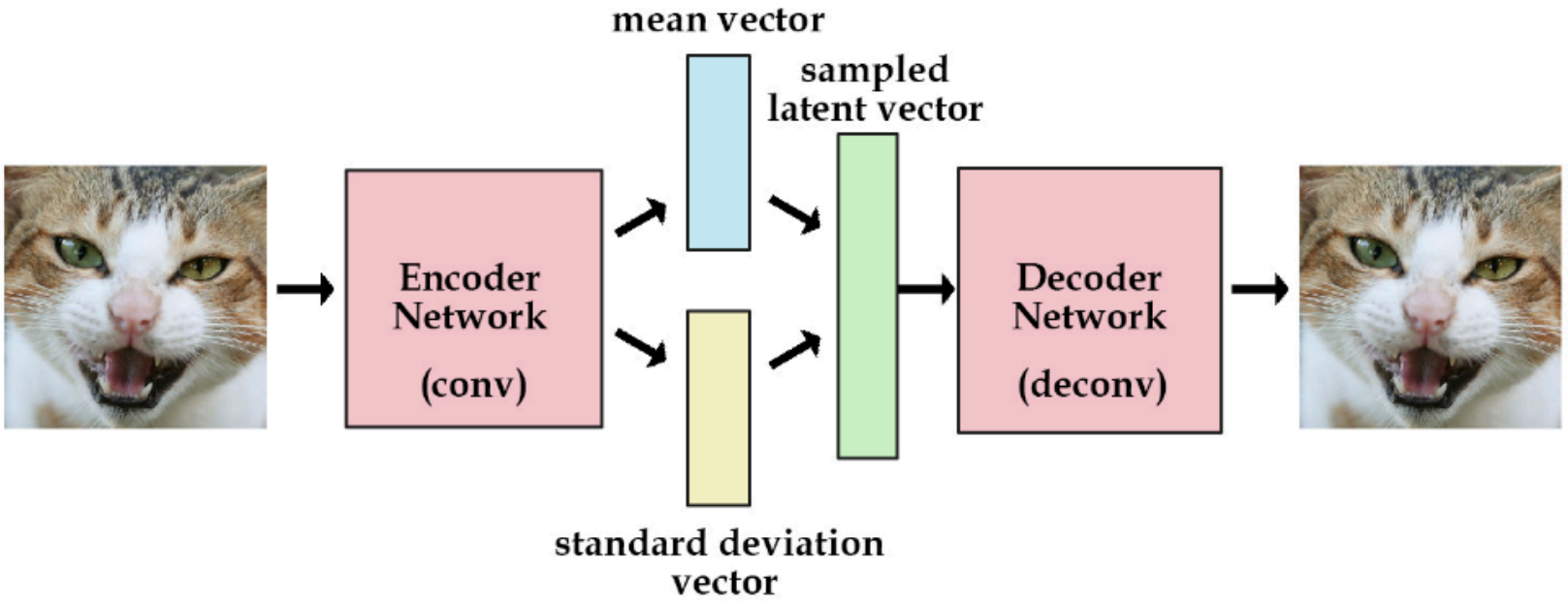
1. Generative Model의 개념을 이해한다.
2. Variational AutoEncoder(VAE)의 개념을 이해한다.
3. Generative Adversarial Networks(GAN)의 개념을 이해한다.
4. TensorFlow를 이용해서 VAE와 GAN을 구현해본다.

Week 7 복습 – Variational AutoEncoder

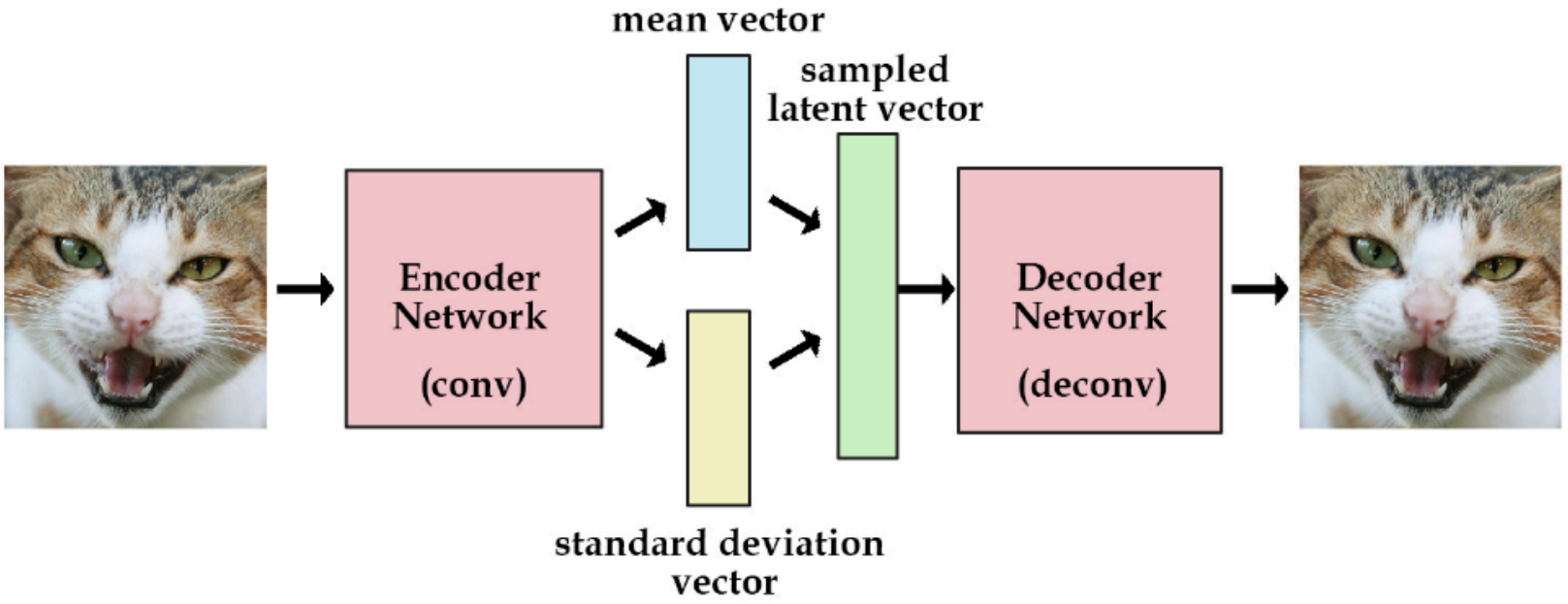
- 확률 분포를 학습하고, 학습한 확률 분포로부터 새로운 데이터를 생성한다.(Generative Model)



Week 7 복습 – Variational AutoEncoder



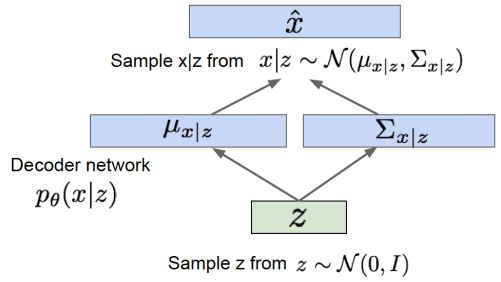
Week 7 복습 – Variational AutoEncoder



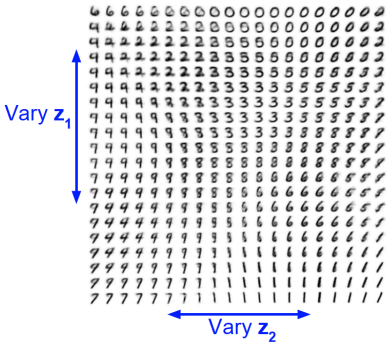
Week 7 복습 – Variational AutoEncoder

Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!

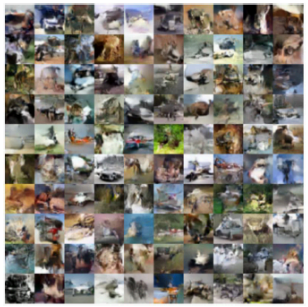


Data manifold for 2-d **z**

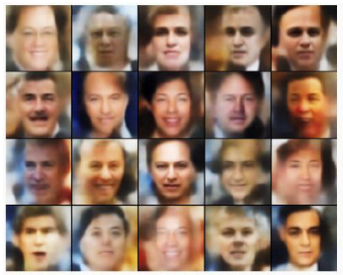


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Generating Data!



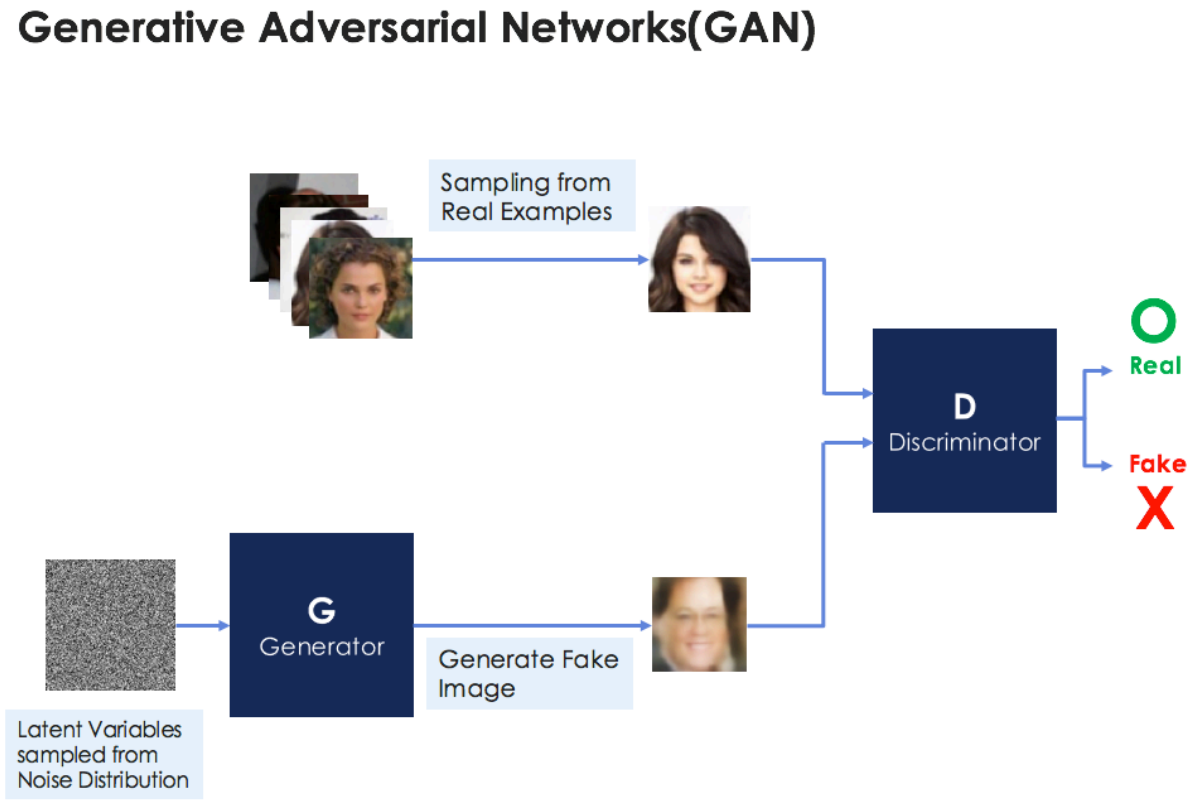
32x32 CIFAR-10



Labeled Faces in the Wild

Week 7 복습 – Generative Adversarial Networks(GAN)

- 경찰(Discriminator)과 위조지폐생성도둑(Generator)
- **Generator(생성자)** – Discriminator를 속이기 위한 이미지를 생성하도록 학습 된다.
- **Discriminator(구분자)** – 주어진 이미지가 진짜 이미지 인지 Generator가 생성한 가짜 이미지인지를 구분하도록 학습 된다.

