# Exploring Convolutional Neural Network Applications to Self-Driving Cars

Arvin Corotana
*School of Physics and Astronomy*
*University of Nottingham*
*ppxac8@nottingham.ac.uk*

Deepak Kumar
*School of Physics and Astronomy*
*University of Nottingham*
*ppxdc3@nottingham.ac.uk*

Hamza Elshafie
*School of Physics and Astronomy*
*University of Nottingham*
*ppxhe1@nottingham.ac.uk*

***Abstract*—With the rise of self-driving cars, enhancing performance to gain public trust is critical. These autonomous technologies have used many artificial intelligence architectures, with a prevalent one being convolutional neural networks which were explored. Different architectures were selected for training on a dataset from tracks, where performance was measured from angle and speed prediction. Manipulations of the data like cropping and balancing the data were used in an attempt to improve track performance along with measures to decrease inference time which resulted in a small and efficient model, MobileNet, being selected as the best performer. However, a number of future improvements can be made through techniques like a different approach to balancing, training metrics and the possibility of developing separate models. Overall, this process demonstrated the viability of CNNs in such tasks and highlighted the importance of understanding the training data.**

## 1. Introduction

Recently, there has been a rise of autonomy with a lot of effort going into making self-driving cars that are able to handle any situation that may arise on the roads [1]. This has been through the developments made in deep learning, with convolutional neural networks (CNNs) contributing massively. [2]. With this in mind, the aim of this research was to explore the performance of different CNNs on driving situations. The architectures of key CNNs must first be understood to choose and analyse the performance of them in an autonomous situation.

### 1.1. Convolutional Neural Networks

CNNs are widely used in computer vision and within the self-driving car area through the ability to perform well on tasks such as object detection, lane detection, segmentation and traffic light/sign detection. Due to their ability to handle vast amount of visual data and extract relevant features, they are ideal for visual decision-making.

CNNs have key components which give them a recognisable architecture. Most CNNs will have images as the inputs feeding into a convolutional layer which applies a set of filters to the image to extract features, which are generally low-level at the start and higher-level further along the network as the receptive field of the filters proportionally increases. The receptive field may increase due to pooling layers which reduce the dimensionality of feature maps while retaining important information, reducing computational cost and reducing overfitting. The most common forms are max pooling and average pooling. Fully connected layers have each neuron apply an activation function upon the input vector and associated weights meaning every input influences every output of the layer [3]. Activation functions can introduce non-linearity into the network, with ReLU being popular due to the gradient only being 0 or 1 so that gradients shouldn't exponentially decay like in tanh and sigmoid functions. However, there may be a problem with dying neurons as a gradient of 0 may lead to no update through back-propagation [4]. Through developments in this field, prompted by the ImageNet Large Scale Visual Recognition Challenge [5], CNN performance was improved greatly with new concepts and architectures.

**1.1.1. AlexNet.** AlexNet consists of 5 convolutional layers, each followed by ReLU activation functions. It incorporates max pooling, with kernel size 3x3 and stride of 2, and local response normalisation (LRN) after the first 2 convolutional layers. This was one of the first times ReLU was used and the first time LRN had been used. The network also has 2 fully connected layers with ReLU and dropout feeding into a fully connected output with a 1000-way softmax function [6]. This structure can be seen in figure 1. AlexNet has been used in regression tasks

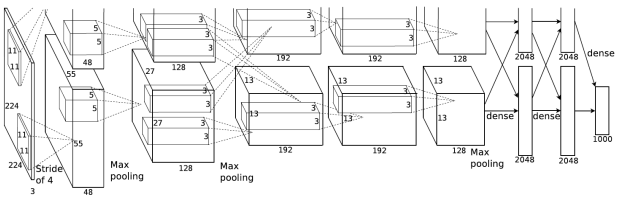


Figure 1. AlexNet Architecture with the 2 GPU structure used for training, specified in [6]

**1.1.2. VGGNet Series.** There are many VGG architectures, but there are general commonalities between them. They

use uniform convolutional layers with 3x3 kernel size with stride of 1 to simplify the architecture. Maxpooling layers follow some of these layers using a 2x2 kernel and stride of 2 instead of 3 in AlexNet meaning there will be more features lost aiding in preventing overfitting. There are generally 3 fully connected layers with one being the output with ReLU being used on 2 of them and dropout [7]. An example structure from the VGGNet series seen in figure 2. A member of the VGG series, VGG-16, has been used in a similar task of predicting steering angle for an autonomous vehicle [8] giving validity to the architecture being suitable for such applications.

