Cats and Dogs Breed Classification Using CNN and Transfer Learning

Marco Venturi
marco.venturi1@studenti.unipg.it

***Abstract*—This project aims to conduct a study on potential approaches and solutions for the Oxford-IIIT-Pet dataset, comprising images depicting various cat and dog breeds, totaling 37 breeds. Different models, ranging from simple custom CNN models to hierarchical models and transfer learning approaches, are employed to predict the actual breed. Initially, a custom CNN is trained and then fine-tuned to achieve optimal performance in breed classification. Subsequently, the same model is used to construct a two-stage hierarchical model to assess whether the custom and hierarchical implementations can outperform the transfer learning approach. Accuracy is chosen as the performance metric since the classes are balanced. After experimenting with different configurations, it appears that the custom model does not achieve better performance than the transfer learning approach, while the hierarchical model matches the performance of the custom model.**

I. Introduction

Animal biometrics has emerged as a critical field at the convergence of computer vision, pattern recognition, and agricultural science, offering solutions for livestock registration, identification, and welfare management. Traditional methods often fall short in recognizing individual animals in diverse and dynamic environments. Recent advancements in deep learning present promising avenues for improving animal recognition accuracy. This paper aims to explore the potential of deep learning in addressing similar challenges, focusing on the recognition of cat and dog breeds using the Oxford-IIIT-Pet dataset.

In parallel, recent studies have highlighted the efficacy of deep learning techniques in animal recognition. One such study proposes a deep learning-based approach for identifying individual cattle based on muzzle point image patterns. This research underscores the significance of accurate animal recognition for insurance claims and livestock management [1]. Another study demonstrates the application of convolutional neural networks (CNNs) in recognizing pig faces [2] for agricultural purposes, emphasizing the importance of individual animal identification for disease monitoring and welfare management in intensive farming settings.

Against this backdrop, our study aims to investigate various approaches, including custom CNN models, hierarchical models, and transfer learning techniques, to classify cat and dog breeds accurately. We initially train a custom CNN model on the Oxford-IIIT-Pet dataset and fine-tune it to optimize breed classification performance. Subsequently, we construct a two-stage hierarchical model to explore whether it can outperform transfer learning approaches. Our evaluation metrics primarily focus on accuracy, given the balanced nature of the dataset.

In summary, this research contributes to the growing body of literature in animal biometrics and deep learning applications in agriculture. By leveraging the insights from related studies on cattle and pig recognition, we aim to enhance the understanding of deep learning methodologies for breed classification and potentially inform future developments in livestock management and welfare monitoring.

II. Related Work

In the realm of fine-grained object categorization, two notable studies have made significant strides in classifying breeds of cats and dogs from images. The study titled Cats And Dogs [by Omkar M Parkhi et al.] [3] introduces a pioneering annotated dataset encompassing 37 distinct breeds of cats and dogs, addressing the intricate challenge of discerning animal breeds and is the same dataset that actually is studied in this paper. Their proposed model integrates shape and appearance features, employing a combination of a deformable part model to detect pet faces and a bag-of-words model to describe pet fur. Through hierarchical and flat classification approaches, alongside experimentation with various spatial layouts, the models showcased remarkable performance, surpassing previous benchmarks on discriminating cats and dogs. Meanwhile, in Identification of Dog Breeds Using Deep Learning [Rakesh Kumar et al.] [4], the focus lies on a multiclass classification task involving 120 diverse breeds of dogs. Leveraging the Stanford Dogs dataset and deep convolutional neural networks (CNNs), particularly ResNet and Inception models, the research achieves notable success in accurately identifying dog breeds. Transfer learning techniques are employed to enhance accuracy, with ResNet101 emerging as the top-performing architecture. However, both studies also shed light on challenges such as dataset imbalance and the adaptation of pre-trained models to new classification tasks. Despite these obstacles, the findings from both endeavors underscore the efficacy of deep learning techniques in tackling intricate classification tasks within the realm of computer vision, offering valuable insights and advancements in the classification of animal breeds from images.

III. Proposed Approach

A. Dataset and Preprocessing

The Oxford-IIIT PET dataset is a collection of $7,349$ images of cats and dogs comprising 37 different breeds, with 25 being dogs and 12 being cats.