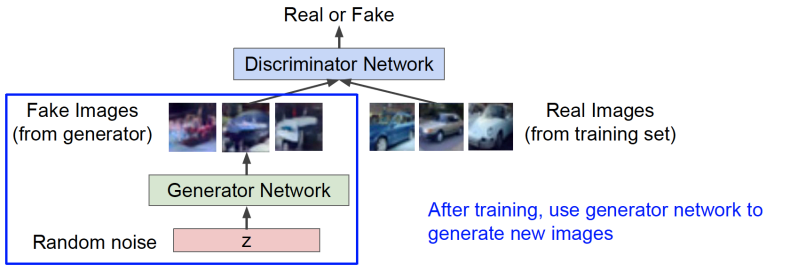


Week 7 복습 – Generative Adversarial Networks(GAN)

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

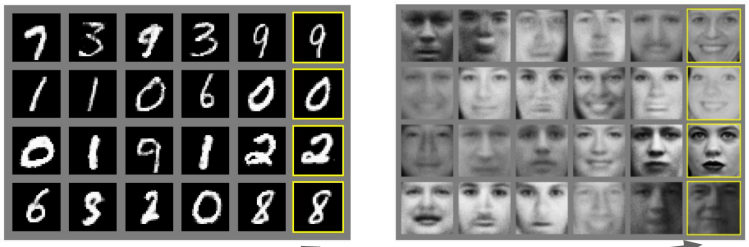


After training, use generator network to generate new images

Generative Adversarial Nets

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generated samples

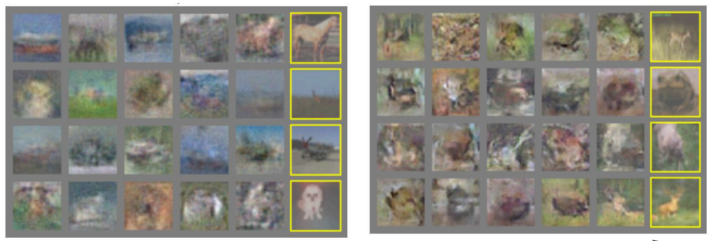


Nearest neighbor from training set

Generative Adversarial Nets

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generated samples (CIFAR-10)



Nearest neighbor from training set

Week 7 복습 – VAE vs GAN

- VAE와 GAN의 비교

1. VAE는 데이터 분포에 대한 일종의 가정(Gaussian Distribution)을 포함하고 있지만 GAN은 순수하게 데이터로부터 데이터 분포를 학습한다.

2. VAE는 Inference P(z|x)를 구할수 있지만 GAN은 그럴 수 없다.

3. 일반적으로 GAN이 더 성능이 좋다.(Blur가 더 적다.)

Outline

- Autopilot (자율주행) Trends
- 논문 리뷰 - Learning a Driving Simulator
- TensorFlow + Keras를 이용한 Learning a Driving Simulator 구현
- 논문 리뷰 - End to End Learning for Self-Driving Cars
- 논문 리뷰 - End-to-end Learning of Driving Models from Large-scale Video Datasets
- ResNet(Residual Networks) 소개
- 논문 리뷰 - Deep Residual Learning for Image Recognition

Week8의 학습목표

- **8강의 학습목표 :**

1. Autopilot을 위한 딥러닝 모델들을 살펴본다.
2. TensorFlow + Keras를 이용해서 Driving Scene Simulator & Steering Angle Prediction을 구현해본다.
3. Residual Networks (ResNet)의 개념을 이해한다.

Autopilot(자율주행) Trends

TC News Startups Mobile Gadgets Enterprise Social Europe Trending Facebook Tesla Snap

TC SESSIONS: ROBOTICS Join TechCrunch in Boston to learn what's next in robotics Get Your Tickets Today

Tesla
Automotive
Popular Posts

Microsoft confirms layoff reports, reorganization expected to impact thousands
2 days ago

Tesla drops 7% after Goldman Sachs says the stock is worth $180
3 days ago

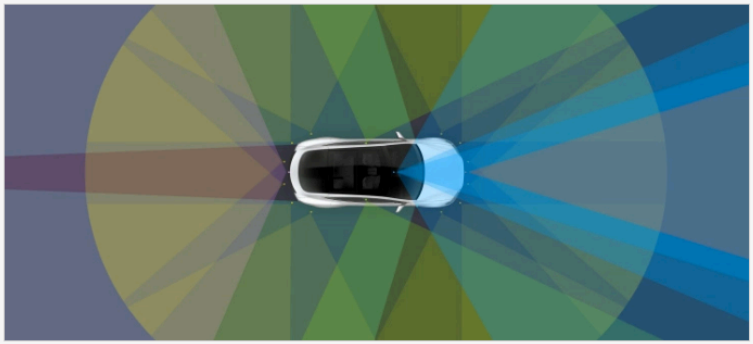
MIT researchers used a $150 Microsoft Kinect to 3D scan a giant T. rex skull
2 days ago

A template for investor/founder sexual harassment policy
3 days ago

Amazon wasn't the only company that tried to buy

Tesla hires deep learning expert Andrej Karpathy to lead Autopilot vision

Posted Jun 20, 2017 by Darrell Etherington (@etherington)



Tesla has hired deep learning and computer vision expert Andrej Karpathy in a key Autopilot role. Karpathy most recently held a role as a researcher at OpenAI, the artificial intelligence nonprofit backed by Elon Musk. He has an extensive background in AI-related fields, having completed a PhD at Stanford University in computer vision.

Karpathy also created one of the original, and most respected, deep learning courses taught at Stanford, and his dissertation work focused on creating a system by which a neural network could identify multiple discrete and specific items within an image, label them using natural language and report to a user. The dissertation also included developing a system that works in reverse, allowing for a model that can use descriptions from a user articulated in natural language (i.e. "white tennis shoes") and find that object in a given

Crunchbase

Tesla

FOUNDED
2003

OVERVIEW
Tesla Motors was started by a group of Silicon Valley entrepreneurs and strives to create a revolution and accelerate the world's transition to electric mobility with a full range of increasingly affordable electric cars. Tesla vehicles are EVs (electric vehicles), which are transforming the way people drive and move. Tesla has gone public as of June 29, 2010 and has a

Autopilot(자율주행) Trends

Drive.ai raises $50 million in funding; Andrew Ng joins board

Reuters Staff

2 MIN READ

Founder and President of Drive.ai Carol Reiley (R) speaks to Ross Anderson of The Atlantic at the "What's Next?" conference in Chicago, Illinois, U.S., October 4, 2016. REUTERS/Jim Young

ALVINN - 1989

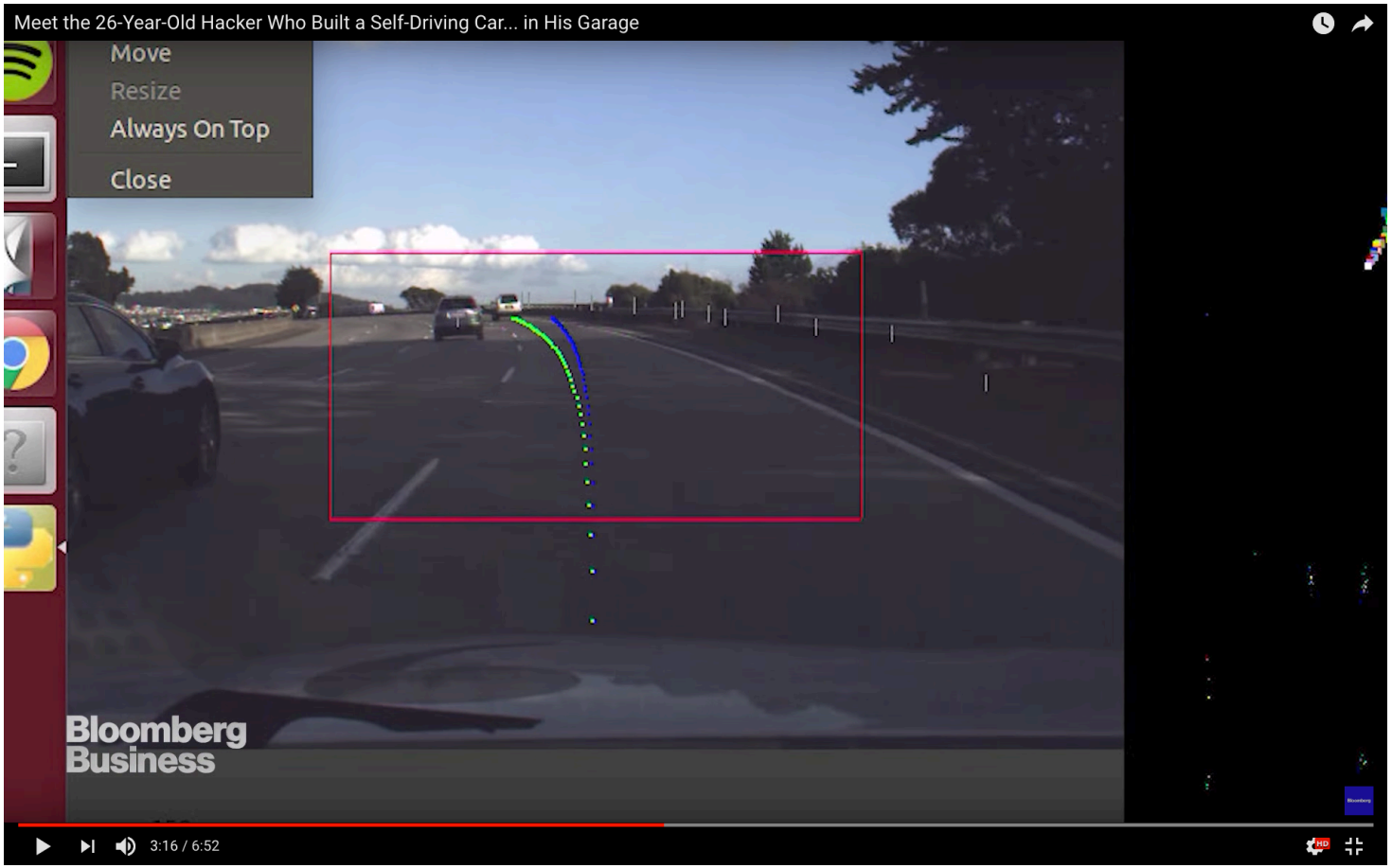
- ALVINN (Autonomous Land Vehicle In a Neural Network) – 1989
- https://www.youtube.com/watch?v=H0igiP6Hg1k

논문 리뷰 - Learning a Driving Simulator

- Santana, Eder, and George Hotz. "Learning a driving simulator." arXiv preprint arXiv:1608.01230 (2016).
- https://arxiv.org/pdf/1608.01230.pdf

Comma.ai Self Driving Car

- https://www.youtube.com/watch?v=KTrgRYa2wbI



Learning a Driving Simulator

- **핵심 아이디어** : 카메라로부터 받은 Raw 도로 Image와 센서들로부터 수집한 정보를 이용해서 다음시간(t+1)에 올 도로 Image를 생성하는 모델(Generative Model)을 만든다. (Driving Simulator 구현)

- Simulated World에서 Control을 어떻게 할 것인지는 다음 논문으로 남겨 둠 –> github 코드는 제공

- **Main Contribution** : We show that it is possible to train a model to do **realistic looking video predictions**, while calculating a **low dimensional compact representation** and action conditioned transitions.

Vison based Control

- 기존의 Vision based Control 시스템 -> 환경에 대한 정보에 무제한적으로 접근할 수 있다.(e.g. Alphago -> 바둑 게임을 무제한으로 시뮬레이션 해 볼 수 있다.)

- 환경에 대한 정보에 무제한적으로 접근할 수 없는 경우도 있다. (e.g. 실제 주행환경에 대한 정보) -> Here instead we focus on learning to simulate aspects of the world from examples of a human agent. We focus on **generating video streams of a front facing camera mounted in the car windshield.**

Driving Dataset

- 실제 주행을 통해 모은 driving dataset

Figure 1: 80 × 160 samples from the driving dataset.

Driving Dataset

- In the released dataset there is a total of **7.25 hours of driving data** divided in 11 videos. The released video frames are 160x320 pixels regions from the middle of the captured screen.
- Beyond video, the dataset also has several sensors that were measured in different frequencies and interpolated to 100Hz.
- Example data coming from sensors are the **car speed, steering angle, GPS, gyroscope, IMU, etc.**
- In this paper we focus on the camera frames, steering angle and speed. We preprocessed the camera frames by **downsampling them to 80x160 and renormalizing the pixel values between -1 and 1.**

Problem Definition

- **Driving Simulator** = 현재 시간 $\boldsymbol{t}$ 에 보이는 영상과 현재 자동차 속도(Car speed), 조향각(Steering Angle)을 기반으로 다음 시간 $\boldsymbol{t+1}$ 에 올 영상을 예측 하는 문제

- $\boldsymbol{X_t} = \{x_{t-n+1}, x_{t-n+2}, \dots, x_t\}$: $\boldsymbol{x_t}$ 는 시간 $\boldsymbol{t}$에서의 프레임을 나타낸다.

