**INTERNATIONAL UNIVERSITY**

**VIETNAM NATIONAL UNIVERSITY, HCM CITY**

**School of Computer Science & Engineering**

Ảnh có chứa vòng tròn, biểu tượng, Nhãn hiệu, Phông chữ

Mô tả được tạo tự động

**PROJECT 1**

**Lecturer: Dr. Ly Tu Nga**

**Course: IT159 Artificial Intelligence**

**Leveraging Pre-trained CNNs for Dog Breed Identification:**

**A Transfer Learning Approach on the Stanford Dogs Dataset**

**Group 3**

|  |  |
| --- | --- |
| Nguyễn Hùng Quốc | ITCSIU21101 |
| Nguyễn Triều Vỹ | ITCSIU21157 |
| Phạm Nguyễn Tiến Đạt | ITDSIU21080 |

**Ho Chi Minh, June 2024**

**CONTENTS**

[**I – INTRODUCTION**](https://docs.google.com/document/d/178VD5bHHZ4PzN54R6OIV3yJdexY5ehnxXlLV8fOfs_w/edit#heading=h.3znysh7)

[**II – DATASET AND PREPROCESSING**](https://docs.google.com/document/d/178VD5bHHZ4PzN54R6OIV3yJdexY5ehnxXlLV8fOfs_w/edit#heading=h.2et92p0)

1. **Dataset**
2. **Preprocessing** 
   1. **Cropping, Resizing and Normalization**
   2. **Split Training/Validation/Testing**
   3. **Data Generator**

[**III – METHODOLOGY**](https://docs.google.com/document/d/178VD5bHHZ4PzN54R6OIV3yJdexY5ehnxXlLV8fOfs_w/edit#heading=h.tyjcwt)

**IV – EXPERIMENT AND RESULTS**

**V – CONCLUSION**

[**REFERENCES**](https://docs.google.com/document/d/178VD5bHHZ4PzN54R6OIV3yJdexY5ehnxXlLV8fOfs_w/edit#heading=h.1t3h5sf)

**List of figures**

**Figure 2.1.a.** 120 breeds of Stanford Dog Dataset

**Figure 2.1.b.** 12 random breeds

**Figure 2.2.a.** Image Classification Process

**Figure 2.2.b.** Special case in cropping images

**Figure 2.2.c.** Display some images after cropping

**Table 1.** Model Architecture

**Figure 3.2.a.** Layers at the end of VGG19

**Figure 3.2.b.** Layers at the end of ResNet152V2

**Figure 3.2.c.** Layers at the end of InceptionResNetV2

**Figure 3.2.d.** Layers at the end of DenseNet201

**Figure 3.** General CNN Architecture

**Table 3.** Performance Metrics

**Figure 4.** Loss and Accuracyof each models

**Figure 4.2.a** VGG19 Confusion Metric

**Figure 4.2.b** ResNet152V2 Augmentation Confusion Metric

**Figure 4.2.c** InceptionResNetV2 Augmentation Confusion Metric

**Figure 4.2.d** DenseNet201 Augmentation Confusion Metric

**I. Introduction:**

For this project, we chose to approach it from an image processing perspective.

The primary objective of this project is to train neural network models to predict dog breeds through transfer learning, combining deep learning models with CNN architectures as their base. Instead of creating convolutional neural networks (CNNs) from scratch, we used transfer learning with pre-trained models known for their robust feature extraction capabilities.

