Programming an Autonomous Driving Car

Mustafa Omar[1]

[1] *Physics and Astronomy, University of Nottingham*

(Dated: June 7, 2022)

In this work we investigate the performance of different deep learning methods in the task of self driving, aided by regularising methods and synthetic data augmentation. We obtain results for models/methods including vanilla Multi layer preceptors, vanilla Convolutional neural networks as well as results using Transfer Learning with MobileNetV3Small and EfficentNetV2 B0 and B1, with performances ranking in the respective order from worse to best. We obtain a best performing theoretical model using the EfficentNetV2B0 model, obtaining a Kaggle MSE score of 0.01161. Furthermore on the practical testing, we used the MobvileNetV3Small and obtained a poorly performing model, only scoring $\frac{9}{35}$ points in the live testing.

INTRODUCTION

Since the development of the first convolutional neural network (CNN) known as Alex net [1], many previously unattainable applications of machine became possible, this is because CNNs opened the way for machine learning systems to interact effectively with image data. One of the most promising applications of CNNs is the self driving car, simply, given that a human driver primarily relies on their vision system to drive, it should in theory be possible to develop a machine learning system that utilises a vision based model architecture to tackle the problem. Indeed the current state of the art systems for self driving cars all utilise CNNs as their primary image processing technology [2], and the most advanced self driving system appears to be the Tesla autopilot system [3], which is continuously being updated an improved to drive with greater generality and safety.

In this work we/(I) will be investigating the effectiveness of vanilla neural networks as well as CNNs in the task of self driving. Using the Sunfounder Picar [4], a set of $14.8k$ images is provided through a Kaggle competition, split such that $13.8k$ images are for training (image + label) and $1.02k$ images are for testing (image only). The images are of dimension (240,320,3) in PNG format, the labels for each image in the training set states the speed and steering angle of the car at that particular time and this is provided within a CSV file.

Furthermore after training and testing the model on unseen data, we additionally test the model on a live testing environment, where the model is loaded onto the Sunfounder Picar and is evaluated on a set of challenges. There is a total of 12 challenges which could be categorised into three types: Lane centering and stable driving, where the car is expected to drive at the center of the lane without swerving too much; Critical risk assessment, this is simply to stop or go depending on what object the car is seeing, objects include: green light sign, red light sign, pedestrian, a tree and a box; And navigation, where the car is expected to respond to right and left turning signs (arrows) such that the car changes directions at the appropriate time.

We will begin by first carrying out exploratory data analysis of the provided data set, this is followed by data cleaning and prepossessing, this is done in order to be able to use the data as input for the machine learning models. Additionally, data augmentation is implemented, this has been shown to regularise machine learning models and classification accuracy [5]. We will then investigate different architecture and observe their effectiveness on the task without utilising data augmentation, this is done in order to pick out the most effective architecture before retraining it completely to produce the best prediction results possible, this is done due to time restrictions.

The methods/models used are; vanilla neural networks, vanilla convolutional neural networks and transfer learning with pretrained TensorFlow models [6], namely MobilenetsV3 models [7] and EfficentNetV2 models [8] are used.

METHODOLOGY

To begin this work, the data is downloaded for cleaning, it is observed that a small number of images appear be missing while the labels for said images were present, to solve this issue the labels were removed. Furthermore some images appeared to have been in RGBA (4 channels) format where an extra alpha channel was present, this was corrected by simply removing the extra channel, reducing the total number of channels for those images form 4 to 3 channels. Finally, we observe that some speeds had anomalous values and since this was only a small number of images ($N = 1$) from the dataset, the issue was resolved by simply giving those image appropriate labels manually.

The CSV file provided is a list of $13.8k$ rows, each with columns; ImageId, Car Steering angle and Car speed.

$$angle_{norm} = \frac{angle - 50}{130 - 50} \tag{1}$$

$$speed_{norm} = \frac{speed - 0}{35 - 0} \tag{2}$$

Equations (1, 2), show how the labels for angle and speed are transformed respectively, this is later used to revert the predicted values back to the correct scales used by the Picar. Additionally, Fig. 1 shows a histograms of the angles and speed after the data is cleaned.

