UNIVERSITY OF THE WITWATERSRAND

School of Computer Science and Applied Mathematics
Faculty of Science
COMS4036 PROJECT REPORT

Snake Species Identification Challenge

Angus Mackenzie (1106817)
Nathan Michlo (1386161)

Course Lecturer: Dr. Richard Klein

November 2019, Johannesburg

Snake Species Identification Challenge

Angus Mackenzie (1106817)
Nathan Michlo (1386161)

I. Introduction

The *Snake Species Identification Challenge* or *SSIC* [1] is a dataset containing 82601 images of snakes with varying backgrounds, crops and viewpoints. The goal of SSIC is to classify each snake contained in each image into one of 45 classes or snake species.

The goal of this report is to investigate the application of various computer vision approaches to this image classification task for the SSIC challenge. Additionally, the report investigates why these methods perform differently.

II. Methodology

We investigated different models to discover a model that excelled at the challenge. Initially, we trained a ResNet-18 [2]. We then created a simple network architecture and used a genetic algorithm to tune the hyperparameters of the network. After that, we experimented with a MobileNet implementation [3] and finally we used EfficientNet [4].

A. Dataset Analysis

Before we could begin training the models we needed to understand the dataset we have been given.

The dataset contains 82601 images, of these images it was found that 184 were corrupt and unusable for training. These images were removed from the dataset leaving us with a total of 82417 valid images.

Of the valid images, 1036 (514 unique images) were duplicates according to md5 hashes. Of the 1036 duplicates, 24 pairs (or 48 images) had conflicting labels. In total after only keeping unique duplicates and removing all conflicts, we were left with 81871 samples.

However, the valid dataset is unbalanced, with class sizes varying from 507 to 10977, with a mean class size of 1819. 93% of classes have less than 5000 samples, 71% of classes have less than 2000 samples, and 33% of classes have less than 1000 samples. This imbalanced dataset presents a problem for training and various techniques can help.

B. Hardware Used

We trained our models on the Nvidia RTX 2060 and Nvidia GTX 1060 GPUs. Hence the models we trained had to fit on a limiting 6 gigabytes of memory.

C. Preprocessing

To add rotational invariance to our model, every photo was rotated by a uniformly random value in the range -90°to 90°then the data was normalised according to the pixel values of ImageNet [5].

D. Pre-trained Models

When training we began with pre-trained models on ImageNet. Pre-trained models reduce the time taken for a network to train. This is because the weights of the model are not randomly initialised, but rather attained training on a different yet correlated problem. This is known as transfer learning, as learned weights for completing one problem can improve learning in another domain [6].

E. Genetic Algorithm

To attain a better understanding of the influence of hyperparameters on CNNs we implemented our genetic algorithm that attempted to tune the hyperparameters of a simplistic CNN architecture. The hyperparameters we tuned were the activation function (choosing between relu, elu, and swish) the dropout probability (from 0 to 0.2), the kernel sizes (3,5 or 7) and the number of filters applied (from 8 up to 64).

Initially, we created a population of 50 individuals with randomly selected parameters and then trained them for 10 epochs each. After all 50 individuals were trained, the 25 individuals with the top accuracy were kept and the bottom 25 individuals were killed. Those 25 individuals were then repopulated using a crossover of parameters from two 'parent' individuals from the top 25, gaining a mixture of the parent's parameters. During this stage, there was also a 31% chance of mutation, where a child had the possibility of attaining random parameters instead of their parents' parameters.

This encourages a survival of the *fittest* scenario, where the models that achieve the best accuracy over the dataset have their genes passed on to future generations.

F. ResNet

The key idea for *ResNets* or Residual Networks are the addition of skip connections or *residual blocks*, which skip one or more succeeding layers. This novel idea allows for the training of extremely deep neural networks which can have hundreds of layers where before such networks were problematic to train. [2] More layers can help the network learn better higher-order representations of your data.

Due to memory limitations, we explore ResNet-18 which accordingly has 18 layers.

Both our final approaches being MobileNet and EfficientNet build upon these ideas with the use of skip connections.

G. MobileNet

MobileNets are efficient models designed with embedded and mobile devices in mind. They use depthwise separable convolutions to create lightweight neural networks [3].

