Vision based speeding with Transformers

Karol Szurkowski

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
kaszu19@student.sdu.dk

Abstract. After success of Transformer architecture in the NLP domain, there comes the time of applying the self-attention mechanism in vision tasks. This work points out traits of commonly used network architectures for working with images and then approaches using Vision Transformers in a regression problem, where the goal is to autonomously steer a car in an simulated environment.

1 Introduction

Advances in computer sciences and progress in the technology might worry some people - e.g. famous writer Noah Harari, whose book '21 lessons for 21st century' has been sold in millions of copies. That indicates that the discussion in the field of AI breached out of the sci-fi and computer science groups. Dreams about driver less cheap taxis are becoming more and more close to our daily life. This is what is this project about - an attempt to train driver less cars mimicking.

Tesla car company has recently revealed an approximation of neural network architecture, and had the director of artificial intelligence and Autopilot Vision at Tesla Andrej Karpathy presenting it [11]. The architecture had two big building blocks in the pipeline - residual layers and transformer part of the network. That is the origin of the idea for the project. Why not just train an obstacle-avoiding car model with getting to know the seemingly promising Transformer architecture. How promising though? Transformer architecture have been first introduced in paper "Attention is all you need" [13] published in 2017 and got a lot of attention in the field of Natural Language Processing field - mostly language translation since the paper is using this field for comparison with other methods and turned to be the state of the art model. What the authors say is that the training took a fraction of the time that other methods used. Moreover they claimed that they plan to apply the same architecture to the other neural network based solutions which require large number of inputs and outputs like images, audio and video. And so they did. The same lab - Google research on the ICLR conference in 2021 published a paper - "An image is worth 16x16 words: Transformers for image recognition at scale" [5], this time with 12 researchers as the authors, while original Transformer paper had 8. Their results turned out to be the state of the art again, beating the Residual Neural Networks [7] in image recognition tasks claiming also using a relatively cheap solution for training.

All of this preface leads to the proper introduction of the scope of this project which in short is an autonomous car driving in an simulation environment.

2 Problem and approach

To test the Vision Transformer network in the car-driving application firstly the description of a car and its environment is needed. Mathematical model of a car build in a simulation is described in subsection 2.1 and in this project is used to avoid obstacles on a simulated 4-lane highway described in subsection 2.2. Gathering of the training data is described in subsection 2.3. Following sections describe common architectures used for solving vision based tasks and further focus on the network training and results.

2.1 Car model

As a mathematical model of a car in this project, to simplify the implementation, a mathematical model of a bicycle was used. This approach eliminates part of the complexity due to the offset of the tires from the center of gravity, while retaining the same inputs and state parameters.

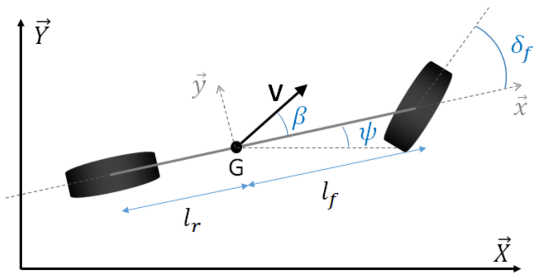


Fig. 1: Autonomous Vehicle Model [12]

Figure 1 illustrates the model in a simple set of equations describing the position in space given the velocity v_k and the heading angle β_k of the vehicle.

$$x_{k+1} = x_k + v_k cos(\beta_k) \Delta t \quad (1)$$
$$y_{k+1} = y_k + v_k sin(\beta_k) \Delta t \quad (2)$$
$$(3)$$

2.2 Simulation environment

The simulated world in which the car is moving is actually a simplified version of a 4-lane highway, where cars facing the AI agent are the obstacles, what can be observed in the figure 2. This approach mimics a crazy driver who is going along the lane of the traffic of the wrong direction and tries not to crash.

For the physics simulator the Pybullet [2] was chosen, where each step of the simulation was chosen to be updated every $\Delta t = 0.1[s]$.