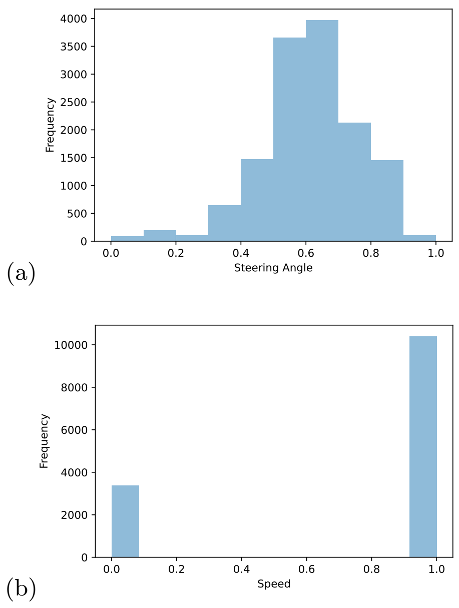


FIG. 1: Showing (a) the steering angle and (b) the speed distributions of the dataset. where for (a), a value of 0 indicates a left turn, 0.5 to be driving straight and 1 a right turn. Furthermore, a speed of 0 suggest a stationary car and speed of 1 suggesting a moving car.

We can observe that the angle distribution is approximately normally distributed with slight skew towards right steering angles, this suggests that for the majority of the time the car is driving forwards and slightly towards the right, while rarely turning in the extremes. Additionally, the speed is skewed towards values of 1, suggesting that the data shows the car driving forwards for the majority of the time. Those imbalances in data representations will need to be accounted for and balanced in order to obtain a representative model that will be able to generalise well both for the Kaggle competition and the live testing, this is done by collecting more data from the car while turning and by using data augmentation methods.

Furthermore, before any preprocessing step is taken, the data is simply split into a representative training and testing sets this is done so that we can obtain a reliable generalisation error after training the models on data that has never been seen by the model before. On the other hand, in order for the machine learning model to be able to take in the training data, the data has to be in a format most suitable for the model being used at that specific time. For models such as the MLP, the input has to be a one dimensional vector, therefore the image is flattened before being given as an input to the model, and since the rest of the models being tested are CNNs, the image is simply given as is or resized to $(224, 224, 3)$ to be used with MobileNetV3 and EfficentNetV2.

The image pixels are given as discrete values $\in [0, 255]$, however, in order for most models to converge quicker an image normalisation step is used where the discrete pixel values are rescaled such that

$$pixel_{scaled} = \frac{pixel_{original}}{255} . \quad (3)$$

Additionally, soft labelling was applied on the speed label only, this is a regularising technique that introduces noise into the given label's distribution, this has been shown to allow the model to learn more effectively [9]. Label softening was applied such that for each speed label of value 0, a sample from a uniform distribution $X \sim U(0.15, 0.35)$ is obtained and used as the label, similarly, for a speed labels of values 1, a sample from a uniform distribution $X \sim U(0.15, 0.35)$ is used in a similar way.

Furthermore, visual distortion data augmentation (primary augmentation) was used in order to regularise the model [10], this includes the following transformation: random contrast, image rotation and zoom-resize (see Fig. 2), the augmentation library Albuminations was used to complete this task [11].