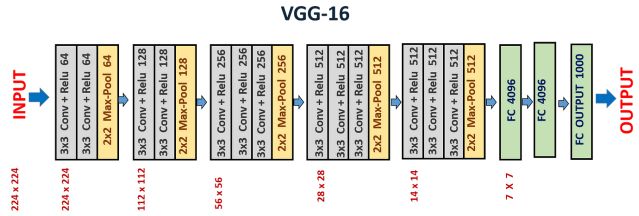


Figure 2. Architecture of VGG-16 showcasing the general structure and repetitive nature of VGGNets. Produced by [9]

**1.1.3. InceptionNet.** This architecture is known for having multiple convolutional filters of varying sizes operating at the same level of the network allowing it to capture spatial hierarchies in data at various scales simultaneously, called inception modules. An example of this in figure 3. This helps InceptionNet be more efficient and adaptable than VGG. The use of pointwise (1x1) kernel convolution before larger convolutions reduces dimensionality, further improving efficiency. The network combines max pooling and average pooling instead of using one exclusively. Fully connected layers are removed, which help reduce parameters and computational load with potential to reduce overfitting [10].

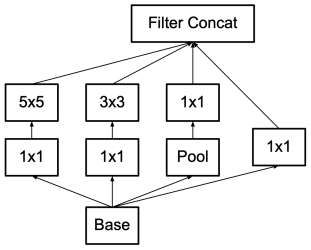


Figure 3. The original inception module from InceptionNetv1 from [10]

**1.1.4. ResNet.** ResNet utilises residual blocks, a basic version in figure 4, with skip connections which help gradients flow more effectively and addresses the vanishing gradient problem. This is due to gradients flowing back to a shallower layer without shrinking exponentially, layer by layer. In common variants like 50 and 101, there are 3 residual blocks and uses bottleneck designs (1x1, 3x3, 1x1 convolutions) to

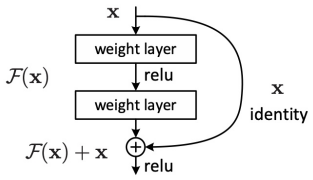


Figure 4. The general structure of a residual block with a skip connection from [11]

reduce computation within them. With improved gradient flow more layers are used, creating a deeper network [11]. This has also been used in a relevant context of predicting steering wheel angle [12].

**1.1.5. MobileNet.** MobileNet is an efficient CNN architecture for mobile and embedded vision applications. It utilises depthwise separable convolutions, which split computation into the 2 steps of depthwise convolution applying a single convolutional filter per input channel and pointwise convolution to create a linear combination of the output of the computation before. This is visualised in figure 5. The use of width and resolution multiplier allows for efficiency and flexibility, through the ability to adjust the number of filters in each layer and the input image resolution. This is furthered by the capping of the ReLU function to 6 [13], lowering precision and computation.

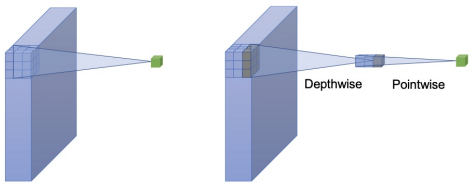


Figure 5. Standard convolution (left) compared to the depthwise separable convolution used in MobileNet from [14]

## 2. Methodology

All processes involving the use of code can be found at our GitHub repository [15].

### 2.1. PiCar Set-up

To implement models developed and test performance, a PiCar was used. The body of the car was the SunFounder PiCar-V kit V2 equipped with a Raspberry Pi 4 , a wide-angle USB webcam and a Coral Edge Tensor Processing Unit (TPU). Before any track testing the PiCar was calibrated using a checkered sheet and wheel adjustment.

### 2.2. Dataset

The dataset provided [16] consisted of around 13,000 images taken by the PiCar and camera going round the tracks and performing scenarios that performance will be tested upon. Along with images, the dataset included a csv file with the corresponding labels of angle and speed.

