Positive Behavioral Canine Training with Neural Networks and Transfer Learning Optimized for Resource Constrained Platforms

Dennis Kim
CS 528 Embedded Systems and Machine Learning
Dept. Computer Science
Colorado State University

***Abstract* - This paper presents a lightweight, resource-efficient solution for reinforcing positive canine behavioral training using neural networks optimized for constrained platforms like smartphones. Building on prior work utilizing GPU-accelerated devices such as the Jetson Nano, we extend the approach by leveraging EfficientNetV2 pre-trained models, transfer learning, and compression techniques, including pruning and quantization, to achieve comparable performance using fewer resources. The system classifies canine behaviors (sit, stand, and lie) with a test accuracy of nearly 92% and a model's size reduced by the same percentage relative to its pre-compressed model. The dataset, derived from the Stanford Dog Dataset, was augmented to improve generalization, and inference was tested on real-world scenarios with a senior Miniature Schnauzer. In an effort to interpret the model's "thinking", GradCAM and integrated gradients were employed to visualize the model's decision-making process. Our results demonstrate that resource-constrained devices can achieve high accuracy and usability for real-time pet training applications, offering a cost-effective alternative to existing frameworks.**

I. Introduction

Dogs are one of the greatest companions to their parents, and they provide a great deal of support. While many families wish they could bring their pets with them everywhere, that is unfortunately not always possible, and we are stuck leaving them home alone from time-to-time. While most dogs can handle themselves admirably, there are a few that require more mental stimulation: senior dogs, puppies, dogs with separation anxiety, dogs with destructive behaviors, etc. In response to this need, I have created a lightweight, effective dog training application that can automatically engage a dog, whether they are in training, require the preoccupation to distract them from other destructive behaviors, or just enjoy putting on a show for a treat. The proposed model uses a neural network on an Android platform and performs reliably, as will be discussed soon. However, this model is suitable for any resource constrained platform and can easily be paired with a treat dispensing apparatus for the full training reinforcement effect.

The underlying model used for this was not arbitrary. Many models were trained and compared on the same dataset with the winning candidate being the network that offered near uniform accuracy across three commands (sit,

lie, stand) as well as overall model accuracy. We will detail each candidate model's training and test accuracy in defense of our ultimate choice. Additionally, we use various optimization techniques to decrease the model size, making it suitable for embedded platforms while preserving accuracy for this task. Furthermore, we test the model in real-time with a senior Miniature Schnauzer to demonstrate its reliability beyond just test accuracies. The application performance as deployed on the Android ecosystem is excellent and exhibits zero lag.

Note: base code for this project was made available by the authors of [1]. Github citation is available in the citations section as reference [8]. Additionally, many of the functions were derived from lectures from CSU course ECE/CS528 [9].

A. Related Works

This work was largely extended off the efforts of fellow CSU students [1], Stock and Cavey. Their platform was built using a more powerful embedded platform, NVIDIA's Jetson Nano along with a camera and treat dispensing apparatus. While their model performs well, the luxury of the Jetson Nano's onboard GPU is not always accessible, motivating some of the techniques we have used to construct our own model. Other previous works have also focused on capturing the behavior of dogs but use techniques involving wearable devices [2]; and while those have also proven successful, it is not very practical to solve the problem at hand. As an alternative, Stock and Cavey leverage the power of convolutional neural networks, and we will as well in a more platform agnostic approach. As with Stock and Cavey's efforts, we aim to create the application so that it can be a self-contained inference algorithm that does not require any outside resources as that is not always available and for the sake of resource preservation. Since dogs are a demonstrative species, convolutional neural networks are well suited for such a task, avoiding wearable devices. Stock and Cavey have found similar research [3] that supports the benefits of algorithm-aided pet training, demonstrating that the training provided relief for separation anxiety-related distress. It is worth noting that the work by Mundel et al. [3] made use of microphones alongside a camera. However, the microphone was used to capture vocalizations, another manifestation of distress. Since observing the presence of distress is not the focus of our work and evidence already exists supporting training as a source of comfort for distressed dogs, we simply focus on the actual behavior presence itself and stick to the camera in order to detect the desired behavior. Stock and Cavey draw upon previous work to justify their use of the Jetson Nano platform, citing previous work using the same platform from real-time fire detection to autonomous driving and sign-language translation. And although their model is suitable for low cost platforms like the Nano, a portion of our aim was to demonstrate that we could do even better by restricting ourselves to even more resource-constrained platforms, like an Android phone. The idea being to demonstrate proof-of-concept that the model can perform reliably even on theoretically cheaper platforms. With all of this in mind, we feel comfortable claiming an Android phone is a good proxy for cheaper platforms with less resources in this work.

