15.S04 - Hands-On Deep Learning

Dog Breed Identification
Final Report

Grace An, Jaya Ren, Jessie Tanaboriboon, Yen Hann Yoo

Section I: Introduction and Problem Statement

Have you ever walked across Sloan and come across a cute, friendly dog, but you always wonder what breed it is? Our project solves your problem. In this Hands-on Deep Learning project, we aim to identify dog breeds through images.

From a technical perspective, we will leverage different deep-learning architectures to extrapolate features and correctly identify dog breeds. More importantly, from a business perspective, accurate identification of dog breeds offers benefits for dog shelters and adoption centers. An improved matching process between potential adopters and dogs becomes possible. Additionally, exact breed information enables centers to provide better insights into each dog's temperament, size, and care requirements, increasing the chances of finding the right home for every animal. Furthermore, this technology saves time and resources spent on manual identification and documentation, allowing staff to concentrate on other essential tasks like animal care and customer service.

Section II: Dataset and Data Preprocessing

Colab Link:
https://drive.google.com/file/d/15y6RVhsQmud-d-NMm73tHH9orPoAAoIZ/view?usp=share_link (Cropping)
https://colab.research.google.com/drive/13GCJG7YQUH05KGl0u-RHQQU_cON4sd1Z?usp=share_link (Splitting Data)

We are using a dataset from Dog Breed Identification - Kaggle Competition [1]. The dataset comprises 120 breeds of dogs and 10,200 images. Each image contains 3 channels (RGB).

Figure 1. Example Images from Dataset

To address the issue of varying image sizes within our dataset, a decision was made to employ a center-cropping technique. This involved adjusting all the images to achieve a square shape, matching the size of the smaller dimension. For instance, if an original image had dimensions of 320x360, it was cropped to 320x320, and if an original image had dimensions of 320x280, it was cropped to 280x280

Once all the images were standardized to a square shape, we proceeded to resize them uniformly. The target size chosen for most models was 244x244, with the exception of one model where the input specification required a size of 224x224.

To facilitate evaluation and training, we partitioned the dataset into two separate portions. Approximately 20% of the data was set aside for testing purposes, while the remaining 80% was allocated for training.

Section III: Model Methodology

We propose 3 main approaches for our classification task: Convolutional Neural Networks (CNNs), Transfer Learning, and a Vision Transformer

1. Convolutional Neural Networks (CNNs):
CNNs are widely used in image classification tasks and have achieved state-of-the-art performance. They work by learning hierarchical representations of the input image by using convolutional layers. The last layer of the CNN can be a softmax layer that outputs the probability distribution of the 120 dog breeds.

Colab Link: https://colab.research.google.com/drive/1QvEAzzAgDQvjqNYTjIhOYbV02AEAk9vE?usp=sharing

Two CNN models were built to identify dog breeds, the architecture of the first model begins with three convolutional layers (Conv2D), each with 16 filters and a kernel size of 2x2. These layers are followed by three max-pooling layers (MaxPool2D) to avoid overfitting. The output of these convolutional layers is then passed through a fully connected layer (Dense) with 256 neurons and a softmax activation function to make prediction of the 120 breeds of dogs. The model is then trained with 10 epochs and 32 as batch size. The accuracy of this model is around 3%.

To improve the performance of CNN, we looked at data augmentation, a technique used to artificially expand the size of the training dataset by applying transformations to the input data. This is implemented through *RandomFlip* and *RandomRotation* of layers to increase the diversity of the training data. Aside from its theoretical benefits, this method made intuitive sense because dogs in the images were not always positioned upright. Hence, the method should be powerful in addressing the diversity of positions of dogs in the images. Data augmentation is applied directly when input image is passed in, and this output will be passed into a CNN with the exact same architecture as before. As a result, the accuracy of this model improves slightly to around 7%.

2. Transfer Learning:
Transfer learning using a pre-trained model like Resnet50 can help by leveraging pre-learned weights that capture general image features, reducing the amount of training required. Resnet50's deep structure also enables it to recognize complex visual patterns, making it ideal for dog breed classification. Using transfer learning with Resnet50 or VGG19 can provide a head start and significantly reduce training challenges, resulting in high accuracy even with limited training examples.

a. **Resnet50**

Colab Link: https://colab.research.google.com/drive/1Sy3--B_mP7KkZpQM0baBgBqW9Be-CK7M?usp=sharing

First, feature extraction is performed using the ResNet50. The preprocessed images are fed through the ResNet50 model, excluding its top layer, to obtain high-level feature representations.

The model architecture is then defined for the classification task. The feature vectors obtained from the ResNet50 model serve as input to the classification model. The model architecture comprises an input layer with a shape of (8, 8, 2048) to match the dimensions of the extracted feature vectors. A flatten layer is added to convert the 3D feature vectors into a 1D vector. Subsequently, a dense layer with 256 units and a ReLU activation function is introduced as a hidden layer to learn complex patterns in the data. Finally, a dense layer with 120 units and a softmax activation function is included as the output layer, representing the 120 different dog breed classes.

