DSTI

Deep Learning

Group Project / A22

Aritra MONDAL Kaan YUCEEL Khalil BENSAID
Github:
https://github.com/Khalilben96/DeepLearningProject

Table of Contents

Introduction

The current project aims to come up with an efficient model of deep learning for the classification of animal images into different classes. Being an essential task in various applications, such as wildlife protection, farming, and automatic image tagging, image classification allows for the exact identification of species.

Here, we applied transfer learning, an approach that would exploit the capabilities of pre-trained models, particularly the VGG16 architecture, which was trained on vast ImageNet. This approach will enable us to take advantage of the features learned from previous learning in order to improve the performance of our model in this new but related task of classifying animal images.

In this regard, we used Grad-CAM as a visualization methodology to highlight the regions of an input image that are most relevant for the predictions made by the model. This would allow a better understanding of the decision process of the neural network and provide insights on which particular features of the images are being used for classification purposes.

In the final analysis, this work synergizes state-of-the-art deep learning with effective visualization techniques for building a robust and interpretable model for the classification of animal images.

Data Preparation and Augmentation

This research project uses an image data set acquired from Kaggle: *https://www.kaggle.com/datasets/alessiocorrado99/animals10/data*. The data set contains about 28,000 medium-quality images, organized in folders corresponding to ten different animal categories: dog, cat, horse, spider, butterfly, chicken, sheep, cow, squirrel, and elephant. Each category contains from 2,000 to 5,000 images, thus providing a balanced representation for the different classes.

Data Augmentation Techniques:

We used a number of augmentation techniques to increase generalization and robustness with the help of the ImageDataGenerator class in TensorFlow/Keras. The transformations applied are as follows:

1. **Rescaling:** Pixel values were normalized by dividing by 255 to bring them into the range [0, 1], which aids in faster convergence during training.
2. **Random Transformations:**
    - **Rotation:** Randomly rotating images to create invariance to orientation.
    - **Translation:** Shifting images horizontally and vertically to simulate different viewpoints.
    - **Shearing:** Applying shear transformations to alter the perspective of the images.
    - **Zooming:** Random zooming in and out to simulate different distances from the subjects.
    - **Horizontal Flipping:** Flipping images to introduce additional variability, particularly beneficial for animals that may appear symmetrically.

The proportion designated for validation was established at 0.15, thereby allocating 15% of the dataset for validation purposes, while utilizing the remaining 85% for training activities. This division is essential for assessing the efficacy of the model on previously unobserved data, contributing to the reduction of overfitting.

Model Architecture and Compilation

In the present study, the VGG16 model architecture, which has been pre-trained on the ImageNet dataset, forms the basis for our image classification endeavors. The VGG16 model is distinguished by its considerable depth and proficiency in extracting comprehensive feature representations, thereby rendering it an appropriate option for transfer learning purposes.

Model Architecture:

The VGG16 architecture was modified by excluding the fully connected layers (i.e., setting include_top=False) to adapt it to our specific classification task that involves ten animal classes. The following changes were implemented:

1. **Global Average Pooling Layer:**
    - This layer replaces the fully connected top layers of VGG16. It aggregates the feature maps by averaging, significantly reducing the number of parameters while retaining essential spatial information.
2. **Dense Layer:**
    - A fully connected layer with 512 units and a **ReLU** (Rectified Linear Unit) activation function was added to process the pooled features. This layer helps in learning complex combinations of features extracted by the convolutional base.
3. **Output Layer:**
    - The final layer consists of 10 units with a **softmax** activation function, designed to classify the images across the ten animal classes. This setup allows the model to output probabilities for each class, enabling us to determine the most likely classification for an input image.