## 2.3. Preprocessing

Before a model was selected, the data preprocessing was decided upon to use each models specific processing process which normalises the pixels to improve training convergence as having a more consistent range of values contributes to this [17]. It should also help prevent vanishing gradients by not having large values and saturating activation functions, mainly tanh and sigmoid, leading to very small gradients in the loss function and updates to parameters that are either too small to matter or nonexistent. However, ReLU activation functions should help avoid vanishing gradients also. This should also stop features dominating the focus purely because it had a larger number [18] compared to others, as the range is between 0 and 255 in images used.

All images were scaled to meet the input size selected in the preprocessing of training along with the inference process. The training dataset was split into training (70%), validation (20%) and test (10%) datasets for all models. For training, angles and speeds were normalised to be between 0 and 1, using (eq1.1), and either 0 or 1, by (eq1.2), respectively. This was reversed for predictions on the PiCar. Due to all models producing linear outputs for speed, the inference code rounds outputs to 0 or 1.

$$\frac{angle - 50}{80} \quad (1) \frac{speed}{35} \quad (2) \tag{1}$$

## 2.4. Model Development

All models had the output layer removed and replaced with one or many layers more suitable for regression, which always had a linear activation function for the new output layer. Any fully connected layers that were not output layers used the ReLU activation function. Mean squared error was used as the loss metric for all models as angle and speed was viewed as regression throughout. Model parameters yielding the best validation score in training were always chosen.

**2.4.1. AlexNet.** To start model development the AlexNet architecture was used as it was one of the first impactful CNNs for computer vision therefore offering natural progression if performance is not satisfactory. With no AlexNet implementation within TensorFlow, the architecture of AlexNet was recreated using keras model layers.

The optimisation algorithm chosen for training was Adaptive Moment Estimation (ADAM), due to its' ability to converge fast and not get stuck in local minima or maxima [19].

**2.4.2. Transfer Learning.** Models used ImageNet parameters for improved generalization [20], with the base layers trainable to adapt to the task.

**2.4.3. Optimisation and Loss Function.** All models from this point underwent Bayesian optimisation using Hyperopt [21] and Optuna [22] for the hyper-parameters like number of neurons, weight decay in l2 regularisation, and dropout rate in the additional layers. Optimisation was also performed on learning rate and momentum in the SGD with momentum loss function.

The loss function of choice from this point was also decided, this was stochastic gradient descent with momentum due its' potential ability to generalise better than ADAM, especially in computer vision [23], due to less adaptations to the loss function landscape and a better optimisation ability compared to stochastic gradient descent due to the velocity term introduction [24].

**2.4.4. VGG16.** After AlexNet, VGG16 was experimented with due to its' potential to learn more complex patterns with a deeper structure. VGG16 was used as a base model. The following layers were added on top of this: 2xConv2D, MaxPooling2D, 2xConv2D, GlobalAveragePooling2D, Dropout, Dense, Dropout, Dense and a Dense output layer.

**2.4.5. VGG16+19 Stacked Ensemble.** As an experiment VGG19 was stacked upon VGG16 sequentially in an ensemble to grow the VGG structure to potentially gain accuracy through a deeper network. The same layers were added on as the VGG16 method.

**2.4.6. InceptionNet.** Due to the increased efficiency of Inception and increased performance on ImageNet, InceptionV3 was selected as the next model to adapt and train for the task at hand. The same layers were added on as VGG16 and the stacked ensemble model.

**2.4.7. TensorFlow Lite.** For ResNet and MobileNet, any concern of inference time the conversion to a Tensorflow Lite model (TFLite) was utilised, where quantization was used to reduce the parameters accuracy after training. This was from a 32-bit float to a selected 16-bit float (half-precision) for a balanced between extreme reduction to 8-bit integer for inference time and retained accuracy [25].

**2.4.8. ResNet.** With ResNet-50 performing even better than Inception on ImageNet, this was trained in hope of better losses. The following layers were added onto the model: GlobalAveragePooling2D, Dropout, Dense layer, and a Dense output layer.

**2.4.9. MobileNet.** MobileNetV3 large was chosen for its edge device suitability and competitive accuracy, with similar layers as ResNet, with inference time the focus. The model had the same layers added as ResNet development and default input size.

Post-dataset exploration revealed a right turn bias, visualised in figure 6, leading to oversampling of underrepresented angles and speeds. Cropping the bottom 50% of images aimed to prevent early turning by focusing on the immediate environment, helping lane keeping.