- $\boldsymbol{S_t} = \{s_{t-n+1}, s_{t-n+2}, \dots, s_t\}$: $\boldsymbol{s_t}$ 는 시간 $\boldsymbol{t}$에서의 자동차 속도(Car Speed)를 나타낸다.

- $\boldsymbol{A_t} = \{a_{t-n+1}, a_{t-n+2}, \dots, a_t\}$: x_t 는 시간 $\boldsymbol{t}$에서의 조향각(Steering Angle)을 나타낸다.

Problem Definition

- 이를 수학적으로 표현하면, 다음의 Function $\boldsymbol{F}$ 를 Learning하는 문제이다.

- $\boldsymbol{F}: \mathbb{R}^{80\times160\times3\times n}\times\mathbb{R}^{n}\times\mathbb{R}^{n} \rightarrow \mathbb{R}^{80\times160\times3\times n}$ that predicts $\boldsymbol{x_{t+1}} = \boldsymbol{F}(\boldsymbol{X_t}, \boldsymbol{S_t}, \boldsymbol{A_t})$

- 최근에 generative video를 위한 모델들이 제안 됨. -> 하지만 기존 모델들은 저차원의 **hidden code**들을 사용하지 않음

- 제안된 모델 -> We believe that **compact intermediate representations are important** for our research

Problem Definition

- First we learned an autoencoder to **embed the frames x_t into a Gaussian latent space z_t ∈ R^2048**, where the dimensionality 2048 was chosen experimentally and the Gaussian assumption enforced with variational autoencoding Bayes [1].

- This first step simplified the problem from learning transitions directly in the pixel space to **learning in the latent space**. Beyond that, assuming that an autoencoder is correctly learned respecting the latent space Gaussianity, **we can generate realistic looking videos** as long as the transition model never leaves the high density region of the embedding space.

Overall Architecture

- Variational AutoEncoders(VAE) + Generative Adversarial Networks(GAN)

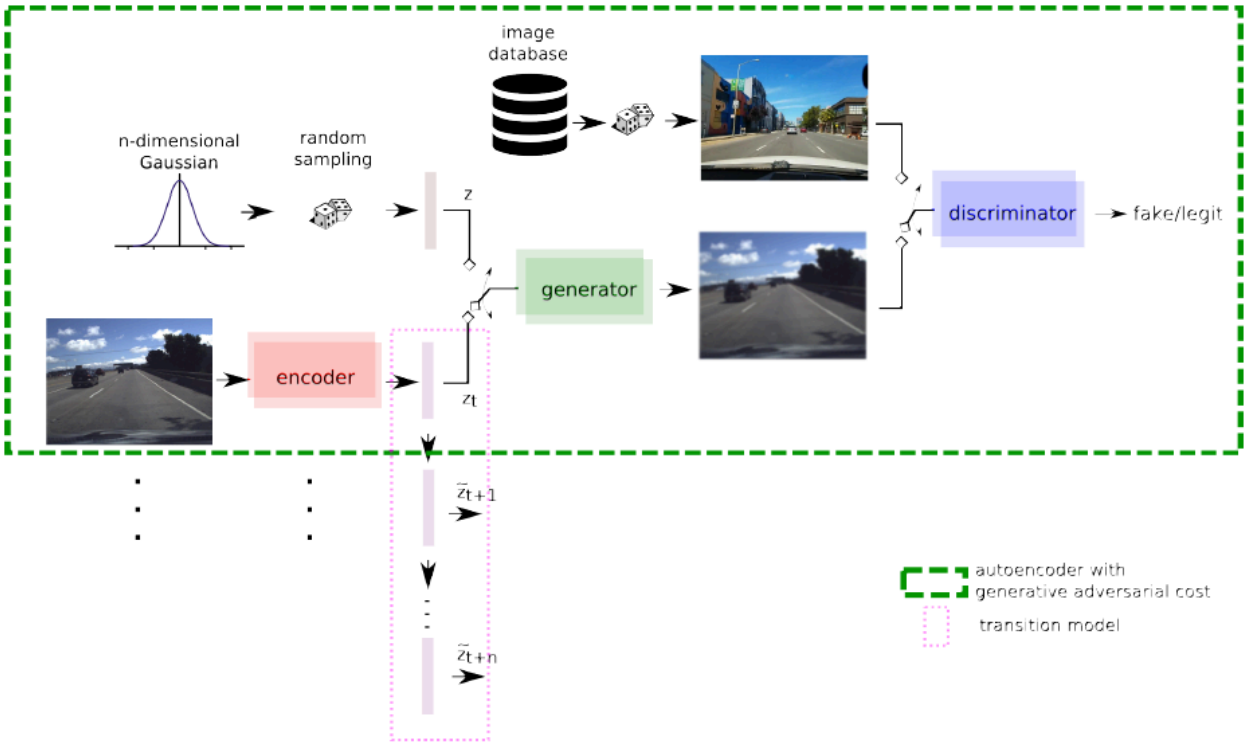


Figure 2: Driving simulator model: an autoencoder trained with generative adversarial costs coupled with a recurrent neural network transition model

Learning a Driving Simulator

- 논문에서는 두가지 모델을 제안한다.

1. an autoenecoder for dimensionality reduction
2. an action conditioned RNN for learning the transitions.

- Unfortunately, the Gaussian assumption in the original data space does not hold for natural images and **VAE predictions usually look blurry** (see Fig. 3).
- On the other hand, **generative adversarial networks (GAN)** [22] and related work [2] [3] learn the generative model cost function alongside the generator.

Learning a Driving Simulator

- The **generative model transforms samples from the latent space distribution into data from a desired dataset**.

- The **discriminator network tries to tell samples from the desired dataset from images sampled by the generator.** The generator is trained to "fool" the discriminator, thus the discriminator can be considered a learned cost function for the generator.

- For our purposes we need to learn not only the generator from the latent to the road images space. But also an encoder from road images back to the latent space. **We need to combine VAE and GANs.** Intuitively, one could simply combine the VAE approach with a learned cost function.

Cost Function

- We used Larsen et. al. [25] approach to train our autoencoder. In Fig. 2 we show the schematic diagram of this model as part of our architecture. According to [25] **the encoder (Enc)**, **generator (Gen)** and **discriminator (Dis)** networks are optimized to minimize the following cost function:

$$\mathcal{L} = \mathcal{L}_{prior} + \mathcal{L}_{llike}^{Dis_l} + \mathcal{L}_{GAN}.$$

VAE + GAN

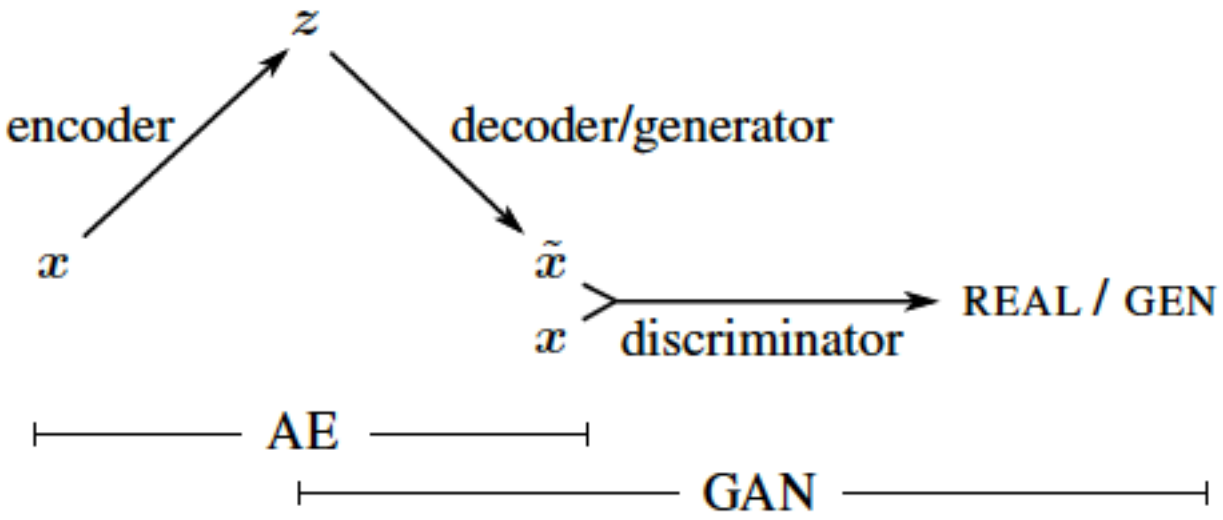


Figure 1. Overview of our network. We combine a VAE with a GAN by collapsing the decoder and the generator into one.

VAE + GAN

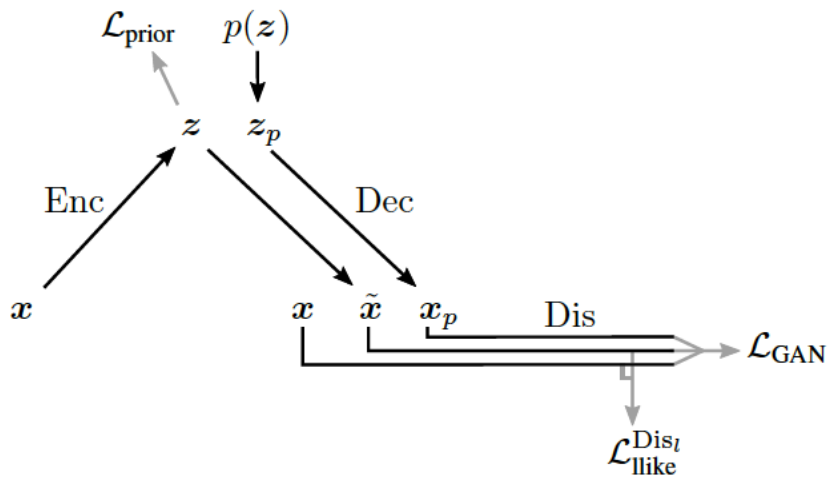


Figure 2. Flow through the combined VAE/GAN model during training. Gray lines represent terms in the training objective.

Algorithm 1 Training the VAE/GAN model

$\boldsymbol{\theta}_{\text{Enc}}, \boldsymbol{\theta}_{\text{Dec}}, \boldsymbol{\theta}_{\text{Dis}} \leftarrow$ initialize network parameters

repeat

$\boldsymbol{X} \leftarrow$ random mini-batch from dataset

$\boldsymbol{Z} \leftarrow \text{Enc}(\boldsymbol{X})$

$\mathcal{L}_{\text{prior}} \leftarrow D_{\text{KL}}(q(\boldsymbol{Z}|\boldsymbol{X}) \| p(\boldsymbol{Z}))$

$\tilde{\boldsymbol{X}} \leftarrow \text{Dec}(\boldsymbol{Z})$

$\mathcal{L}_{\text{llike}}^{\text{Dis}_l} \leftarrow -\mathbb{E}_{q(\boldsymbol{Z}|\boldsymbol{X})}\left[p(\text{Dis}_l(\boldsymbol{X})|\boldsymbol{Z})\right]$

$\boldsymbol{Z}_p \leftarrow$ samples from prior $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$

$\boldsymbol{X}_p \leftarrow \text{Dec}(\boldsymbol{Z}_p)$

$\mathcal{L}_{\text{GAN}} \leftarrow \log(\text{Dis}(\boldsymbol{X})) + \log(1 - \text{Dis}(\tilde{\boldsymbol{X}})) + \log(1 - \text{Dis}(\boldsymbol{X}_p))$

// Update parameters according to gradients

$\boldsymbol{\theta}_{\text{Enc}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Enc}}}(\mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{llike}}^{\text{Dis}_l})$

$\boldsymbol{\theta}_{\text{Dec}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dec}}}(\gamma\mathcal{L}_{\text{llike}}^{\text{Dis}_l} - \mathcal{L}_{\text{GAN}})$

$\boldsymbol{\theta}_{\text{Dis}} \overset{+}{\leftarrow} -\nabla_{\boldsymbol{\theta}_{\text{Dis}}}\mathcal{L}_{\text{GAN}}$

until deadline

Cost Function

$$\mathcal{L} = \mathcal{L}_{prior} + \mathcal{L}_{llike}^{Dis_l} + \mathcal{L}_{GAN}.$$

- **L_prior = D_KL(q(z|x)||p(z))** is the Kullback-Liebler divergence between the distribution of the encoder outputs, q(z|x), and a prior distribution, p(z). This is the same **VAE regularizer**. Here the prior distribution is p(z) is a Gaussian N(0, 1) and we use the reparametrization trick [1] to optimize that regularizer. Thus, during training we have z = $A = \mu + \epsilon\sigma$ and at test time z = μ, where μ and σ are outputs of the encoder network and ϵ is a Gaussian random vector with the same number of dimensions as μ and σ.