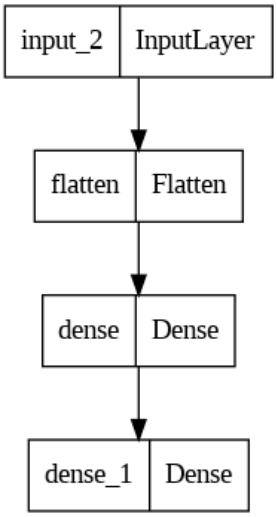

Figure 2. Representation of ResNet50 Model

Following the architecture definition, the model is compiled with the Adam optimizer, which adapts the learning rate during training, minimizing the loss function. The loss function chosen is sparse categorical cross-entropy, suitable for multi-class classification tasks with integer labels. Additionally, the accuracy metric is specified to monitor the model's performance during training.

For model training and evaluation, the training features and labels obtained previously are used. The model is trained using the training data in batches of size 32. After training, the model's performance is evaluated using the test features and labels. Due to computational limitations, only a portion of the training data could be used. Initially, 1000 training images were fed to the model with 50 epochs, resulting in a training accuracy of 9.89% and a test accuracy of 1.04%. Then, an attempt was made to increase the training data by feeding 2000 images with 1000 epochs, yielding a training accuracy of 92.4% and a test accuracy of 23.96%. Unfortunately, it was not possible to feed more data or increase the computational workload further, as the system (Colab) crashed under the heavier load.

b. VGG19:

Colab Link: https://colab.research.google.com/drive/1-N4V5SNR140Mlqf6o7Pb-5tl8PXYEJP1?usp=sharing

The methodology involved the utilization of the VGG19 model as a base for our task. The pre-trained VGG19 model was employed with the following configurations: the top layers were excluded (making it headless), and weights were initialized with the "imagenet" dataset.

To obtain the embeddings from the VGG19 model, the model was used to predict the training images. The output embedding has the size of (7, 7, 512).

A custom model was then created, starting with an input layer of the same shape as the embeddings. The subsequent layers included a global average pooling layer to reduce spatial dimensions, followed by three dense layers with respective activation functions of ReLU. Dropout layers were incorporated between the dense layers to reduce overfitting. Finally, the output layer consisted of a dense layer with a softmax activation function, producing probabilities across 120 classes.

The model was compiled with a sparse categorical cross-entropy loss function, Adam optimizer with a learning rate of 0.0001, and accuracy as the evaluation metric.

The training process was conducted by fitting the model on the embeddings obtained earlier and the corresponding label-encoded training data. The training was performed for 50 epochs with a batch size of 32 and a validation split of 20% for monitoring the model's performance. The training history, including loss and accuracy metrics, was recorded in the "history" variable for further analysis. The result is plotted below.

