# What is your Dog Feeling?

Multi-Classification Image Recognition with TensorFlow and Keras

Doug Andrade, 18 June 2024

### Introduction

Endless articles, tutorials, and discussion boards exist about convolutional neural networks (CNN) for image recognition. However, most literature lacked detailed analysis around iterative model design and improvement processes, such as how to balance image pixel sizes versus image batch size or the impact of pooling layers on CNN layer channel architecture design.

This project may serve to inform individuals looking for details beyond common recommendations, on why or how one would select an image batch size, pixel size, evaluation metric, or fine-tuning layer. Throughout this project, I constantly had the urge to find better parameters and layer architectures other than the standard recommendations or defaults.

Provided is an overview of my personal CNN project using Python with TensorFlow (with a single graphics processing unit [GPU]). Links to my Python script and image dataset are available in the report's conclusion.

### Process Summary

The project's basis was to apply an image recognition CNN model with transfer learning and fine-tuning to classify the emotions of my wife’s late American Bulldog, Perogi. I used my laptop’s GeForce RTX 2060 GPU to run the CNN model.

My project’s primary intent was critical yet simple and achievable. The project’s secondary objectives were ambitious and unguaranteed, yet not critical for project success.

##### Project Goals:

1. Primary Intent
   1. Pursue self-learning to develop an independent understanding of deep learning applications.
2. The secondary objectives:
   1. Consistently classify the project’s title image of Perogi as “happy.”
   2. Achieve a validation accuracy ≥ 70%.
   3. Achieve an F1-Score ≥ 25% for each class (five).

##### The Project Flow:

1. Exploratory data analysis
2. Design a baseline CNN model (inspired by VGGNet16)
3. Design a transfer learning model (EfficientNet2VS selected)
4. Train the model for the optimal parameter combination
5. Iterate over a set of fine-tuning layers for best performance
6. Evaluate and analyze model performance
7. Apply the final model to 201 unlabeled images of Perogi
8. Assess project goals

### Exploratory Data Analysis

I came across an existing dataset of 15,921 images divided into four classes on Kaggle, which, to my surprise, was precisely what I thought I needed. Unfortunately, I discovered the images were far from reliable when I was in step three (described above). My initial EDA of the Kaggle dataset failed to recognize the severe class imbalance, poor image labeling, and the 6,596 non-dog images.

Necessarily, I decided to manually evaluate each of the 15,921 images for usability (of which 6,596 images were removed), as well as relabel each image based on a self-created set of standard criteria (label name selection was intentional for string length purposes during coding):

* Alert – The appearance of attention (wide eyes, stiff ears, rigid body).
* Angry – The appearance of growling, with an aggressive display of teeth.
* Frown – The appearance of dejection, pain, or abuse.
* Happy – The display of the tongue, with a near human-like smile.
* Relax – The appearance of resting or doing nothing.

### Baseline Model Development

To begin the transfer learning and fine-tuning process with an understanding of my machine’s hardware limitations (and my academic understanding of deep learning), I chose to recreate the Visual Geometry Group’s VGGNet16 (D) model. Consequently, I learned my machine could not effectively process image pixel sizes of 224x224, as used by VGGNet, without significant trade-offs to modeling convergence and training time (adopt batch sizes of 16 or 32 and reduce the CNN layers and parameters due to the max pooling to fit to smaller pixel sizes). Nevertheless, I created the best variation of the VGGNet16 model my machine could process.

##### Baseline Model Description:

* Image batch size: 128; Image pixel size: 128x128x3
* 12 layers:
  + 1st Group
    - 1 CNN input layer, 32 channels, 3x3 kernel size, ReLU activation
    - 1 max pooling layer, 3x3 pool size, 2x2 stride
    - 1 batch normalization layer
  + 2nd Group
    - 2 CNN input layers, 64 channels, 3x3 kernel size, ReLU activation
    - 1 max pooling layer, 3x3 pool size, 2x2 stride
    - 1 batch normalization layer
  + 3rd Group
    - 2 CNN input layers, 128 channels, 3x3 kernel size, ReLU activation
    - 1 max pooling layer, 3x3 pool size, 2x2 stride
    - 1 batch normalization layer
  + 4th Group
    - 4 CNN input layers, 256 channels, 3x3 kernel size, ReLU activation
    - 1 max pooling layer, 3x3 pool size, 2x2 stride
    - 1 batch normalization layer
  + 5th Group
    - 2 fully connected dense layers, 4096 channels, ReLU activation
    - 2 dropout layers
    - 1 fully connected dense output layer, 5 channels, softmax activation