4 types of visual models of a car were obtained from a free 3d object library [1] and were scaled appropriately to the dimensions of the lanes. Each car is possible to get randomly assigned one out of 7 predefined colors. To include more realism to the scene and increase the level of difficulty for network to train, the background view simulating real life landscape and grass near the road was added.

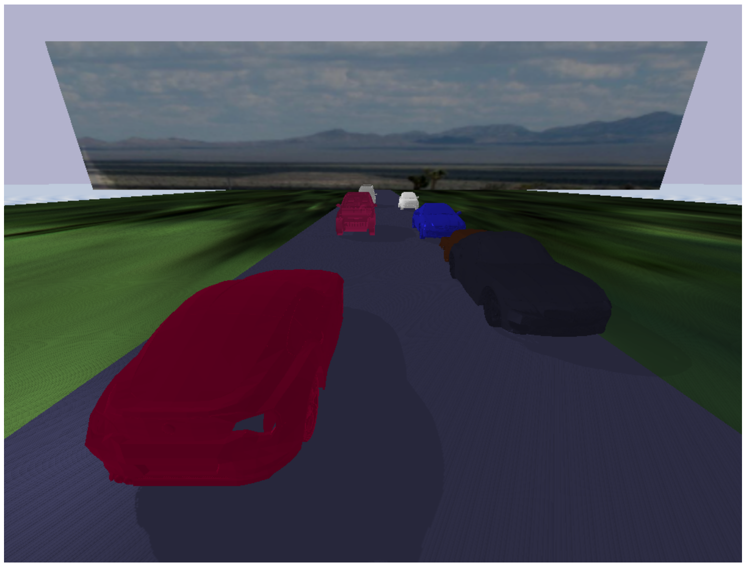


Fig. 2: Simulation environment.

For ease of programming of the movement of the objects in the environment car steered by an agent stays in place and all other cars move relative to it with the same velocity.

2.3 Training data gathering

As any other supervised machine learning methods, neural networks need a lot of training data. Data set were created by recording a human player who was able to change the turning angle of a car and its speed with pressing the keys on the keyboard. With the frequency of 10 times per second an *RGB* image with resolution of $400x400px$ from the camera mounted at the front of the car as seen in the figure 4 and the gathered targets are saved. Saving the data is terminated by an occurrence of collision of the agent's car with an obstacle. Using original 3d model collision meshes turned out to be exhaustive for detecting collision in physics engine simulation so it was decided to use simplified cube shapes as seen in image 3. Other constraint of the car's position is the boundaries of asphalt.

When an agent touches the grass, the effect is the same as detecting a collision - end of a run.

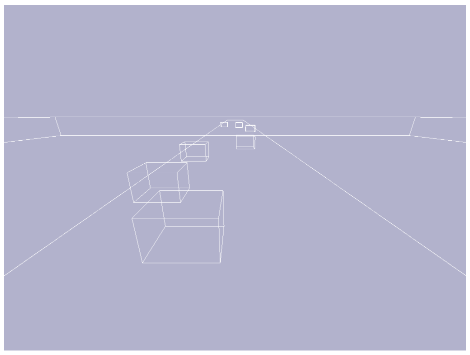


Fig. 3: Collision meshes of simulated environment.

Moreover when collision is detected last 20 saved images and corresponding training outputs were being discarded to avoid training the network with harmful to repeat examples. After each termination of a run user is asked if this data should be saved on disk serving the same purpose.

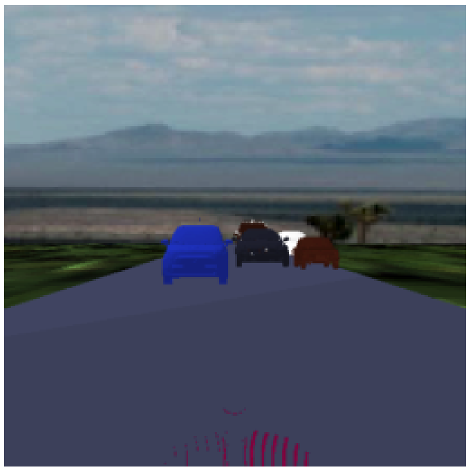


Fig. 4: Input from the camera with the saved data of $angle = -0.4[rad]$, $velocity = 10[m/s]$ indicating moving towards the unoccupied lane to the left.