B. Contributions

Our extensions to Stock and Cavey's proposed model include using newer CNN networks that were unavailable at the time of their work. We also explore various optimizations and compression techniques to bolster and preserve inference performance while making our model suitable for low-cost platforms - something that was not done previously. The project in its entirety is provided alongside this paper and will soon be made available as a Github repo.

II. Dataset Details

The dataset used to train these models were largely provided by Stock and Cavey themselves who annotated images from the Stanford Dog Dataset [4]. Although one of the original goals was to supplement those images with new hand-labeled instances in hopes of improving transfer learning, time did not permit any amount of additions, meaningful or not. This was disappointing, but the circumstances of my group changed and my sole focus was spent on fine-tuning the ultimate candidate model. This was not a detrimental miss. The original dataset was quite large and included a healthy number of examples for each type of command. The dataset included 20,578 images: 4143 for standing, 3038 for sitting, 7090 for lying down, and 6307 for undefined positions. Figure 1 provides a sample of images for each command. Only instances of standing, sitting, and lying down were used, presenting an issue with imbalance classes; but as we will see later, the imbalance did not have a material effect on the model's performance. The undefined instances were deemed to be too ambiguous or context-less for accurate annotation and therefore were not used by either Stock and Cavey nor this model.

Each image consisted of customary RGB characteristics, the overall dataset contained 120 unique breeds in a diverse range of image quality. Roughly half of the images were between

Fig. 1: Dataset samples for each command class

a resolution of 361x333 and 500x453, but the diversity of image sizes in the dataset went beyond those bounds. As Cavey and Stock point out, this allows for downsampled images that can still retain much of the original encoded information.

The entire dataset was preprocessed using various augmentation before training and testing as a means to improve the generalizability of our model. The augmenting techniques included: random rotations, random reflections about its horizontal access, and resizing to a uniform 224x224 pixel layout. Given the success of our model as well as that of [1], it

is evident that these augmentation techniques were beneficial to generalization.

III. Experimental Methodology

A. *Models and Training Details*

Like [1], we decided to use a convolution neural network as the underlying framework to operationalize confirmation of command. Since the purpose of our framework is simply to determine if a command was successfully completed, we felt like a CNN was more than adequate to capture the desired effect. RNNs were also considered as we felt the ultimate product should have a time component associated to any command to capture that a command was followed and held. However, the added benefit of a temporally-based RNN did not outweigh the complexity required to implement it. Instead, the duration a command was held was implemented in the application source code itself. The CNN still classifies images as sitting, standing, or lying; but it does so as an ensemble of images captured over the desired duration of the command. As an illustration, the app itself captures images of the dog over a user-specified period at a rate of 3 FPS. If 90% of the images are classified by our CNN as the chosen command, the app signals success. Otherwise, it signals failure. .

Multiple models were considered for our successor platform, specifically from the newer versions of the EfficientNet family (EfficientNetV2). These models were chosen because [1] uses the original EfficientNet (B0-B4) architecture [5]; and although the comparisons would not be exactly apples-to-apples, it seemed a more appropriate comparison than any of the more complex, time-consuming Keras models. Variations in data preprocessing and augmentation were not explored as [1] had previously. This decision was justified by the desire to best preserve final model comparison with those found by [1].

Training took place in a Google Colab environment, utilizing their available TPUv-2.8 accelerator. Specs were not readily published, but as of the date of this paper, the system RAM and disk space made available to this effort was ~335GB and ~225GB, respectively. Since the new EfficientNetV2 models were of mostly larger parameter size than their predecessors, training times ranged from 100s/epoch to a shade under 1400s/epoch.