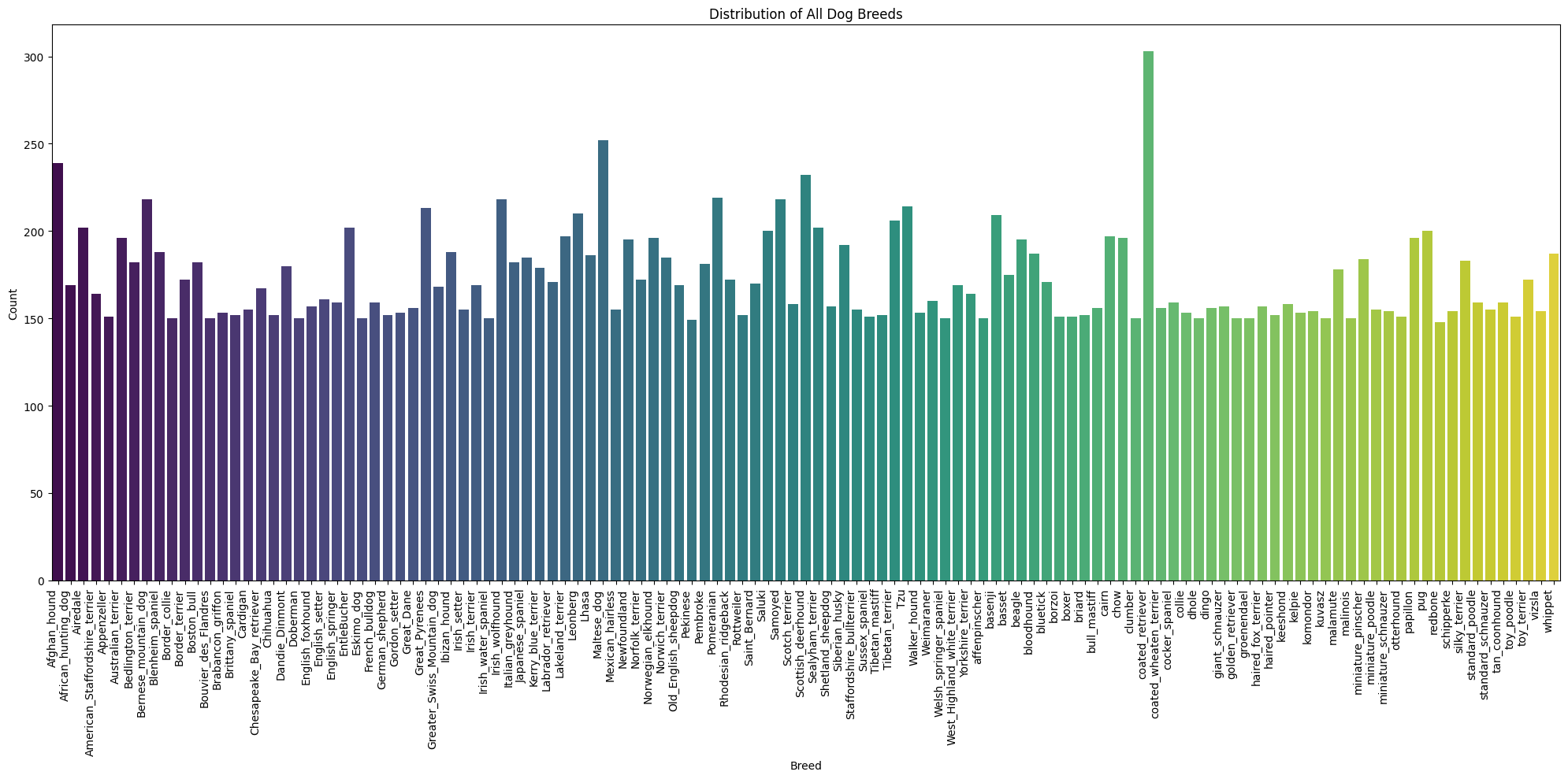
The study focuses on neural networks and how to train them for image classification tasks. We specifically want to examine the performance of ResNet152V2, VGGNet19, InceptionResNetV2, and DenseNet201 in distinguishing different dog breeds. By studying these models, we hope to understand and learn more about the most effective architecture for image processing while also learning about the intricacies of neural networks and their training methods for image processing applications.

Through this investigation, we seek to determine the best model architecture for accurately classifying dog breeds from photographs, thereby improving our understanding of transfer learning and the practical application of deep learning models in the field of image identification.

**II. Dataset and Preprocessing**

**1. Dataset**

For this project, we are using the Stanford Dogs Dataset, which includes photos of 120 distinct dog breeds from around the world. The dataset contains 2 main folders: ‘Images’ and ‘Annotations’.