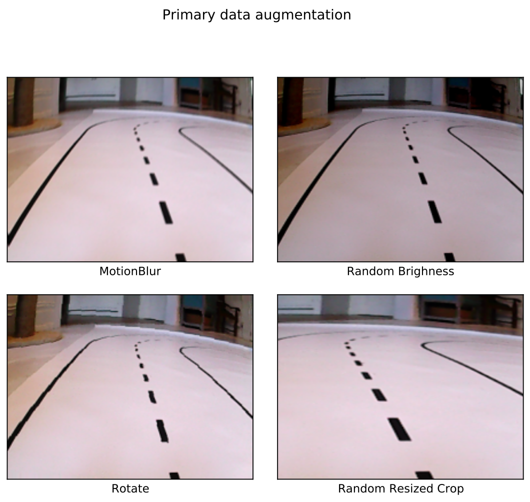


FIG. 2: Showing examples of data augmentation, where the same image undergoes different predefined transformations.

Additionally, we introduce problem specific data augmentation (secondary augmentation), this augmentation aims to expand the dataset, such that the model is able to learn general rules that would help it make more accurate predictions for the challenges previously detailed, and this includes: adding cropped images of pedestrians and objects to each training example, followed by a change to the speed label such that it has a value of zero, effectively teaching the model to stop whenever an object is placed in front of it; we also add cropped images of right and left turn signal, followed by random label changes $X \sim U(0.15, 0.35)$ for a left turn and $X \sim U(0.65, 0.85)$ for a right turn; also red and green stop light signs were cropped and added into the image and the speed label was either changed to zero or left unchanged respectively for each scenario. Fig. 4 shows examples of augmentations on a reference image, the augmented images in Fig. 4 are again augmented using the primary augmentation, to regularise the model even further.

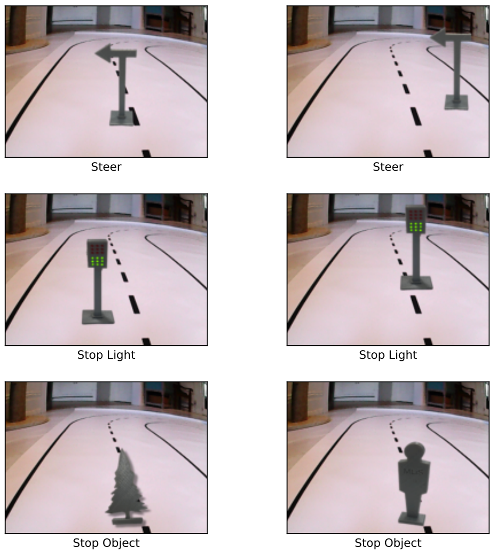


FIG. 4: Showing an example of secondary augmentation carried out on the reference image, with the first row showing a steering sign, second row showing a stop light, and the third row showing an object placed in front of the car's view.

one potential issue with this technique is the fact that some images may already contain the object used in the augmented image, this may lead the model to be confused and to generalise poorly, therefore, to work around this, the augmentation was done for specific label value ranges for the angle label only: for example the turning signs are added only if the initial steering angle is relatively straight $angle \in [0.4 : 0.6]$, on the other hand this doesn't apply to the speed label as the nature of the problem allows for this. However, for stop signs (red and green light), the augmentation was done for images where the speed is 1 for both cases, this is done to avoid scenarios where the red light for stopping is potentially already present in images with speed label $= 0$.

Finally, in this work, a few models are trained and tested on the non-augmented data in order to figure out which model is the best performing model. A simple feed forward neural network (see Appendix for shape), was the first model that was trained and tested, this is used to allow for a comparison with image based techniques (e.g CNNs). The MLP uses ReLU for activation layers, Adam as its optimiser, a learning rate of 0.001, for 15 epochs, with average pooling and a dropout rate of 0.2 after each layer.

After testing a simple MLP, two vanilla convolutional neural networks; deep and shallow CNNs (see Appendix for shapes), were trained and tested. We expected that those models will perform a lot better than MLPs due to the convolutional operation that draws relation from neighbouring pixels. Furthermore, both CNNs use ReLU for activation layers, Adam as an optimiser, a learning rate of 0.001, for 10 epochs, with average pooling and batch normalisation layers.