Cost Function

$$\mathcal{L} = \mathcal{L}_{prior} + \mathcal{L}_{llike}^{Dis_l} + \mathcal{L}_{GAN}.$$

- L^Disl_llike is an error calculated using Dis_l, the **hidden activations of the l-th layer of the discriminator network, Dis**. The activations are calculated using a legit image x and its corresponding encoded-decoded version Gen(Dis(x)). Assuming $\boldsymbol{y_l}$= **Dis_l(x)** and $\tilde{\boldsymbol{y}}_{\boldsymbol{l}}$ **= Dis_l(Gen(Enc(x)))**, we have $\boldsymbol{L}_{\boldsymbol{llikie}}^{\boldsymbol{Dis_l}} = \mathbf{E}[(\boldsymbol{y_l} - \tilde{\boldsymbol{y}}_{\boldsymbol{l}})^{\boldsymbol{2}}]$.

Cost Function

$$\mathcal{L} = \mathcal{L}_{prior} + \mathcal{L}_{llike}^{Dis_l} + \mathcal{L}_{GAN}.$$

- Finally, **L_GAN is the generative adversarial network cost** [22]. That cost function represents the game between Gen and Dis.

- **When training Dis, both Enc and Gen are kept fixed and we have**

$$\mathcal{L}_{GAN}^{Dis} = log(Dis(x)) + log(1 - Dis(Gen(u))) + log(1 - Dis(Gen(Enc(x))),$$

- where **u ~ N(0, 1)** is another random variable. The first r.h.t of (2) represents the log-likelihood of Dis recognizing legit samples and the other two terms represents its log-likelihood to tell fake samples generated from both random vectors u or codes z = Enc(x).

- **When training Gen, we keep both Dis and Enc fixed and we have**

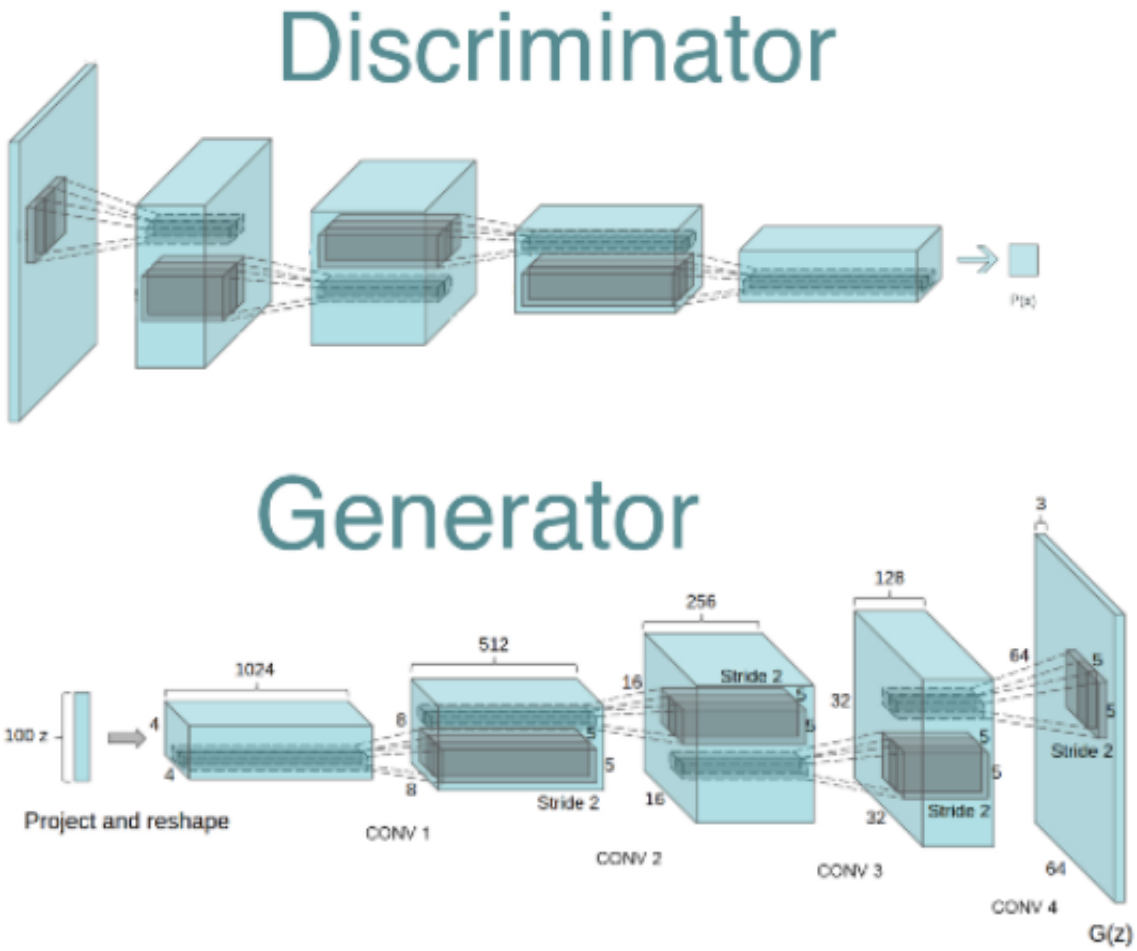
$$\mathcal{L}_{GAN}^{Gen} = log(Dis(Gen(u))) + log(Dis(Gen(Enc(x))),$$

- which accounts for **Gen being able to fool Dis.**

Training Parameters

- 트레이닝에 사용한 parameter들은 다음과 같다.
- Epochs : 200
- Batch size : 64
- Optimizer : Adam
- The **autoencoder architecture followed Radford et. al.[3]**, with the generator made of 4 deconvolutional layers each one followed by batch normalization and leaky-ReLU activations. The discriminator and encoder consisted of convolutional layers where each but the first layer was followed by batch normalization. The activation function was ReLU.
- **Dis_l is the output of the 3rd convolutional layer of the decoder**, before batch normalization and ReLU are applied.
- The output size of the discriminator network is 1 and its cost function was binary cross-entropy.
- The output size of the encoder network is 2048. This compact representation is almost 16 times smallet than the original data dimensionality.
- **After training the autoencoder, we fix all its weights and use Enc as preprocessing step for training the transition model.**

Deep Convolutional Generative Adversarial Networks(DCGAN)



Transition model - RNNs

- After training the autoencoder described above we obtain the transformed dataset applying Enc : $x_t \mapsto z_t$. We train RNN : $x_t, h_t, c_t \mapsto z_{t+1}$ to represent the **transitions in the code space**:

$$h_{t+1} = tanh\left(Wh_t + Vz_t + Uc_t\right),$$
$$\tilde{z}_{t+1} = Ah_{t+1}$$

- Where W, V, U, A are trainable weights, h_t is the **hidden state of the RNN** and c_t are the **concatenated control speed and angle signal**. We leave LSTMs, GRU and multiplicative iterations between c_t and z_t for **future investigations**.

- The cost function for optimizing the trainable weights is simply the mean square error:

$$\mathcal{L}_{RNN} = \mathbb{E}\left[(z_{t+1} - \tilde{z}_{t+1})^2\right].$$

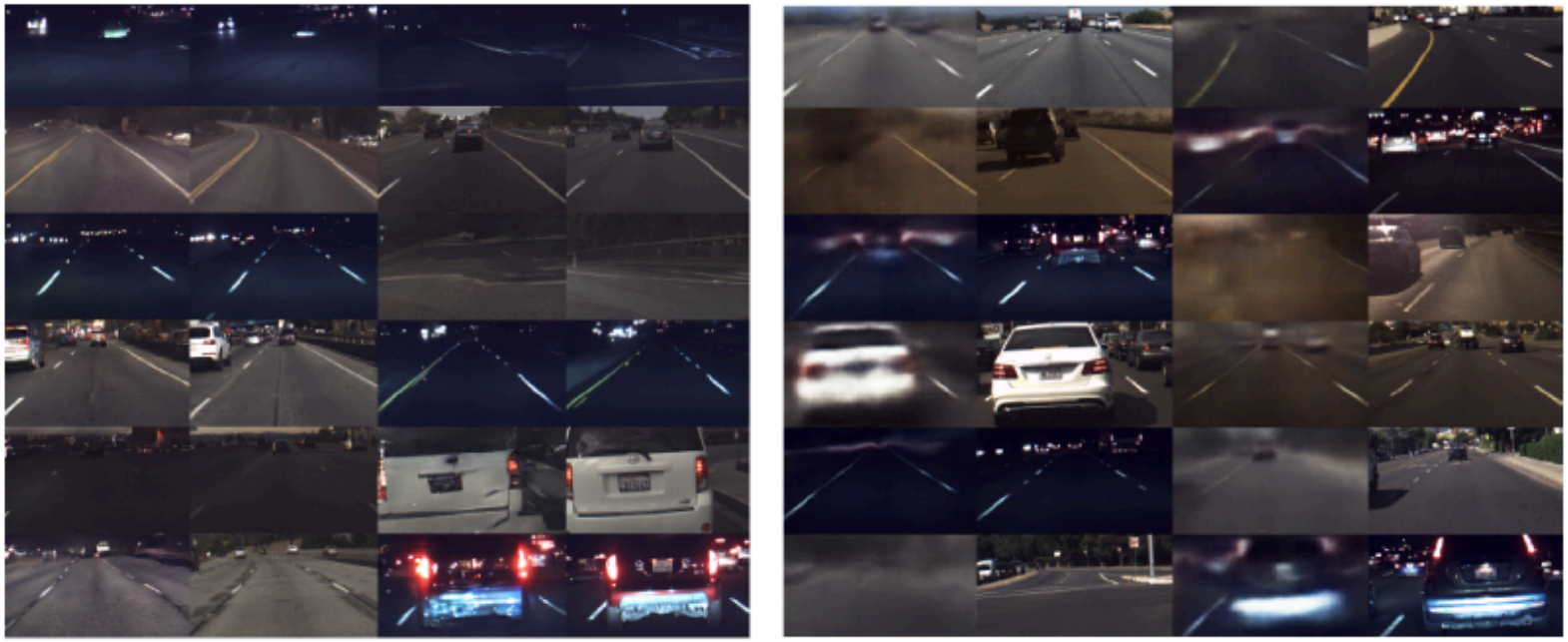
- It is easy to see that (5) is optimal because we impose a Gaussian constraint L_{prior} in the distribution of the codes z when training the autoencoder. In other words, MSE equals up to a constant factor the log-likelihood of a normally distributed random variable.

- **Given a predicted code $\tilde{z}_{t+1}$ we can estimate future frames as $\tilde{x}_{t+1} = Gen(\tilde{z}_{t+1})$.**

Simple MSE vs GAN

- Although results were similar in terms of MSE, a learned cost function using generative adversarial networks had the most visually appealing results.
- In Fig. 3 we show decoded images with models trained using two different cost functions. As expected, **MSE based neural networks generate blurred images.** The main issues for our purposes is that blurring connects the lane marks into a single long lane.

Experiment Result



(a) (b)

Figure 3: Samples using similar fully convolutional autoencoders. Odd columns show decoded images, even columns show target images. Models were trained using (a) generative adversarial networks cost function (b) mean square error. Both models have MSE in the order of 10^{-2} and PSNR in the order of 10.