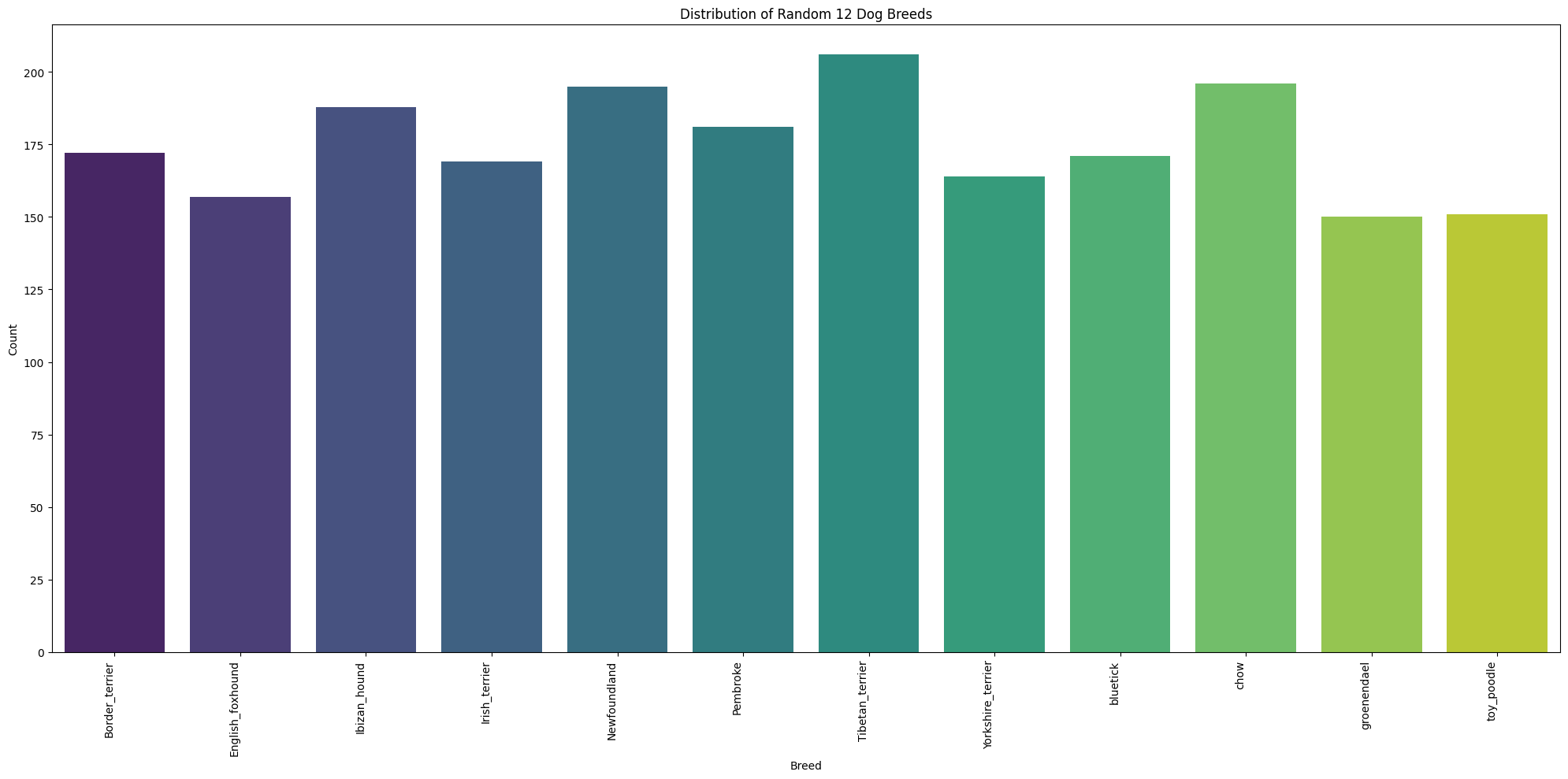
* Images count: 20,580.
* Classes: 120.

**Figure 2.1.a.** 120 breeds of Stanford Dog Dataset

The annotation files contain the labels for the dog breeds and coordinates for their bounding box.