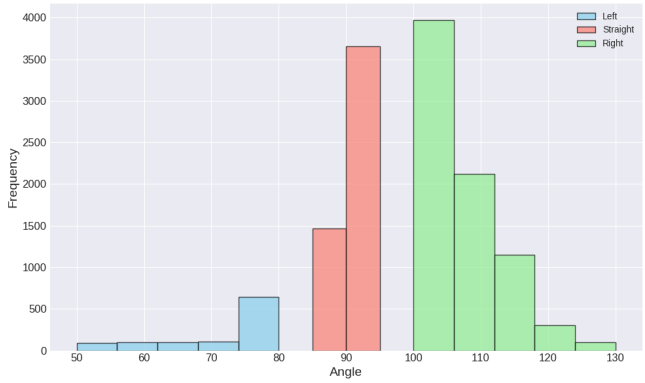


Figure 6. Distribution of angles within the original dataset before balancing

## 3. Results and Discussion

Model performance was evaluated by losses in training, Kaggle upload losses, and track performance on a t-junction, oval, and figure-eight, shown in figure 7. The t-junction straight-line test assessed basic model ability.

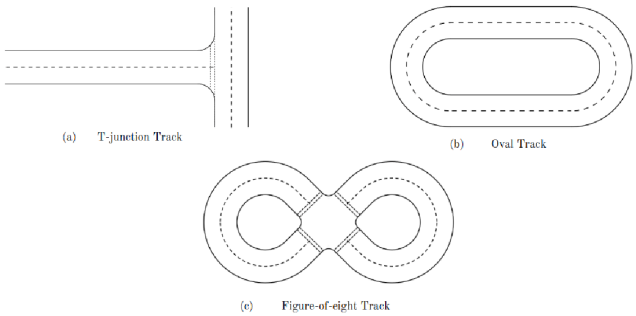


Figure 7. Tracks used to measure performance.

As model complexity increased, the general performance in terms of training, validation, test and Kaggle losses improved, represented in table 1. However, this changes when model development reached MobileNet, which could be due to its efficiency in architecture design. On the track, all models performed poorly in the straight line test as they were veering off the track within a second, which did not improve even with the better losses. Inference times were always high, which may have been causing a problem as minor imperfections in prediction would not be corrected in time of it leaving the track. MobileNet improved inference times greatly with times hovering around 70 to 150ms for all models with TFLite and around 250ms without it. This was a huge improvement compared to ResNet with TFLite operating with an inference time of 2500ms.

TABLE 1. LOSSES OF EACH MODEL USED IN DEVELOPMENT AND KAGGLE PUBLIC AND PRIVATE SUBMISSIONS

| | Training | Validation | Test | Public | Private |
|---|---|---|---|---|---|
| AlexNet | 0.17000 | 0.18000 | 0.20000 | 0.29021 | 0.28943 |
| VGG16 | 0.01300 | 0.00101 | 0.00110 | 0.01924 | 0.01852 |
| VGG16+19 | 0.01220 | 0.00987 | 0.00860 | 0.02063 | 0.01437 |
| Inception | 0.00960 | 0.00820 | 0.01070 | 0.01512 | 0.01796 |
| ResNet | 0.00320 | 0.00751 | 0.00793 | 0.01423 | 0.01307 |
| MobileNet | 0.00650 | 0.00510 | 0.00580 | 0.01317 | 0.01265 |

A problem within development was the confusion caused by validation loss being better than the training loss after the implementation balancing, hinting at a data leakage. This occurred because balancing was done before splitting the data, making the validation score misleading. Fixing this issue by splitting data before, as shown in figures 8 to 9, made validation loss a better indicator of generalization.