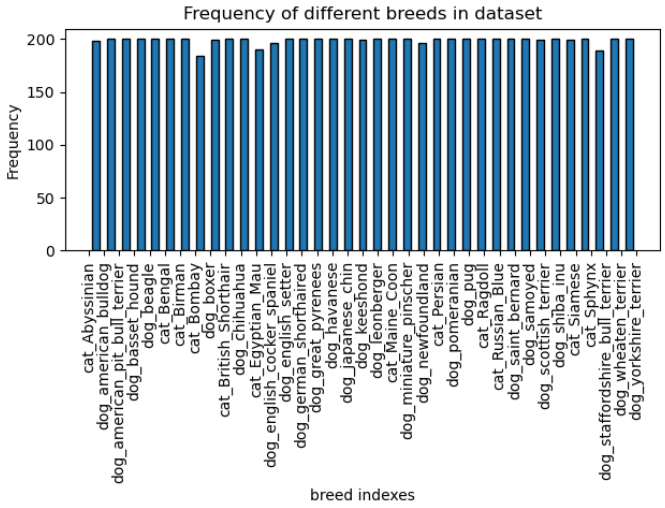

Fig. 1: Histogram of breed distribution in the dataset

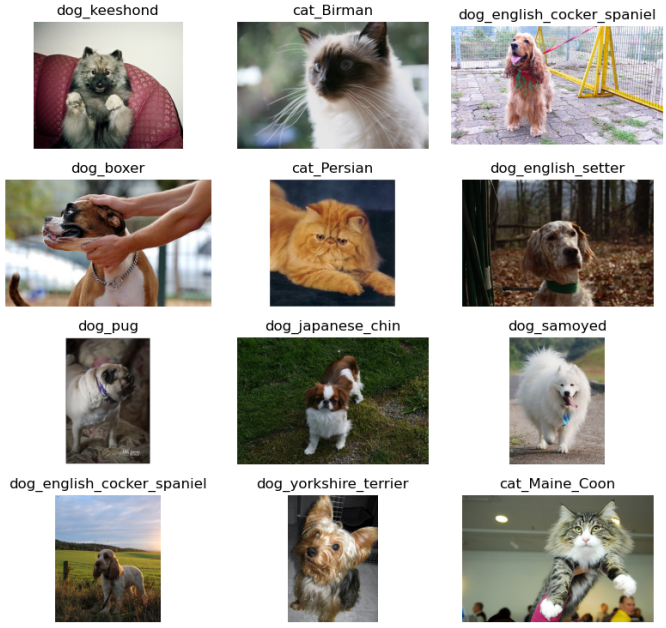

Fig. 2: Sample images

The images are divided into training, validation, and test sets. The dataset contains approximately 200 images for each breed. Figure 1 shows all the breeds and their counts contained in the dataset. The dataset has been randomly split into 70% for training and validation and 30% for testing. Then, the training and validation datasets are further split with 70% for training and 30% for validation using a stratification approach with respect to the target class labels. With this dataset approach, it is possible to generate an unknown test set and a balanced train and validation set with respect to the target classes. In some dataset configurations, augmentation techniques are applied to the training set after splitting the data into train and validation as suggested in this review [5]. The augmentation includes generating new images with different transformations such as Random Horizontal Flip and Random rotation of 10 degrees, resulting in a final dimension of $7,200$ images in the training_augmented set. Changes regarding brightness, contrast, and saturation are not applied because it is assumed that the images in the dataset already exhibit sufficient diversity in these properties. Figure 2 shows a sample of images from the dataset, illustrating how some images contain human presence, in those images is possible to see the differences in brightness and contrast too.

The sizes (width and height) of the images vary throughout the entire dataset. In Figure 3, the different size pairs of images are shown. It is assumed that in 2012, when the dataset was released, full HD images were not as common, so the selected image pairs are only under 1000 pixels per dimension. In Figure 4, the rarest size pairs of images are shown. Since the images have varying sizes, before the dataset splitting phase, all images are resized to two different dimensions: 128x128 px and 256x256 px. These two different scales are trained independently to determine if the models are affected by different scale images. In both cases, the model input and the fully connected layers of the custom and Hierarchical models are adapted to the given input.