While generating the training data, the main focus was put on using the appropriate turning angle, less focus on the speed. In the example above the speed was higher than normal, due to the significant distance from the agent to the nearest obstacle.

3 Different network architectures to check

3.1 Convolutional Neural Networks

Widely popular method for solving vision-based problems with neural network is convolution neural network architecture, first introduced in 1990s. Main building block and improvement from classical multi layer perceptron networks was using the convolution operation which introduced 3 important features - sparse interactions, parameter sharing and equivariant representation. CNNs were well suited to working with high dimensional data due to the reduced storage of parameters thanks to the use of kernels which provides parameter sharing [6]. Deep neural network architecture with convolutional layers can be seen on the left side of the picture 13.

3.2 Resnet approach

In 2015 Microsoft Research group came up with an idea how to successfully train deeper than before networks with hundreds and up to a thousand layers. Their framework outcompeted previous best architecture of VGG networks in classification tasks having lower complexity. To overcome the descrepancy between the training and testing error and lack of the convergence of the training with many layers the solution was introducing shortcut connections. Connections like that mean that the output values of one layer can skip one or more layers and connect to the deeper layer with its value as seen in the figure 5.

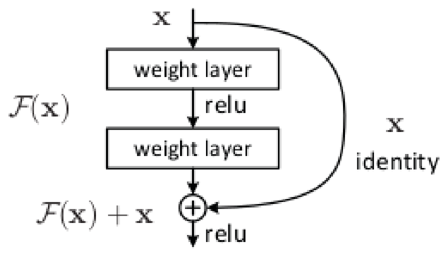

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Fig. 5: Residual learning block [7].

Applying the residual learning block in practice can be seen on the right side of the figure 13 when it's directly compared to the classical CNN.

Take home message is that the ResNet approach seems to be performing really well with a smart trick of skipping connections which doesn't introduce more learning parameters to the model. Moreover computational complexity stays nearly the same, since it introduces only an additional addition operation.

3.3 Vision Transformers

Following the succes of regular Transformer architecture similar solution was applied to the vision classification system. One of the successful papers in this field was [5] with the architecture presented in the figure 6. As an direct inspiration of the Vision Transformer code this source [8] was used.

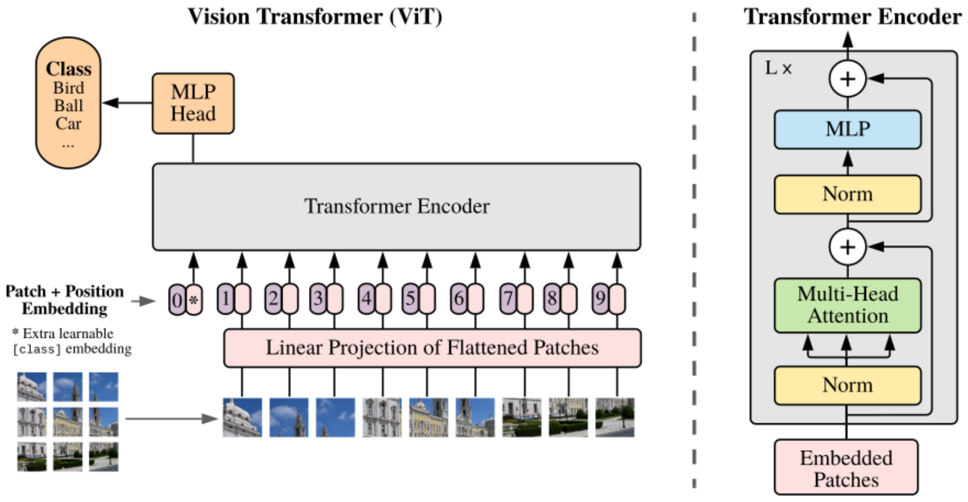


Fig. 6: Vision Transformer architecture presented in [5].

