Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Department of Computer Science,
Communication and Business
Degree Program: Quantitative Finance and
Data Science
HTW Berlin

Fine-grained Image Classification (Dog Breeds)

Written report for the module
Deep Learning Seminar

by
Jamli, Mohamed Amine
Personal GitHub Repository
https://github.com/aminjamli/Fine-grained-classification-Project
Course GitHub Repository
https://github.com/HTW-WM-FAR/DeepLearningWS2025/tree/main/fine%20grained%20classification

submitted to
Prof. Petukhina, Alla

Berlin, February 20, 2026

Contents

1 Introduction

Many real-world image classification tasks require distinguishing between visually similar categories, where differences between classes are subtle and often difficult to identify. Traditional image classification methods usually focus on coarse-grained categories, such as distinguishing between broad object classes like dogs and cats. While these methods are effective for general image recognition, they are often not sufficient for more complex real-world problems in which visually similar categories must be accurately separated, such as different breeds of dogs or cats.

Fine-grained image classification addresses this challenge by focusing on subordinate-level categories within the same general class. In this setting, models must learn subtle and discriminative visual features, including variations in texture, color, shape, and spatial structure. As a result, fine-grained classification is considered more challenging than standard image classification, since intra-class variation can be high while inter-class differences remain small.

Fine-grained classification is particularly important in applications where high precision is required, such as medical image analysis, autonomous systems, and visual quality inspection. In medical imaging, small visual differences can indicate different disease stages, making accurate classification essential. Similarly, autonomous systems rely on precise visual recognition to ensure safe and reliable decision-making. These applications highlight the importance of developing models capable of capturing fine-level visual details.

Dog breed classification is a widely used benchmark for evaluating fine-grained image classification methods. The Stanford Dogs dataset, introduced by Krause et al. [1], contains images of dogs labeled by breed and is specifically designed to reflect the challenges of fine-grained recognition. Different dog breeds often share very similar visual characteristics, while images within the same breed may vary significantly due to pose, lighting conditions, background, and age. The dataset used in this project consists of images of dogs labeled by breed and is designed to challenge models to capture subtle yet meaningful visual distinctions. The primary goal of this fine-grained classification task is to train and evaluate models that can effectively learn discriminative features at a fine level of detail, thereby demonstrating their ability to generalize across visually similar categories. As such, dog breed classification serves as an ideal testbed for studying fine-grained visual recognition and assessing the performance of modern deep learning architectures in complex real-world scenarios.

2 Data Pre-processing

2.1 Dataset Overview

The Stanford Dogs dataset is a fine-grained image classification dataset containing images of 120 different dog breeds from around the world[1, 2].. It was constructed using images and annotations from the ImageNet dataset (https://www.image-net.org/) specifically for the task of fine-grained visual categorization, where the objective is to distinguish between visually similar classes such as different breeds of dogs. The dataset includes a total of 20,580 images, with annotations in the form of class labels and bounding boxes for the objects of interest. For common training/testing splits, 12,000 images are typically used for training and 8,580 for testing. The images are 3 color channel-images but vary in size and resolution and are organized into subfolders corresponding to each breed category. This dataset is widely used as a benchmark for convolutional neural network architectures and transfer learning in fine-grained classification tasks.

Source: http://vision.stanford.edu/aditya86/ImageNetDogs/

2.2 Data Visualisation

As an initial step, we verify the integrity of the dataset by examining the total number of classes and images to ensure that the dataset has been correctly loaded. This validation step is essential to confirm that no data corruption or incomplete uploads occurred during preprocessing.

Subsequently, we analyze the number of images per class in order to assess the class distribution. Ensuring a relatively balanced number of samples across classes is crucial, as significant class imbalance may lead to biased learning, negatively affect model generalization, and distort performance metrics. Identifying underrepresented classes at this stage allows for appropriate corrective measures, such as data augmentation.

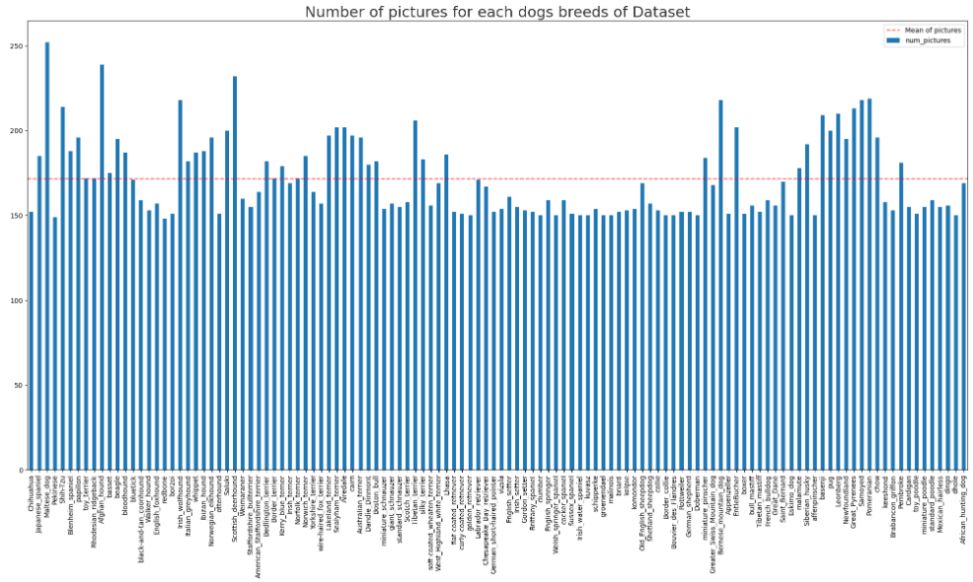

Figure 1: Distribution of images across dog breed classes

It can be observed that all dog breed classes are sufficiently populated with images. On average, each class contains approximately 171 images. No breed is underrepresented; therefore, all classes can be retained for subsequent analysis.

We now examine a selection of example images from different dog breeds available in the dataset:

Figure 2: Sample images from a single dog breed class

As observed in Figure 2, the original images exhibit varying dimensions, necessitating a uniform resizing step. Since the Xception model, which will be employed for trainin later, requires input images of size 299×299 pixels, all images in the dataset are resized accordingly to match this requirement.