Experiment Result

- Once we obtained a good autoencoder, we trained the transition model. Results of predicted frames are shown in Fig. 4 . We trained the transition model in 5Hz videos. **The learned transition model keeps the road structure even after 100 frames.**

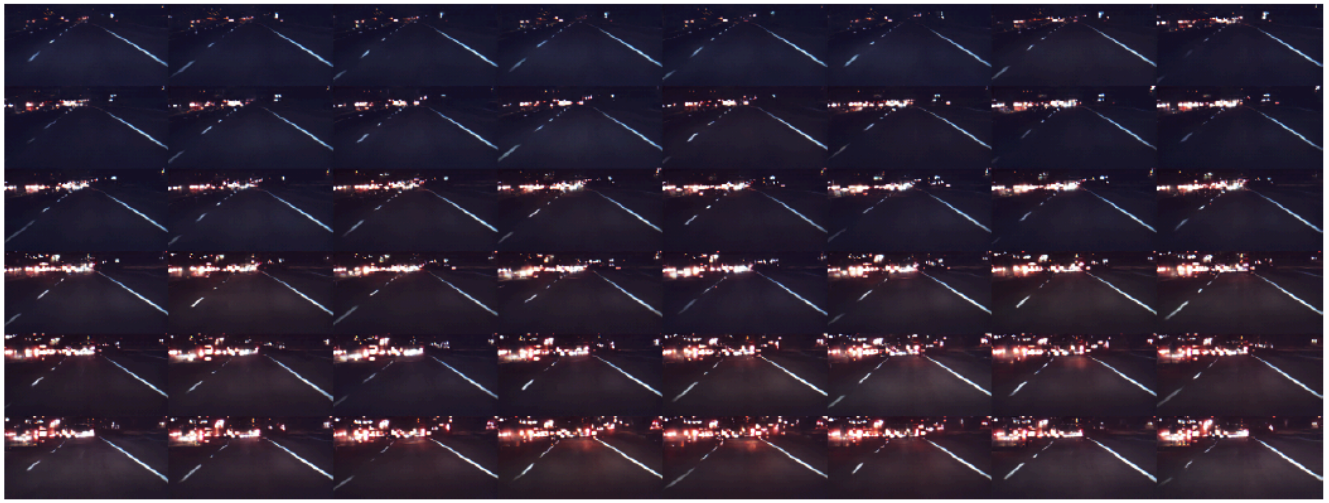
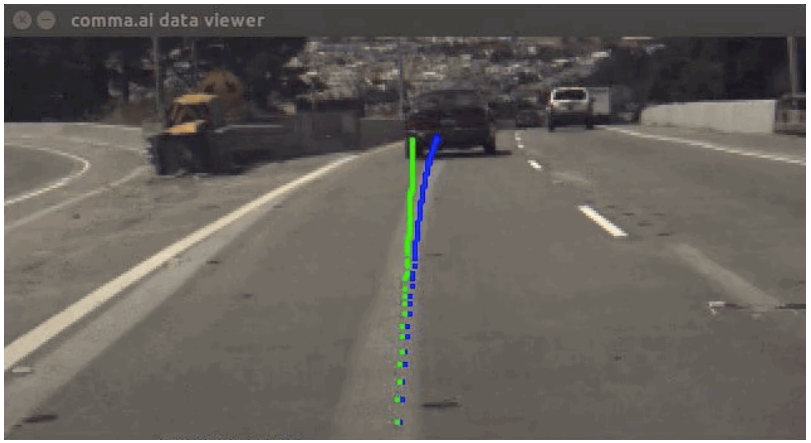


Figure 4: Samples generated by letting the transition model *hallucinate* and decoding $\tilde{z}$ using Gen. Note that Gen was not optimized to make these samples realistic, which support our assumption that the transition model did not leave the code space.

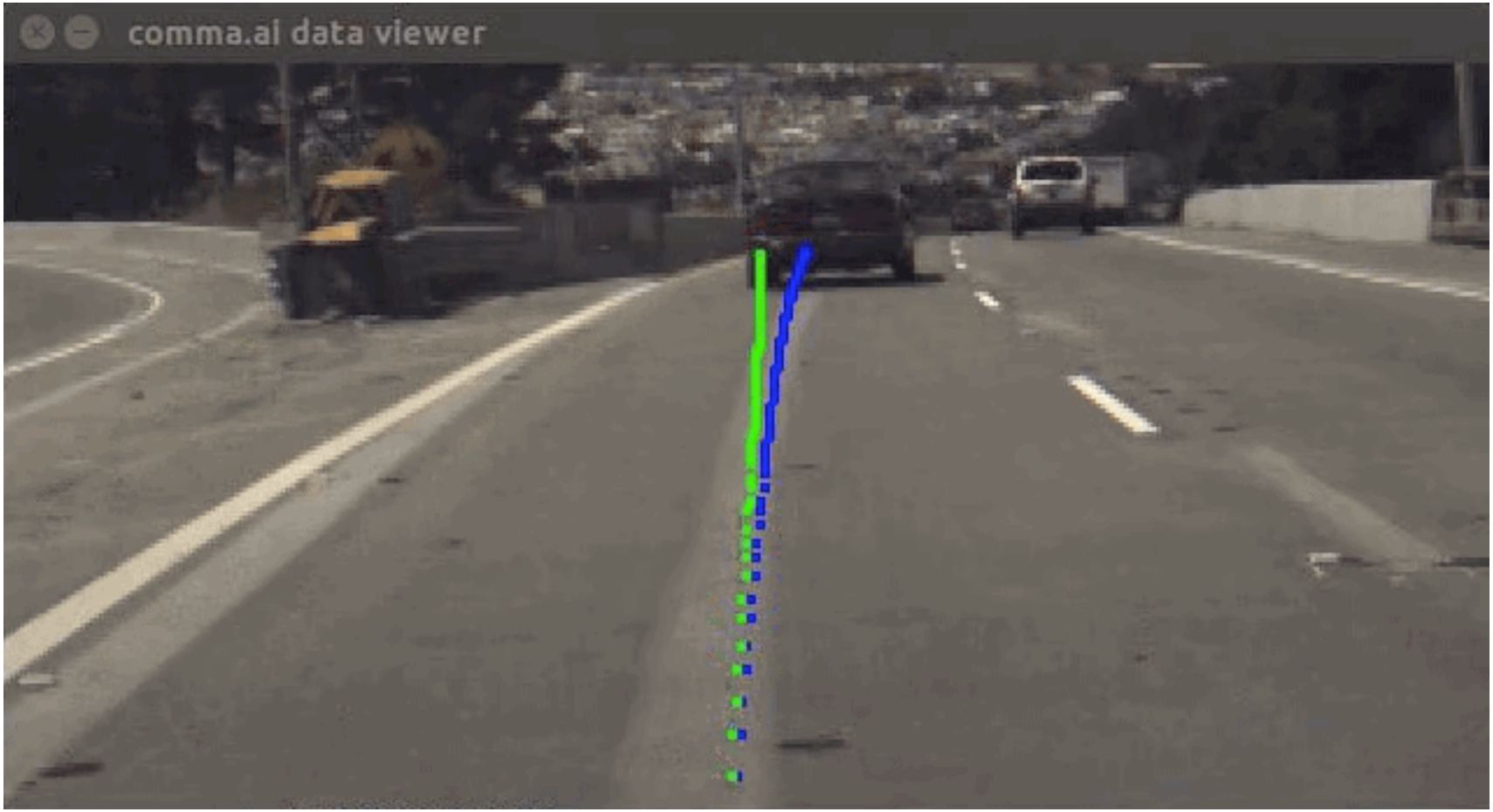
TensorFlow + Keras를 이용한 Learning a Driving Simulator 구현

- https://github.com/solaris33/dl_cv_tensorflow_10weeks/tree/master/week7/research



Training Steering Angle Prediction

- https://github.com/solaris33/dl_cv_tensorflow_10weeks/blob/master/week7/research/DriveSim.md



Training a generative image model

- https://github.com/solaris33/dl_cv_tensorflow_10weeks/blob/master/week7/research/SelfSteering.md

논문 리뷰 - End to End Learning for Self-Driving Cars

- Bojarski, Mariusz, et al. "End to end learning for self-driving cars." *arXiv preprint arXiv:1604.07316* (2016).

- **핵심 아이디어** : Human Driver의 steering wheel angle control Data와 Camera로 획득한 Image Data를 토대로, CNNs을 이용해서 해당 Image 상태에서 적절한 steering wheel angle을 prediction하는 모델을 제안.

- https://arxiv.org/pdf/1604.07316.pdf

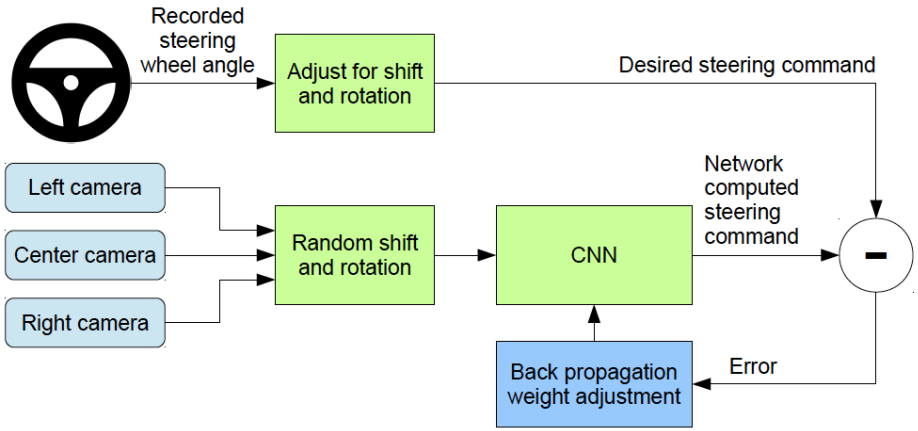


Figure 2: Training the neural network.

Data Collecting

- Three cameras are mounted behind the windshield of the data-acquisition car. Time-stamped video from the **cameras is captured simultaneously with the steering angle applied by the human driver.**

- Training with data from only the human driver is not sufficient. **The network must learn how to recover from mistakes.** Otherwise the car will slowly drift off the road. The training data is therefore augmented with additional images that show the car in different shifts from the center of the lane and rotations from the direction of the road.

- Images for two specific off-center shifts can be obtained from the left and the right camera.

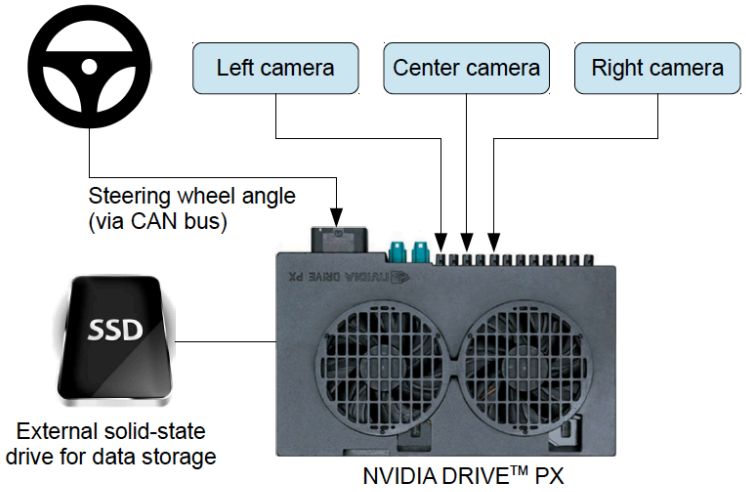


Figure 1: High-level view of the data collection system.

Overall Architecture

- Images are fed into a CNN which then computes a proposed steering command.
- **The proposed command is compared to the desired command for that image** and the weights of the CNN are adjusted to bring the CNN output closer to the desired output.