The main idea is to divide the input image into smaller images (patches) with a 16x16 pixel dimension in case of original paper. Each patch is embedded with a number of a patch from the image. All of the patches are flattened and then projected with an embedding matrix E of projection that helps the flattened image to fit the transformer input dimension. To the projected embedding there is added an extra learnabale class embedding which is trying to represent the entire image.

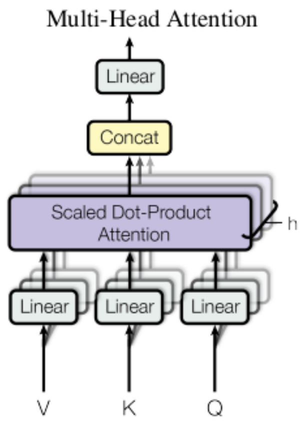
Then the embedded patches are the input of the Transformer Encoder. Firstly the values are going through the layer normalization. Basically we want to normalize the activations of neurons with respect to the mean and deviation of the activations (weighted sum) of the previous layers. This approach is batch size independent unlike the batch normalization.

Then having the input normalized the next step is calculating Multi-Head Attention. Each attention filter is calculated in a way presented in figure 7b and then concatenated to let the model to jointly attend the information from different subspaces as shown in figure 7a.

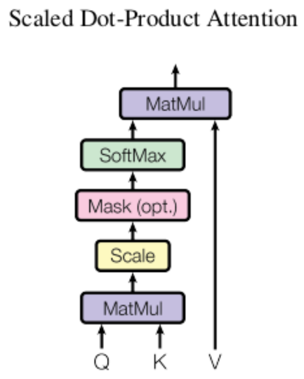
What happens in these steps is that the flattened embedded patches are layer normalized and for each of h heads of layers of multi-head attention the self attention is calculated. That means each layer normalized and flat patch is reshaped to 3 vectors with a linear layer - Value (V), Key (K) and Query (Q), which then are linearly transformed with one fully connected layer to fit. For the reshaped Q, K, V vectors the attention function is calculated which is, when no mask applied, scaled vector multiplication, where scale is dependant of the dimension of K vector d_k. Math behind the whole Multi-Head Attention process in the Transformer Encoder from figure 6 is presented below, where W^a is a vector of weights of single fully connected linear layer a.

Next following the skipping connections also used in ResNet, to the multi-head attention output (attention filter) the connection adds unnormalized flat embedded patches and then what follows is the normalization and then an MLP layer with substantial amount of hidden layers. After another residual connection, the output of transformer encoder is fed to the MLP which normally is used for classification but in this paper, the final activation function is linear and dimension of the output of the MLP was changed to fit the regression approach of the project.

$$MultiHead(Q,K,V) = Concat(head_1, ..., head_h)W^O \tag{4}$$
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{5}$$
$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}}V) \tag{6}$$



(a) Multi head attention mechanism [13].



(b) Dot-Product attention mechanism [13].

4 Network training

For training the network certain learning parameters had to be chosen. Although the network architecture was kept the same with the same following parameters.

The dimension of the image was $256x256px$, each patch of the word in the image has dimensions $8x8px$, an embedding dimension of 64 units, a depth of 1 Transformer block, 2 transformer heads, and 128 units in the hidden layer of the output MLP head.

For updating the network parameters in the backward pass of regression problem the classical L1 loss function was used.

4.1 Data augmentation

Since the simulated cars have varying colors, to increase generalization of the model some data augmentation for the image data has been added. Mostly to introduce more colors in the dataset color jitter, that randomly transforms the brightness and saturation of an image [3], was used along with random equalization of the histogram of an image [4]. Other methods that was tested is random horizontal flip.

Examples of randomly augmented or not data can be seen in the figure 8 below.

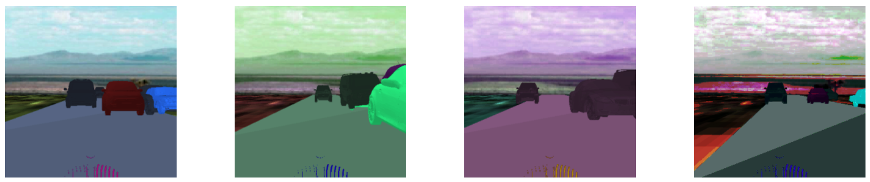


Fig. 8: Examples of input image without data augmentation and with random color jitter and histogram equalizer.