B. *Smartphone and Ecosystem*

The chosen embedded device for inference in this study was the Motorola Moto G Stylus 5G, an android platform smartphone. This device came with 4GB of ram, 128 GB of storage capacity, and a 48 MP camera resolution, but no spec that rivals the Jetson Nano, making it the perfect device to demonstrate the suitability of our proposed model to low-resource devices. We unfortunately were unable to build an apparatus to dispense treats upon successful command execution as we ran out of time, but it is not unreasonable to think that that could be done in a straight-forward implementation. As a hypothetical example, such an apparatus would simply need a binary signal sent to its receiver that dispenses the treat mechanically. The binary signal would be the signal from the deployment platform indicating a successful command.

C. *Post-Training Quantization & Other Deployed Compression Techniques*

Finding techniques to shrink our final candidate model while preserving its test performance was a challenging task. [1] provided code scaffolding, but it was clear that aged dependencies and libraries meant the code itself no longer ran as is, and a number of edits had to be made in order to experiment with post-training quantization techniques. One of the most frustrating challenges was the inclusion of data augmentation directly into the proposed model from [1]. As a custom layer, it meant that many quantization options were off the table. Google's Gemini made some helpful contributions but was largely hit-or-miss in its suggestions.

Ultimately, the best solution was to spin out the data augmentation layer as a separate function and apply it outside of the model. This did smooth quantization application, but it did not fix every issue. The EfficientNetV2 family came with many unique layers that presented a number of problems. And although each problem was overcome, it was done so after much research over the StackOverflow forums.

Ultimately, our proposed model went with the default quantization techniques (via tf.lite.Optimize.Default) without a representative dataset, meaning that only weights were quantized to 8-bit integers while leaving activation functions as is. And while it applied pruning and weight clustering, performance was best preserved without PCQAT which was surprising. These combined efforts ultimately shrank our Keras model from roughly 102MB to ~9MB, a nearly 92% size reduction. Such compression had minimal effect on the test accuracy as well amounting to <1% drop in accuracy (relative to its uncompressed state). The model that applied everything plus PCQAT actually experienced an increase in accuracy, suggesting that it improved generalization; but the increase in performance paled in comparison to the loss in compression evident in our chosen candidate model. For detail, the PCQAT version was roughly 32MB in size and had a test accuracy of just over 92%.

IV. Architecture Development

Transfer learning was the primary chosen methodology for building our model. This decision was based on Stock and Cavey's comments regarding the laborious search for a custom model as well as its lackluster performance. Additionally, the candidate base models used (e.g., EfficientNet and EfficientNetV2) were proven to perform well on image classification tasks. Given the size of the dataset as well as its distinct qualities relative to the imagenet dataset, fine-tuning our transfer learning efforts made use of the entire network rather than a portion.

For our training efforts, the dataset was split into training, validation, and test sets with 75%, 10%, and 15% of the images, respectively. Just like [1], the images were augmented by randomly rotating them by $\pm$8%, random horizontal flips, and image translations to their width and height. This was done in an effort to improve generalization. Unlike [1], taking the data augmentation layer out of the original approach by [1] meant that our training times were not greatly affected in the same way theirs had been.

A. *Transfer Learning*

As previously stated, this model leaned heavily on transfer learning off the EfficientNetV2 family the same way that [1] did with the original EfficientNet family. Not only does this somewhat preserve comparability, but it provided an opportunity to test the benefits of larger base models in terms of accuracy. Since [1] had not previously used pruning, weight clustering, or PCQAT, it was hypothesized that larger models could potentially lead to better (or at least comparable) accuracy in a much smaller package. All of our candidate base models made us of 224x224 images. While Stock and Cavey experimented with other resolutions, this project only experimented with 224x224 because of the time each model took to train.

Our final candidate model was based on the EfficientNetV2B network originally trained on imagenet. It exhibited the most uniform performance across the three commands at high accuracy (see Table 3). The transfer learning model was built by initially removing the final layer and adding a handful of new layers. The additional layers were a global average pooling layer, a dense layer using the RELU activation function, a dropout layer of 35%, batch normalization, and a dense layer with a softmax activation function for probability outputs. The final model trained using adaptive moment estimation to optimize the spar categorical cross entropy, which is suited for our task as traditional vectors, not one-hot encoded vectors. The learning rate used was .00001 over 20 epochs and a batch size of 32.