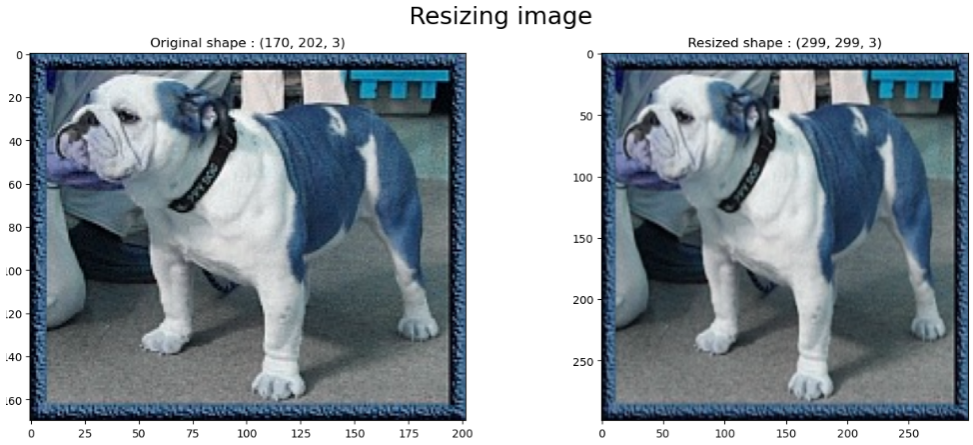

Figure 3: Example of resized Image

It is observable in Figure 3 that resizing the images introduces distortions compared to the original photographs. Once this resizing operation is applied uniformly across the dataset, all images are likely to exhibit similar geometric distortions.

Furthermore, given the relatively large and diverse image set, variations in exposure, contrast, and overall illumination are evident across samples. To address these inconsistencies, we next employ histogram-based image processing techniques in order to improve data normalization and enhance the quality of the preprocessing pipeline.

The histogram of a digital image is a statistical representation of the distribution of its pixels according to their intensity values [3]. We begin by examining a particular image in detail.

Next, the image is transformed into different color spaces. The YUV color space is derived from an RGB source and consists of three components: Y, representing luminance (brightness information), and U and V, representing chrominance (color information). This format allows for a clear visualization of the histogram across all three dimensions.

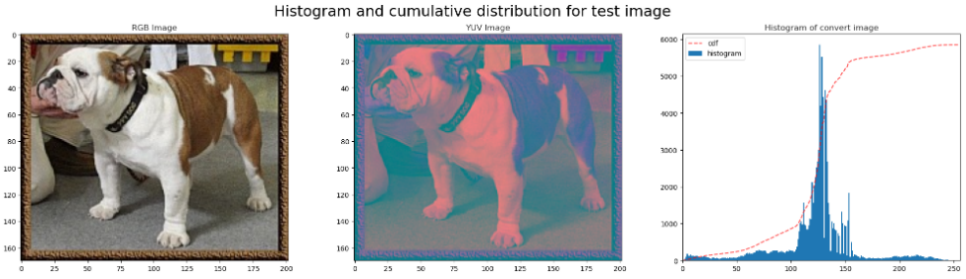

Figure 4: Histogram and cumilative distibution for test image

Here, we observe significant peaks in the center of the histogram. For proper histogram equalization (contrast enhancement), it is necessary to redistribute the pixel intensities across the entire dynamic range of the image.

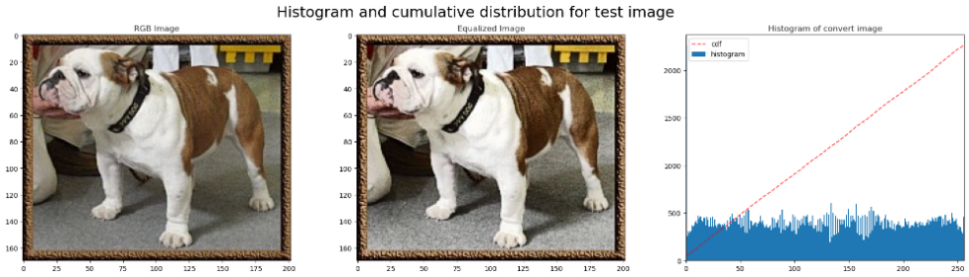

Figure 5: Histogram and cumilative distibution for test image

Images may be affected by noise for various reasons, meaning they contain atypical pixel values that can negatively impact feature detection. Common sources of noise include camera sensor characteristics, JPEG compression artifacts, and insufficient image resolution.

To mitigate the presence of noise, image filtering techniques can be applied. These techniques can be broadly categorized into linear filters, such as Gaussian filtering, and non-linear filters, such as median filtering. Among the most effective approaches for image denoising is the Non-Local Means (NLM) filter.

Unlike local averaging filters, which smooth an image by replacing each pixel with the average value of its neighboring pixels, the Non-Local Means filter computes a weighted average over all pixels in the image. The weights are determined by the similarity between the target pixel and other pixels, regardless of their spatial distance. This approach results in significantly improved post-filtering clarity and preserves fine image details more effectively than local averaging methods.

We now apply the Non-Local Means filter to the selected test image:

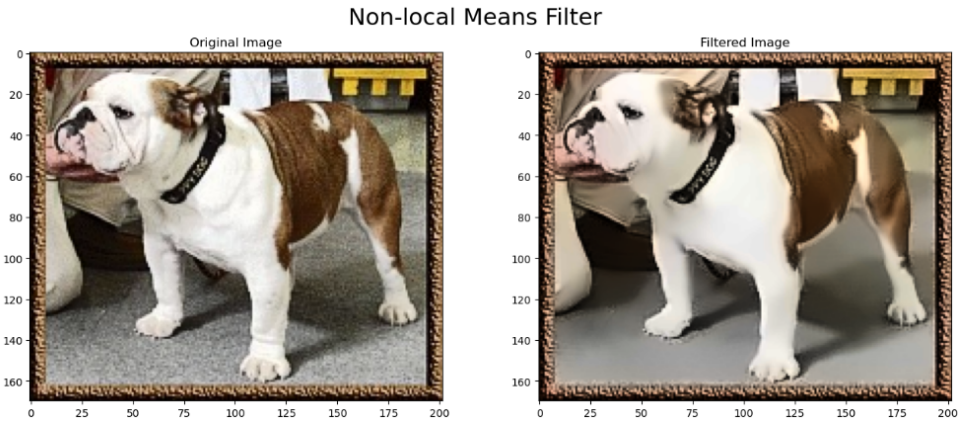

Figure 6: Example of filtered image

A major risk when working with a relatively small dataset, such as the present one containing approximately 20,000 images, is overfitting. In such cases, the model may fail to learn decision rules that generalize effectively to unseen data. To mitigate this issue, it is necessary to artificially increase the diversity of the training data through data augmentation.