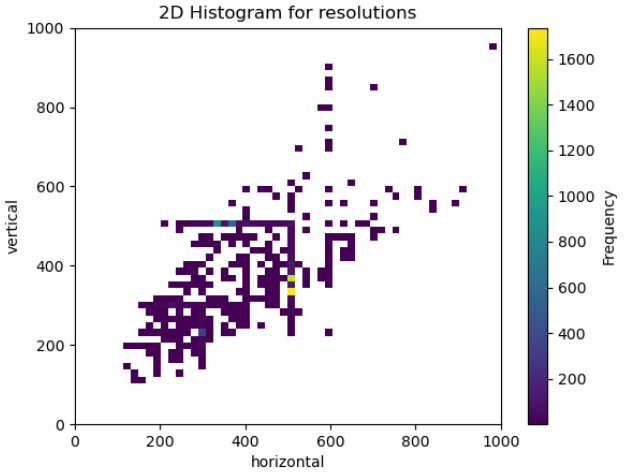

Fig. 3: 2D histogram of image resolution pairs

B. Model Architecture

As previously mentioned, in this project, we will compare three models: a custom CNN model, a Hierarchically CNN custom model, and a pretrained one. The final goal is common among them. Different batch sizes, 128 and 256, are tested in all models. The metric used for evaluation is accuracy, defined as the ratio of correctly predicted instances to the total instances, expressed as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

This choice is made because the dataset is balanced. All models are described separately in the next sub-chapters.

1) Custom Model Architecture: The custom CNN architecture, illustrated in Fig. 5, follows this structure:

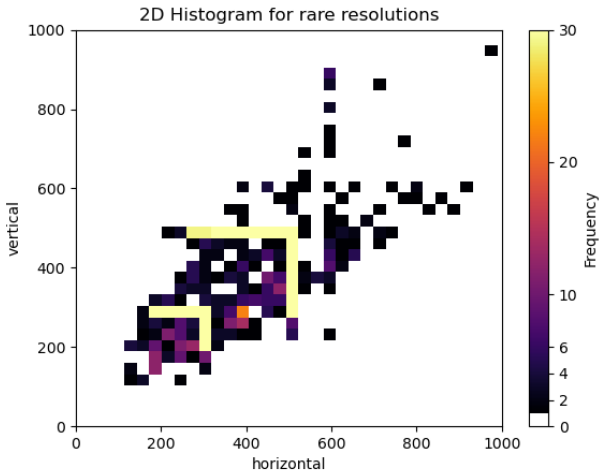

Fig. 4: 2D histogram of rare image resolution pairs

$$\texttt{INPUT} \rightarrow [\texttt{CONV} \rightarrow \texttt{BATCHNORM} \rightarrow \texttt{RELU} \rightarrow \texttt{MAXPOOL}] \times 2 \rightarrow$$

$$\rightarrow [\texttt{CONV} \rightarrow \texttt{RELU} \rightarrow \texttt{MAXPOOL}] \rightarrow$$

The output of the last Maxpooling layer is then flattened to connect to a Fully connected layer:

$$\rightarrow \texttt{FC} \rightarrow \texttt{DROPOUT} \rightarrow \texttt{RELU} \rightarrow \texttt{FC}$$

resulting in thirty-seven output neurons that perform a softmax function.

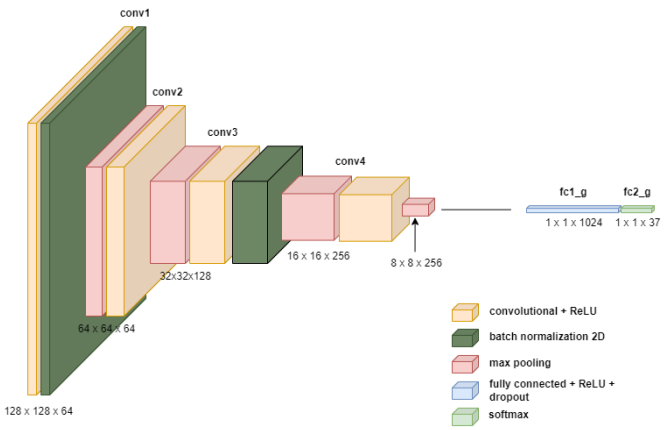

Fig. 5: Graphical representation of CNN custom model architecture (128×128 image size)

Let's explore in more detail the structure of the various layers:

- The input, with dimensions $128 \times 128 \times 3$, enters the first 2D convolutional layer comprising 64 filters of size 128×128. This layer applies a padding of 1 to visit every pixel during convolution. The stride is set to 1 in the convolutional layer and set to 2 in the subsequent pooling layer. This setting avoids requiring the network to learn to subsample and is repeated in all the subsequent layers.
- Following this, a batch normalization layer is applied to the output of the previous convolutional layer. Batch normalization helps stabilize and accelerate the training of neural networks by normalizing the batch to its mean and standard deviation instead of doing it with the whole dataset, improving the test set performance too [6]. One batch normalization is between the first convolutional layer and the first pooling layer, and the second normalization is between the third convolutional layer and the third pooling layer.
- The output of the batch normalization then enters the Rectified Linear Unit (ReLU) activation function, introducing non-linearity to the model, allowing it to learn from more complex patterns in the data.
- Before entering another convolutional layer, the output of the preceding layer undergoes pooling through a max pooling layer with a window size of 2×2 and a stride of 2. As a result, the size of the produced map is halved.
- The previous steps are repeated one more time, with the only difference being the number of filters, which is doubled in each subsequent convolutional layer, except for the last convolutional layer which maintains the same number of filters as the previous one.
- The output of the last pooling layer is flattened and fed into the fully connected layer with 1024 neurons, followed by a dropout layer and ReLU activation.
- The output of the last fully connected layer is fed into a fully connected layer with thirty-seven neurons, one for each class of the classification problem, followed by a softmax function.

A dropout layer is included in the fully connected part. A Cross Entropy Loss is used as the loss function with the Adam algorithm to optimize it.

2) Hierarchical Custom Model Architecture: The Hierarchical custom model architecture, depicted in Fig. 6, is divided into two sub-models. The first model has the same structure as the custom model architecture, with the only difference in the output layer, which has 2 neurons (`Cat`: 0 or `Dog`: 1). The second part has the same architecture as the custom model, with the difference in the first fully connected layer where the input includes not only the flattened CNN maps but also the predicted labels of the first model. Each model trains independently of the other, with the first training of the binary classifier with respect to the Cat and Dog classes. After training the classifier, the test set is used to predict the binary label, which will be used as input for the breed classifier. The breed classifier initially uses the images as inputs to the convolutional layers, while the binary class information is then concatenated to the flattened vector. The training, validation, and test sets are the same sets used in the binary model.

3) Transfer Learning Model Architecture: Transfer learning capitalizes on a pre-trained model's ability to learn generic features that are applicable to a broader range of tasks. This pre-trained model, typically trained on a large and diverse dataset, serves as a foundation for a new model tackling

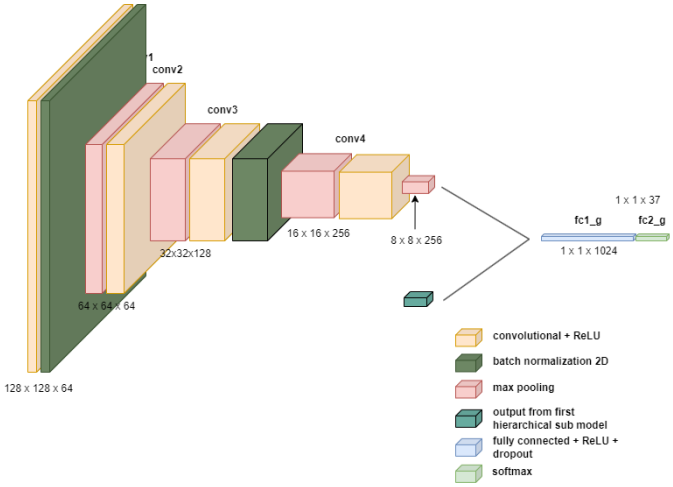

Fig. 6: Graphical representation of the second hierarchical sub-model architecture

a related but potentially smaller dataset. The core concept lies in leveraging the pre-trained model's knowledge as a springboard, reducing the training time and data requirements compared to building a model entirely from scratch. There are three main approaches to implementing transfer learning:

- CNN as a fixed feature extractor: In this approach, the pre-trained model's convolutional layers are frozen, essentially acting as a feature extractor. These layers learn low-level and mid-level features like edges, shapes, and textures, which are often transferable across different visual recognition tasks. A new classifier head is then added and trained on the target dataset, allowing the model to adapt the learned features to the specific classification problem at hand.
- Fine-tuning: This method builds upon the fixed feature extractor approach. Here, a few of the final layers (often the fully connected layers) in the pre-trained model are unfrozen and allowed to adapt during training on the new dataset. This fine-tuning process enables the model to adjust the higher-level features learned on the original task to better distinguish the classes within the target dataset.
- Using weights directly from a pre-trained model: In this approach, all or most of the pre-trained model's weights are directly copied into the new model. This is often used when the source and target tasks are very similar. The model is then trained on the new dataset, allowing it to refine the weights and tailor them to the specific nuances of the target problem.

