USC Marshall

School of Business

Deep Learning Model for Dog Breed Classification

Kaiyi (Katherine) Wang, Manhui Zhu, Wenyi Lin

DSO 569 - Deep Learning for Business Applications

May 6, 2024

Introduction

We are conducting dog breed classification using a dataset from Kaggle, which includes 120 dog breeds. This task presents significant challenges due to the limited number of samples available for each breed. To address this, we have developed a model using the TensorFlow framework and have utilized two highly regarded pre-trained image classifiers: ResNet-101 and Inception V3. We have also experimented with Vision Transformer. Our project notebook is powered by a Nvidia A100 GPU to enhance processing speed. This project is particularly valuable not only for its ability to deepen our understanding of leveraging and refining image classification models using pre-trained networks but also for its practical implications in areas such as dog license registration, animal shelter adoption processes, and educational programs about dog breeds.

Why This Question is Important

Identifying dog breeds through visual inspection can be challenging, even for experts. Manual classification is prone to errors with over 340 breeds recognized by various kennel clubs and a considerable degree of variation within each breed. Automating this process with machine learning models introduces several advantages. First, these models can consistently analyze visual features, which enhances accuracy beyond what human experts can achieve. Additionally, automated systems can efficiently handle large volumes of data, making them particularly suitable for processing thousands of images. Moreover, such tools increase accessibility for pet owners, veterinarians, and animal rescue organizations, enabling them to quickly and accurately identify dog breeds. Although our analysis did not fully resolve the issue, it provides a valuable starting point.

Dataset Overview

The dataset from Kaggle comprises two subsets: a training set with 10,222 labeled images, each associated with the corresponding dog breed, and a test set containing 10,357 unlabeled images. For our project, we are utilizing only the labeled images from the training set. These have been further divided into training, validation, and testing subsets, with a distribution ratio of 70/20/10, respectively. The labels for each image, including the breed and a unique image identifier, are stored in a CSV file. The dataset encompasses a total of 120 dog breeds. The most well-represented breeds have up to 126 image samples each, while the least represented breeds have as few as 66 image samples (Figure 1.1). Certain breeds are strikingly similar, making them difficult to distinguish, even for human observers. For instance, Norfolk Terriers and Norwich Terriers are two breeds that are often confused due to their close resemblance(Figure 1.2).

Dataset Preprocessing

First, we import a dataframe from a CSV file that contains breed labels and image identifiers. We then encode the labels using scikit-learn's LabelEncoder, which converts the breed names from strings to integers. To partition the data into training, validation, and testing sets, we apply the *train_test_split* method twice (Figure 2.1), adhering to the previously mentioned ratios. The splits are stratified to address the imbalance among the different classes.

Through the TensorFlow API, we utilize the *Dataset.from_tensor_slices* method to create three datasets consisting of images and labels. This approach helps in loading images on the fly, significantly reducing memory usage. At this stage, the images undergo minimal preprocessing; the only steps taken are reading the image from its file path and decoding the JPEG image. This design is intended to increase the flexibility of image preprocessing to accommodate various pre-trained models.

We further preprocess the images for each model by resizing, normalizing, and applying model-specific preprocessing steps. Additionally, we enhance the datasets and efficiency by batching, shuffling, and prefetching. We opt for a high buffer size of 7,500 to shuffle the elements of the train dataset. Although this uses more memory, it increases the randomness of the images. It is important to note that shuffling is only performed on the training set.

Resnet

The ResNet model is a convolutional neural network (CNN) developed with residual blocks and shortcut connections (Figure 3.1), ResNet addresses the vanishing gradient problem by allowing gradients to bypass certain layers. This facilitates the training of substantially deeper networks than previously possible without performance degradation.

Inception V3

The Inception V3 model is a CNN known for its high performance and features a unique architecture with factorized convolutions that break down larger filters into smaller, more manageable ones (Figure 3.2). The model's architecture, pre-trained on the ImageNet dataset, efficiently handles spatial dimensions and processes image details at multiple scales through its multi-path design, making it well-suited for environments with strict memory and computational constraints.

Vision Transformer (ViT)

Vision Transformer (ViT) is a CNN architecture introduced in 2021. It utilizes transformer and attention mechanisms that are commonly found in Natural Language Processing (NLP), to extract image information effectively. The figure 3.3 shows a demo of ViT model. Unlike traditional CNNs, ViT represents input images as a sequence of patches, like word embeddings in text transformers. Each patch is flattened into a vector and linearly projected to the desired input dimension. ViT doesn't enforce sequence order naturally; hence, Position Encodings establish order. The ViT encoder comprises Layer Norm, Multi-head Attention Network (MSP), and Multi-Layer Perceptrons (MLP). Layer Norm adapts to image variations, MSP generates attention maps focusing on informative parts, and MLP acts as a two-layer classification network, employing GELU activation. The output of MLP, after softmax, provides classification labels based on special (class) embeddings.