The objective of data augmentation is to generate additional training samples by applying random transformations to existing images. To this end, Keras provides the `ImageDataGenerator` utility, which enables the application of various transformations such as horizontal flipping (mirroring), rotation, zooming, and other geometric variations in a stochastic manner. These transformations improve the robustness of the model by exposing it to a wider range of plausible image variations during training.

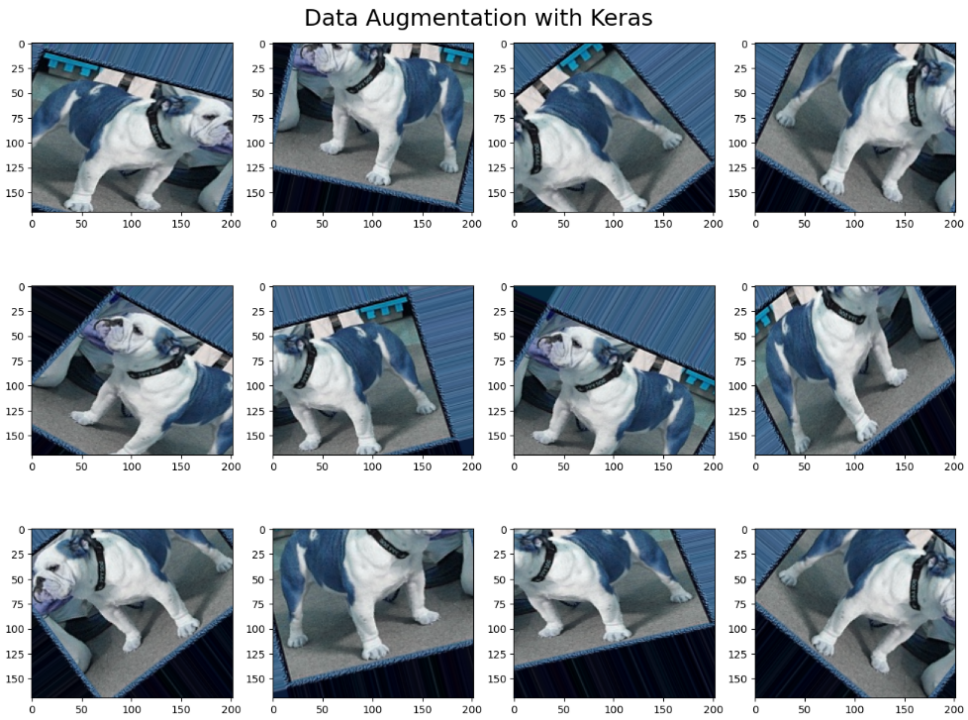

Figure 7: Example of image augmentation

3 Models

3.1 Baseline CNN Architecture

The baseline convolutional neural network (CNN) is constructed as a composition of parametric nonlinear transformations that progressively extract hierarchical representations from the input images.

All input images are resized to $299 \times 299 \times 3$ prior to training.

Convolutional Blocks The network implemented consists of three convolutional blocks. Each block is composed of:

- A convolutional layer with 3×3 kernels,
- A Rectified Linear Unit (ReLU) activation function,
- A max-pooling layer for spatial downsampling.

Formally, the convolutional transformation at layer l is given by

$$h^{(l)} = \sigma\left(W^{(l)} * h^{(l-1)} + b^{(l)}\right),$$

where

- $h^{(l-1)}$: input feature map,
- $W^{(l)}$: learnable convolutional filters,
- $b^{(l)}$: bias term,
- $*$: convolution operator,

- $\sigma(\cdot)$: nonlinear activation function (ReLU),
- $h^{(l)}$: output feature map.

The ReLU activation introduces nonlinearity and is defined as

$$\sigma(z) = \max(0, z),$$

where z denotes the pre-activation value.
Following each convolution, spatial dimensionality is reduced via max-pooling:

$$h^{(l)} = \text{MaxPool}_p\left(h^{(l-1)}\right),$$

Fully Connected Classifier After the final convolutional block, the feature maps are flattened into a vector $z \in \mathbb{R}^d$. A fully connected layer performs the transformation

$$h^{(l)} = \phi(W^{(l)}z + b^{(l)}),$$

where $W^{(l)} \in \mathbb{R}^{m\times d}$ is the weight matrix, $b^{(l)}$ the bias vector, and $\phi(\cdot)$ a nonlinear activation function.

To mitigate overfitting, a dropout layer with rate $p = 0.5$ is applied during training.

Output Layer and Softmax For the multi-class classification problem with $K = 120$ classes, the final linear layer produces logits

$$a = W^{(L)}h^{(L-1)} + b^{(L)}.$$

where

- $h^{(L-1)}$ denotes the output feature vector of the pre-last layer (i.e., the learned representation before classification), where d is the feature dimension,
- $W^{(L)}$ represents the weight matrix of the final fully connected (linear) layer,
- $b^{(L)}$ denotes the bias vector of the final layer,
- a is the vector of logits (unnormalized class scores).

The logits are transformed into class probabilities using the softmax function:

$$\hat{y}_k = \frac{\exp(a_k)}{\sum_{j=1}^{K}\exp(a_j)}, \quad k = 1, \ldots, K.$$

where

- $\hat{y}_k$ represents the predicted probability that the input sample belongs to class k,
- $\exp(a_k)$ ensures that all transformed scores are strictly positive,
- the denominator $\sum_{j=1}^{K}\exp(a_j)$ acts as a normalization constant,
- $K = 120$ denotes the total number of classes in this classification problem.

The softmax function maps the logit vector a to a probability distribution $\hat{y}$ such that $\hat{y}_k \in (0, 1)$ for all k and $\sum_{k=1}^{K} \hat{y}_k = 1$. Consequently, the output layer produces a valid categorical probability distribution over the K classes.