In this paper, a CNN is used as a fixed feature extractor. The chosen model is the VGG16 [7]. The VGG16 is first downloaded, and then only the convulationals layers are selected. Afterward, an adaptive global average pooling layer with padding and stride of 1 is added, followed by a fully connected layer with 512 inputs and 1024 neurons with a ReLU activation, a dropout layer with a rate of 0.25, and finally, a fully connected layer with 37 neurons, one for each class. The pre-trained model is chosen over other models because at the beginning of the experiments, it was interesting to find a model that performs well, but not excessively well compared to a custom approach [8], so that the comparison could make sense.

IV. Experiments

A. Training Environment and Hardware Specifications

The model training was conducted within a Python environment in Anaconda, utilizing PyTorch libraries with GPU acceleration. The machine used features an AMD Ryzen 7 5800H CPU with 8 cores and 16 threads, operating at 3.2 GHz. It is equipped with 32 GB of RAM running at 3600 MHz and a PNY NVidia RTX 3070 graphics card (Laptop Series) with 8GB of VRAM and 5120 CUDA cores, limited to 100W.

B. Training Process

The number of epochs for each training network is presented below:

Model	**Num. of epochs**
Custom CNN	14
Hierarchical custom (Binary) CNN	11
Hierarchical custom (Multi) CNN	11
VGG16 pretrained CNN	22

TABLE I: Number of epochs in each model

An early stopping mechanism was implemented with a patience value set to 5 for both architectures. The patience hyper-parameter was determined by monitoring the validation loss of the model during various training sessions and stopping the process if the loss did not improve for a consecutive number of epochs. This determined number was then fixed and used to automate the process.

The training process was stopped when the loss values on the training set reached:

Model	**Last loss value (training)**
Custom CNN	0.0057
Hierarchical custom (Binary) CNN	0.001
Hierarchical custom (Multi) CNN	0.0093
VGG16 pretrained CNN	0.3633

TABLE II: Training loss value

and on the validation set:

Model	**Last loss value (validation)**
Custom CNN	0.0225
Hierarchical custom (Binary) CNN	0.0032
Hierarchical custom (Multi) CNN	0.0434
VGG16 pretrained CNN	1.2531

TABLE III: Validation loss value

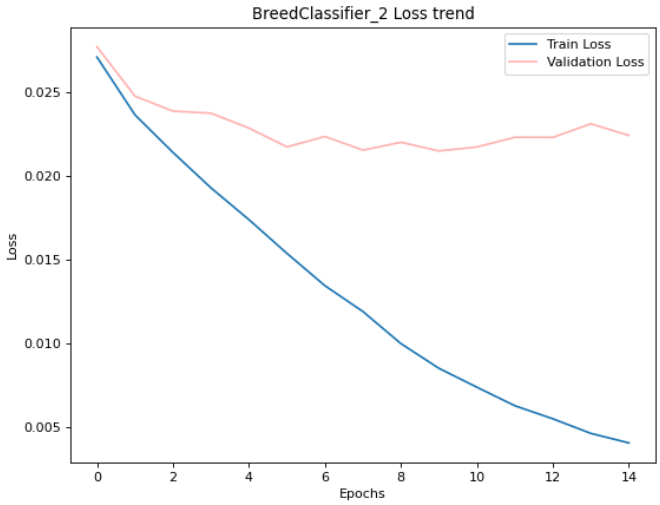

Fig. 7: Loss trend in the custom CNN

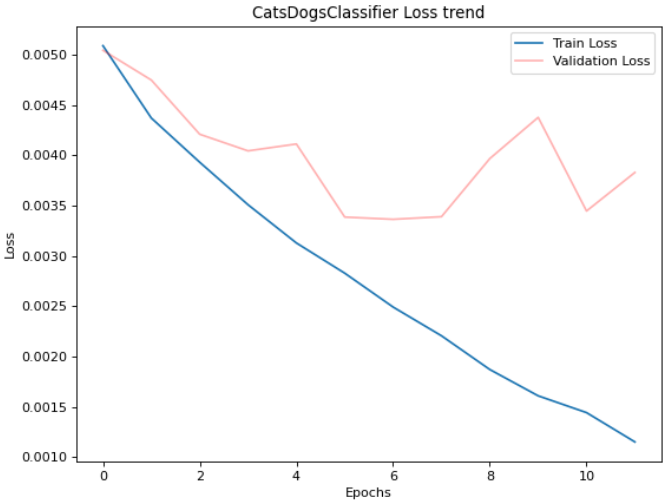

Fig. 8: Loss trend in the Hierarchical First section custom CNN Model

In Fig. 7, Fig. 8, Fig. 9 we can observe the trends through the epochs of the loss in the custom CNN approach, the Hierarchical First section custom CNN Model and the Hierarchical First second custom CNN model respectively.