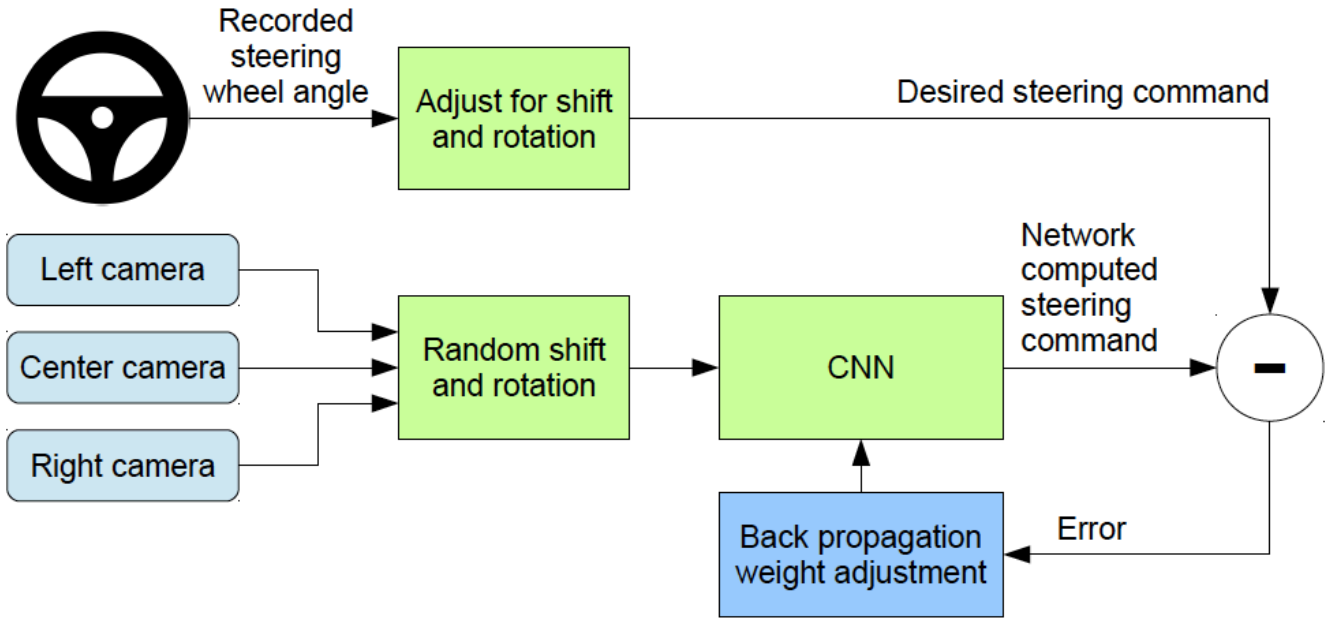


Figure 2: Training the neural network.

Driving by CNNs Prediction

- Once trained, the network can generate steering from the video images of a single center camera.

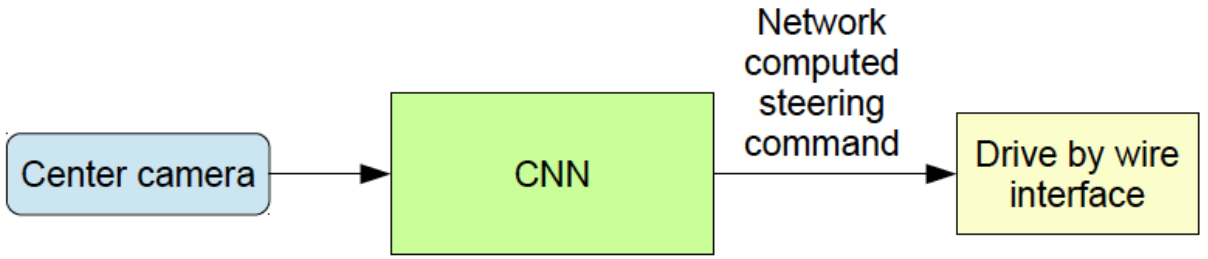


Figure 3: The trained network is used to generate steering commands from a single front-facing center camera.

CNNs architecture

- **The first layer of the network performs image normalization**. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing.

- The **convolutional layers were designed to perform feature extraction** and were chosen empirically through a series of experiments that varied layer configurations. We use strided convolutions in the **first three convolutional layers with a 2x2 stride and a 5x5 kernel** and a **non-strided convolution with a 3x3 kernel size in the last two convolutional layers.**

- We follow **the five convolutional layers with three fully connected layers leading to an output control value** which is the inverse turning radius.

CNNs architecture

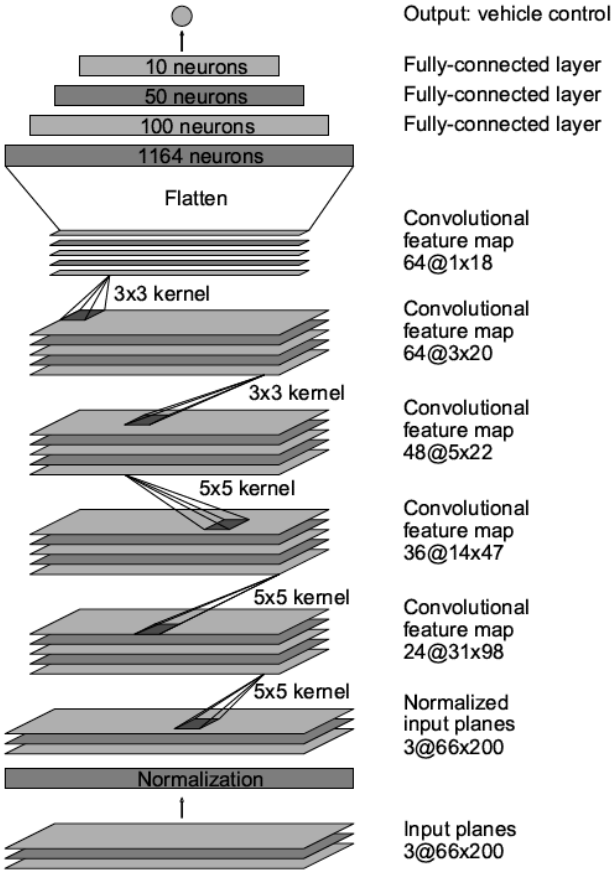


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

Performance Measure

- 성능을 측정하기 위해서 **전체 주행 시간중에 몇 %가 자율주행(Autonomy)인지**를 측정한다.
- 만약 차선을 일정거리 이상 벗어나면 인간이 개입해서 정상적인 차선으로 되돌려놓는다. 다시 정상적인 차선으로 되돌려 놓는데 6초의 시간이 걸린다고 가정하였다.
- We calculate the percentage autonomy by counting the number of interventions, multiplying by 6 seconds, dividing by the elapsed time of the simulated test, and then subtracting the result from 1:

$$\text{autonomy} = (1 - \frac{(\text{number of interventions}) \cdot 6 \text{ seconds}}{\text{elapsed time [seconds]}}) \cdot 100$$

- Thus, if we had 10 interventions in 600 seconds, we would have an autonomy value of

$$(1 - \frac{10 \cdot 6}{600}) \cdot 100 = 90\%$$

On-road Tests

- For these tests we measure performance as the fraction of time during which the car performs autonomous steering. This time excludes lane changes and turns from one road to another. For a typical drive in Monmouth County NJ from our office in Holmdel to Atlantic Highlands, we are autonomous **approximately 98% of the time.**

- https://www.youtube.com/watch?v=NJU9ULQUwng

CNNs Feature Map Visualization

- Figures 7 and 8 show the activations of the first two feature map layers for two different example inputs, an unpaved road and a forest. In case of the unpaved road, the feature map activations clearly show the outline of the road while in case of the forest the feature maps contain mostly noise, i. e., the CNN finds no useful information in this image.

- This demonstrates that the **CNN learned to detect useful road features on its own**, i. e., with only the human steering angle as training signal. **We never explicitly trained it to detect the outlines of roads, for example.**

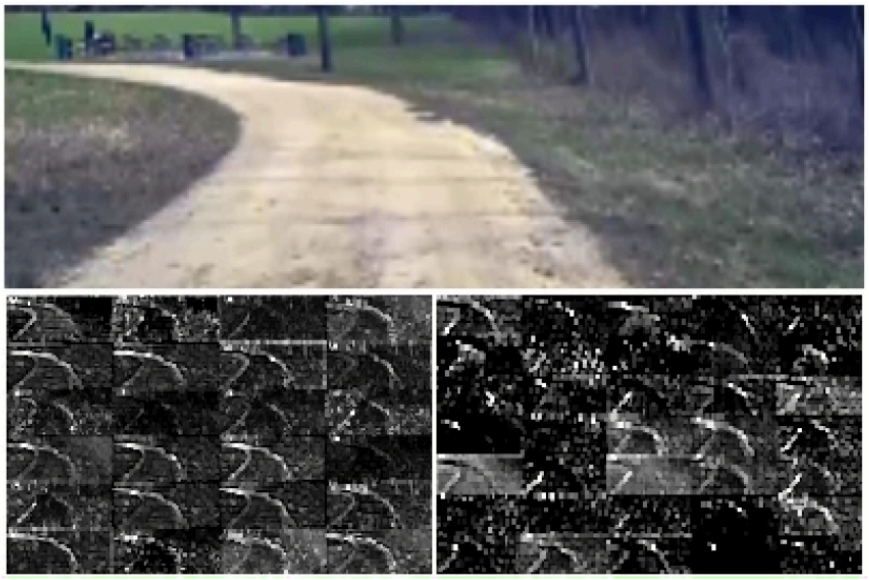


Figure 7: How the CNN "sees" an unpaved road. Top: subset of the camera image sent to the CNN. Bottom left: Activation of the first layer feature maps. Bottom right: Activation of the second layer feature maps. This demonstrates that the CNN learned to detect useful road features on its own, i. e., with only the human steering angle as training signal. We never explicitly trained it to detect the outlines of roads.

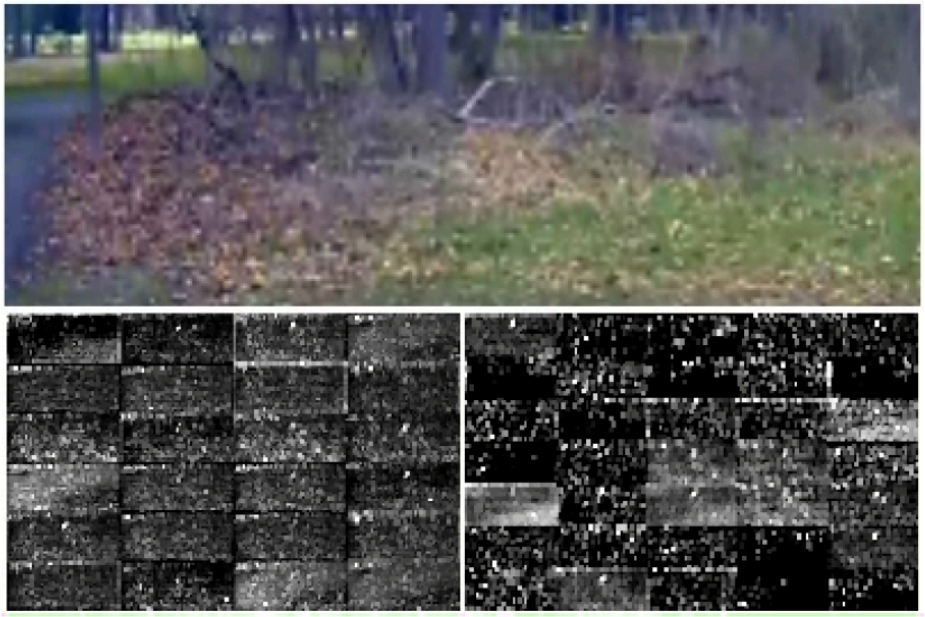


Figure 8: Example image with no road. The activations of the first two feature maps appear to contain mostly noise, i. e., the CNN doesn't recognize any useful features in this image.

논문 리뷰 - End-to-end Learning of Driving Models from Large-scale Video Datasets

- Xu, Huazhe, et al. "End-to-end learning of driving models from large-scale video datasets." *arXiv preprint arXiv:1612.01079* (2016).
- **핵심 아이디어** : Autopilot을 위한 Large Dataset을 공개하고, FCN-LSTM 모델을 제안
- https://arxiv.org/pdf/1612.01079.pdf

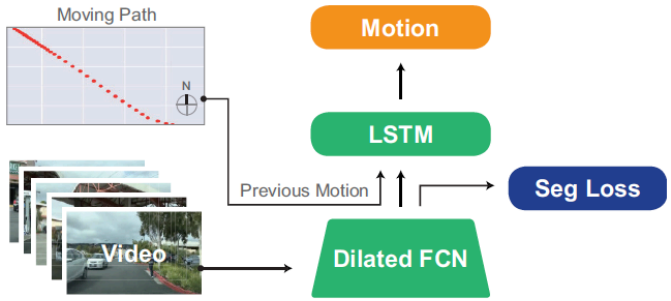


Figure 1: Autonomous driving is formulated as a future egomotion prediction problem. Given a large-scale driving video dataset, an end-to-end FCN-LSTM network is trained to predict multi-modal discrete and continuous driving behaviors. Using semantic segmentation as a side task further improves the model.