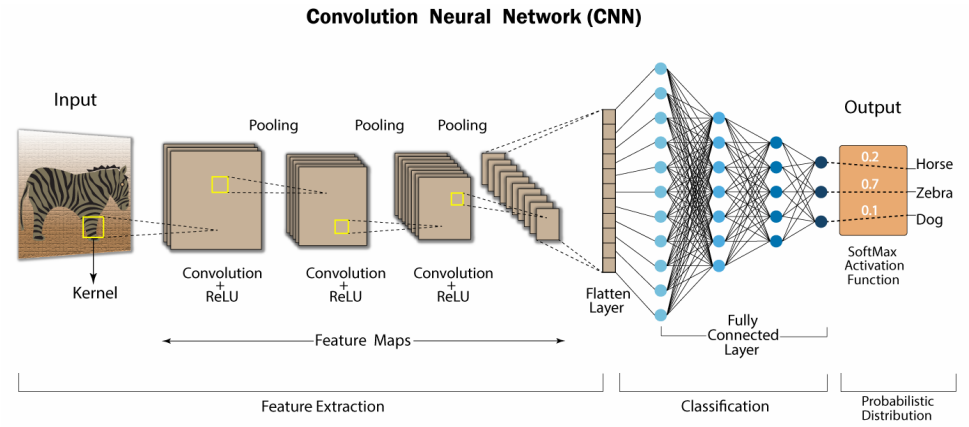

Figure 8: CNN architecture

3.2 Xception Model Construction

Xception (Extreme Inception), proposed by *François Chollet* in 2017 [4], is a deep convolutional neural network architecture that builds upon the conceptual foundation of the Inception family of models. The central idea behind Xception is the complete replacement of standard convolutional layers with a factorized two-step process known as *depthwise separable convolutions*. This modification is based on the hypothesis that spatial correlations and cross-channel correlations can be learned more efficiently when modeled separately. By decoupling these two processes, Xception achieves improved parameter efficiency and reduces computational complexity.

In classical convolutional neural networks, convolutional layers simultaneously capture spatial patterns and channel-wise feature interactions. Let the input tensor be defined as $X \in \mathbb{R}^{H \times W \times C_{in}}$, where H and W are the spatial dimensions and C_{in} the number of input channels. A standard convolution using C_{out} filters of spatial size $k \times k$ produces an output tensor according to

$$Y_{i,j,c} = \sum_{u,v} \sum_{c'=1}^{C_{in}} W_{u,v,c',c} \, X_{i+u,j+v,c'},$$

where

- $X \in \mathbb{R}^{H \times W \times C_{in}}$: input feature tensor with height H, width W, and C_{in} input channels
- $Y_{i,j,c}$: output feature value at spatial location (i, j) and output channel c
- $W_{u,v,c',c}$: convolutional kernel weight at spatial offset (u, v) connecting input channel c' to output channel c

This formulation shows that each output channel depends on all input channels simultaneously, meaning that spatial feature extraction and channel mixing are performed in a single operation. The total number of learnable parameters in this case equals

$$k^2 \cdot C_{in} \cdot C_{out},$$

which can become computationally expensive for large channel dimensions. Xception replaces this joint operation with depthwise separable convolution. The first step is the *depthwise convolution*, defined as

$$Z_{i,j,c} = \sum_{u,v} W^{(d)}_{u,v,c} \, X_{i+u,j+v,c}.$$

where

- $X \in \mathbb{R}^{H \times W \times C_{in}}$ denotes the input tensor with spatial dimensions H and W, and C_{in} input channels,
- $Z_{i,j,c}$ represents the intermediate output at spatial position (i, j) in channel c,
- $W^{(d)}_{u,v,c}$ is the depthwise convolution kernel weight at spatial offset (u, v) for channel c,
- (u, v) are the spatial indices within the $k \times k$ kernel window,

Thus, each input channel is convolved with its own spatial filter, and no cross-channel interaction occurs in this step.

The second step is the *pointwise convolution*, implemented as a 1×1 convolution:

$$Y_{i,j,c} = \sum_{c'=1}^{C_{in}} W^{(p)}_{c',c} \, Z_{i,j,c'}.$$

where

- $Z \in \mathbb{R}^{H \times W \times C_{in}}$ is the intermediate tensor obtained from the depthwise convolution,
- $Y_{i,j,c}$ denotes the final output feature at spatial location (i, j) and output channel c,
- $W^{(p)}_{c',c}$ represents the pointwise convolution weight connecting input channel c' to output channel c,

The pointwise convolution combines the information from different channels at each pixel location. This allows the network to learn how the extracted spatial features interact with each other. The computational cost of this factorized operation becomes

$$k^2 \cdot C_{in} + C_{in} \cdot C_{out},$$

which is significantly lower than that of a standard convolution when k is small and C_{out} is large.

To illustrate the efficiency gain, consider a convolution with $k = 3$, $C_{in} = 256$, and $C_{out} = 256$. A standard convolution requires

$$9 \cdot 256 \cdot 256 = 589{,}824$$

parameters, whereas the depthwise separable version requires

$$9 \cdot 256 + 256 \cdot 256 = 2{,}304 + 65{,}536 = 67{,}840.$$

This represents a reduction by nearly a factor of nine, demonstrating the substantial improvement in parameter efficiency.