Finally, Transfer learning, which is a method based on the principle that problems in a common domain share similarities, and that knowledge can be transferred from one similar application to another is used. Keras provides models that were trained on the Imagenet dataset [12]. Transfer learning allows us to make use of those models, we are simply removing the top layers and utilising the trained layers/parameters to solve an image based (regression) problem; we take the pretrained transfer learning model and remove the inference layer, this is replaced with a two neuron dense layer with ReLU activation, where each neuron represents a label (angle and speed).

we then train the 2 neuron dense layer while the rest of the model layers are frozen and after, we being to fine tune the model by unfreezing $\approx 2\%$ of the layers and training the model again but with a much lower learning rate. We initially start by using MobileNetV3 as our first pretrained model [7]. Additionally we train different EfficentNetV2 models, namely B0, and B1 each of which are provided through the Keras API, All transfer learning models use ReLU for activation on the inference layers, Adam as an optimiser, a learning rate of 0.001, for 10-20 epochs, with average pooling and no regularisation layers (batch normalisation or dropout).

Furthermore after training all of the aforementioned models, a single model namely, B0, was picked for further training, the model was trained using similar hyperparameters, however it was trained for a longer period of time multiple times, with different types of augmentation: initially with no augmentation, followed by primary augmentation, secondary augmentation and back again for a final training period with the primary augmentation, this was done at different layers such that the deepest layers saw the most augmented data and shallower layers saw the raw dataset at the end. After fine tuning, the model was further tuned using selected secondary augmentation scenarios for 1-2 epochs, before removing soft labelling and again training the head layer for 1 epoch.

[Note: See appendix for learning curves]

RESULTS AND DISCUSSION

Table I shows the model size in terms of the number of trainable parameters, the inference time on the car for models that were chosen as candidates for live testing, and the initial Kaggle score measured is MSE. We found that the MLP is the worst performing model and this is as expected, because MLPs aren't capable of easily drawing relationships between neighboring pixels due the input structure (vector). Furthermore, the two vanilla CNNs perform better than the MLP with the shallow CNN performing slightly better than the deep CNN, this is due to the vanilla architecture, where a large kernel was used (10×10), and no up sampling along with max pooling, all of which contributed to a smaller number of parameters, however a lot of information from the image is destroyed due to this, in the meanwhile a smaller kernel size (5×5) and fewer layers were used for the shallow CNN, this allows the model to draw some correlation while not losing much information, and results in the model performing slightly better than the deep CNN.

Furthermore we find that the MobileNetV3 CNN architecture performs better at the task than a shallow CNN after fine tuning a small number of layers $(\approx 2\%)$ for a short period of time. Similarly, we find that EfficentNetV2 models are general more accurate with B0 being the best model, and therefore was chosen as the primary model for training. After training and tuning the B0 model multiple times as described previously in the methodology, a final Kaggle score of 0.01161 was obtained.

For the live testing the model we chose was the MobileNetV3Small, despite the EfficentNetV2B0 model performing much better on the Kaggle competition, we ran into the problem of a slow inference time, where the B0 model took $\approx 1.25s$ to make a single prediction on the car, this is too long and results in the car driving off of the track before a new prediction is made. MobileNetV3Small is $\approx 5\times$ smaller than Efficient net B0, and it's initial Kaggle score performance is much worse than B0, however it's inference time is $\approx 11\times$ quicker (Table I), this allows the model some leeway with inference accuracy, as it could make multiple error and still recover an accurate course on the track through a few correct predictions, this is a trade off between model complexity and accuracy. On the live testing however, the MobileNet model performed poorly scoring only $\frac{9}{35}$ points, this is because the MobileNet model was poorly trained due to time restrictions. This was also due to a failure to strike a balance between model complexity and real world performance when different models were initially chosen, the B0 model was a very good theoretical model, however practically speaking, due to the poor inference time, it is a very poor model, on the other hand, MobileNetV3small is a very simple model, making it a slightly worse theoretical model but practically it is a lot better.