The results were favorable and primarily demonstrated the benefits of various compression techniques. The final model of this paper achieved a nearly 92% test accuracy, including pruning, weight clustering, and tflite conversion. This represented only a marginal decrease (<.5%) in accuracy relative to the best performing model in, EfficientNetB4 [1]. Training time was much longer for the proposed model in this work relative to [1], but as a one-time cost, this is not a huge problem. Our model was trained on a single TPU while Stock and Cavey had the benefit of multiple GPUs. This likely explained the difference in training time (150 vs 52 minutes, respectively).

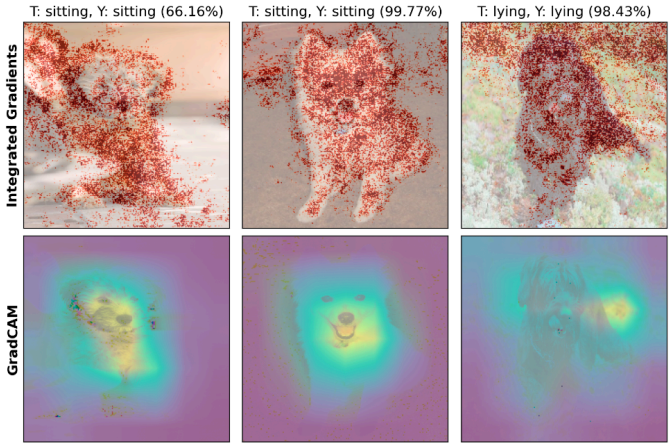

Fig. 2: GradCAM and Integrated Gradients results and insight into model "thinking". "T" represents the ground truth while "Y" is the model prediction.

	Lying	Sitting	Standing
Lying	86.4	8.3	5.4
Sitting	4.2	89.4	6.4
Standing	2.4	4.9	92.7

Table 3: Confusion Matrix for the EfficientNetV2B2, which was used as the base model for transfer learning.

V. Model Interpretation

While high test accuracies were achieved, the underlying neural network is inscrutable to the everyday user possibly detracting from trust (in any neural network architecture really) in the

model itself. To mitigate that opacity, the proposed model was evaluated using two methods to explain its "thinking": gradient-weighted class activation mapping (GradCAM)[6] and integrated gradients [7]. Both techniques aid in model transparency by studying how input features attribute specific predictions. Figure 2 illustrates findings from both techniques as features of focus over input images from each class. "Integrated gradients obtained by accumulating all the gradients along a straight line path from a baseline zero-intensity image to the input sample" [1]. It appears that the EfficientNetV2B2 built model improves on feature focus than that suggested by [1]. While [1] found that their models focused on individual parts of a posture, the model in this work focuses on the entire entity when inferring a prediction. As a caveat, the lying posture was somewhat less exact as focus by our model included the space above the entity. This was not the case for Stock and Cavey.

While integrated gradients highlight granularity of focus at the pixel-level, "GradCAM localizes relevant regions in an input image through the activation maps of successive convolutional layers using the desired class signal and back propagating gradients to compute a localization heat map" [1]. In other words, GradCAM highlights the area that our model found to be most pertinent when making predictions. As can be seen in the second row of figure 2, the focus of our model is exactly on the body of each entity, regardless of position. This promising result was supported by the observation that sitting postures were more vertical than lying postures where the focus remained more horizontal. These observations support the efficacy of the model in this work as demonstrated by these two common neural network explanatory techniques results.

VI. Experiments

Given the project scope, the paper does not dive into as many model traits as Stock and Cavey. The aim here was to create an application that could be used by pet parents to train their dog on an economical platform that may not have the luxury of high compute resources, like the Jetson Nano. As a result, the experimental results were primarily focused on accuracy characteristics, model size, and inference speed last. Inference speed was less important in this case as the application here would not require the highest FPS over the desired command duration. Instead, our application uses a modest FPS rate in order to reduce platform strain, resulting in zero degradation in efficacy or accuracy.