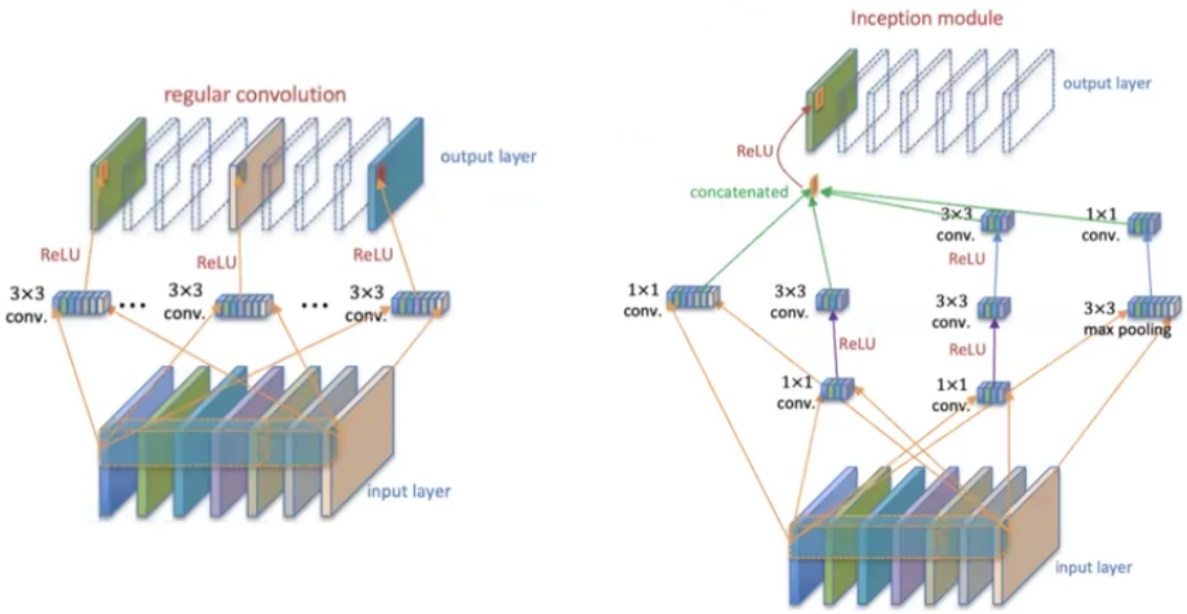

Figure 9: Comparison between standard convolution and depthwise separable convolution.

The overall Xception architecture is organized into three sequential flows: an entry flow, a middle flow, and an exit flow. The entry flow performs initial feature extraction and progressive spatial downsampling while increasing the channel depth. The middle flow forms the computational core of the network and consists of repeated depthwise separable convolutional blocks. These blocks are connected through residual connections inspired by the ResNet architecture, allowing stable gradient propagation and facilitating the training of deep networks. Each residual block learns a transformation $\mathcal{F}(x)$ and adds the input x to the output, resulting in $\mathcal{F}(x) + x$. This residual learning mechanism mitigates vanishing gradients and improves optimization stability.

The exit flow further refines high-level representations and concludes with a `GlobalAveragePooling2D` layer. Instead of using fully connected layers, global average pooling aggregates each feature map into a single scalar by averaging across spatial dimensions. This operation reduces the number of parameters, lowers the risk of overfitting, and produces a compact global feature representation suitable for classification tasks.

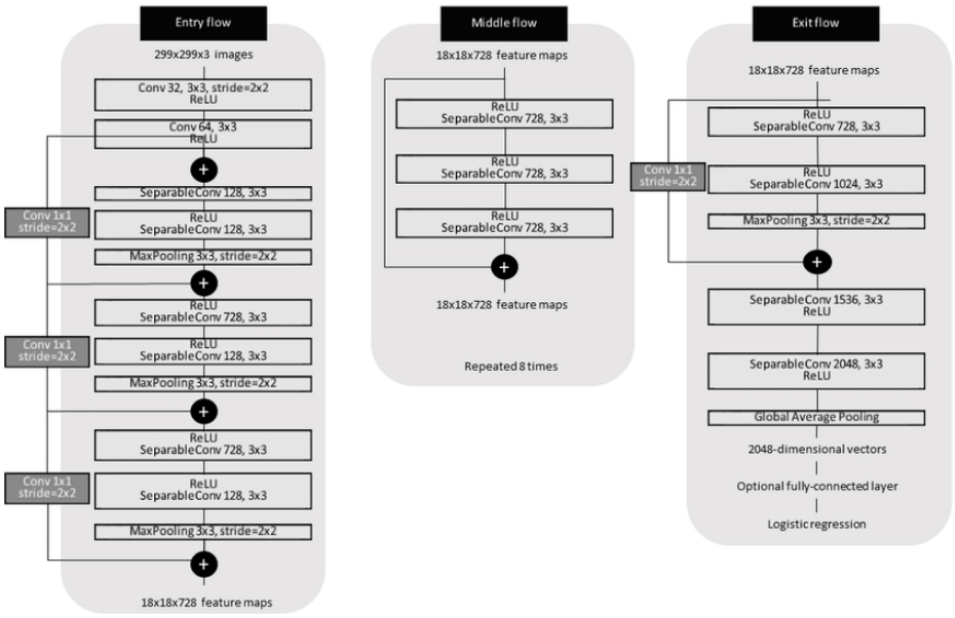

Figure 10: Xception-Flow Architecture

Conceptually, Xception can be interpreted as a fully linear stack of depthwise separable convolutional layers combined with residual learning. By explicitly separating spatial feature learning from channel interaction, the architecture achieves a more efficient and structured representation of image data while maintaining strong predictive performance.

3.3 ResNet Architecture

Residual networks (ResNet) were introduced by He et al. in 2016 [5] and allow stable training of very deep networks. In deep convolutional neural networks, multiple layers are stacked and trained to perform a given task. As depth increases, the network progressively learns hierarchical feature representations, from low-level features (such as edges and textures) to mid-level and high-level features in the deeper layers (such as body parts and objects).

In residual learning, instead of directly learning a direct mapping, the network is designed to learn a residual function. The residual can be understood as the difference between the desired output of a layer and its input. Let x denote the input to a residual block. Instead of directly approximating a mapping $H(x)$, the residual block learns a function $F(x, \Theta)$ parameterized by weights Θ, such that:

$$y = F(x, \Theta) + x.$$

Here, $F(x, \Theta)$ typically represents a sequence of convolutional layers, batch normalization, and nonlinear activation functions. The term x is added via an identity shortcut connection. In its basic form, the residual function is composed of two convolutional layers:

$$F(x) = \sigma\left(W^{(2)} * \sigma\left(W^{(1)} * x\right)\right),$$

where

- $W^{(1)}, W^{(2)}$: weight tensors,
- $*$: convolution operation,
- $\sigma(\cdot)$: nonlinear activation function (ReLU),

- x: input feature map of the residual block.

If the dimensionality of $F(x)$ differs from that of x, a linear projection W_s is applied to the shortcut connection:

$$y = F(x, \Theta) + W_s x,$$

ensuring compatible dimensions for the element-wise addition.
In this formulation,

- $F(x, \Theta)$ is the residual mapping learned by the block as mentioned above
- W_s is a learnable linear projection matrix
- $W_s x$ maps the input tensor to a new representation with dimensions matching those of $F(x, \Theta)$,
- y denotes the output tensor of the residual block after element-wise addition.