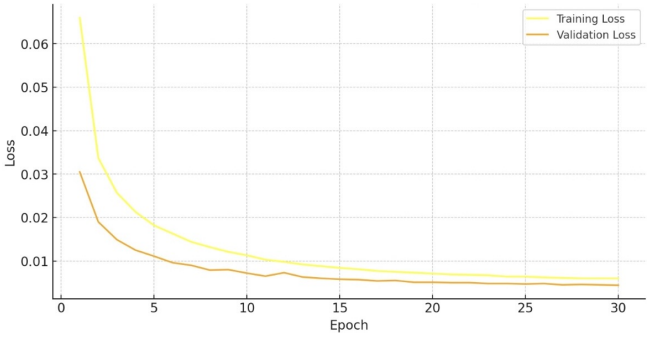


Figure 8. Training of MobileNet with balancing before data spliting with cropping

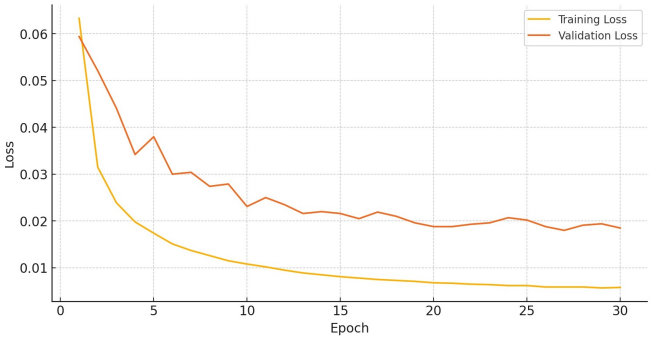


Figure 9. Training of MobileNet with balancing after data splitting with cropping

Darkening augmentation was experimented with in MobileNet development, yielding no improvement leading to it being removed for time-saving. The best model was the MobileNet model with cropping, balancing but with data leakage as it did not decrease performance, only our perception of generalisation from validation loss. The best model had a Kaggle private loss of 0.01265 with an inference time of 70ms. Cropping most likely reduced the information in the image that was causing a problem with potential early turning with a turn in the top of the image. This change led to the performing better on the straights of the tracks.

Overall, the results show that the model development, going from less complex to more complex models, led to the increase in accuracy but not better performance on the track, due to inference times. Inference times were improved once the mindset was shifted to speed of predictions alongside accuracy, which was aided by the introduction of TFLite models and MobileNet. This enforced the need for predictions needing to meet a threshold of being around 150ms or below with accuracy being as higher as possible with good generalisation ability.

### 3.1. Future Improvements

The generalisation ability of most models, especially those tested on an external test dataset, was suboptimal . In general, this can be due model complexity and overfitting but this explanation does not offer much insight as differing levels of model complexities were experimented with. A more viable explanation could be that the models could have all benefited from the transfer learning being utilised slightly more through a smaller learning rate. This would lead to the parameters learned through training would diverge less from the learned parameters from a large general dataset of ImageNet, potentially leading to a better ability to generalise.

Further improvements can be made, especially in terms of situations that may be under-represented. Even though the data was balanced in terms of angles and speed, it does not mean the data represents each situations with suitable proportions as driving on a straight lane and driving straight at an intersection can created the same angle. Due to this, balancing angles and speeds alone may not be enough to increase the number of under-represented scenario meaning balancing scenarios, through oversampling or data collection is also required to not bias the predictions due to being specialised to a set of situations over others.

Comprehensive dataset exploration and preprocessing should also precede model development to ensure fair comparisons in further developments. This would have allowed for a more accurate comparison of model performance.

The chosen model failed to continue with a pedestrian on the side. Improvements could be made by developing the model with the output of speed being a classification task instead of regression, with a sigmoid output pushing predictions towards stopping (0) and driving (1) instead of a linear output, where more predictions may be closer to 0.5. With less predictions near 0.5 there may be less chance of middling predictions leading to classifications fluctuating between stopping and going, which could be seen even when stopping for a pedestrian in the road. This would require the training to be altered with a classification metric of binary cross-entropy for speed classification.

Speed and angle predictions could be completely separated in training through the development of 2 models, a speed prediction classification-specific model and an angle prediction regression-specific model. This may improve performance due to the fact that the labels are independent of each other, showcased by figure 10. If the labels are independent, training separate models will allow for the learning of different features that are more specific for each task. To combat the inherent increased inference time and computation from the processing of 2 model, simpler models could be used for both or just one task as features learned are now more specific to each task so potentially less features need to be learned to perform well on the task.