TABLE I: Model Information and results

Model	N.Params (N)	Inference Time (s)	First Kaggle score (MSE)
MLP	$\approx 500e03$	-	0.1304
CNN deep	$\approx 300e03$	-	0.0834
CNN Shallow	$\approx 740e03$	-	0.0630
MobileNetV3Small	$\approx 1e06$	≈ 0.11	0.0541
EfficentNetV2B0	$\approx 5e+06$	≈ 1.25	0.04

CONCLUSION

In this work we successfully trained and tested multiple machine learning models to tackle a real world problem, effective techniques were used to create a very accurate theoretical model, however when tested in the real world, there was a failure due to complexity, and again, a very practical but theoretically poor model was used, that again sinned at striking a good balance between theory and real world application.

To advance this work a few things could be done, firstly and most importantly, is to find a model slightly more complicated than the MobileNetV3Small and simpler than EfficentNetV2B0, such a model should be perform well theoretically and then practically when deployed on the car, secondly, we could add a scaling property to the secondary augmentation, such that the model can see the same object at different sizes, this can then be further advance by logic to teach the model to only respond to objects that are close by e.g "To only stop when the red light sign is large".

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12 (Curran Associates Inc., Red Hook, NY, USA, 2012) pp. 1097–1105.

[2] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz, L. de Paula Veronese, T. Oliveira-Santos, and A. F. De Souza, Self-driving cars: A survey, Expert Systems with Applications **165**, 113816 (2021).

[3] M. Dikmen and C. Burns, Trust in autonomous vehicles: The case of Tesla Autopilot and Summon, in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (2017) pp. 1093–1098.

[4] Sunfounder focuses on steam education with open-source robots (2019).

[5] L. Perez and J. Wang, The Effectiveness of Data Augmentation in Image Classification using Deep Learning (2017).

[6] Martín~Abadi, Ashish~Agarwal, Paul~Barham, Eugene~Brevdo, Zhifeng~Chen, Craig~Citro, Greg~S.~Corrado, Andy~Davis, Jeffrey~Dean, Matthieu~Devin, Sanjay~Ghemawat, Ian~Goodfellow, Andrew~Harp, Geoffrey~Irving, Michael~Isard, Y. Jia, Rafal~Jozefowicz, Lukasz~Kaiser, Manju-

nath~Kudlur, Josh~Levenberg, Dandelion~Mané, Rajat~Monga, Sherry~Moore, Derek~Murray, Chris~Olah, Mike~Schuster, Jonathon~Shlens, Benoit~Steiner, Ilya~Sutskever, Kunal~Talwar, Paul~Tucker, Vincent~Vanhoucke, Vijay~Vasudevan, Fernanda~Viégas, Oriol~Vinyals, Pete~Warden, Martin~Wattenberg, Martin~Wicke, Yuan~Yu, and Xiaoqiang~Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015).

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (2017).

[8] M. Tan and Q. V. Le, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks 10.48550/ARXIV.1905.11946 (2019).

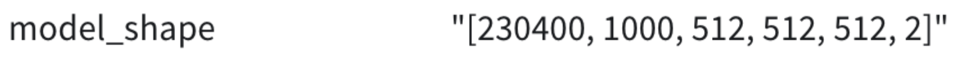
[9] R. Müller, S. Kornblith, and G. Hinton, When Does Label Smoothing Help? (2019).

[10] A. Herná ndez-García and P. König, Further advantages of data augmentation on convolutional neural networks, in *Artificial Neural Networks and Machine Learning – ICANN 2018* (Springer International Publishing, 2018) pp. 95–103.

[11] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, Albumentations: Fast and Flexible Image Augmentations, Information **11**, 10.3390/info11020125 (2020).