Experiment

In our models, both of which incorporate the ResNet-50 and Inception-V3 architectures, we have employed a similar shallow classifier design. Instead of doing image augmentation during the data loading and preprocessing phase, we have introduced augmentation layers directly at the beginning of the model, before the input reaches the pretrained networks. We experimented with various augmentation techniques including random horizontal and vertical flips, translations, rotations, zooms, and adjustments in contrast and brightness. However, incorporating all these augmentations resulted in a significant drop in accuracy across training, validation, and testing phases. To avoid over-augmentation, we opted to limit the augmentation methods to just random rotations and flips. The augmentations applied vary with each epoch due to randomness, providing vital variability especially given the limited data available per class. Similar to dropout layers, neither the validation nor the testing datasets went through these augmentation layers.

We conducted experiments with batch sizes of 32, 64, and 128, a batch size of 128 gave its slightly superior performance. We also experimented with various configurations of the shallow classifier, adjusting the number of dropout layers, the dropout intensity, the number of densely connected layers, and the number of neurons per layer. The shallow classifier began with a 2D global average pooling layer, which aggregated information and converted the output of the pretrained model from a tensor to an array. All densely connected layers used *ReLU* as the activation function, and the output layer employs a softmax function with 120 neurons (Figure 4.1, 4.2).

In the model with ResNet-101, we achieved an accuracy of 70.58% on the testing dataset. From Figure 5.1, we can see that the validation accuracy converges around 12 epochs, while the training accuracy shows signs of convergence around 50 epochs. However, the validation accuracy remains similar despite the continued improvement in training accuracy. This indicates that the model is overfitting in the later epochs.

In the model with Inception-V3, we achieved an accuracy of 85.53% on the testing dataset. From Figure 5.2, we observe that the training accuracy never surpasses the validation accuracy. To determine if the training accuracy ever converges, we deliberately set a large patience value for early stopping. We notice that the validation accuracy fluctuates after 12 epochs, suggesting that the improvement in the training dataset is likely due to overfitting. To investigate the cause, we experimented with different random seeds during the train-test split to ensure that we did not accidentally create an unrepresentatively easy testing dataset. However, the issue persisted. We also carefully examined the splitting process to ensure that there was no information leakage during the training process. Overall, we conclude that this phenomenon is due to a more challenging training set, resulting from larger variations within the samples as well as the additional augmentation process.

For the Vision Transformer, we utilized a pre-trained ViT-B/32 model, freezing all parameters in the pre-trained transformer and only training the parameters in Multi-layer Perceptrons (MLP), which was the output classifier. Since images in our dataset were in different sizes, we first resized images into size 224 x 224 size, normalized them, and did the data augmentation in data loading and preprocessing phase. Unlike ResNet and Inception-V3, data augmentation significantly improved performance of ViT model. Then we patched images with size 32 x 32, each image had 7 x 7 = 49 patches (224/32 = 7). We used *RectifiedAdam* optimizer and learning rate of 0.001. The evaluation matrix was *accuracy*. When training the model, we used batch size = 128 and epoch = 50. By applying early stop with patience = 5, the training stopped at the 19-th epoch. Figure 5.3 shows training and validation accuracy. The highest validation accuracy is 71.05% and the test accuracy is 70.28%.

Compared to ResNet and Inception-V3, the model performance of ViT is lower. It may be due to insufficient data and computing power. Vision Transformer models often require a large amount of data for effective learning, but our training set only has more than 7000 images. Since ViT first patches the image and applies a transformer encoder, it is computationally expensive and might require extensive hyperparameter tuning. Even though we buy more GPUs to train our model, we don't have enough time to train for a sufficient number of epochs or with different learning rates, or even unfreeze some parameters in hidden layers and fine tune them. (It takes us more than 1 hour to train for 20 epoches). Figure 1.2 is two examples of image data in 2 classes, we can find the two dog breeds Norfolk Terrier and Norwich Terrier look very similar. Compared to ImageNet that used to train ViT-B/32, which images are pretty different from each class, our dataset is much harder for the ViT model to extract effective features to classify different breeds.

Conclusion

In conclusion, our exploration into dog breed classification using advanced deep learning models such as ResNet-101, Inception V3, and Vision Transformer has illuminated both the capabilities and limitations of these approaches within a practical context. While Inception-V3 demonstrated robust performance, owing to their sophisticated architectures that effectively handle deep network training and intricate image details, ResNet and Vision Transformer (ViT) lagged slightly, possibly due to the dataset's complexity and its smaller size relative to what ViT architectures typically require. These findings underscore the critical importance of dataset characteristics in model performance and highlight the necessity for further optimization and tuning of hyperparameters, especially when adapting models pre-trained on datasets like ImageNet to more specialized tasks. This project not only advances our understanding of leveraging pre-trained models for image classification but also contributes practical insights into the challenges of machine learning applications in real-world scenarios, such as dog breed identification which has significant implications for animal welfare and biological research.