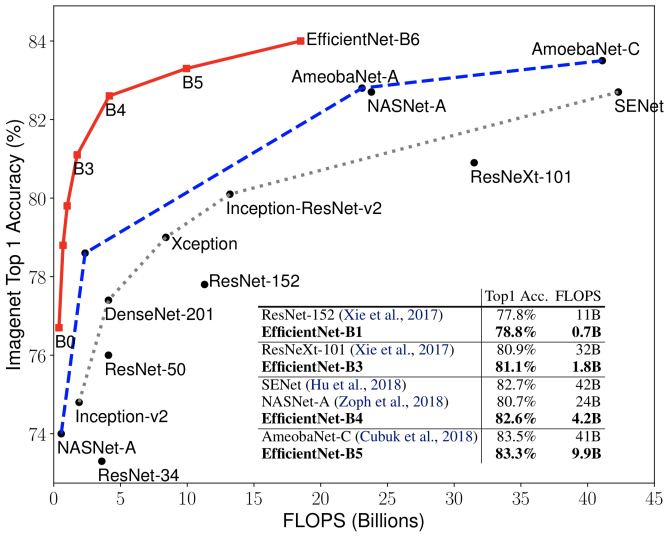
Depth wise separable convolutions first apply a filter to each channel in the previous layer and stack the output followed by the application of a 1x1 filter. This is in comparison to standard convolutions which have weights for all channels in the previous layer.

Depth wise separable convolutions significantly reduce the number of parameters in neural networks compared to standard convolutions, with only a minor loss in accuracy for comparable networks.

This network was chosen for its computational efficiency which is an important aspect to consider when developing real-world applications, an appropriate sub-goal for SSIC.

H. Efficient-Net

EfficientNet is a novel CNN implementation that introduces a method to scale all dimensions of a neural network (depth/width/resolution) to achieve better accuracy on tasks while being smaller and faster than many other implementations [4]. This is highlighted in Figure 2, which contains a breakdown of the number of Floating Point Operations (FLOPS) in Billions, compared to the accuracy achieved on the ImageNet dataset [5]. As shown, the EfficientNet-B0 outperforms the ResNet model in terms of accuracy gained, while using fewer FLOPS.



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**78.8%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.1%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.6%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2018)	83.5%	41B
EfficientNet-B5	**83.3%**	**9.9B**

Fig. 1. FLOPS vs ImageNet Accuracy. Figure from Tan and Le [4].

We used the B0, B1, B2, and B3 EfficientNet models in our training, taking considerably less time to train than other approaches, while achieving far better accuracy. Due to memory limitations, we could not effectively train EfficientNet models B4 and above.

I. Model Optimisation

Various techniques exist for model optimisation. For the majority of our experiments we used the *rmsprop* optimiser [7]. However an alternative we considered was the Adam optimiser [8], but familiarity lead us not to use this method.

a) Loss Function: We chose to use *label smoothing* cross-entropy loss over regular cross-entropy loss as it can significantly improve generalisation and model calibration. Label smoothing does this by encouraging representations in the final layer to group in tight equally distant clusters, rather than close together. [9]

The only downside to label smoothing is that it can affect the representations that the models learn, resulting in a loss of information in the logits about similarities between instances of different classes. [9] This may have affected our results in III-D on page 4.

b) One Cycle Policy: All training was conducted using the 1Cycle Policy, which takes the training through multiple learning rate phases, a warm-up phase where the learning rate is progressively increased to a maximum, cool-down phase where the learning rate is decreased to a minimum and an annihilation phase where we decrease the learning rate even further. The higher learning rates act as additional regularization, while the final lower learning rates help the model settle into local minima. Performance is improved over fixed learning rates. [10]

c) Future Work: Interestingly there has been a recent extension to the Adam optimizer called *Rectified Adam* or *RAdam* that provides an automated, dynamic adjustment to the adaptive learning rate. This technique arose after a study on the effects of variance and momentum when training. [11] The most notable result of RAdam is its relative invariant nature to incorrect choices for the learning rate, which can help considerably.

Furthermore, *Lookahead* is a new technique that can work in conjunction with any optimiser. [12] The approach keeps two sets of weights which are averaged at regular intervals but are updated at different rates, one set faster and the other set more slowly. Lookahead improves learning stability and lowers variance with minimal overhead.