4.2 Hyperparameter tuning

For finding the best parameters a grid search in the parameter space was firstly introduced. The grid search was looking for the smallest test loss given changing learning rate of optimizer and L2 weight regularization (weight decay) [6] since in the Vision Transformer architecture there are lots of layers and it seemed to be not straight forward where to apply dropout to improve regularization [10].

In the figure below 9 we can see the which learning rate of Adam optimizer and weight decay combination gives the best results over 60 epochs of training.

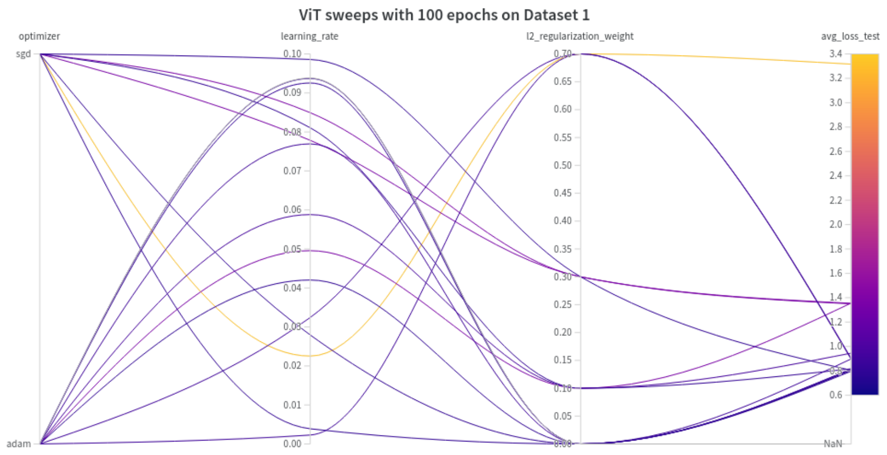


Fig. 9: Hyperparameter sweeps for Vision Transformer architecture with different optimizers and L2 weight decay. Graph done using Weights and Biases platform [9]. Each line represents parameters for one run for 100 epochs, where first column indicates optimizer, second learning rate, third weight decay and lead to the average loss of evaluation data.

5 Model evaluation

To properly test the decisive model which steers the car, the non-random obstacle course was designed 10. Neural network model's performance was measured in the time that the agent was driving before hitting and obstacle on average of 10 runs.

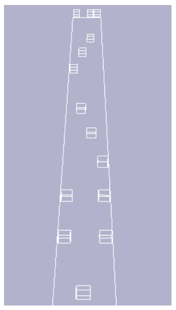


Fig. 10: View of collision meshes of obstacle course to test neural network models in practice.

5.1 Experiments and results

To validate the training of the network the experiment setup was defined. Models were trained on 3 datasets.

First dataset is short but complicated - it consists of 76 images and was covering the recorded driving from 7.6 seconds of human driving, while every car spawned on the road was in one out of 7 possible car colors.

Second dataset to make things easier for the model to learn was obtained where all of the obstacle cars were in white color and there were only 67 images meaning 6.7 seconds of recording.

Third dataset is bigger, it consists of 976 images recorded during 97.6 seconds of driving footage where all of the obstacle cars were in white color.

dataset name	NN architecutre	training epochs	training loss	evaluation loss	horizontal flip augmentation
dataset 1	ViT	700	0.02084	0.02971	yes
dataset 2	ViT	700	0.021	0.02205	no
dataset 3	ViT	700	0.01725	0.01792	no

The most successful model from the table 5.1 had its performance checked by running 10 times and recording the effective position of the AI agent. Obtained trajectories from different runs can be observed in the figure 11. Although the model was trained on the data with 7 different colors of the model, the best performance was observed best when all of the obstacles were white, thus the before mentioned figure shows the most optimistic variant with all the obstacle cars white.