References

Cukierski, W. (2017). Dog Breed Identification. Kaggle. Retrieved from https://kaggle.com/competitions/dog-breed-identification

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Retrieved from https://arxiv.org/abs/2010.11929

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2818-2826.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. Retrieved from https://arxiv.org/abs/1505.00393

Appendix

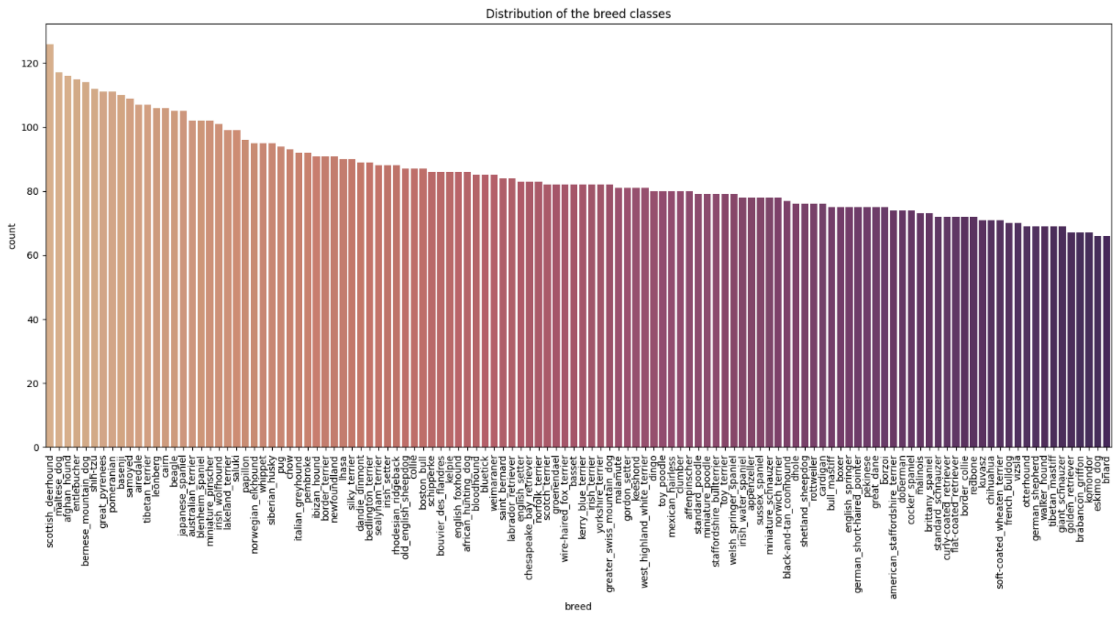


Figure 1.1 Histogram of Dog Breeds

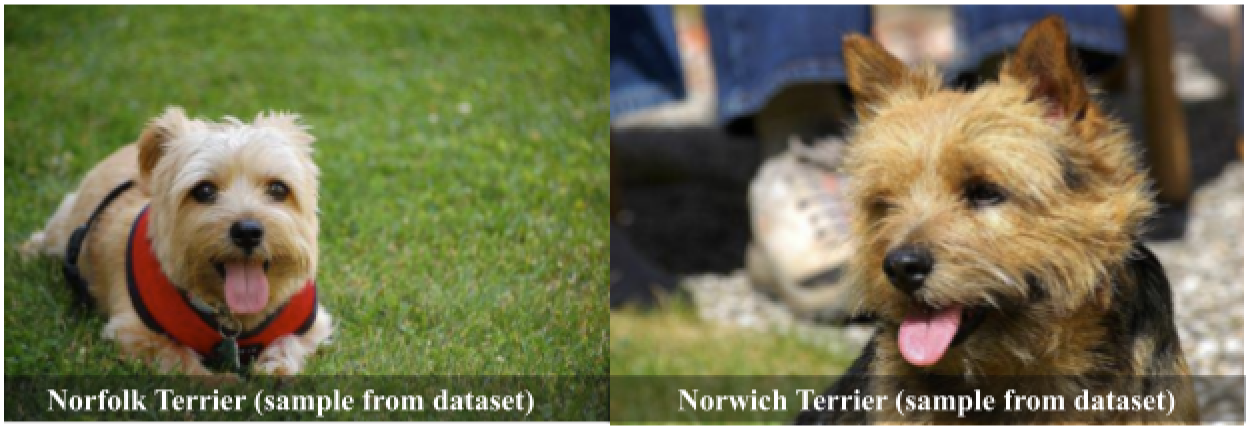
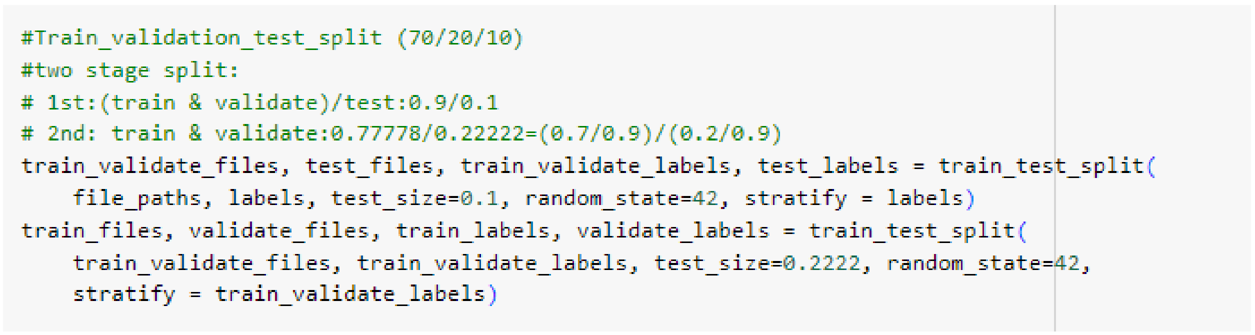