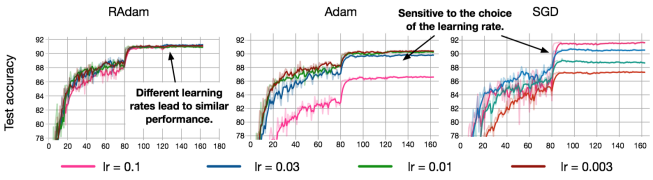


Fig. 2. Future work: Comparison of learning rates for the RAdam Optimizer. Figure from Liu et al. [11].

The combination of RAdam with Lookahead is called *Ranger* [13]. We applied this to optimizing one of the MobileNet models. Interestingly we used the same learning rate for rmsprop and ranger, the model optimised with rmsprop got to 62% accuracy while that of Ranger got to 66%. RAdam is described as a drop-in replacement for Adam that has no negative drawbacks.

Future work should consider using RAdam or even Ranger instead of rmsprop, and is expected to improve results in all cases.

J. Metrics

Several different metrics were used to get a holistic understanding of the model, and its performance.

a) Accuracy: Classification *Accuracy* is the fraction correct of predictions. Although it should be noted that measuring accuracy on an unbalanced dataset such as SSIC is not a good metric.

$$\text{accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

The *balanced accuracy* metric is more suitable for unbalanced data, normalising the prediction counts by the number of positive and negative samples.

The *top k accuracy* makes a minor modification and denotes whether the correct class was contained within the k most likely predictions.

b) Precision and Recall: Precision is the number of true positives overall positive classifications. *Recall* is the number of true positives over all samples that should have been classified as positive.

$$\text{precision} = \frac{tp}{tp + fp} \text{recall} = \frac{tp}{tp + fn}$$

c) F_1 Score: The F_1 Score, is a measure of a test's accuracy, defined as the harmonic mean of the precision and recall. An F_1 score of 1 indicates perfect precision and recall while F_1 towards 0 is undesirable. F_1 is given by the equation:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

d) Cohen's Kappa Coefficient: Cohen's Kappa κ is a measure of the agreement between two different classifiers, each of which classifies items into the same mutually exclusive classes. Cohen's Kappa has the additional advantage of being able to handle unbalanced data, similar to the SSIC. This metric is a good measure of how well our models agree with the targets.

III. Experiments

A. Genetic Algorithm

The genetic algorithm was time-intensive to train and yielded poor results. However, during the construction of the algorithm insight into the inner-workings of a CNN was gained. After training for 10 iterations, models using the swish activation function, a dropout probability of 0.1, kernel size of 3 and 16 filters performed the best - however, with an average accuracy for the top 5 individuals of 30% we deemed this line of query too cumbersome and costly.

While genetic algorithms may produce excellent results if given enough computational resources, proven hand-tuned models are more appropriate for this task.

B. EfficientNet

We used the Pytorch implementation of EfficientNet [14]. The B0 and B1 models used batch sizes of 32, and full precision 32bit floating-point operations. The B2 and B3 models both used batch sizes of 16, with the B2 model using full precision and the B3 using half-precision floating-point operations. These modifications were required due to memory limitations.

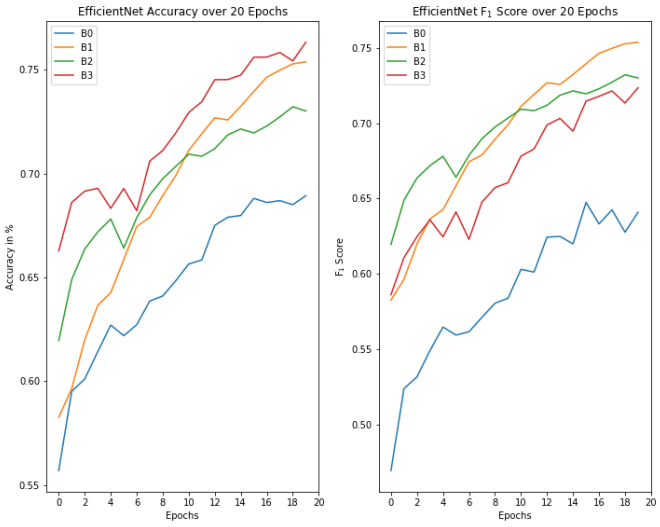


Fig. 3. Accuracy and F_1 *Score* of the B0, B1, B2 and B3 EfficientNet models over 20 epochs.