Experiments were broken up into stages: model accuracy results and distributions as well as compression technique evaluations. Given these criterion of focus, it was determined that the models based on EfficientNetV2B2 had a high test accuracy characterized by a uniform accuracy across commands. The spread of validation accuracies between classes was 6.3 percentage points while the spread of other models was in the double digits. Uniform accuracy across commands was determined to be of importance as the model should be as consistent in performance as possible to avoid canine confusion and poor training. Model sizes ranged from between 9.3 and 102 MBs (refer to Table 1 for model comparison details).

Model Base	Pruning	Clustering	PCQAT	Tflite Conversion	Size (in MB)
EfficientNetV2B2	✔				69.2
EfficientNetV2B2	✔	✔			80.9
EfficientNetV2B2	✔	✔	✔		102
EfficientNetV2B2	✔	✔	✔	✔	9.3

Table 1: Compression technique effect on model size

Training experiments included varying hyperparameter values as well as base models. Hyperparameter variations (epochs, batch sizes, early stopping delta, patience, and image input shapes) all confirmed the observations by [1]/ Namely, that the best image input shape was 224x224, Adam learning rate of .0001 had a good tradeoff between training time and model conversion, and 20 epochs combined with a batch size of 32 led to the highest accuracies. Additionally, early stopping was used for convenience as well as regularization of all models. Cavey and Stock did not use early stopping, but the best hyperparameters for our use were a minimum delta of .0001 and a patience of 5 epochs of no improvement.

Three primary compression techniques were explored post-training of the final candidate model: pruning, weight clustering, and quantization. For pruning, various initial and final sparsity percentages were used, but it was found that even aggressive sparsity requirements did not materially affect model performance. However, due to EfficientNetV2B2's construction, the only pruning that took place was at the final dense layers where nearly 75% sparsity was achieved. Stackoverflow community users suggested using various tools from the PyTorch library, but there was little time to add learning PyTorch to my model. Regardless, the pruning still resulted in an 85% decrease in overall model size. Weight clustering was less effective by comparison. While the last two layers did achieve clustering of 16 and 17 clusters, respectively, it contributed nothing to the compression of the pruned model. There was also a similar story of unclustered weights due to the complexity of EfficientNetV2B2. This was an unfortunate outcome, but the compression achieved so far using pruning was encouraging enough to experiment with post-training quantization. PCQAT was actually detrimental to compression efforts though it did improve test accuracy, albeit marginally. Overhead costs attributable to PCQAT actually increased the model size from 34MB to nearly 104MB. Since this was a step in the wrong direction without any meaningful benefit (accuracy improvement of <1%), PCQAT was skipped. The resulting pruned and clustered model was compressed into a tflite format and decreased another 73%. This combined with pruning shrunk the model from 102MB to just over 9MB making this model incredibly light weight and suitable for smartphone deployment.

With the final tflite model detailed, comparisons between models from [1] and this model can be made. Table 2 (appendix) compares the EfficientNetB(0-4) family with the model proposed here. Please note that quantization was applied to Stock and Cavey's family of models whereas the model here did not. This means that pruning had a meaningful impact in achieving our lightweight goals. Even more impressive is that EfficientNetV2B2 is larger than 60% of the EfficientNetB(0-4) models used in [1] yet was still smaller than 73% of the model variations in [1], regardless of quantization techniques used. Additionally, the techniques deployed here actually improved accuracy over the best of nearly every EfficientNetB(0-4) model highlighted in [1]. Specifically, our final model

experienced a 91.4% test accuracy while the best of [1]'s EfficientNetB(0-4) family topped out at 91.2%. Thus, we argue that the new EfficientNetV2 family did benefit accuracy despite having more parameters to start yet through pruning and tflite conversion still achieved a model size that beat out the majority of its predecessors. Note given the lack of focus on inference speed here, FPS was not tested nor compared. Again, our goals differed slightly from the original work.

Furthermore, apart from the Table 3, precision, recall, and f1-scores were calculated to further support our models efficacy. Table 4 provides each category.

Metric	Lie (%)	Stand (%)	Sit (%)
Precision	93.89%	94.39%	82.82%
Recall	89.77%	93.95%	88.69%
F1 Score	91.78%	94.17%	85.65%

Table 4: Precision, Recall, and F1 scores for our model

Precision and recall values indicate that false positives and negatives were minimal, and the overall F1-score supports those two findings.