The training process ended with the following metrics on the training set:

Model	**Train Accuracy**
Custom CNN	0.8065
Hierarchical custom (Binary) CNN	0.9579
Hierarchical custom (Multi) CNN	0.8393

TABLE IV: Training accuracy value

with the following time in second as indicated in the Table. V.

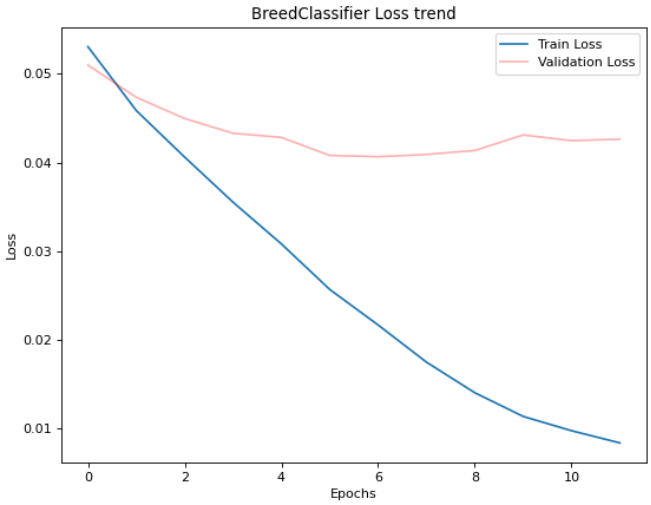

Fig. 9: Loss trend in the Hierarchical Second section custom CNN Model

Model	**Time in second**
Custom CNN	511
Hierarchical custom (Binary) CNN	652
Hierarchical custom (Multi) CNN	487
VGG16 pretrained CNN	1008

TABLE V: Models time table

The validation set accuracy is indicated in Table. VI. It's

Model	**Validation Accuracy**
Custom CNN	0.3565
Hierarchical custom (Binary) CNN	0.8549
Hierarchical custom (Multi) CNN	0.3406

TABLE VI: Models validation Accuracy

noticeable that there are variations in the results obtained between the training and validation sets. It's crucial to emphasize that the definitive evaluation of the models' performance will emerge from the testing set. Only then can we thoroughly assess whether one architecture holds an advantage over the other. In the context of the provided sample image in Fig. 10, taken from the validation set, we can observe the corresponding mean attention heatmaps for each convolutional layer in the neural network. Specifically, Fig. 11 illustrates the Custom CNN attention heatmap, while Fig. 12 depicts the Hierarchical custom (Binary) CNN heatmap and Fig. 13 showcases the Hierarchical custom (Multi) CNN heatmap.

C. Model Selection

During the training process, various model configurations were trained and tested. The presented models and approach were selected based on the best configuration, which includes using the Adam optimizer with a learning rate of 0.0001 and applying L2 normalization with a hyper-parameter of 0.001. Additionally, a dropout rate of 0.5 was employed.

Fig. 10: Sample image from the validation dataset

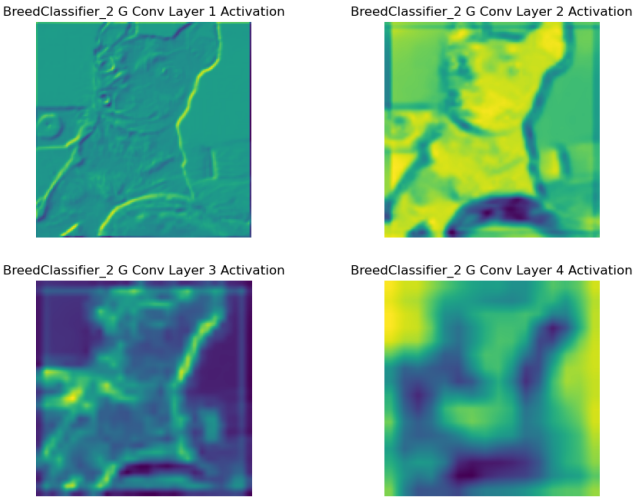

Fig. 11: Attention heatmap from custom CNN model

D. Training Result

After concluding the model training, we proceed to the testing phase. In this stage, we load the testing images, which were kept separate and unprocessed initially, to assess the CNN's performance in interpreting this data. Following, you'll find the table displaying the accuracy scores for all architectures: along with Fig. 14, Fig. 15, Fig. 16 and Fig. 17

Model	**Test Accuracy**
Custom CNN	0.3505
Hierarchical custom (Binary) CNN	0.8580
Hierarchical custom (Multi) CNN	0.3406
VGG16 pretrained CNN	0.67

TABLE VII: Test accuracy value

illustrating all confusion matrices for Breed Classification.