The projection W_s is required when the spatial dimensions or the number of channels of x differ from those of $F(x, \Theta)$. By aligning dimensionality before addition, the residual connection remains well-defined and preserves stable gradient propagation during backpropagation.

This approach is implemented in ResNet architectures through the use of shortcut (skip) connections, which directly connect the input of a given layer to the output of a deeper layer. These shortcut connections allow information and gradients to flow more effectively through the network, alleviating the vanishing gradient problem and enabling the successful training of very deep architectures.

As a result, ResNet models achieve improved convergence behavior and superior performance compared to traditional deep convolutional networks without residual connections.

3.4 EfficientNet Architecture

EfficientNet is a family of convolutional neural network architectures proposed by Tan and Le (2019) [6] with the objective of systematically improving the trade-off between predictive accuracy and computational efficiency. While traditional convolutional networks are typically scaled by arbitrarily increasing depth (more layers), width (more channels), or input resolution (larger images), such single-dimension scaling strategies often lead to diminishing returns or inefficient use of parameters. EfficientNet addresses this limitation by introducing a compound scaling framework that jointly scales depth, width, and resolution in a mathematically balanced manner.

To understand the motivation behind EfficientNet, consider a baseline convolutional neural network defined by a function $f(X; \theta)$, where $X \in \mathbb{R}^{H \times W \times C}$ denotes the input image tensor with spatial dimensions H and W and C channels, and θ represents the learnable parameters. Increasing the network depth improves feature abstraction, increasing the width enhances channel-level representation capacity, and increasing input resolution allows finer-grained spatial detail to be captured. However, scaling only one of these dimensions may create representational bottlenecks. For example, increasing depth without increasing width may restrict information flow, while increasing resolution without sufficient capacity may under-utilize spatial detail. EfficientNet therefore proposes to scale all three dimensions simultaneously in a coordinated fashion.

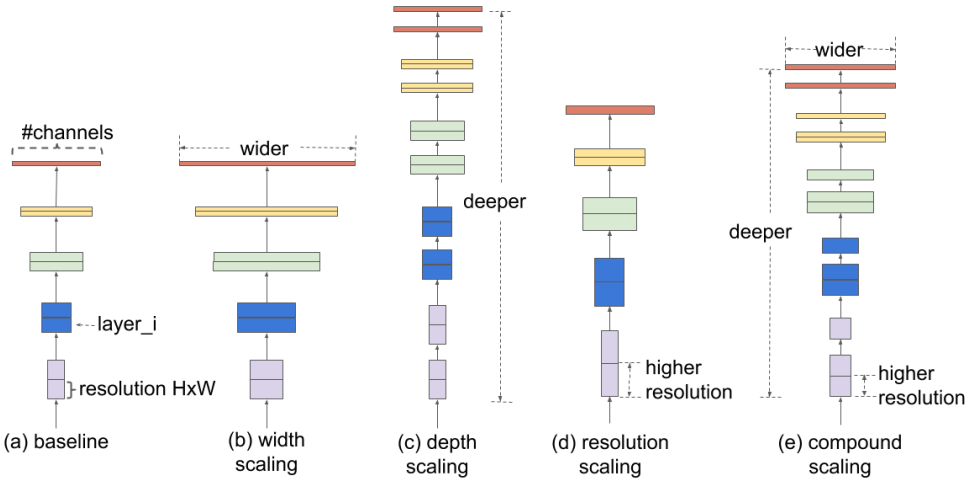

Figure 11: Compound scaling

The core computational building block of EfficientNet is the Mobile Inverted Bottleneck Convolution (MBConv). Unlike classical bottleneck layers that first reduce dimensionality and then expand it, the inverted bottleneck structure first expands the channel dimension and then projects it back to a lower-dimensional space. Let the input tensor be defined as $X \in \mathbb{R}^{H \times W \times C_{in}}$. The first step is the expansion phase, implemented as a 1×1 convolution:

$$X'_{i,j,c} = \sum_{c'=1}^{C_{in}} W^{(e)}_{c',c} X_{i,j,c'},$$

where $X' \in \mathbb{R}^{H \times W \times tC_{in}}$, t denotes the expansion ratio, $W^{(e)}_{c',c}$ are the expansion weights, and c indexes the expanded channels. This operation increases the representational capacity by projecting the input into a higher-dimensional feature space.

The second step is a depthwise convolution applied independently to each expanded channel:

$$Z_{i,j,c} = \sum_{u,v} W^{(d)}_{u,v,c} \, X'_{i+u,j+v,c},$$

where $Z \in \mathbb{R}^{H' \times W' \times tC_{in}}$, $W^{(d)}_{u,v,c}$ represents the depthwise kernel weights for channel c, and (u, v) are the spatial kernel indices. This operation captures spatial correlations while maintaining computational efficiency since no cross-channel mixing occurs at this stage.

The final step is the projection phase, again implemented as a 1×1 convolution:

$$Y_{i,j,c} = \sum_{c'=1}^{tC_{in}} W^{(p)}_{c',c} \, Z_{i,j,c'},$$

where $Y \in \mathbb{R}^{H' \times W' \times C_{out}}$ and $W^{(p)}_{c',c}$ are the projection weights mapping the expanded channels back to C_{out}. This stage reintroduces channel interactions and produces the block output. When input and output dimensions match, a residual connection is added such that the block computes $Y + X$, improving gradient propagation and optimization stability.