Norfolk Terrier (sample from dataset)
Norwich Terrier (sample from dataset)

Figure 1.2 Dataset Sample



```
#Train_validation_test_split (70/20/10)
#two stage split:
# 1st:(train & validate)/test:0.9/0.1
# 2nd: train & validate:0.77778/0.22222=(0.7/0.9)/(0.2/0.9)
train_validate_files, test_files, train_validate_labels, test_labels = train_test_split(
    file_paths, labels, test_size=0.1, random_state=42, stratify = labels)
train_files, validate_files, train_labels, validate_labels = train_test_split(
    train_validate_files, train_validate_labels, test_size=0.2222, random_state=42,
    stratify = train_validate_labels)
```

Figure 2.1 Data Split

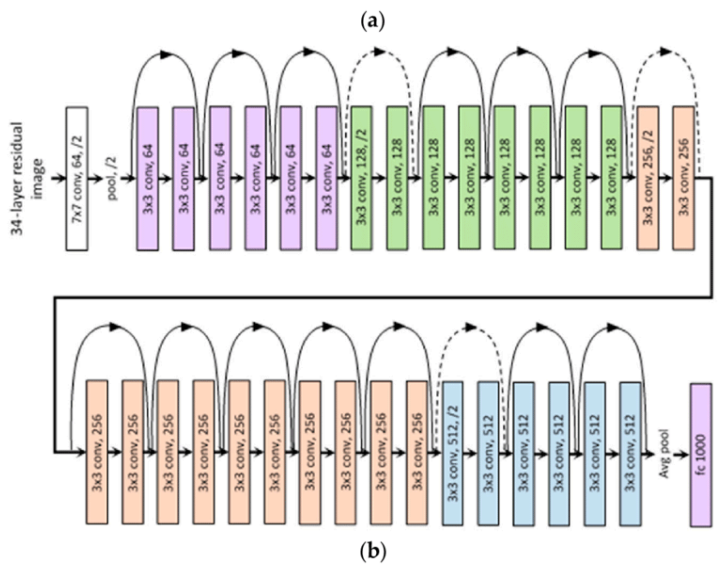


Figure 3.1 Model Architecture of ResNet

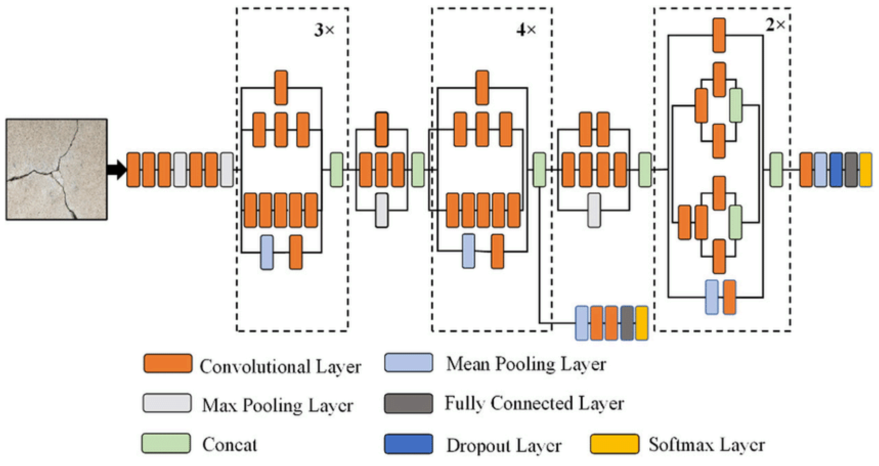


Figure 3.2 Model Architecture of Inception-V3

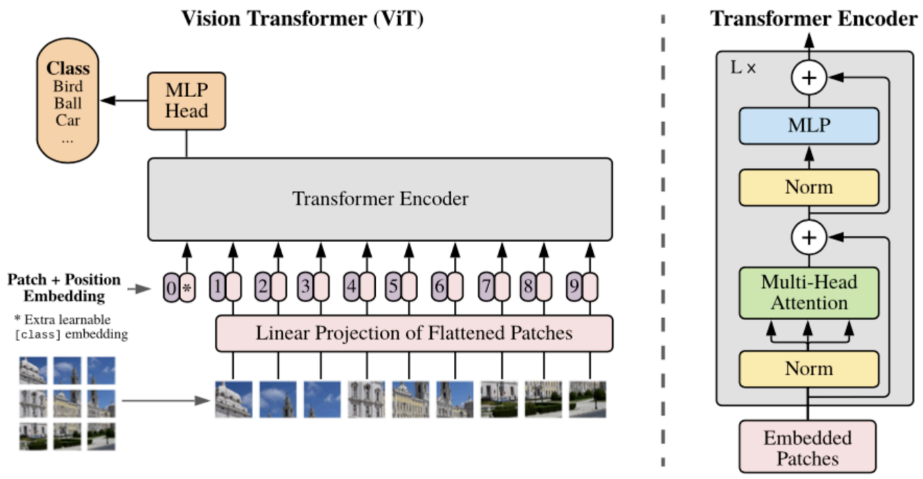
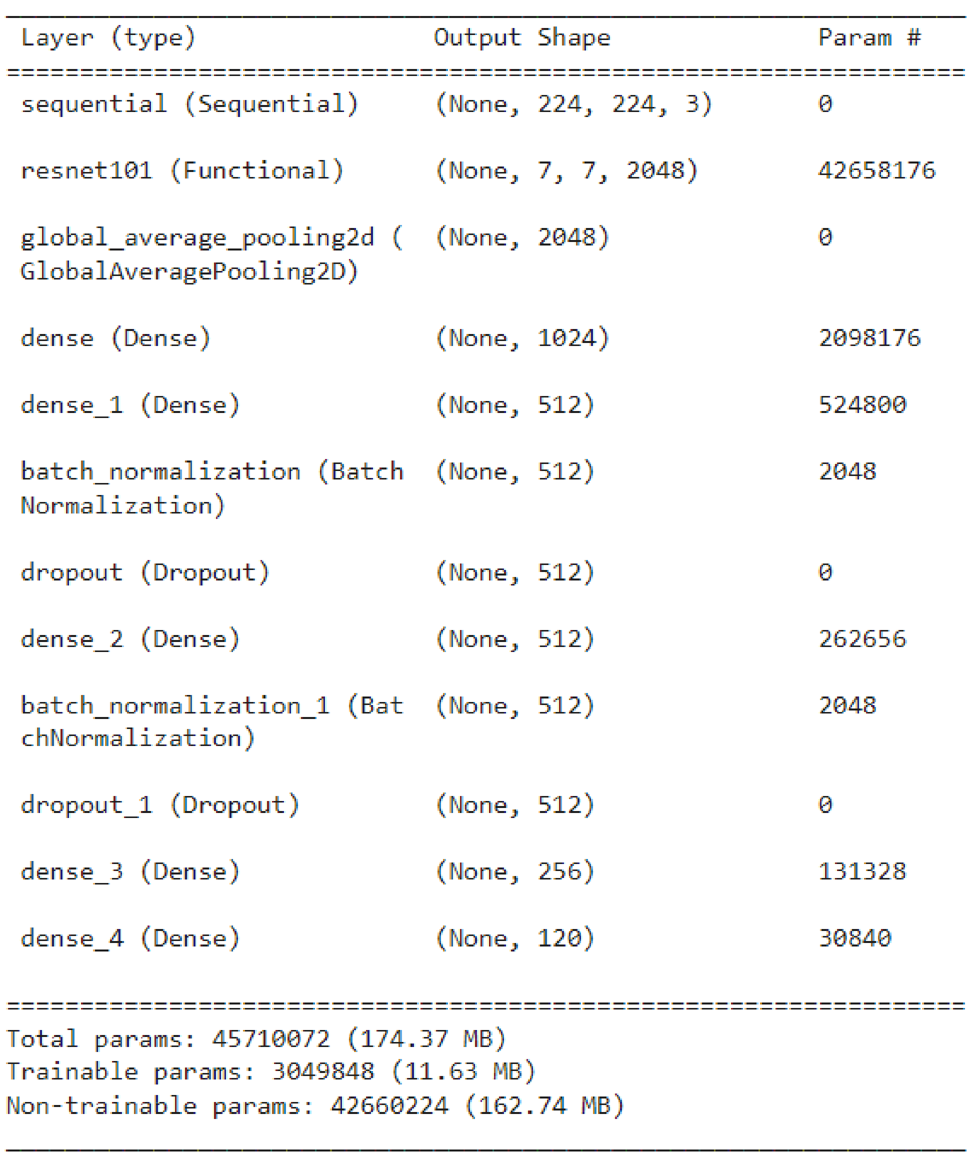
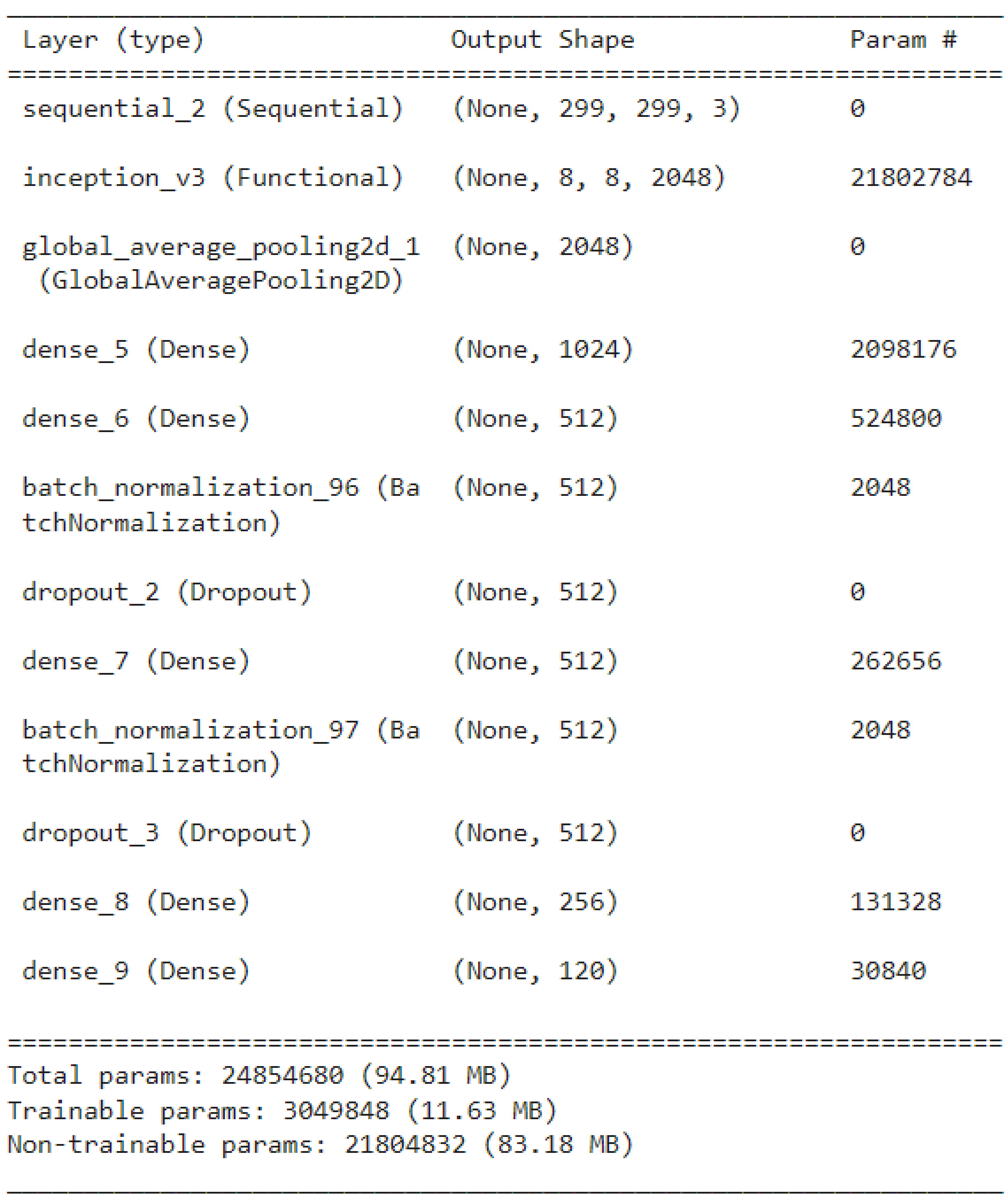