Figure 3 shows that scaling the size of EfficientNet is only advantageous to a certain extent. B0 performs the worst, but it is also the smallest of the models. Surprisingly, B1 seems to perform better than B2 and B3 in recall, this may be due to over-fitting present in the other models or greater sensitivity to hyperparameters during training.

The EfficientNet B0 model had an average epoch time of 11 minutes, B1's was 23 minutes, B2's was 20 minutes, while B3 ran for roughly 38 minutes. Interestingly, the F_1 score indicates that the B1 and B2 do better than their counterparts - and seeing as they train in roughly half the time of the B3 - that is a considerable achievement. Although, it does seem that after running for another 5 epochs B2 does dip under B3. On the whole, the accuracy results of our training are in line with the EfficientNet paper [4].

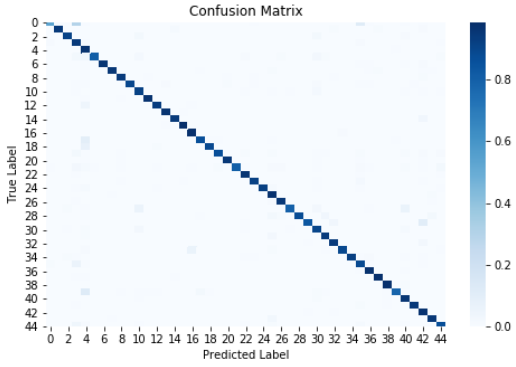


Fig. 4. Confusion Matrix created using our trained B3 model, the x-axis represents the predicted class label for inputs, and the y-axis represents the true label.

We used the B3 model from above to create a confusion matrix (see figure 4) of the predicted vs true labels, the model

predicts the correct target the vast majority of the time, but what is interesting is that the noise outside of the diagonal is mostly uniform except for a few outliers along the 4th class (zero-indexed). This 4th class corresponds to the snake species with the largest number of sample images, almost double that of the next largest class, it is thus not surprising that there are more outliers along this column.

In summary, EfficientNet achieves remarkable results, while also allowing for a large degree of freedom when it comes to computation time and complexity.

C. Alternative Models

The remaining models described in II on page 1 achieved notable results although still below the performance of Efficient-Net in 5. In this figure, we examine both the accuracy and Cohen's κ, where Cohen's κ gives a better measure of performance when the underlying dataset is unbalanced, as with the SSIC. Despite this, both measures have similar characteristics, which indicate that despite the class imbalances, the models have still learnt to distinguish between them, potentially a characteristic of the modified loss function (see section II-I0a on page 2).

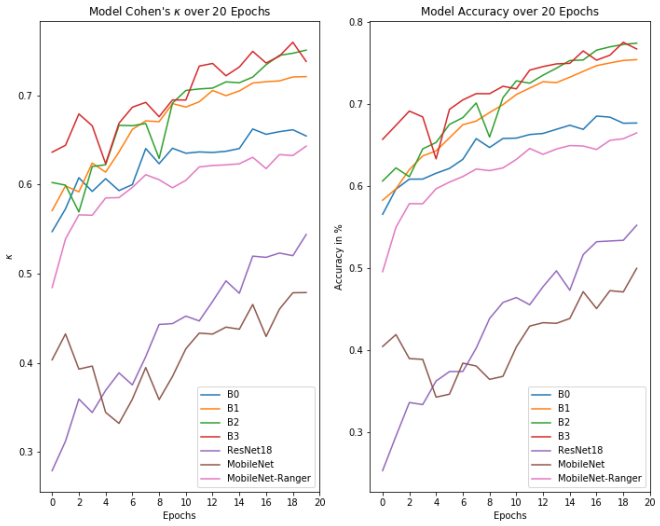


Fig. 5. Accuracy and *Cohen's* κ for the B0, B1, B2, B3, MobileNet and ResNet-18 models over 20 epochs.

The MobileNet and ResNet-18 implementations that used rmsprop achieved lower values across the board, with the MobileNet using ranger achieving significantly better values than its counterpart. The training time for ResNet-18 was approximately 22 minutes per epoch, MobileNet was 7 minutes per epoch and MobileNet with Ranger was also 7 minutes per epoch. This emphasises the trade-off between performance and computational efficiency, as well as the selection of hyperparameters. This also highlights the need for further research into the use of RAdam and Ranger, as drop-in replacements they can improve performance considerably on existing tasks.