EfficientNet further integrates Squeeze-and-Excitation (SE) modules within each MBConv block to model channel-wise dependencies explicitly. Given an intermediate tensor Z, global average pooling is applied to obtain a channel descriptor s_c:

$$s_c = \frac{1}{H'W'} \sum_{i=1}^{H'} \sum_{j=1}^{W'} Z_{i,j,c}.$$

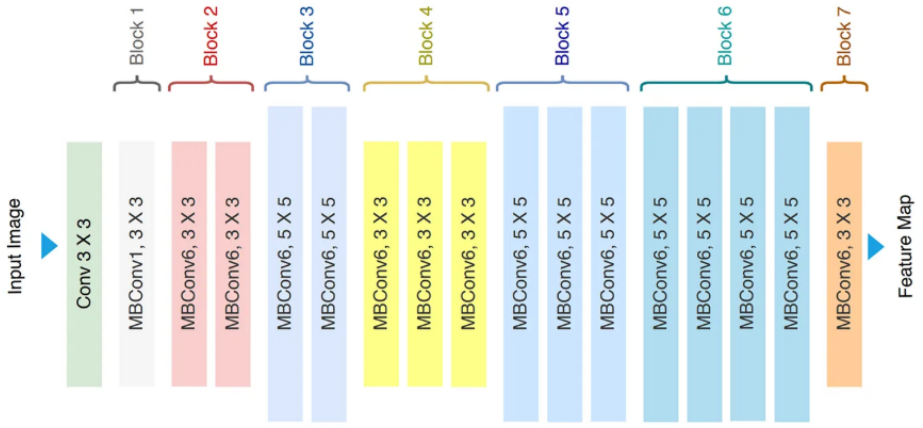

Figure 12: EfficientNet Architecture

Conceptually, EfficientNet can be understood as combining three efficiency principles: (i) inverted bottleneck structures that reduce unnecessary parameter usage, (ii) depthwise separable convolutions that decouple spatial and channel learning, and (iii) compound scaling that avoids representational bottlenecks during model enlargement. This coordinated design enables EfficientNet to achieve state-of-the-art performance on image classification benchmarks while using significantly fewer parameters and floating-point operations than conventional convolutional networks. Due to this favorable accuracy–efficiency trade-off, EfficientNet models are particularly effective in transfer learning scenarios, where pretrained weights on large-scale datasets such as ImageNet can be fine-tuned for domain-specific tasks with limited computational resources.

4 Model Modification

4.0.1 Transfer Learning and Layer Freezing

Deep convolutional neural networks require large amounts of labeled data to learn robust and generalizable feature representations[7]yx. Training such architectures from scratch on a relatively small dataset often results in severe overfitting, as the number of trainable parameters can reach tens of millions. In order to mitigate this issue, a transfer learning strategy is adopted. Transfer learning refers to the process of leveraging knowledge obtained from a large-scale source task and transferring it to a related target task. Instead of initializing all parameters randomly, the network is initialized with weights learned from a large benchmark dataset, allowing the model to start from a well-structured feature space rather than from scratch.

In this study, all three deep convolutional architectures are pretrained on the ImageNet dataset, which contains over one million labeled images across 1000 object categories. ImageNet serves as a large-scale benchmark dataset that enables models to learn generic low-level and mid-level visual features such as edges, textures, shapes, and object parts. Since these fundamental visual patterns are largely transferable across image recognition tasks, the pretrained weights provide a strong initialization for downstream classification problems.

Formally, let $f(x;\theta_{pre})$ denote a pretrained network with parameters θ_{pre} learned on the source dataset (ImageNet). In transfer learning, these pretrained parameters are reused for the target task. The network is then adapted by replacing the final classification layer with a new task-specific output layer, while optionally freezing part of the pretrained parameters. In this work, all convolutional layers of the pretrained backbone are frozen, meaning that their weights are not updated during backpropagation:

```
for layer in model.layers:
```

```
layer.trainable = False
```

Freezing the convolutional base ensures that the general-purpose feature extractors learned from ImageNet remain unchanged. This significantly reduces the number of trainable parameters and lowers the risk of overfitting, particularly when the target dataset is small. Additionally, freezing layers reduces computational cost and stabilizes training, as the pretrained feature representations are already well-structured.

Early convolutional layers capture low-level visual features such as edges and gradients, which are largely universal across image domains. Deeper layers encode more task-specific patterns. By freezing the backbone and training only the final layers, the model effectively acts as a fixed feature extractor followed by a lightweight classifier. This approach is especially appropriate when the target dataset is visually similar to ImageNet, as is the case in this study.

In summary, transfer learning enables the reuse of powerful pretrained feature representations from large-scale datasets. By freezing the convolutional layers of ResNet, Xception, and EfficientNet, the models retain their learned general visual knowledge while adapting only the final classification layers to the specific task, thereby improving generalization performance and reducing overfitting.

4.0.2 Custom Classification Head

Since the pretrained Xception model ends with a `GlobalAveragePooling2D` layer, a custom classification head is appended to adapt the network to the dog breed classification task. This head consists of a fully connected layer with 128 neurons and ReLU activation, followed by a Dropout layer with a dropout rate of 0.2 to further reduce overfitting. Finally, a softmax output layer is added to perform multi-class classification over the dog breed categories.

Formally, the final layers are defined as:

- A dense layer with 128 units and ReLU activation
- A Dropout layer with a rate of 0.2
- A dense softmax classifier with C output units, where C denotes the number of dog breeds

The resulting model is trained using the Adam optimizer and sparse categorical cross-entropy loss. This architecture allows the model to leverage powerful pretrained feature extractors while learning a task-specific decision boundary through the newly added classifier layers.

5 Results

Figure **??** presents the performance of the implemented convolutional neural network architectures over the course of training epochs. The baseline CNN demonstrates the lowest performance, with a mean accuracy of approximately 0.18, reflecting its limited capacity to capture fine-grained features. The ResNet50 model shows improved performance, reaching a mean accuracy of 0.66, highlighting the benefit of residual connections for stable gradient propagation and deeper feature learning. Xception achieves a higher mean accuracy of 0.78, demonstrating the efficiency and effectiveness of depthwise separable convolutions in capturing spatial and channel-wise correlations. EfficientNet outperforms the other models with a mean accuracy of 0.81, benefitting from its compound scaling strategy that balances network depth, width, and resolution while maintaining computational efficiency. Overall, these results indicate that modern architectures such as Xception and EfficientNet are particularly suitable for fine-grained image classification tasks, providing superior accuracy compared to traditional CNNs.