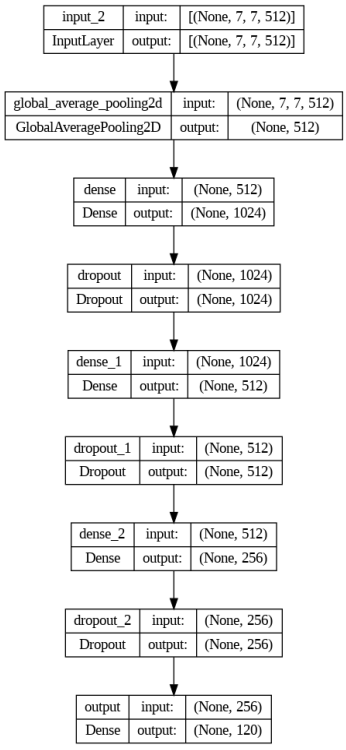

Figure 3. Representation of VGG19 Model

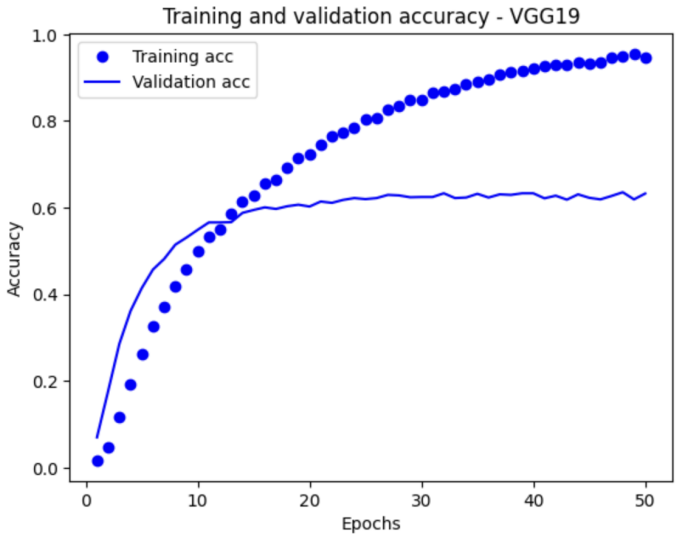

Figure 4. Accuracy Comparison for VGG19

3. Vision Transformer:

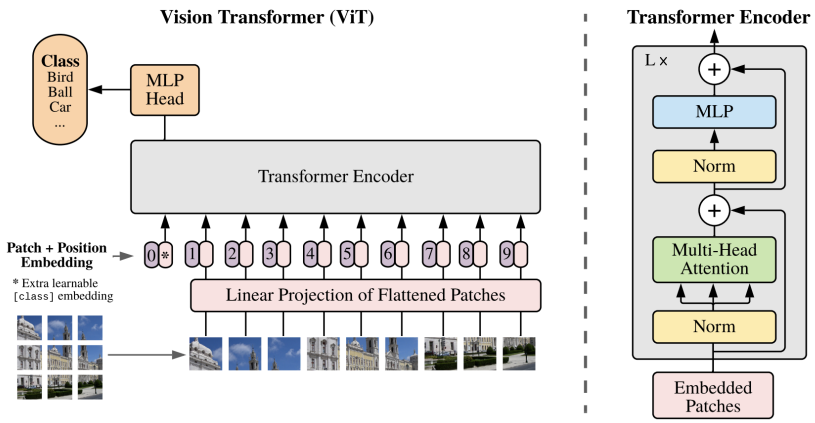

Figure 5. Vision Transformer Architecture (Developed by Google)

A pre-trained vision transformer developed by Google was imported through Hugging Face [2] and fine-tuned. This vision transformer was originally trained on 14 million images with 21,843 classes. Once imported, the vision transformer was fine-tuned on our dog breed training set. In contrast to the output layer shown above, the number of classes was also modified to be 120 to match the number of dog breed labels.

The transformer first splits the image into patches of size 16x16. Each patch is then fed through a linear projection layer, which maps the RGB values of the patch to an embedding. The resulting embeddings are then combined with learnable positional embeddings, which provide the model with spatial information about the patch's location within the image.

Next, the embeddings are fed through 12 subsequent transformer encoders. Each encoder consists of two sub-layers: a multi-head self-attention layer and a position-wise fully connected feed-forward network, as shown in the figure above. The self-attention layer allows the model to attend to different parts of the image when encoding each patch's information, while the feed-forward network provides non-linear transformations to the embeddings. Additionally, each encoder contains a residual connection and a layer normalization operation, which helps the model avoid the vanishing gradient problem during training.

The output embeddings after passing through the transformer encoders are aggregated by a mean pooling operation along the patch dimensions, resulting in a single embedding for the entire image. This embedding is then fed through a linear projection layer to map it to the number of output classes, followed by a softmax activation function to generate class probabilities.

Colab Link: https://colab.research.google.com/drive/195oyKOlM-_wymqBB6sXc6ZlXnw3APxwp?usp=sharing

Section IV: Results

MODEL	TRAIN	TEST
Baseline (Random Guess)	0.8%	
Vanilla CNN	20.9%	7.1%
ResNet50	92.4%	24.0%
VGG19	94.6%	60.9%
Vision Transformer	98.4%	90.7%

Table 1. Performance Comparison Across Models

Section V: Discussion and Lessons Learned

The initial attempts using a simple CNN model proved inadequate in capturing the intricate complexities of the task at hand. To overcome this limitation, we turned to the powerful technique of transfer learning by leveraging pre-trained models. This approach allowed us to significantly enhance the model's performance.

In our comparative analysis of the ResNet50 and VGG19 models, we initially expected similar results. Surprisingly, the VGG19 model outperformed the ResNet50 model by a substantial margin. Upon closer examination, we discovered that the key difference between the two models lies in the treatment of their respective embeddings. In the ResNet50 model, the embedding is flattened, while in the VGG19 model, max-pooling is applied to the embedding. This observation highlights the critical role played by the choice of appropriate layers in influencing model performance. By incorporating max-pooling, we were able to capture the relationships between the same block for different filters within the image effectively. Conversely, flattening the embedding resulted in a loss of this valuable relationship information.

Notably, the Transformer model surpassed all other models in performance. This superiority can be attributed to the presence of multiple objects within our images. For instance, an image may contain both humans and dogs. The Transformer approach addresses this challenge by dividing the image into patches and selectively attending to specific patches. In this manner, it can prioritize attending to patches containing dogs rather than humans, thereby optimizing its object recognition capabilities. However, the transformer also had some shortcomings. In general, the model misclassified images where even differentiating between some of the dog breeds proved difficult even through manual inspection. An example is shown in Figure 6 below, where the fur colour and features also resemble that of a golden retriever, which the model predicted. Additionally, there were also edge cases where the dog itself was cropped out of the image entirely, as shown in Figure 7, during pre-processing. Hence, the misclassification issues demonstrated by the transformer are unsurprising upon further analysis.

Figure 6. An Example of a “Hard-to-Classify” Image

Figure 7. An Example of the Dog Being Cropped Out

Overall, our findings emphasize the significance of selecting the appropriate pre-trained model and the importance of understanding the model's internal mechanisms to maximize performance on complex tasks involving multiple objects within images.

Figure 8. Training Images Containing Humans or Other Object

Reference

[1] https://www.kaggle.com/competitions/dog-breed-identification/data?select=train
[2] https://arxiv.org/pdf/2010.11929.pdf
[2] https://huggingface.co/google/vit-base-patch16-224