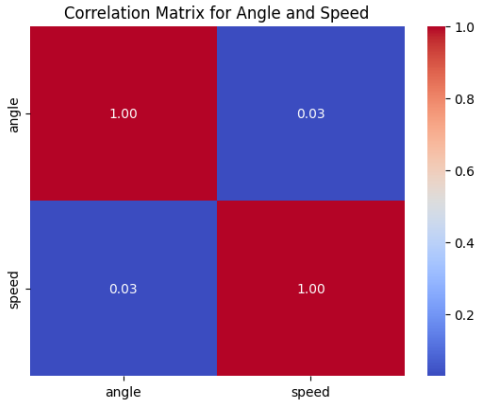


Figure 10. Correlation Matrix of Angle and Speed from the original dataset

With any of the suggested improvements, an explainable AI metric could be used to learn about what the CNN is viewing as important. Shapley Additive Explanations (SHAP) could be used in an attempt to show contribution of features to the output [26]. This would help the researchers understand the models focus along with potentially building public trust in these technologies.

## 4. Conclusion

Overall, the need for lower inference time along with accuracy and generalisation is highly important. This led to the selection of MobileNet with a TFLite model conversion. There are also several improvements possible to improve accuracy, generalisation or inference time including smaller learning rates, different and sooner dataset manipulations, different training metrics for speed, potential ensemble methods of models with separate outputs and grey-scale. With these potential improvements, here could be a better generalisation ability and performance on the track with more examples of under-represented scenarios.

## Acknowledgment

The author would like to thank the following people. Dr Maggie Lieu for creating example models and inference codes, creating the project structure, and explaining the use of the car for recording data and deploying models.

Dr Adam Moss for creating a GitHub repository containing template code for the autopilot software on the PiCar with all files required to run the models and examples on different model setup and inference codes.

Dr Steven Bamford for contributing to the Github repository with code for the deque method.

Dr Simon Dye for contributing to the judging of the performance of the PiCar on the track along with Dr Adam Moss and Dr Maggie Lieu.

## References


[1] McKinsey & Company. The autonomous vehicle industry moving forward. 2023. Accessed: 2024-05-15.

[2] Augmented Startups. Convolutional neural networks (cnn) in self-driving cars. 2023. Accessed: 2024-05-15.

[3] Aakanksha Shrestha, Umapada Pal, and Akshay Asthana. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *IEEE Access*, 7:135420–135447, 2019.

[4] Stamatis Mastromichalakis. ALReLU: A different approach on leaky ReLU activation function to improve neural networks performance. *arXiv preprint arXiv:2012.07564*, 2020. Version 2, last revised 29 Apr 2021.

[5] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[8] Xin Song, Huili Cao, and Haitao You. Steering wheel rotation angle prediction based on vgg-16 and data augmentation. In *2022 4th International Conference on Control and Robotics (ICCR)*, pages 1–5, 2022.

[9] Siddhesh Bhobe. Vgg net architecture explained, 2023. Accessed: 2024-05-15.

[10] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[12] Mariusz Bojarski et al. End-to-end learning of driving models from large-scale video datasets. *arXiv preprint arXiv:1604.07316*, 2016.

[13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[14] Yunhui Guo, Yandong Li, Liqiang Wang, and Tajana Rosing. Depthwise convolution is all you need for learning multiple visual domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8368–8375, 2019.

[15] Machine Learning in Science PiCar GitHub Repository. https://github.com/HamzaElshafie/PiCar-MLiS.

[16] Machine Learning in Science 2024 Kaggle Dataset. https://www.kaggle.com/competitions/machine-learning-in-science-ii-2024/data, 2024. Accessed: 2024-05-15.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[18] Zhengxin Shao, Zhimin Zhu, Yue Li, and Minghui Yan. Normalization and batch normalization. *Applied Soft Computing*, 82:105524, 2019.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[20] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NIPS 2014)*, volume 27, pages 3320–3328, 2014.

[21] James Bergstra, Daniel Yamins, and David D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, pages 115–123, 2013.

[22] Takuya Akiba, Shuji Sano, Takeru Yanase, Toshihiko Ohta, and Manabu Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2019)*, pages 2623–2631, 2019.

[23] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems (NIPS 2017)*, volume 31, 2017.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[25] Paulius Micikevicius, Sharan Narang, Jonah Alben, Greg Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Harris Mindzak. Mixed precision training. In *Proceedings of the 6th International Conference on Learning Representations (ICLR 2018)*, 2018.

[26] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, volume 30, pages 4765–4774, 2017.