FCN-LSTM Architecture

- We propose a novel architecture for time-series prediction which fuses an LSTM temporal encoder with a fully convolutional visual encoder.

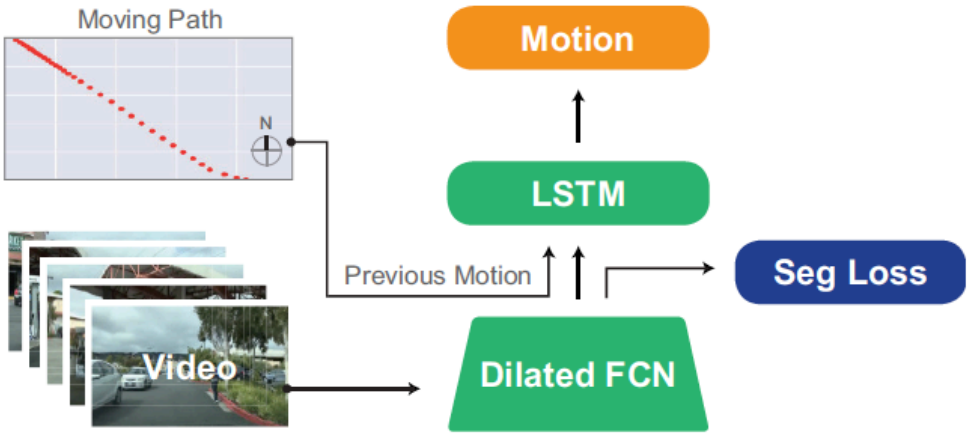


Figure 1: Autonomous driving is formulated as a future egomotion prediction problem. Given a large-scale driving video dataset, an end-to-end FCN-LSTM network is trained to predict multi-modal discrete and continuous driving behaviors. Using semantic segmentation as a side task further improves the model.

Visual Encoder

- In our architecture, a **dilated fully convolutional neural network** [26, 5] is used to extract the visual representations. We take the ImageNet [19] pre-trained AlexNet [11] model, remove POOL2 and POOL5 layers and use dilated convolutions for conv3 through fc7.

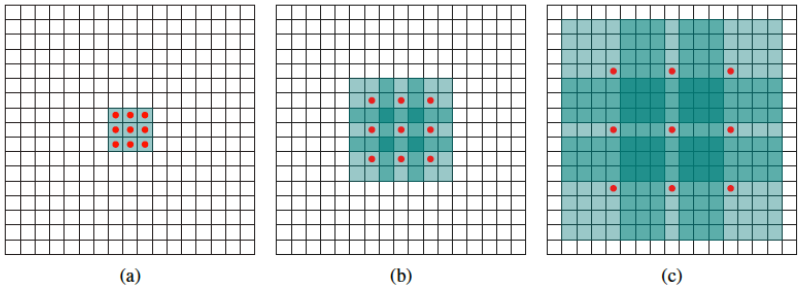


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

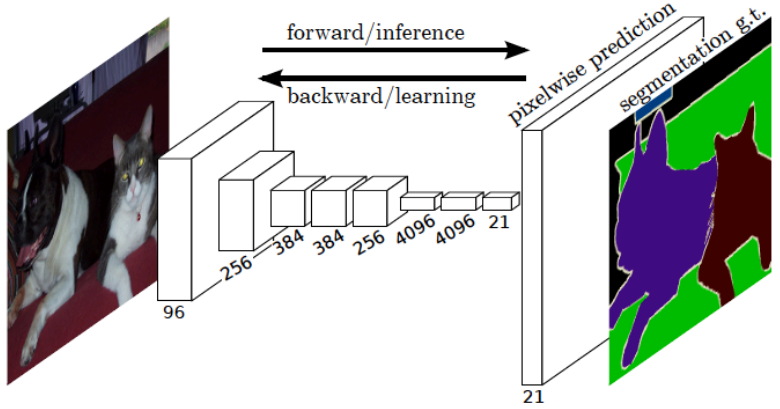


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Dilated Convolution

- 팽창된(Dilated) Convolution
- Filter의 크기가 커지더라도 유지해야할 Parameter개수는 그대로 유지한다. (필터의 크기가 커진 부분은 모두 0으로 채운다.)

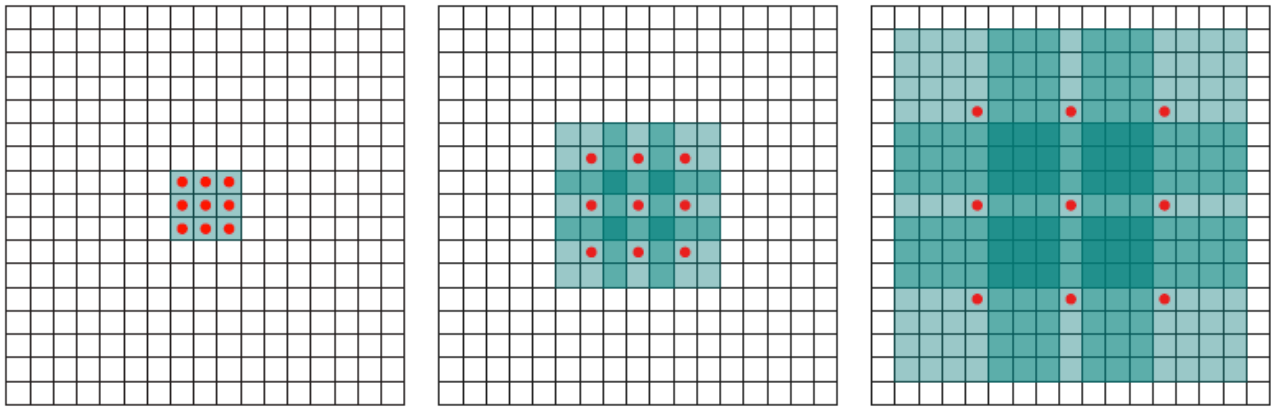


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

Fully Convolutional Networks (FCN) for Semantic Segmentation

- "Fully convolutional networks for semantic segmentation.", Long, Jonathan, Evan Shelhamer, and Trevor Darrell, CVPR 2015

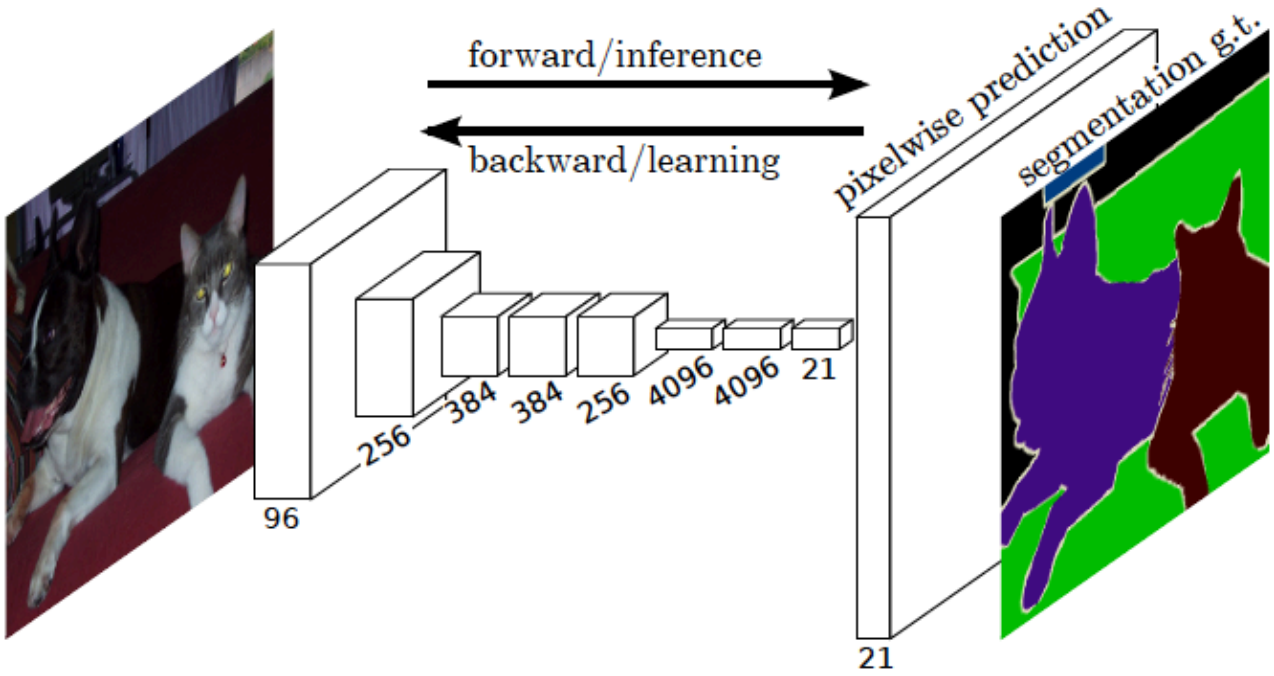
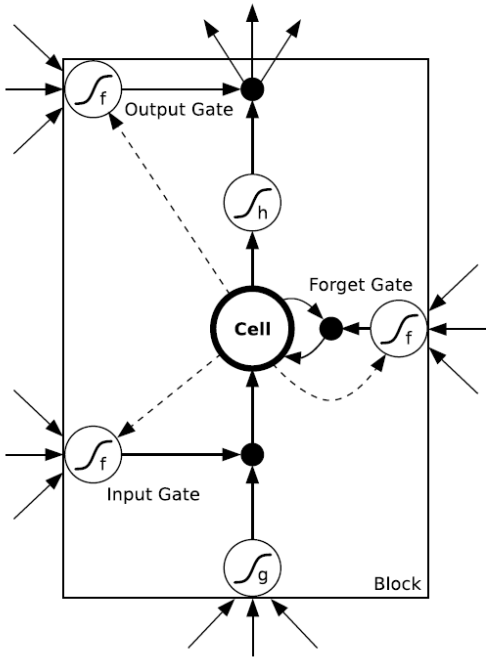


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Temporal Fusion – LSTM Networks

- LSTM 네트워크를 이용해서 encoded visual feature + car speed + angular velocity를 결합해서 다음 행동을 예측한다.

- We optionally concatenate the **past ground truth sensor information**, such as **speed and angular velocity**, with **the extracted visual representation**. With the visual and sensor states at each time step, we use an **LSTM to fuse all past and current states into a single state**, correspondina to the state s in our drivina model F**(**s, a**)** .



Learning with Privileged Information (LUPI)

1. **Motion Reflex Approach** : 기본적인 FCN-LSTM approach
2. **Privileged Training Approach** : Loss function에 Segmentation Loss를 추가해서 학습 함
3. **Mediated Perception Approach** : Segmentation을 먼저 수행하고, Segmentation 결과를 LSTM의 Input으로 넣어서 학습 함

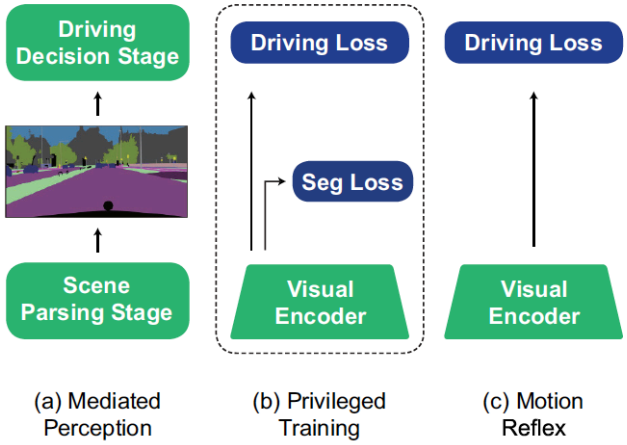


Figure 3: Comparison of learning approaches. Mediated Perception relies on semantic-class labels at the pixel level alone to drive motion prediction. The Motion Reflex method learns a representation based on raw pixels. Privileged Training learns from raw pixels but allows side-training on semantic segmentation tasks.