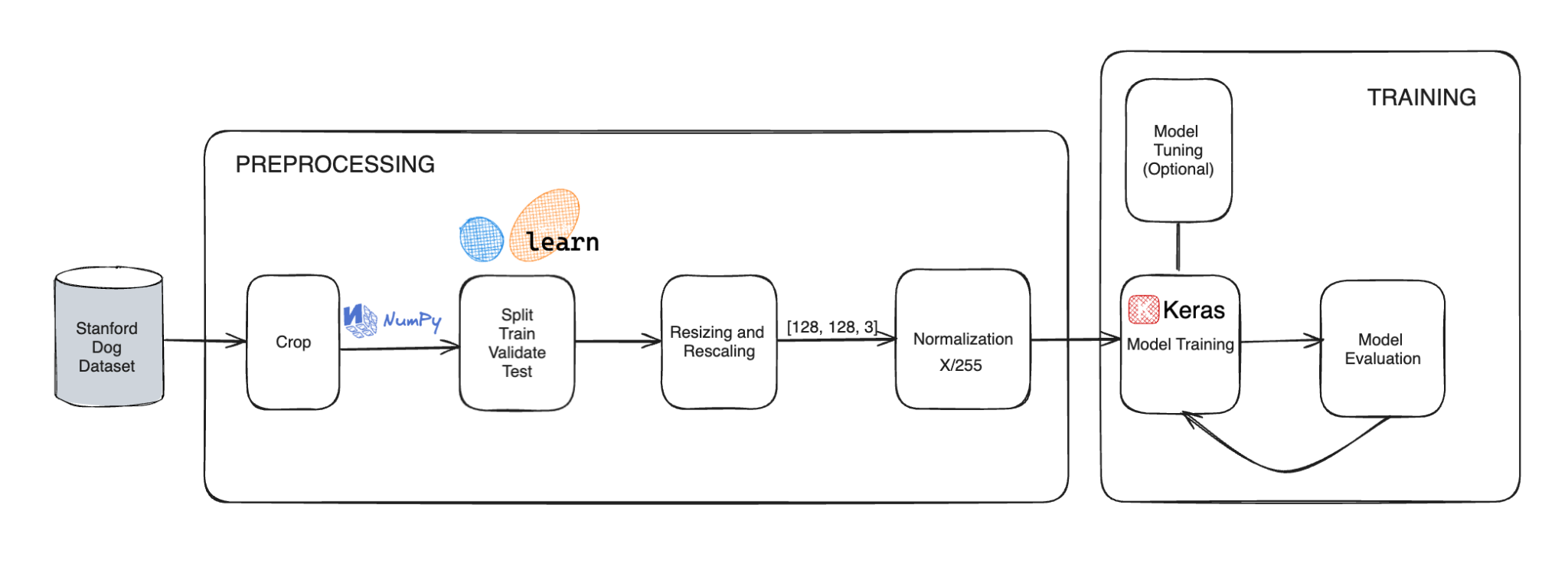
During the process of training this dataset, we realized that the sheer size of this dataset introduced various problems that led to overall poor performances across all models. Additionally, training a dataset with 120 classes proved to be too much for our computer’s performance. Thus, in order to maintain a reasonable level of consistency across our team’s computers, we decided to reduce the number of classes to train to 12 classes.

* Images count: 2221.
* Classes: 12.



**Figure 2.1.b.** 12 random breeds

1. **Preprocessing**



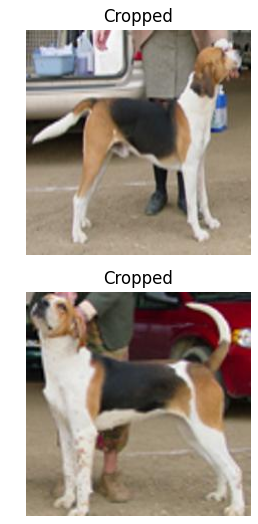
**Figure 2.2.a.** Image Classification Process

**a. Cropping, Resizing and Normalization**

**Cropping**

One of the challenges of this dataset is that some images contain not only the dogs but also other unrelated objects like humans, trees, obstructions, etc. One approach we initially considered using was ‘Object Detection’ for identifying where the dogs are in the images. However, the annotation files already contain information for the bounding boxes to detect where the target is. This means we only need to extract the information about the bounding boxes to properly crop the images.

There is one particular scenario where the image contains more than one dog; the bounding box solves this by dividing separate bounding boxes for each dog that is in the image.



**Figure 2.2.b.** Special case in cropping images

**b. Resizing and Normalization**

**Resizing and rescaling**

Standardizing the image dimensions to 128x128 guarantees uniformity throughout the dataset. This consistency is critical for compatibility with deep learning models such as VGG19, ResNet, Inception, and DenseNet, which require set input sizes. By scaling all photos to the same dimensions, we ensure that the input tensors have the expected shape, preventing compatibility difficulties during model training and inference. Additionally, we found out that working with smaller-sized images facilitates faster training iterations and enhances the scalability of the machine learning pipeline. This makes it more feasible to train deep learning models with limited computational resources.