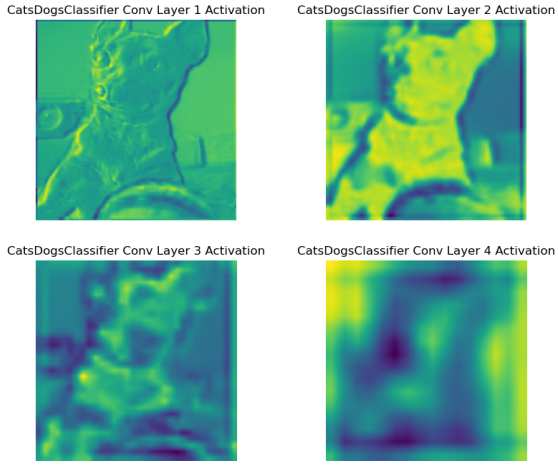

Fig. 12: Attention heatmap from Hierarchical custom (Binary) CNN model

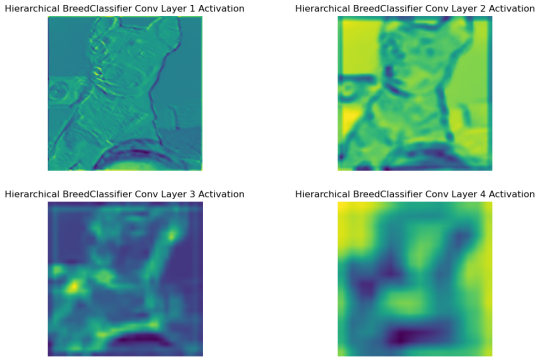

Fig. 13: Attention heatmap from Hierarchical custom (Multi) CNN model

V. Conclusion

In this study, we have presented a comparison between two custom CNN architectures for cats and dogs breed recognition and a pre-trained approach. As we can observe, the custom approaches as a similar performance with a little improvement in the local approach rispect to the Hierarchical approach. Unfortunately, the custom CNN doesn't achieve results close to the pre-trained model, while VGG16 achieves performance close to that observed in studies. The first hierarchical classifier is still working well when it comes to distinguishing between the binary class problem. An interesting solution could be to use the model used for classifying cats and dogs as a fixed feature extractor CNN and concatenate more convolutional layers, as well as the final fully connected part, to induce the network to train respecting breeds while retaining the knowledge to distinguish cats from dogs.