The Berkeley DeepDrive Video (BDDV) Dataset

- 다른 데이터셋들과의 비교

Datasets	settings	type	Approx scale	Diversity	Specific Car Make
KITTI	**city, rural area, highway**	**real**	less than 1 hour	one city, one weather condition, daytime	Yes
Cityscape	city	**real**	less than 100 hours	German cities, **multiple weather conditions**, daytime	Yes
Comma.ai	mostly highway	**real**	7.3 hours	highway, N.A. , **daytime and night**	Yes
Oxford	city	**real**	214 hours	one city (Oxford), **multiple weather conditions**, daytime	Yes
Princeton Torcs	highway	synthesis	13.5 hours	N.A.	N.A.
GTA	city, highway	synthesis	N.A.	N.A.	N.A.
BDDV(ours)	**city, rural area, highway**	**real**	**10k hours**	**multiple cities, multiple weather conditions,daytime and night**	**No**

Table 1: Comparison of our dataset with other driving datasets.

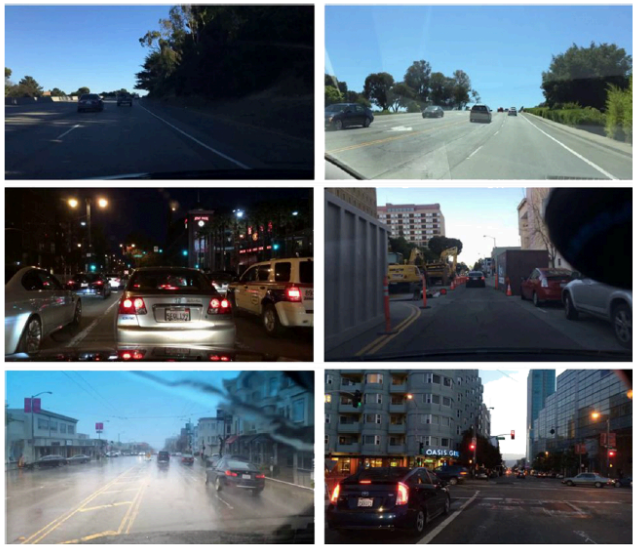


Figure 5: Sample frames from the BDDV dataset.

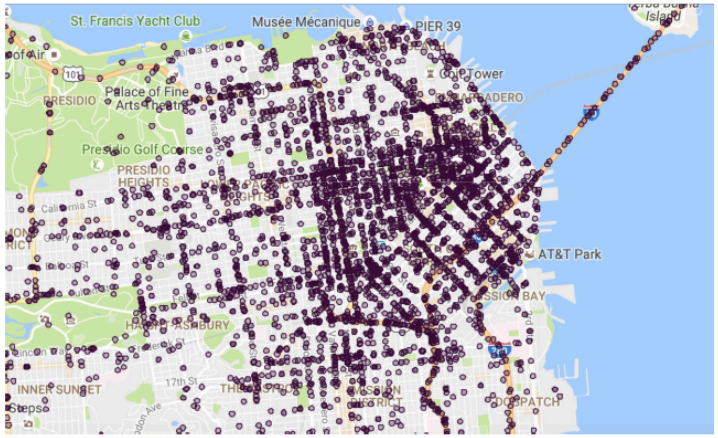


Figure 4: Example density of data distribution of BDDV in a major city. Each dot represents the starting location of a short video clip of approximately 40 seconds.

Experiment Setting

- we used a subset of the BDDV comprising **21,808 dashboard camera videos as training data**, **1,470 as validation data** and **3,561 as test data**. Each video is **approximately 40 seconds in length**.

- Since a small portion of the videos has duration just under 40 seconds, we truncate all videos to 36 seconds. We downsample frames to **640x360** and temporally downsample the video to 3Hz to avoid feeding near-duplicate frames into our model. After all such preprocessing, we have **a total of 2.9 million frames**, which is approximately **2.5 times the size of the ILSVRC2012 dataset**.

- Optimizer : Stochastic Gradient Descent (SGD) with an initial learning rate of 10^-4, momentum of 0.99

- Batch size : 2.

- Learning Rate : decayed by 0.5 whenever the training loss plateaus.

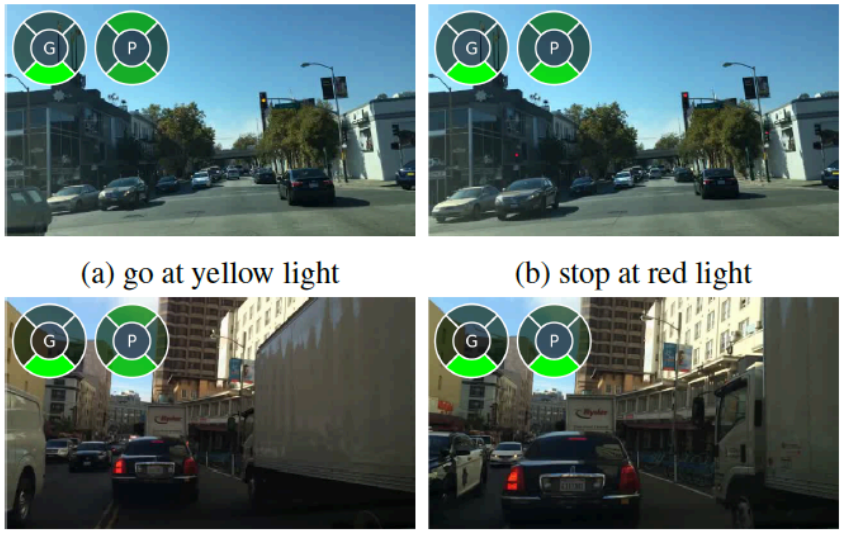
- Gradient Clipping : 10

- Number of hidden units in LSTM : 64

- Performacne Measure : predictive perplexity and accuracy, where the maximum likelihood action is taken as the prediction.

Discrete Action Driving Model

- We first consider the discrete action case, in which we define four actions: **straight, stop, left turn, right turn.** The task is defined as predicting the feasible actions in the next 1/3rd of a second.



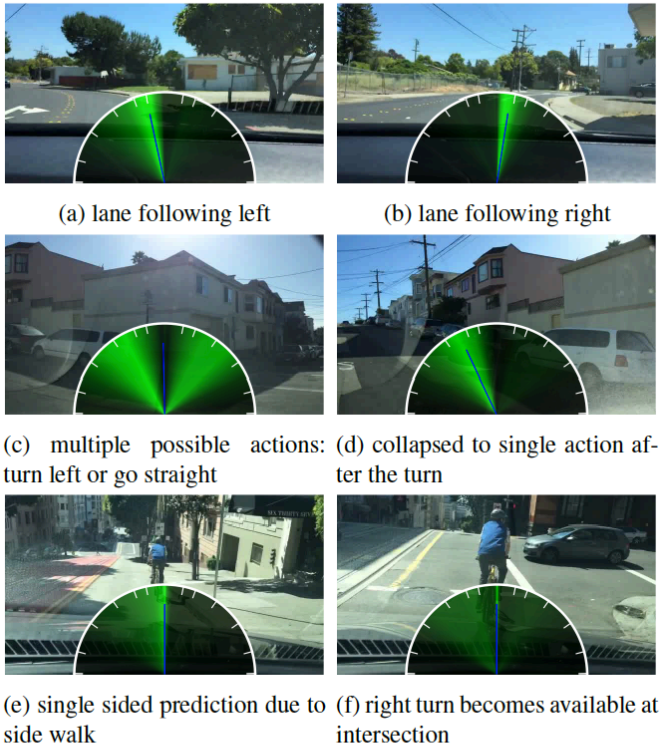
(a) go at yellow light (b) stop at red light

(c) stop & go equal weight at medium distance (d) stop when too close to vehicle ahead

Figure 6: Discrete actions predicted by our FCN-LSTM model. Each row of 2 images show how the prediction changes by time. The green bars shows the probability of doing that action at that time. The red bars are the driver's action. The four actions from top to bottom are going straight, slow or stop, turn left and turn right.

Continuous Action Driving Model

- In this section, we investigate the continuous action prediction problem, in particular, lane following. We define the lane following problem as **predicting the angular speed of the vehicle** in the future 1/3 second.



(a) lane following left

(b) lane following right

(c) multiple possible actions: turn left or go straight

(d) collapsed to single action after the turn

(e) single sided prediction due to side walk

(f) right turn becomes available at intersection

Figure 7: Continuous actions predicted by our model. The green sector with different darkness shows the probability map of going to a particular direction. The blue line shows the driver's action.

Learning with Privileged Information (LUPI)

1. **Motion Reflex Approach** : 기본적인 FCN-LSTM approach

2. **Mediated Perception Approach** : Segmentation을 먼저 수행하고, Segmentation 결과를 LSTM의 Input으로 넣어서 학습 함 -> we first compute the segmentation output of every frame in the videos using the Multi-Scale Context Aggregation approach described in [26]. We then feed the segmentation results into an LSTM and train the LSTM independently from the segmentation part, mimicking stage-by-stage training.

3. **Privileged Training Approach** : Loss function에 Segmentation Loss를 추가해서 학습 함 -> The motion prediction loss (or driving loss) and the semantic segmentation loss are weighted equally.)

method	perplexity	accuracy
Motion Reflex Approach	0.718	71.31%
Mediated Perception Approach	0.8887	61.66
Privileged Training Approach	**0.697**	**72.4%**

Table 4: Comparison of the privileged training with other methods.

Learning with Privileged Information (LUPI)

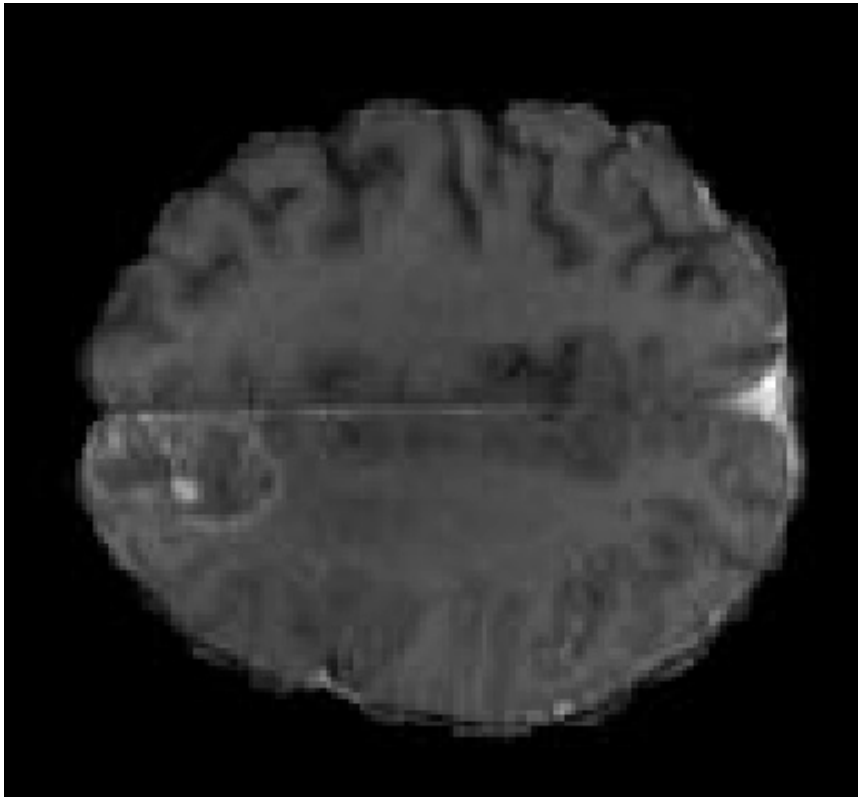
- The First column of images shows the prediction of the three methods when the **brake light of the car goes out but the vehicle has not yet started to move.** The Privileged Training approach in (c) predicts stop with high probability. The **other two methods behave more aggressively and predict going straight with high probability.**

- In the second column, there is **a red light far ahead in the intersection**.

Figure 8: We show one example result in each column from each of the three models. (a) is the Behavior Reflex Approach. (b) is the Mediated Perception Approach and (c) the Privileged Training Approach.

과제 – BRATS Dataset에 대한 FCN 구현

- https://github.com/solaris33/dl_cv_tensorflow_10weeks/tree/master/week9/FCN.tensorflow 코드를 BRATS 데이터셋에 대한 FCN Semantic Image Segmentation 모델을 구현해봅시다.



Questions & Answers

Thank You!