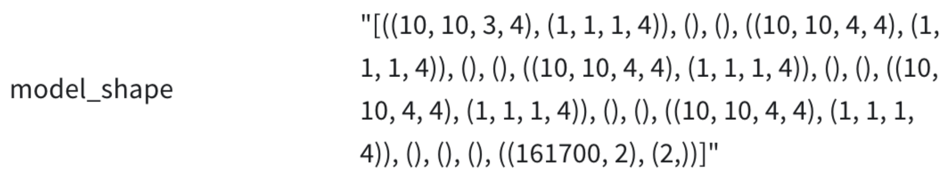
[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in *2009 IEEE conference on computer vision and pattern recognition* (Ieee, 2009) pp. 248–255.

APPENDIX



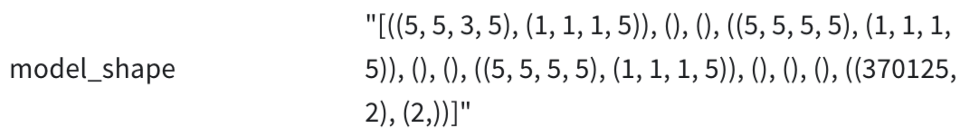
model_shape "[230400, 1000, 512, 512, 512, 2]"

FIG. 5: Showing the raw shape of the MLP model



model_shape "[((10, 10, 3, 4), (1, 1, 1, 4)), (), (), ((10, 10, 4, 4), (1, 1, 1, 4)), (), (), ((10, 10, 4, 4), (1, 1, 1, 4)), (), (), ((10, 10, 4, 4), (1, 1, 1, 4)), (), (), ((10, 10, 4, 4), (1, 1, 1, 4)), (), (), (), ((161700, 2), (2,))]"

FIG. 6: Showing the raw shape of the Deep CNN model



model_shape "[((5, 5, 3, 5), (1, 1, 1, 5)), (), (), ((5, 5, 5, 5), (1, 1, 1, 5)), (), (), ((5, 5, 5, 5), (1, 1, 1, 5)), (), (), (), ((370125, 2), (2,))]"

FIG. 7: Showing the raw shape of the Shallow CNN model

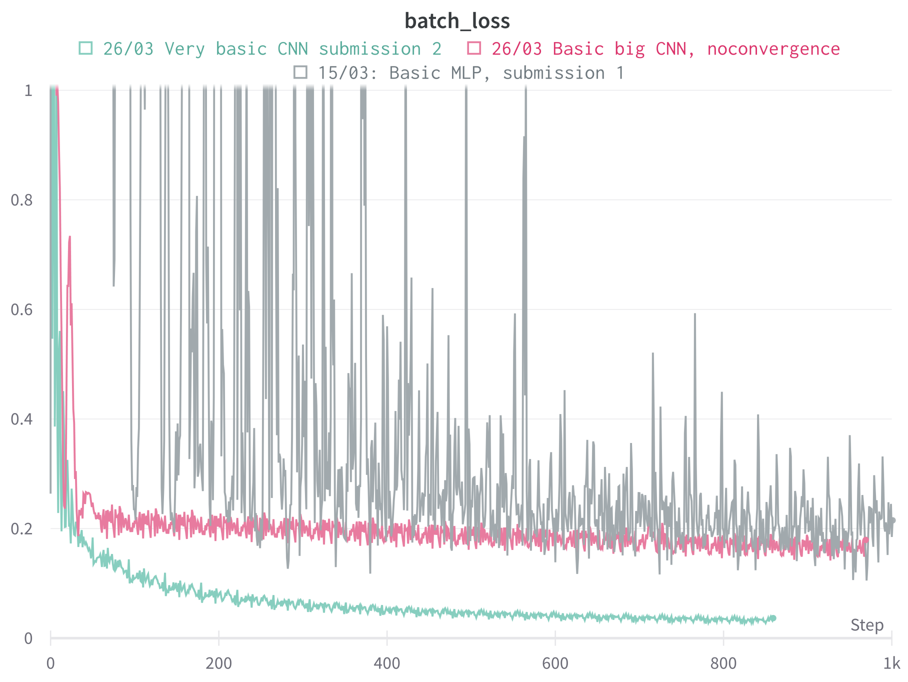


FIG. 8: Showing The learning curves for vanilla models, Grey: MLP, Pink: Large CNN, Green: Small CNN.