D. Class Specific Image Generation

Class-specific image generation is a technique for iteratively optimising an image to maximize the output neuron for a specific class.

The technique itself is very simple, starting with a noise-filled image which is treated as a set of weights, feeding these weights/image through your trained model, then setting your loss equal to the negative output from your target neuron, and finally back-propagating to optimize the image.

Images generated often do not appear real but can give insight into understanding texture or high-level features that the neural network has learnt. Different families of neural networks can produce varying results, and visualisations are highly dependent on the learning rate and weight decay of the image.

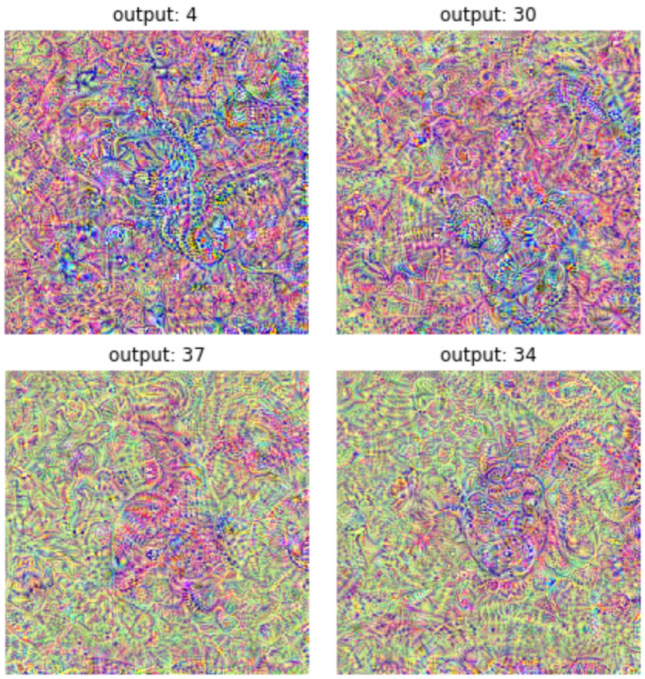


Fig. 6. Class specific images generated for various neurons for EfficientNet-B3 trained on the SSIC dataset. RGB channels of the pptimised images were normalised using Contrast Limited Adaptive Histogram Equalisation for visual aid, actual colors do not match.

Figure 6 gives us an idea of the type of features in the data that our best performing neural network, EfficientNet-B3, has learnt when trained on the SSIC dataset for various output neurons. It is interesting to note how shape wise the features appear very similar and line-like corresponding to the high-level concept of snakes and patterns on the snakes. Accordingly, the outputs are activated for the single appearance of some object that is marked as blue in our visualisations, except for output 4 which may be problematic as it has possibly learnt incorrect features. Although the images look very similar for a human perspective if you look closely there are minor changes to texture across the images, suggesting that the neural network has learnt about individual snake species. It should be noted that Label Smoothing may have affected these visualisations, see section II-I0a on page 2

The downside to this visualisation approach is that visually it works better for humans when there is significant variation in the structure of the underlying classes in the dataset. However, our results still correspond to the general analysis of occlusion sensitivity in the next section.

E. Occlusion Sensitivity

Occlusion sensitivity mapping is a method for visualising the relative importance of regions of an image for a neural network when making predictions for a specific class.

Occlusion mapping works by moving a sliding window that is a contiguous block of colour across the image, measuring the difference in activations for that point for the occluded and non-occluded images.