**Normalization**

We also applied ‘Normalization’ to the dataset, normalizing pixel values is critical for stabilizing the training process and increasing convergence rates during model optimization. Normalization promotes numerical stability by scaling pixel intensities to a defined range; specifically, we ensured that it was within the [0,1] range. The formula is as follows:

= ​

Each pixel has 3 channels: red, blue, and green, and their values range from 0 to 255. For example, let's consider an original pixel with RGB values (100, 200, 50). The normalized output would be (0.392, 0.784, 0.196).

**Split Train/Validate/Test**

When it comes to splitting the dataset into train, validate, and test sets, the original dataset already has a dedicated testing set aside from the main set. This means we would only need to divide the dataset into training and validation, which initially we did, with an 8/2 split.

However, after scaling down the dataset to 12 classes, we decided to split the set into train/validate/test like normal. This is because we are strictly training the models to predict the 12 random breeds of dogs. The testing set not only contains all of the original 120 breeds, but they are also not organized the same way the main set was; this makes the original splitting scheme a lot more tedious to set up without much benefit. Thus, we decided to split the dataset as follows:

*train\_split\_percent = 0.7*

*validation\_split\_percent = 0.15*

*test\_split\_percent = 0.15*

**Data Generator**

Our first attempt at data preparation was non-parallel, which led to extremely poor performance and hours of training time for each model. We created a function to create data generators that expedite the process of providing data for the training, validation, and test sets in order to overcome this inefficiency. These generators guarantee that big datasets are handled well and that they work with TensorFlow's data pipeline.

Our data generators are designed to incorporate aforementioned preprocessing steps, including resizing images and normalization. The construction of these generators involves creating TensorFlow Datasets for the training, validation, and test sets, ensuring that the data is efficiently handled and preprocessed.

**c. Creating TensorFlow Datasets**:

We create TensorFlow Datasets using tf.data.Dataset.from\_tensor\_slices for the training, validation, and test sets.

**d. Parallelizing Preprocessing**:

We apply preprocessing steps, such as loading images and normalization, to the data in parallel using the map function with num\_parallel\_calls=AUTOTUNE. This technique optimizes the preprocessing performance by utilizing multiple CPU cores, which speeds up the data preparation process.

**e. Shuffling and batching**:

For the training and validation sets, we shuffle the data to ensure diverse batches, which helps in better model generalization. The data is then batched into smaller subsets to fit into memory during training.

Overall, after applying all the techniques and preprocessing steps mentioned, we were able to successfully preprocess the dataset much more efficiently than before. There is a starking contrast between the performance of our old approach of the preprocessing pipeline and this newly modified approach. This improvement helped us gain a new insight into which aspect is important and affects the performance of training models the most. Here is a a visualization of the all preprocessing steps applied to the dataset:

 **Figure 2.2.c.** Display some images after preprocessing

**III. Methodology**

**1. Transfer Learning Approach**

In this study, we leverage transfer learning to classify dog breeds using the Stanford Dogs Dataset. Transfer learning enables the utilization of pre-trained models, which have already learned robust feature representations from large-scale datasets such as ImageNet. By fine-tuning these models on the target dataset, we aim to achieve higher performance with reduced training time and improved generalization.

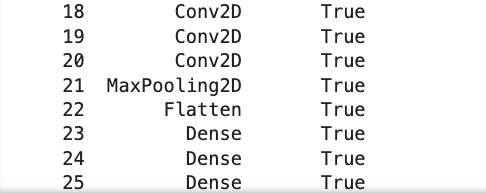
**2. Model Architecture**

We employ four convolutional neural network architectures for this task:

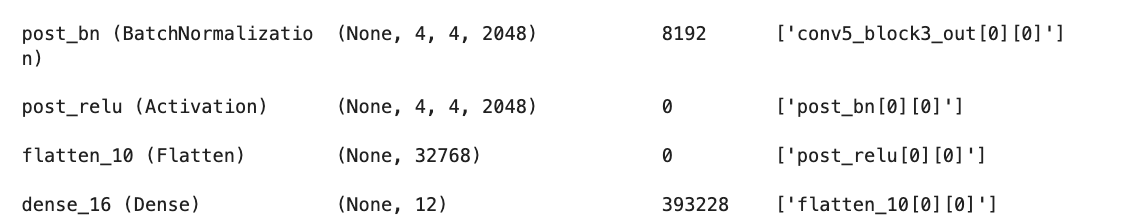
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Layers** | **Filter Size** | **Parameters** | **Description** |
| VGG19 | 19 | 3x3 | 29.50M | Simple design, popular choice |
| ResNet152V2 | 152 | Variable (Residual Connections) | 58.70M | Balanced performance, efficient |
| InceptionResNetV2 | 572 | Mixed  (1x1, 3x3, 5x5) | 54.40M | Deep architecture, captures multi-scale features |
| DenseNet201 | 201 | NA (Dense Connections) | 18.70M | Efficient, smaller model size |