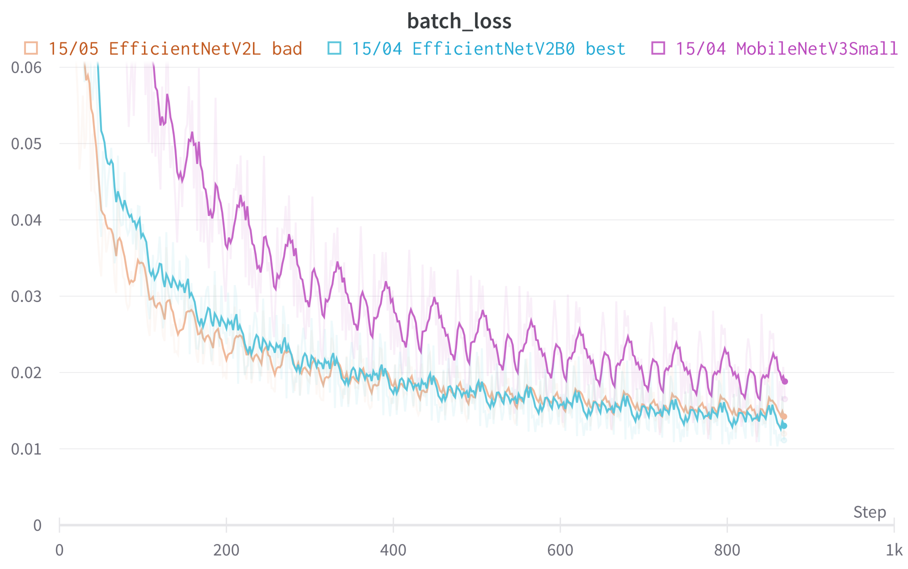


FIG. 9: Showing the learning curves for different transfer learning models, Orange: EfficentNetV2L, Blue: EfficentNetV2B0, and Purple: MobileNetV3Small.

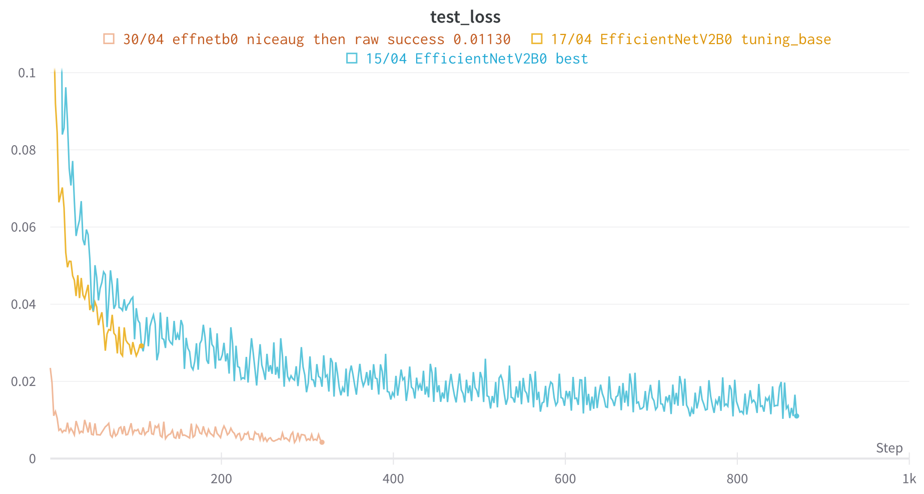


FIG. 10: Showing the learning curve of transfer learning model. Blue: full training no fine tuning, Yellow: Fine tuning base Orange: Opening base (yellow model) layers for fine tuning.