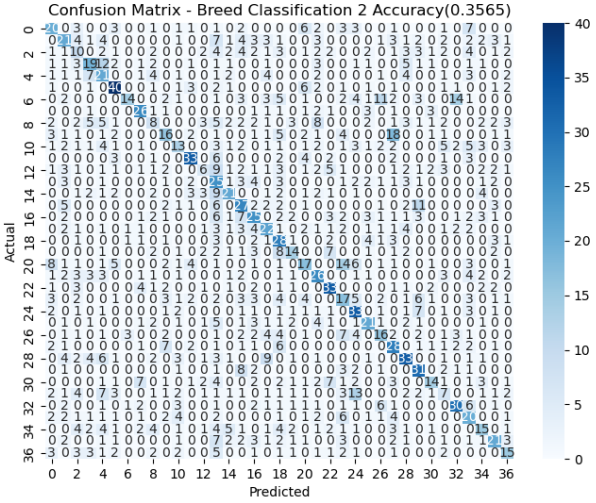

Fig. 14: Confusion Matrix for Custom CNN

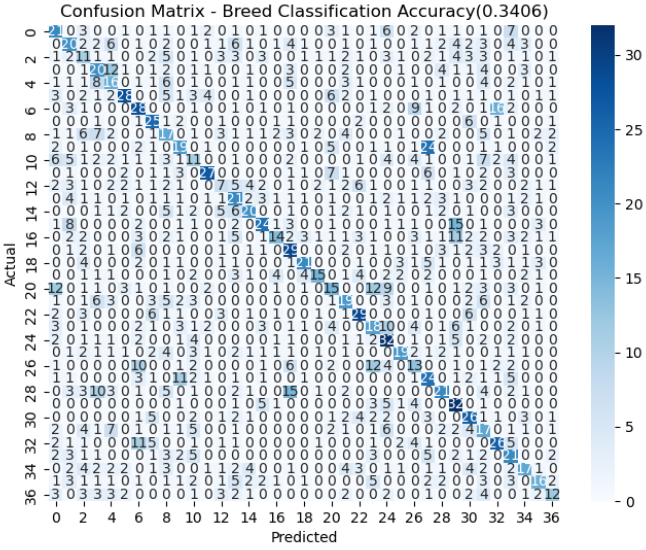

Fig. 16: Hierarchical custom (Multi) CNN

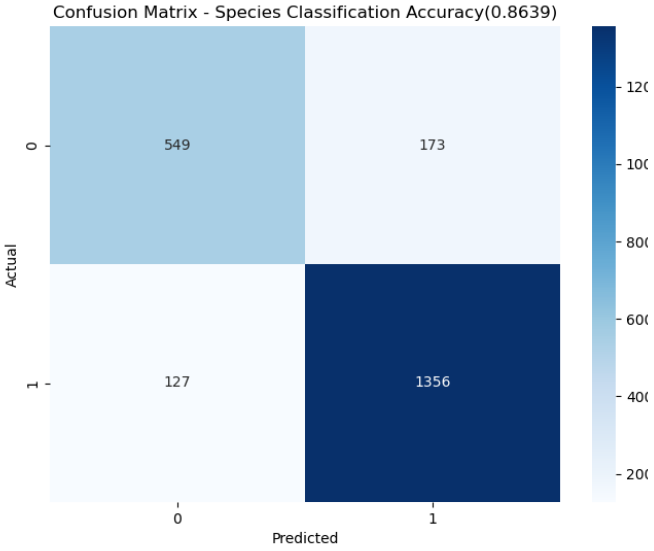

Fig. 15: Hierarchical custom (Binary) CNN

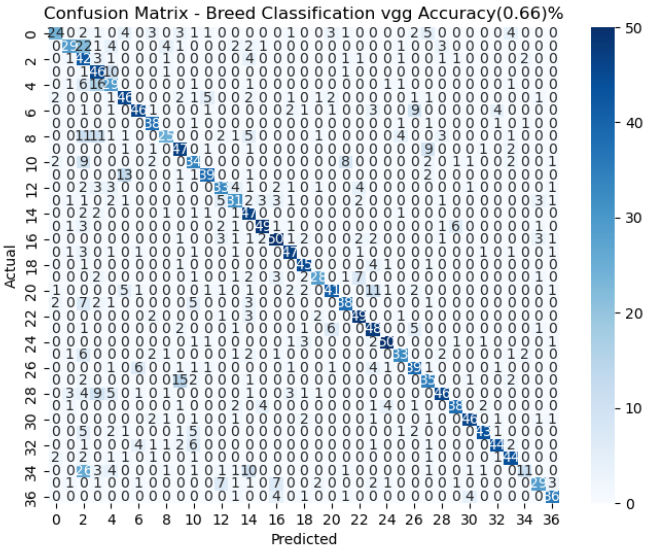

Fig. 17: VGG16 pretrained CNN

REFERENCES

[1] K. S. R. S. S. K. S. K. S. A. K. S. A. M. Santosh Kumar, Amit Pandey, "Deep learning framework for recognition of cattle using muzzle point image pattern," *Measurement*, vol. 116, no. 1–17, 2018.

[2] X. S. L. Y. P. F. X. Y. C. L. Y. Z. Mathieu Marsot, Jiangqiang Mei, "An adaptive pig face recognition approach using convolutional neural networks," *Computers and Electronics in Agriculture*, vol. Volume 173, p. 105386, 2020.

[3] A. Z. O. M. Parkhi, A. Vedaldi and C. V. Jawahar, "Cats and dogs," pp. 3498–3505, 2012.

[4] K. D. R. Kumar, M. Sharma and G. Singal, "Identification of dog breeds using deep learning," pp. 193–198, 2019.

[5] A. F. D. S. P. Mingle Xu, Sook Yoon, "A comprehensive survey of image augmentation techniques for deep learning," *Pattern Recognition*, vol. Volume 137, pp. 193–198, 2023.

[6] S. T. V. Thakkar and C. Chakraborty, "Batch normalization in convolutional neural networks — a comparative study with cifar-10 data," pp. 1–5, 2018.

[7] A. Z. Karen Simonyan, "Very deep convolutional networks for large-scale image recognition," pp. 1–14, 2014.

[8] A. K. S. A. Varshney, A. Katiyar and S. S. Chauhan, "Dog breed classification using deep learning," pp. 1–5, 2021.