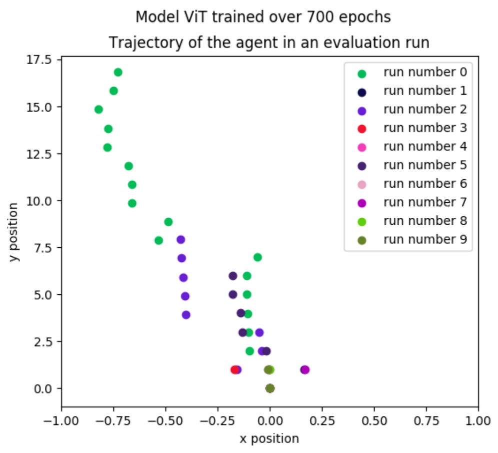


Fig. 11: Trajectory covered in 10 runs of an AI agent using Vision Transformer architecture and trained over 700 epochs of the first dataset. It can be seen that even though it is 1 trained model, and the position of the different looking obstacles is the same, the trained agent gets the correct notion to go to left bank of the road, but still collides with one of the first obstacles.

6 Analysis and Discussion

As can be seen in the table 5.1 good results were hard to obtain. This could be an effect of multiple choices which had to be taken when training the network. First of those choices is the training data - as seen in figure 11 model managed to learn how to change the angle of the steering which is a small success. Nevertheless when watching the car's behavior live in the simulation trying to avoid the obstacles still the crashes happen due to hitting an obstacle which is outside of camera's image with the side of the car. This is unfortunate and shows an importance of training data, because different models of cars used had different visual size meaning more complex problem.

Another important factor is data augmentation. The best model turned out to be the one that besides histogram equalization and color jitter as data augmentation was using also random horizontal flips. This method wasn't used during other 2 datasets training simply because of a mistake.

Other observation from the figure 12 show that it is hard to train the model properly. Experimenting with changing the L2 norm weight decay didn't seem to solve the issue as can be inferred from image representing different run experiments 9. Even though dropout was not introduced in original paper this could be an approach to introduce some noise to the training to finally achieve better generalization.

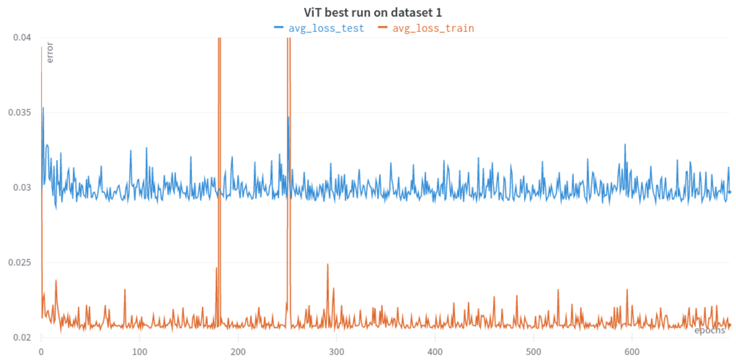


Fig. 12: ViT model trained on Dataset 1 over 700 epochs.

All these results should point towards improving the training data. Since human driver doesn't change the steering angle that often, especially in the intervals of 0.1 second that produces data that sparsely have the information other than default 0 deg when car is driving straight. In case of solving the problem as a classification - whether to change lane of the car, this issue would be named ”imbalanced classes”, where most of the data would say ”stay on your current lane”. Classification ViT examples are more common to find than regression applications. This issue could be then solved by introducing more training data with good quality.

To achieve good quality of training data, good approach would be instead of using human-input, just use an obstacle avoiding algorithm that could solve the very problem that is tried to be solved in this work, meaning avoiding immediate obstacle and choosing the best direction of avoidance based on the obstacles behind. In that case, the training data gathering could be run without human intervention.

When using the bigger dataset (number 3), which has significantly more images, even though results in the table 5.1 show improvement in minimizing loss, though the actual tests of the model in simulation doesn't show any behavior of taking a decision. This observation again points towards the argument for using training data of higher quality.