**Table 1.** Model Architecture

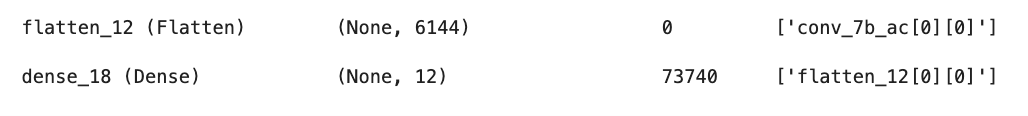
**Note:** While VGG19, ResNet152V2, InceptionResNetV2, and DenseNet201 have distinct architectures, they likely share these two key elements at the end for multi-class classification. The Flatten layer prepares the data, and the Softmax activation interprets the results as probabilities for multi-class classification.



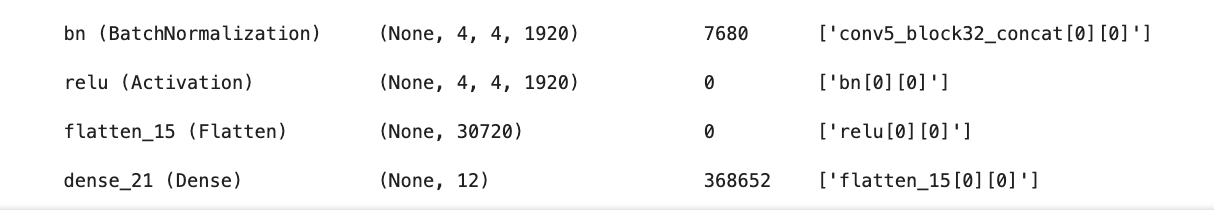
**Figure 3.2.a.** Layers at the end of VGG19



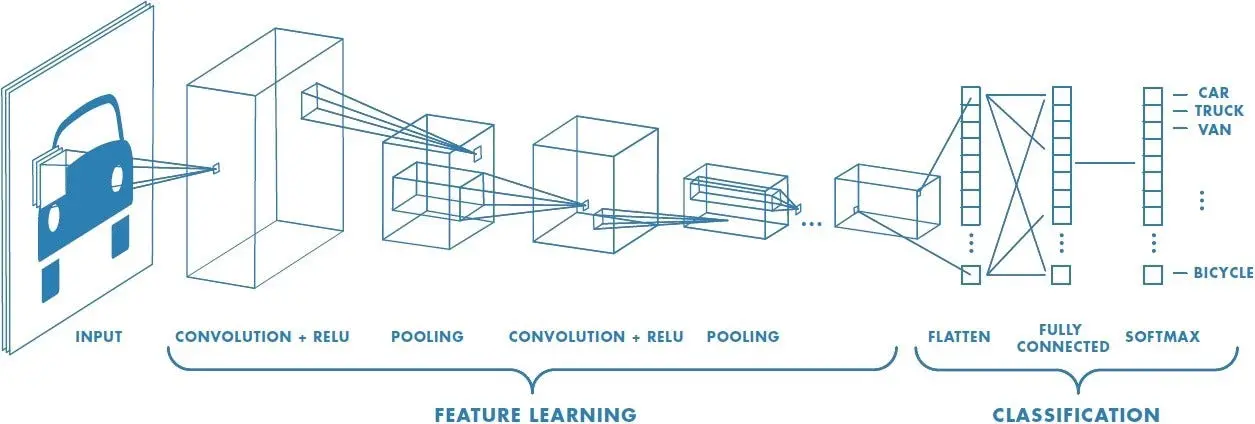
**Figure 3.2.b.** Layers at the end of ResNet152V2



**Figure 3.2.c.** Layers at the end of InceptionResNetV2

****

**Figure 3.2.d.** Layers at the end of DenseNet201



**Figure 3.** General CNN Architecture

**3. Training Process**

The training process includes several key steps:

First, model initialization:

* Initializing models with pre-trained weights from ImageNet.
* Modifying the final classification layer to match the number of dog breeds (12 classes).

Second, fine-tuning:

* Fine-tuning the models by unfreezing the top layers while keeping the initial layers frozen.
* Using the Adam or SGD optimizer with a learning rate of 0.001 and a batch size of 32.