The main takeaway was this model achieved an unimpressive ~40% accuracy and an entropy of ~1.4. My goal was not to build a high-performing baseline CNN model but to create a baseline understanding of what my machine could do with minimal effort. I learned the impact of image pixel size, image batch size, and pooling parameters trade-offs that would limit the detail of information my machine could process to learn specific image characteristics for classification.

### Transfer Learning and Fine-Tuning Model Development

Most literature recommends that random parameter tuning is usually best. I found this hard to accept, and it kept me constantly wondering if there was a better combination of parameters that I missed. I ran over 250 iterations of my transfer learning and fine-tuning models to narrow in on the optimal combination of parameters. I created a composite score for each model trained to narrow the best-in-class model selection process (further detailed in the Other Modeling Details section).

I focused on creating the best-in-class fully connected top layer for transfer learning while keeping the fine-tuning part of the model in inference mode. Once I exhausted the combinations of parameters, I selected the top ten models for fine-tuning model testing specifically focused on identifying the best EfficientNetV2S layer to begin tunning.

##### Transfer Learning (fine-tuning in inference mode) Training Methodology:

1. Train 103 iterations of specified dense layers and channels on a set image batch size of 16 and pixel size 128x128x3.
2. Train 40 iterations on the prior top 10 models on unique batch sizes (16, 32, 64, 128).
3. Train 40 iterations on the prior top 10 models using four unique dropout ratios (0.2, 0.4, 0.6, 0.8).

##### Transfer Learning with Fine-Tuning Training Methodology:

1. Train 32 iterations on the prior top four models on eight unique Efficientnet2VS CNN layers (from least to most layers – 6h, 6a, 5a, 4a, 3a, 2a, 1a, all).
2. Train 12 iterations on the prior top three models on four ranges of dropouts (0.0-0.4, 0.2-0.6, 0.4-0.8, 0.0-0.8).
3. Select re-train the final model for additional analysis.

##### The Final Model Architecture:

* Input Layer
  + RGB (red, green, and blue) image size 128x128x3.
* Image Augmentation Layer
  + Random flipping on the horizontal and vertical axis.
  + Random rotation at a random range between [-25% \* 2pi, 25%\* 2pi].
  + Random translation at a range between a random lower and upper bound of [-10%, 10%] in both the height and width dimensions.
  + Random contrast between a random lower and upper bound range of [1.0 - 0.25, 1.0 + 0.25].
* EfficientNet2VS Transfer Learning Layer
  + EfficientNet2VS’ top layer was disabled since it is designed to classify on ImageNet’s 1000 labels.
  + The parameter trainable = False, to preserve and leverage the EfficientNet2VS pre-trained weights for the transfer learning benefit.
* CNN Top Layer
  + Omitted from the final model (degraded performance).
  + A single 32-channel CNN layer of 3x3 kernel size, with ReLU activation.
* Top Pooling Layer
  + Part of the CNN top layer. Removed for the final model.
  + Max pooling of size 3x3 and a stride of 3x3.
* Flatten Layer
  + The layer collapses the high-dimensional CNN output arrays from EfficientNet2VS into a single dimension of operations at the fully connected top layer.
* Fully Connected Top Layer
  + A dense layer of 256 channels and Leaky ReLU activation.
    - A batch normalization layer.
    - A dropout layer to ignore 40% of input units.
  + A dense layer of 512 channels and Leaky ReLU activation.
    - A batch normalization layer.
    - A dropout layer to ignore 60% of input units.
  + A dense layer of 512 channels and Leaky ReLU activation.
    - A batch normalization layer.
    - A dropout layer to ignore 80% of input units.
  + Fully Connected Output Layer
    - A dense layer of 5 channels (class labels), with softmax activation.