Figure 3.3 Demo of Vision Transformer



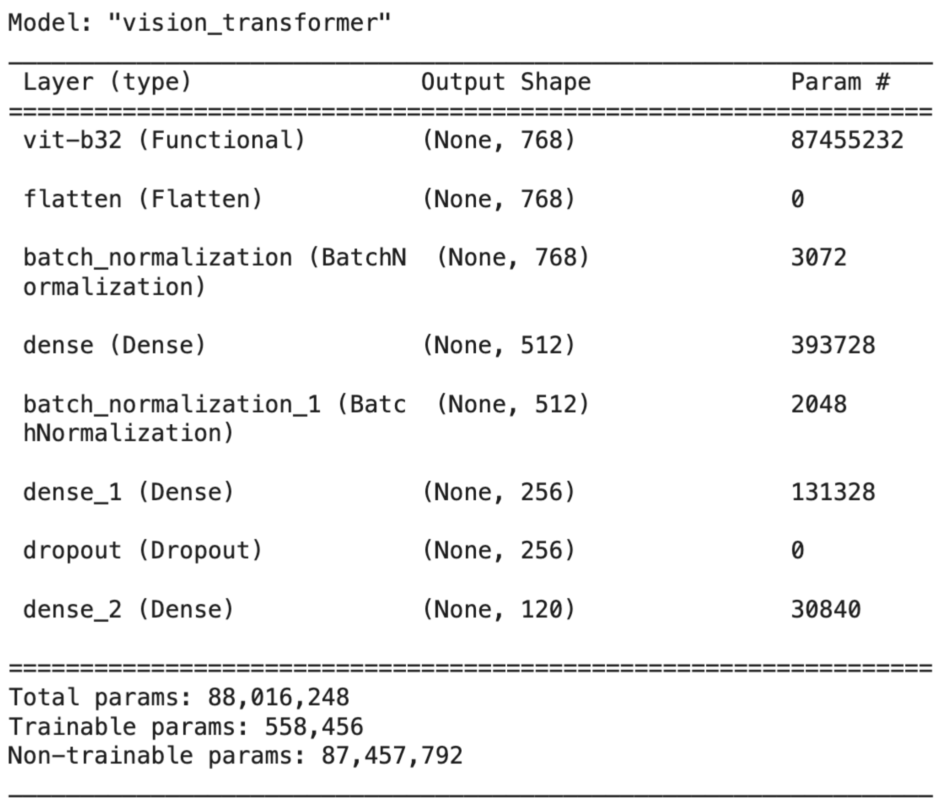
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 resnet101 (Functional)      (None, 7, 7, 2048)        42658176

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 1024)              2098176

 dense_1 (Dense)             (None, 512)               524800

 batch_normalization (Batch  (None, 512)               2048
 Normalization)

 dropout (Dropout)           (None, 512)               0

 dense_2 (Dense)             (None, 512)               262656

 batch_normalization_1 (Bat  (None, 512)               2048
 chNormalization)

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 256)               131328

 dense_4 (Dense)             (None, 120)               30840

=================================================================
Total params: 45710072 (174.37 MB)
Trainable params: 3049848 (11.63 MB)
Non-trainable params: 42660224 (162.74 MB)
_________________________________________________________________
```

Figure 4.1 Model Summary of ResNet-101



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_2 (Sequential)   (None, 299, 299, 3)       0

 inception_v3 (Functional)   (None, 8, 8, 2048)        21802784

 global_average_pooling2d_1  (None, 2048)              0
  (GlobalAveragePooling2D)

 dense_5 (Dense)             (None, 1024)              2098176

 dense_6 (Dense)             (None, 512)               524800

 batch_normalization_96 (Ba  (None, 512)               2048
 tchNormalization)

 dropout_2 (Dropout)         (None, 512)               0

 dense_7 (Dense)             (None, 512)               262656

 batch_normalization_97 (Ba  (None, 512)               2048
 tchNormalization)

 dropout_3 (Dropout)         (None, 512)               0

 dense_8 (Dense)             (None, 256)               131328

 dense_9 (Dense)             (None, 120)               30840

=================================================================
Total params: 24854680 (94.81 MB)
Trainable params: 3049848 (11.63 MB)
Non-trainable params: 21804832 (83.18 MB)
_________________________________________________________________
```

Figure 4.2 Model Summary of Inception-V3



```
Model: "vision_transformer"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vit-b32 (Functional)        (None, 768)               87455232

 flatten (Flatten)           (None, 768)               0

 batch_normalization (BatchN  (None, 768)              3072
 ormalization)

 dense (Dense)               (None, 512)               393728

 batch_normalization_1 (Batc  (None, 512)              2048
 hNormalization)

 dense_1 (Dense)             (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 120)               30840

=================================================================
Total params: 88,016,248
Trainable params: 558,456
Non-trainable params: 87,457,792
_________________________________________________________________
```

Figure 4.3 Model Summary of Vision Transformer (ViT)