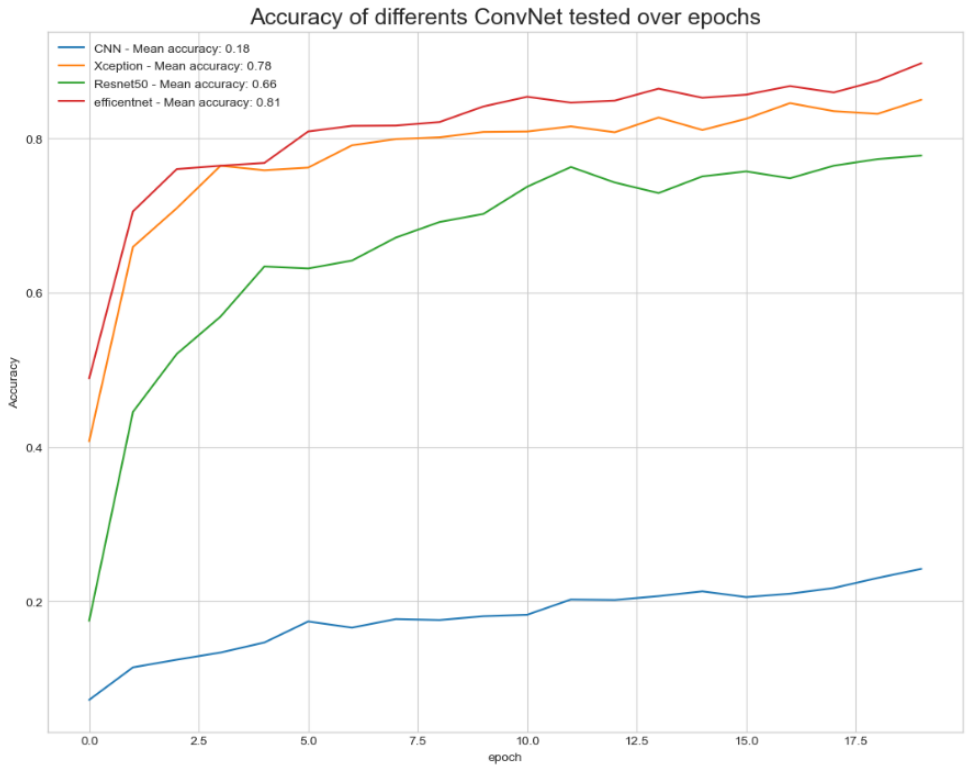

Figure 13: Accuracy of different ConvNet architectures tested over epochs

The figure below presents the Grad-CAM attention heatmaps generated for the Xception model. These heatmaps visualize the spatial regions within the input images that contribute most strongly to the model's classification decision. Warmer colors (red and yellow) indicate areas of higher importance, while cooler colors (blue) represent regions with lower influence on the prediction.

It can be observed that the model primarily focuses on discriminative features of the dog, such as the facial structure, ears, and body contours, rather than background elements. This behavior suggests that the Xception network has successfully learned meaningful and task-relevant visual representations. The attention maps therefore provide interpretability and confirm that the model bases its predictions on semantically important regions of the image.

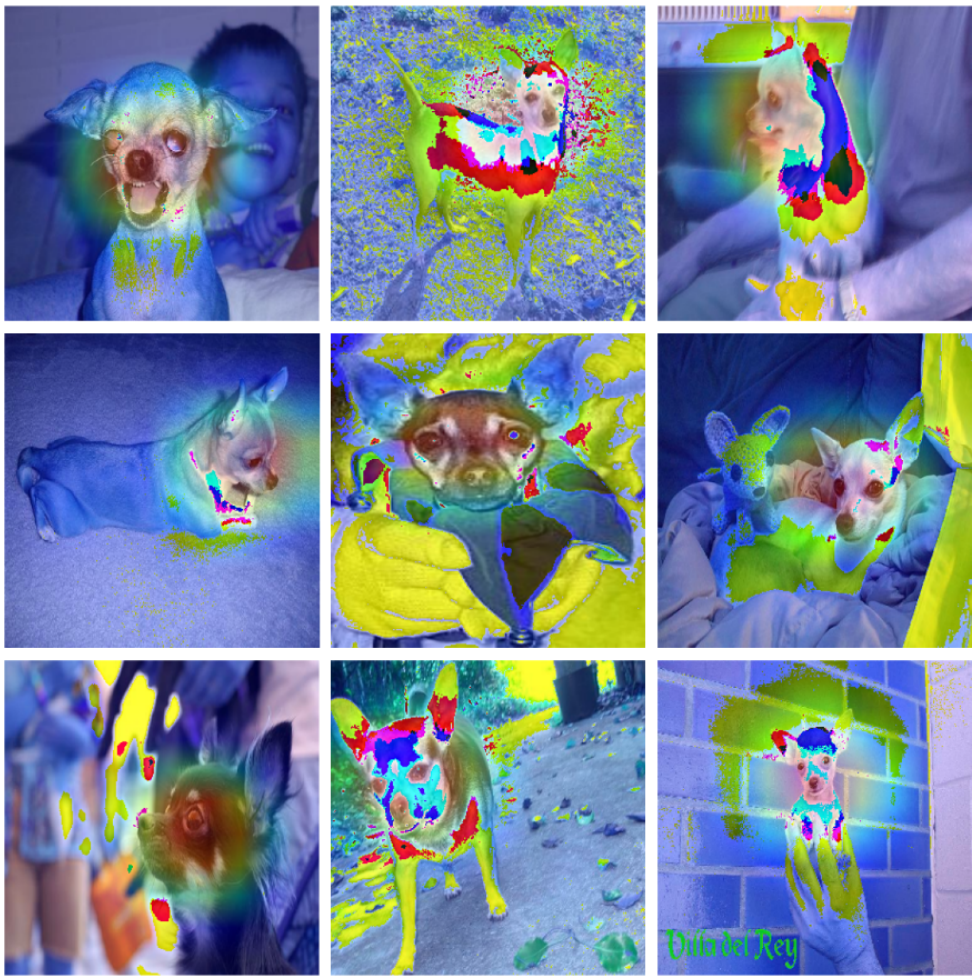

Figure 14: Attention heatmap of Xception Model

Due to the large number of classes in the dataset (120 dog breeds), a conventional correlation heatmap between predicted and true classes is not presented. Visualizing a full 120×120 correlation matrix would result in a highly dense and difficult-to-interpret figure, limiting its practical value. Instead, attention visualization techniques such as Grad-CAM are used to provide more meaningful insights into the model's decision-making process.

References

[1] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "The stanford dogs dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2013.

[2] G. Van Horn *et al.*, "The inaturalist species classification and detection dataset," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[3] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Prentice Hall, 2002.

[4] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[6] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the International Conference on Machine Learning*, 2019.

[7] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.