7 Conclusion

This work presents a brief summary of important properties of different neural network model architectures used for working with images - Convolutional Neural Networks, Residual Neural Networks [7] and finally Vision Transformers [5]. Furthermore this work presents descriptions of implementing a pybullet [2] environment for simulation of a car on 4-lane highway, saving the human-recorded data and an approach to train a neural network with Vision Transformer architecture to make the model predict the correct change of steering angle based only on an image from 1 camera. Surprisingly, even though the biggest training dataset achieved the least L1 loss, the best performance of the model in evaluation obstacle course was observed for the most color-varying dataset with little samples. Even though those results were best, agent was hitting one of the first obstacles, which leave some room for improvement. This improvement could be achieved by replacing a human for data generation by a smart obstacle-avoiding algorithm.

8 Appendix

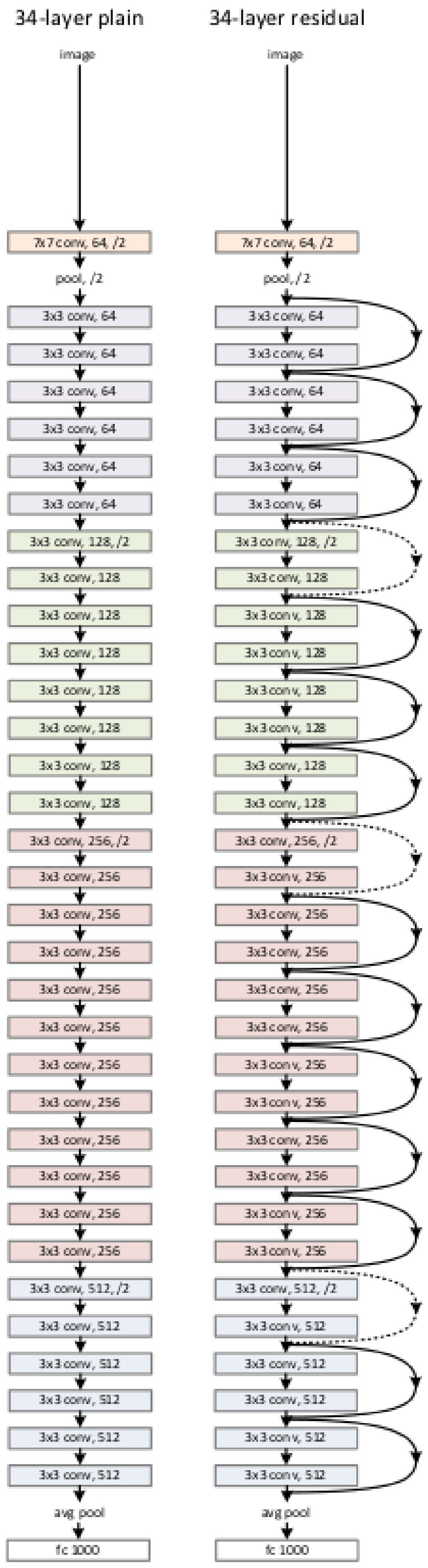


Fig. 13: Comparison of classical CNN architecture (left) with ResNet architecture

References

[1] *ameede.net*. 2021. URL: https://www.ameede.net/.

[2] *Bullet Real-Time Physics Simulation*. URL: https://pybullet.org/wordpress/.

[3] Torch Contributors. *Color Jitter*. 2017-present. URL: https://pytorch.org/vision/stable/transforms.html#torchvision.transforms.ColorJitter.

[4] Torch Contributors. *Random Equalize*. 2017-present. URL: https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#randomequalize.

[5] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[8] lucidrains. *Vision Transformer - Pytorch*. URL: https://github.com/lucidrains/vit-pytorch.

[9] Lavnaya Shukla. *Intro to Pytorch with WB*. 2019. URL: https://wandb.ai/site/articles/intro-to-pytorch-with-wandb.

[10] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[11] Tesla. *Tesla AI Day - Augutst 2021*. URL: https://www.youtube.com/watch?v=j0z4FweCy4M.

[12] *The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?* URL: https://www.researchgate.net/publication/318810853_The_kinematic_bicycle_model_A_consistent_model_for_planning_feasible_trajectories_for_autonomous_vehicles (visited on 11/18/2020).

[13] Ashish Vaswani et al. "Attention is All You Need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.