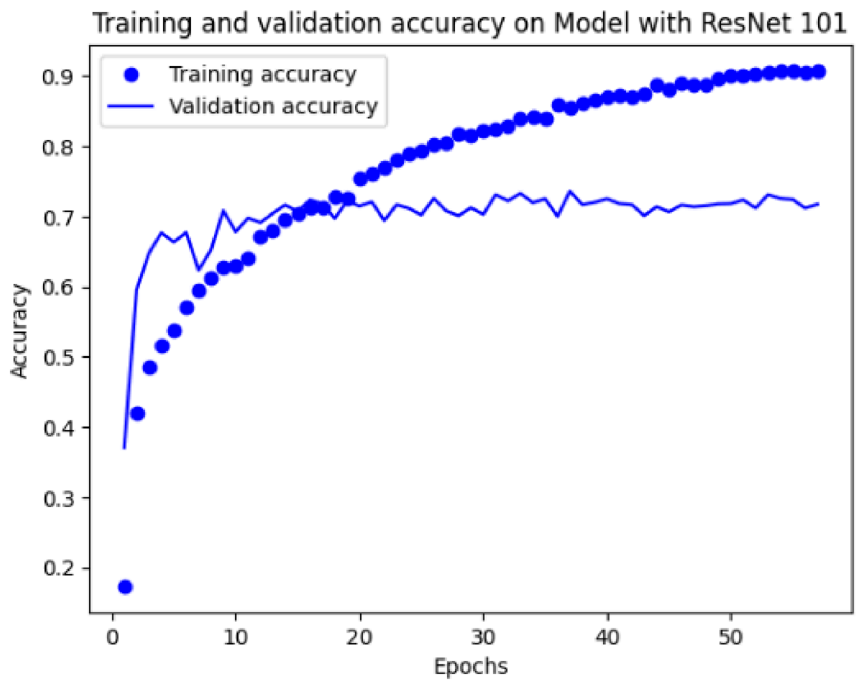


Figure 5.1 Training and Validation Accuracy on Model with ResNet 101

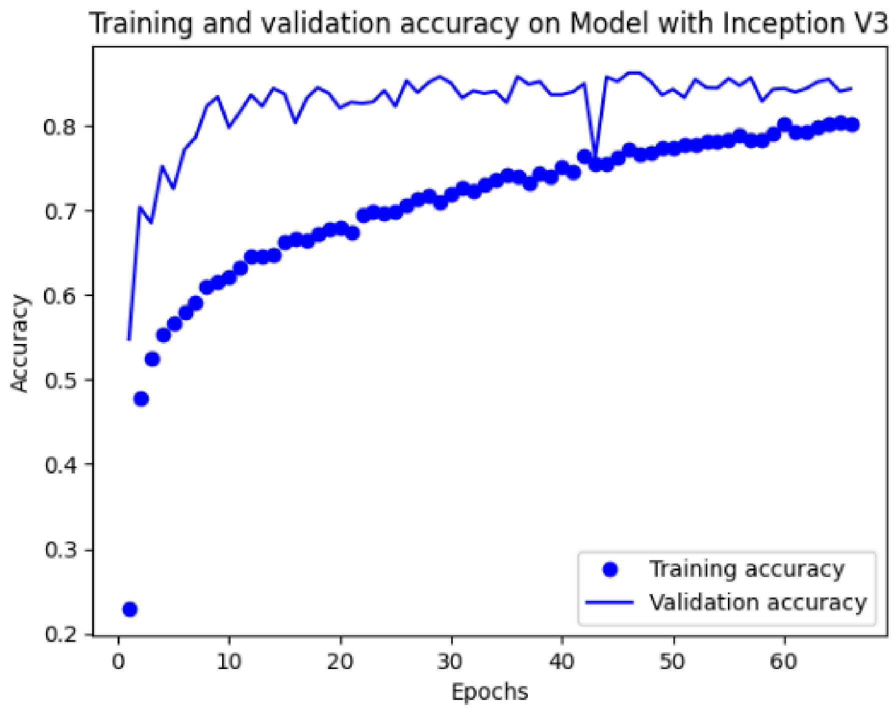


Figure 5.2 Training and Validation Accuracy on Model with Inception V3

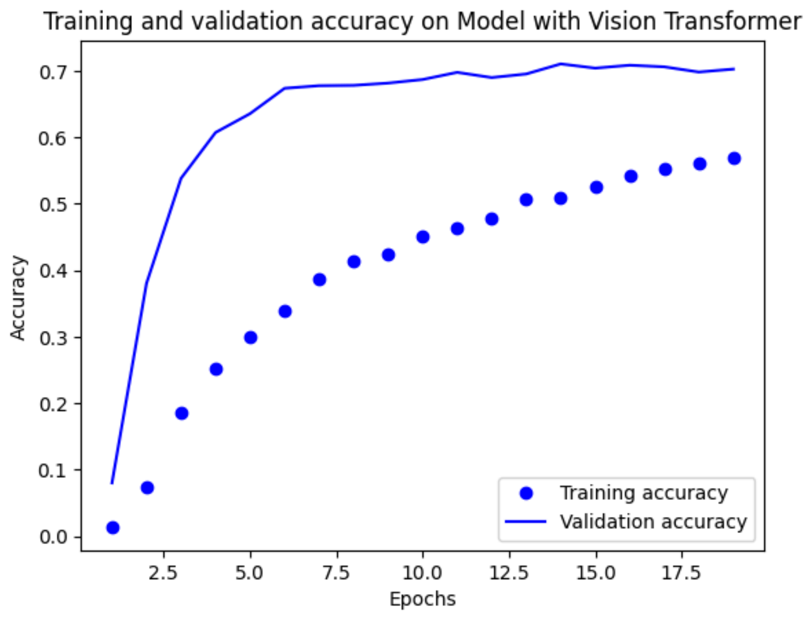


Figure 5.3 Training and Validation Accuracy on Model with Vision Transformer (ViT)