Third, training:

* Training each model for 50 epochs with cross-entropy loss.
* Implementing early stopping and model checkpointing to avoid overfitting.

**4. Evaluation**

Firstly, these models are evaluated based on accuracy, loss, and execution time. The evaluation metrics are measured on a test set to ensure unbiased performance metrics. Additionally, confusion matrices are generated to analyze misclassifications.

Secondly, these models are also evaluated based on :

* Precision: The ratio of true positive predictions to the total predicted positives.
* Recall: The ratio of true positive predictions to the total actual positives.
* F1-Score: The harmonic mean of precision and recall, providing a single metric that balances both concerns.

**IV. Experiments and Results**

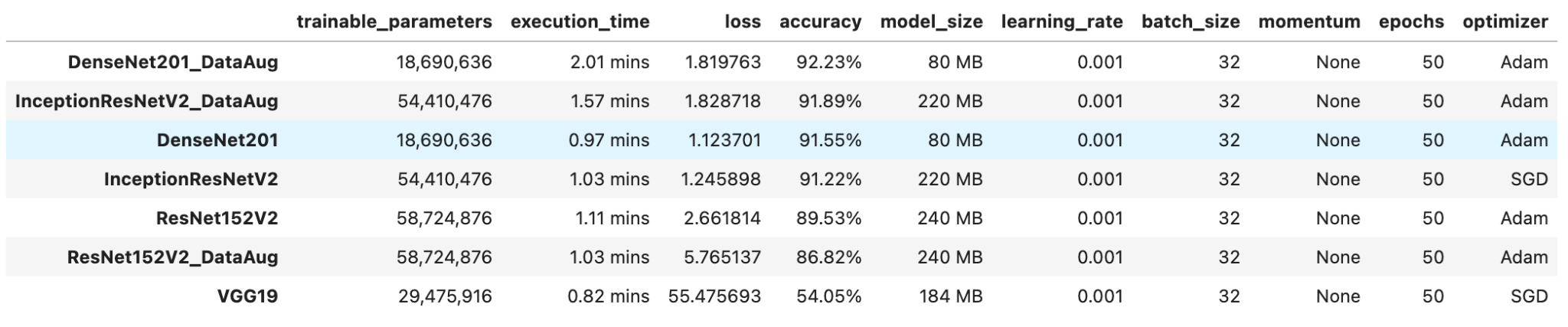
|  |  |
| --- | --- |
| **VGG19** | **ResNet152V2** |
| **ResNet152V2 Augmentation** | **InceptionResNetV2** |
| **InceptionResNetV2 Augmentation** | **DenseNet201** |
| **DenseNet201 Augmentation** |  |

**Figure 4.** Loss and Accuracyof each model

**Training vs. Validation Performance**: Across all models, training loss and accuracy typically improve smoothly, while validation metrics are more variable, indicating possible overfitting or variability due to the dataset.

**Effect of Augmentation**: Data augmentation seems to introduce more variability in validation loss and accuracy, which could indicate the models are facing more challenging augmented data, but it can also help improve the loss.

**1. Based on accuracy, loss, and execution time metrics**



**Table 3.** Performance Metrics

**VGG19** recorded the lowest accuracy (54.05%) and the highest loss (55.475693), highlighting its limited effectiveness compared to more advanced architectures. Its simplicity and fewer parameters did not compensate for the performance gaps.

**DenseNet201** achieved high accuracy (91.55%) and the lowest loss (1.123701) among all models, showing its robustness and efficiency. The execution time was short (0.97 minutes), making it a competitive option. **DenseNet201 with data augmentation** demonstrated a balance of high accuracy (92.23%) and low loss (1.819763), with a reasonable execution time (2.01 minutes). Data augmentation further enhanced its performance, making it one of the most effective models in this study.

**InceptionResNetV2** achieved an accuracy of 91.22% with a reasonable loss (1.245898), this model performed well, though it required substantial resources given its large number of parameters and model size. The execution time was also efficient (1.03 minutes).

The **InceptionResNetV2\_DataAug** model achieved high accuracy (91.89%) with an efficient execution time (1.57 minutes). The use of data augmentation significantly improved performance, although it exhibited a high loss (1.828718), indicating potential overfitting despite its high accuracy.