VII. Conclusion

With a likely successor model in hand, we have achieved our goals to both shrink the model to a size better suited for an even more resource-constrained platform than that used in [1], like a smartphone. We did our best to keep models as comparable as possible though some hyperparameters differed from the previous work. Unlike the Stock and Cavey, we applied pruning and weight clustering techniques, which were arguably required given we tested even larger base models than used in [1]. Practical tests via apps demonstrate that our model works exactly as intended, indicating when a dog successfully follows a command. With this app, pet parents can now rest easy that their pets are not bored at home, suffering separation anxiety, or resorting to destructive behaviors. At a nearly 92% accuracy rate, this model beats out the majority of its predecessors and at a significantly lower memory cost. Aside from a few parameter changes for improved training, these results are directly attributable to the new techniques used in this paper: pruning and weight clustering, leveraging new base models for transfer learning, and regularization effects. However, future work could improve on this model as well. Stock and Cavey specifically mention that generalization could be improved by supplementing the training data with even more pictures. Furthermore, these models focus on a small subset of commands and could be expanded to even more commands (e.g., shake). Additionally, an RNN could be a better neural network architecture if a user wants to cover more dynamic commands (e.g., rolling over, turn left, etc).

References

1. J. Stock and T. Cavey, “Who’s a Good Boy? Reinforcing Canine Behavior using Machine Learning in Real-Time,” *arXiv (Cornell University)*, Jan. 2021, [Online]. Available: https://arxiv.org/abs/2101.02380v1

2. J. Majikes *et al.*, “Balancing noise sensitivity, response latency, and posture accuracy for a computer-assisted canine posture training system,” *International Journal of Human-Computer Studies*, vol. 98, pp. 179–195, Apr. 2016, doi: 10.1016/j.ijhcs.2016.04.010.

3. P. Mundell, S. Liu, N. A. Guérin, and J. M. Berger, “An automated behavior-shaping intervention reduces signs of separation anxiety–related distress in a mixed-breed dog,” *Journal of Veterinary Behavior*, vol. 37, pp. 71–75, May 2020, doi: 10.1016/j.jveb.2020.04.006.

4. A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, “Novel dataset for fine-grained image categorization,” in First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition, Colorado Springs, CO, June 2011.

5. M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” ArXiv, vol. abs/1905.11946, 2019.

6. R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” International Journal of Computer Vision, vol. 128, pp. 336–359, 2019.

7. M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” in ICML, 2017.

8. J. Stock and T. Cavey, “canine-embedded-ml,” *Github*, Jan. 2021, [Online]. Available: https://github.com/stockeh/canine-embedded-ml

9. S. Pasricha. (2024). ECE/CS 525 Module 4: Software Model Optimizations [PowerPoint Slides]. Available: https://colostate.instructure.com/courses/191420/module

Appendix

Table 2

Model	Tflite	Dynamic	FP16	Pruning	Clustering	Accuracy (in %)	Size (in MB)
EfficientNetB0_dyn		✔				76.68	4.16
EfficientNetB1_dyn		✔				87.48	6.67
EfficientNetB2_dyn		✔				89.91	7.82
EfficientNetB0_fp16			✔			90.79	7.98
EfficientNetV2B2 (proposed)	✔			✔	✔	91.48	9.30
EfficientNetB3_dyn		✔				88.27	10.73
EfficientNetB1_fp16			✔			91.03	12.84
EfficientNetB2_fp16			✔			90.84	15.12
EfficientNetB0_lite	✔					90.75	15.68
EfficientNetB4_dyn		✔				90.33	17.41
EfficientNetB3_fp16			✔			91.40	20.87
EfficientNetB1_lite	✔					91.03	25.27
EfficientNetB2_lite	✔					90.84	29.82
EfficientNetB4_fp16			✔			92.06	34.02
EfficientNetB3_lite	✔					91.36	41.27
EfficientNetB4_lite	✔					92.06	67.46
EfficientNetV2B2						90.16	102.00

Note: The models highlighted in blue refer to the models developed in this paper while all other models originated from [1]. Our final, compressed model beat out all but two predecessor models. However, the models (EfficientNetB4_fp16 and EfficientNetB4_lite) that did beat ours were ~4x and ~7x the size, respectively.