##### Transfer Learning Step:

The model training was initially executed on the pre-trained weights (inference mode) from the EfficientNet2VS model with the previously described architecture (input, regularization, top layers, and output).

After the early stop callback requirements were satisfied, the transfer learning stage ended, and the fine-tuning stage of training began, picking up with the very next epoch.

##### Fine-Tuning Step:

For the final best-in-class model, all of EfficientNet2VS’ CNN layers were set to trainable = True, except the batch normalization layers and output layer.

Training repeated on the full EfficientNetV2S model with weights updated based on the optimization (loss) function in the EfficientNetV2S model architecture. The top layer’s weights were trained on the fine-tuned weights from the EfficientNet2VS model.

### Other Modeling Details: Computational Efficiency and Time Management

##### Mixed Precision

The use of Kera’s mixed precision function to maximize the GPU’s opportunity to execute operations on float16 data types over float32 made training time more efficient (~25 seconds of training per epoch to ~10 seconds of training per epoch). Of note, Keras’ fully connected output layers required a precision operation on float32 data types, necessitating the dtype = ‘float32’ parameter to be set into the final dense output layer.

##### AUTOTUNE API

TensorFlow’s AUTOTUNE API leverages the GPU memory and processing capability to execute data loading via prefetching. This is done by overlapping the data preprocessing and training steps (https://www.tensorflow.org/guide/data\_performance), allowing the model to prefetch elements of the input dataset ahead of the time they are trained, significantly reducing the model’s idle time and training time.

##### Model Scoring Record Keeping

Developing a systematic way to record and evaluate models was critical to maintaining a historical log for reference and, most importantly, for consistency. Due to the substantial number of parameters and architectural design combinations, I recognized I was losing awareness of the optimal combinations and trends for all my testing factors. I implemented a table to auto-populate with key parameters, metrics, and a composite score, following the training of each model iteration. The composite score was a continuously updated normalized weighted combination of the model’s accuracy, entropy, and F1 score.

### Final Model Evaluation

The final model was selected based on the iterative selection process described in the Transfer Learning Training Methodology and Fine-Tuning Training Methodology sections. Additionally, a confusion matrix supplemented the model metrics evaluation to assess the secondary objective of achieving a class F1 score ≥ 25% across all classes.

The ultimate evaluation was to observe how the model performed on unlabeled images of Perogi, particularly the title image. The potential irony in this evaluation process is that even if I had built a CNN model with perfect validation accuracy and near-zero entropy, if the predicted label of Perogi’s emotion made no sense to me or my wife then the model would be a failed machine learning tool.

### Conclusion

My primary intent was fully achieved and even exceeded my expectations. My secondary objectives were partially achieved:

1. Primary Intent
   * Pursue self-learning, specifically to develop an independent understanding of deep learning applications.
2. The secondary objectives:
   * Consistently classify the project’s title image of Perogi as happy
   * Achieve a validation accuracy ≥ 70%
   * Achieve a class F1-Scores ≥ 25%

An unanticipated lesson learned lies in the idea of stakeholder engagement. My wife inspired the idea of this project, and therefore her opinion of what a happy, angry, or sad dog looks like is essential to the success of the CNN model. Unfortunately, the relabeled training dataset was based on my opinion of a dog’s emotion and not my wife’s. Had I thought to keep my wife involved in the EDA, modeling, and evaluation process, then she may have been more satisfied with the final prediction outcomes. Unfortunately, my wife and I will have to agree to disagree on what a happy dog looks like, for now!

The Python Jupyter Notebook is available for reference at my GitHub repository (https://github.com/dougrandrade/Dog\_Emotions\_CNN\_Repo). The associated image dataset used for training is available on Kaggle (https://www.kaggle.com/datasets/dougandrade/dog-emotions-5-classes).