**ResNet152V2** provided solid performance with an accuracy of 89.53% and moderate loss (2.661814). The execution time (1.11 minutes) was also competitive, indicating a well-balanced model suitable for this task. When trained with data augmentation, **ResNet152V2\_DataAug** showed lower accuracy (86.82%) and higher loss (5.765137), indicating that the augmentation might have introduced complexities not well handled by this architecture.

Based on this metric, **DenseNet201\_DataAug** and **InceptionResNetV2\_DataAug** are recommended due to their balance of high accuracy and efficiency in terms of parameter count and model size. The application of data augmentation generally improved model performance.

**2. Based on accuracy, precision, recall, f1-score**

|  |  |
| --- | --- |
|  |  |

**Figure 4.2.a** VGG19 Confusion Metric

We began by experimenting with transfer learning combined with data augmentation. Keras offers a pre-trained VGG19 model, which was trained on the massive ImageNet dataset. Transfer learning is applicable here because the VGG19 model was trained on a vast amount of data, and the underlying image inputs share fundamental similarities between the two tasks. We opted to remove the final layers of the VGG19 model and instead append two fully connected layers followed by a softmax classification head with 12 output classes tailored to my specific problem. To leverage the pre-trained knowledge, I froze the weights within the remaining VGG19 network (after removing the top layers). During training, only the weights of my newly added fully connected layers (Dense) and the softmax layer were updated using my own training dataset. The optimization approach remained unchanged, minimizing cross-entropy loss without any regularization and employing the Adam optimizer. Unfortunately, this experiment yielded less-than-ideal results.

VGG19 struggles significantly across all classes, with particularly poor performance in breeds like Blenheim\_spaniel and affenpinscher. VGG19 has a significant spread in misclassifications, indicating poor model performance. VGG19 is the least effective model for this task, with significantly lower accuracy and more widespread misclassifications.

|  |  |
| --- | --- |
|  |  |

**Figure 4.2.b** ResNet152V2 Augmentation Confusion Metric

ResNet152V2 has notable misclassifications in classes such as vizsla and malinois. ResNet152V2 has a reasonable concentration along the diagonal but shows more spread in misclassification. ResNet152V2 provides decent performance but is outperformed by the other two models mentioned below.

InceptionResNetV2 also performs well, but is slightly less consistent compared to DenseNet201 in specific classes. InceptionResNetV2 is a close second, showing high accuracy and good class-wise performance.

|  |  |
| --- | --- |
|  |  |

**Figure 4.2.c** InceptionResNetV2 Augmentation Confusion Metric

|  |  |
| --- | --- |
|  |  |

**Figure 4.2.d** DenseNet201 Augmentation Confusion Metric

DenseNet201 shows strong precision, recall, and f1-score across most classes, especially for breeds like Gordon\_setter, Old\_English\_sheepdog, and Great\_Dane. DenseNet201 is the best performing model for this dog breed classification task, with the highest accuracy and strong class-wise performance.

Building upon the exploration with VGG19, we delved into the effects of network depth on classification performance. This section details our investigations with three deeper pre-trained models: ResNet152V2, InceptionResNetV2, and DenseNet201.

In summary, for practical purposes, either DenseNet201 with augmentation or InceptionResNetV2 with augmentation would be recommended due to their high accuracy and reliable performance across different classes.

**V. Conclusion**

This project investigated the use of transfer learning for classifying dog breeds using the Stanford Dogs Dataset. We evaluated four advanced CNN architectures: VGG19, ResNet152V2, InceptionResNetV2, and DenseNet201. Each model was fine-tuned and assessed on key metrics such as accuracy, loss, and execution time.

Data augmentation generally improved performance, enhancing the robustness of the models. This study highlights the effectiveness of transfer learning for image classification and suggests DenseNet201 and InceptionResNetV2 with augmentation as top performers.

**References**

[1] [Transfer learning and fine-tuning | TensorFlow Core](https://www.tensorflow.org/tutorials/images/transfer_learning?hl=en)

[2] [CSCI-S89 | Kartik Lal | Image Classification | Stanford dogs dataset (youtube.com)](https://www.youtube.com/watch?v=QdDtgUqewhY&ab_channel=KartikLal)

[3] [Deep learning for identifying dog breed - YouTube](https://www.youtube.com/watch?v=fxEv6RjpeIw&ab_channel=YangZhou)

[4] [DogClassifier.ipynb](https://drive.google.com/file/d/1DfFUELadDXva1kVwTzedRIWW_QmsgcPl/view)