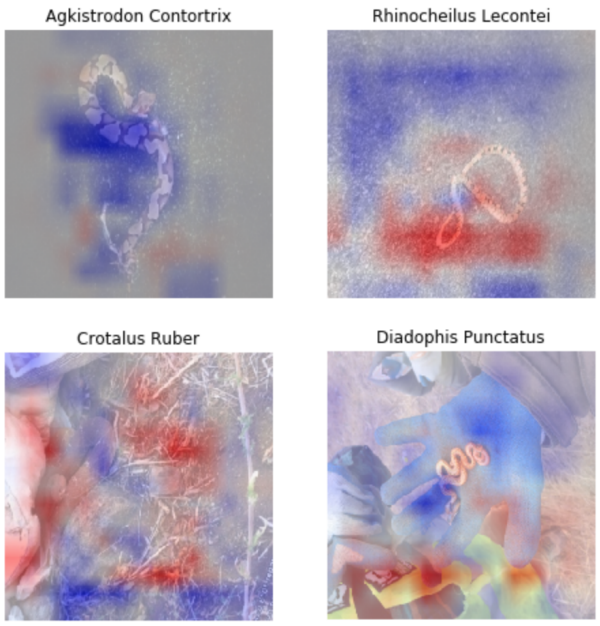


Fig. 7. Occlusion maps for the highest predicted class of various images. Red regions indicate a relative increase in the class activation when that region is occluded, while blue indicates *critical* regions which decrease activations when occluded. Critical, blue, regions should generally lie over snakes, while red regions should generally comprise the background.

Figure 7 contains interesting occlusion maps that explain problems with generalisation and performance.

The top left map for Agkistrodon Contortrix is a good example of a neural network that has learnt appropriate features. These features do not take the background into account and rely solely on the snake within the image.

The remaining three maps are examples of problematic understandings of the images. Rhinocheilus Lecontei relies on the tarmac background and performance for this activation only increases when the snake is occluded. The network has a sporadic understanding of the scene with Crotalus Ruber, the texture of the grass, leaves and twigs confuses that neural network and the actual snake is irrelevant. The final example for Diadophis makes its prediction based off of the human hand holding the snake rather than the snake itself. All these problematic examples can be attributed to biased training data which may not contain examples of those classes or species with varied backgrounds or environments.

The different neural network architectures we used produced similar results when occlusion mapping was applied. This is further indicative of problems in the underlying data, which can be resolved by applying further computer vision techniques to pre-process the data. As snakes vary in size and location between images, our suggestion is to train a model to predict the bounding boxes of snakes and crop out the identified regions. These new regions are used to re-train our various approaches. The expectation is that limiting the background and standardising the size and position of the snakes helps the classifiers learn more meaningful and general representations of the data.

F. Summary of Results

Network	Top 1	Top K	Cohen's K	F_1
ResNet	0.65	0.89	0.63	-
MobileNet	0.61	0.88	0.59	-
MobileNet + Ranger	*0.67*	*0.90*	*0.64*	-
EfficientNet-B0	0.72	0.91	0.66	0.72
EfficientNet-B1	0.75	0.93	0.72	0.75
EfficientNet-B2	0.77	0.95	0.75	**0.77**
EfficientNet-B3	**0.80**	**0.95**	**0.75**	-

IV. Conclusion

In this paper, we explored various models in an attempt to classify images of snakes into their relevant species based on their appearance. Our model's achieved admirable results, with the best accuracy being 80%.

Neural networks are an inherently black box technique, we investigated various methods for visualising what a neural network has actually learnt and sees in an image when making a prediction. This allowed us to draw better conclusions about our models and approaches, how the approaches are limited and how they may be improved in future work.

Furthermore, we examined metrics of various techniques and used alternative metrics to accuracy drawing conclusions for the class imbalanced SSIC dataset.

Computer Vision techniques have become widespread as the state of the art solution in many fields, acting as a catalyst for change.

References

[1] S. Mohanty and S. Khandelwal, "Snake species identification challenge," https://github.com/AIcrowd/snake-species-identification-challenge-starter-kit, 2019.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] 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," *arXiv preprint arXiv:1704.04861*, 2017.

[4] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[5] 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.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[7] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, p. 8, 2012.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[9] R. Müller, S. Kornblith, and G. Hinton, "When does label smoothing help?" *arXiv preprint arXiv:1906.02629*, 2019.

[10] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[11] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[12] M. Zhang, J. Lucas, J. Ba, and G. E. Hinton, "Lookahead optimizer: k steps forward, 1 step back," in *Advances in Neural Information Processing Systems*, 2019, pp. 9593–9604.

[13] "New deep learning optimizer, ranger: Synergistic combination of radam + lookahead for the best of both." https://medium.com/@lessw/2dc83f79a48d, (Accessed on 11/20/2019).

[14] L. Melas-Kyriazi, "Efficientnet-pytorch: A pytorch implementation of efficientnet," https://github.com/lukemelas/EfficientNet-PyTorch, (Accessed on 11/19/2019).