Compilation:

The model was compiled with the following specifications:

- **Loss Function: Categorical Cross-Entropy**
    - This loss function is appropriate for multi-class classification problems, measuring the performance of the model by comparing predicted class probabilities with actual labels.
- **Optimizer: Adam Optimizer**
    - The Adam optimizer was selected due to its adaptive learning rate capabilities, enhancing convergence speed and efficiency. A learning rate of 1×10−41 \times 10^{-4}1×10−4 was set, balancing the exploration of the loss landscape.
- **Metrics:**
    - **Accuracy** was chosen as the primary evaluation metric to monitor model performance during training and validation.

Below is the architecture diagram showing changes made in the VGG16 model, illustrating the flow from input images to the final classification probabilities through the various layers.

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 14,982,474
Trainable params: 14,982,474
Non-trainable params: 0
_________________________________________________________________
```

Callbacks

During the training phase of deep learning models, callbacks are used as tools to monitor and adjust the training process in real time. For this project, three critical callbacks were used to enhance the model's effectiveness and reduce the chance of overfitting:

1. **ModelCheckpoint:**
    - This callback saves the model at the end of each epoch if the validation loss improves. By retaining only the best-performing weights, it ensures that we can retrieve the optimal model state, effectively preventing overfitting during training.
    - **Implementation:**
        - This was configured to monitor the validation loss and set to save the model in a specified filepath.
2. **EarlyStopping:**
    - EarlyStopping monitors the validation loss and halts training when it no longer improves for a specified number of epochs (in this case, set to five). This helps in avoiding overfitting by preventing the model from continuing to learn from noise in the training data after it has reached its peak performance.
    - **Implementation:**
        - The patience parameter was set to five, indicating that training will stop if there is no improvement in validation loss for five consecutive epochs.
3. **ReduceLROnPlateau:**
    - This callback reduces the learning rate when the validation loss plateaus, indicating that the model has stopped making significant improvements. A lower learning rate can help in fine-tuning the model’s weights, allowing it to converge more effectively.
    - **Implementation:**
        - The monitoring metric is set to validation loss, and a factor to reduce the learning rate was specified.

Effectiveness of Callbacks:

To evaluate the impact of these callbacks on the training process, a snip from the output below, highlights the model's performance over epochs.

```
Epoch 10/15
1392/1392 [==============================] - ETA: 0s - loss: 0.1848 - accuracy: 0.9428
Epoch 10: val_loss did not improve from 0.27366
1392/1392 [==============================] - 942s 677ms/step - loss: 0.1848 - accuracy:
0.9428 - val_loss: 0.2935 - val_accuracy: 0.9176 - lr: 1.0000e-04
Epoch 11/15
1392/1392 [==============================] - ETA: 0s - loss: 0.1667 - accuracy: 0.9479
Epoch 11: val_loss improved from 0.27366 to 0.26785, saving model to model.keras
1392/1392 [==============================] - 720s 517ms/step - loss: 0.1667 - accuracy:
0.9479 - val_loss: 0.2678 - val_accuracy: 0.9227 - lr: 1.0000e-04
Epoch 12/15
1392/1392 [==============================] - ETA: 0s - loss: 0.1634 - accuracy: 0.9486
Epoch 12: val_loss improved from 0.26785 to 0.21310, saving model to model.keras
1392/1392 [==============================] - 706s 507ms/step - loss: 0.1634 - accuracy:
0.9486 - val_loss: 0.2131 - val_accuracy: 0.9355 - lr: 1.0000e-04
```

The use of these callbacks significantly improved the model's training efficiency and performance, ensuring that the model converged appropriately while maintaining generalization capabilities.

Model Training and Evaluation

The training step of the deep learning model is the most crucial step to ensure the model works well with unseen data. In this work, the model was trained for a total of 15 epochs with certain settings to enhance learning and generalization.

Training Configuration

- **Batch Size:**
  - Batch size 32 was selected in order to properly balance between memory efficiency and convergence speed. Smaller batch sizes may cause higher noise, while very large batch sizes may result in longer training time and less frequent updates.
- **Data Splitting:**
  - The dataset was divided into training and validation subsets using a 15% validation split. It means that 15% of the data is used for validation, which will fairly evaluate how effective the model is on unseen data while training.

Training Process

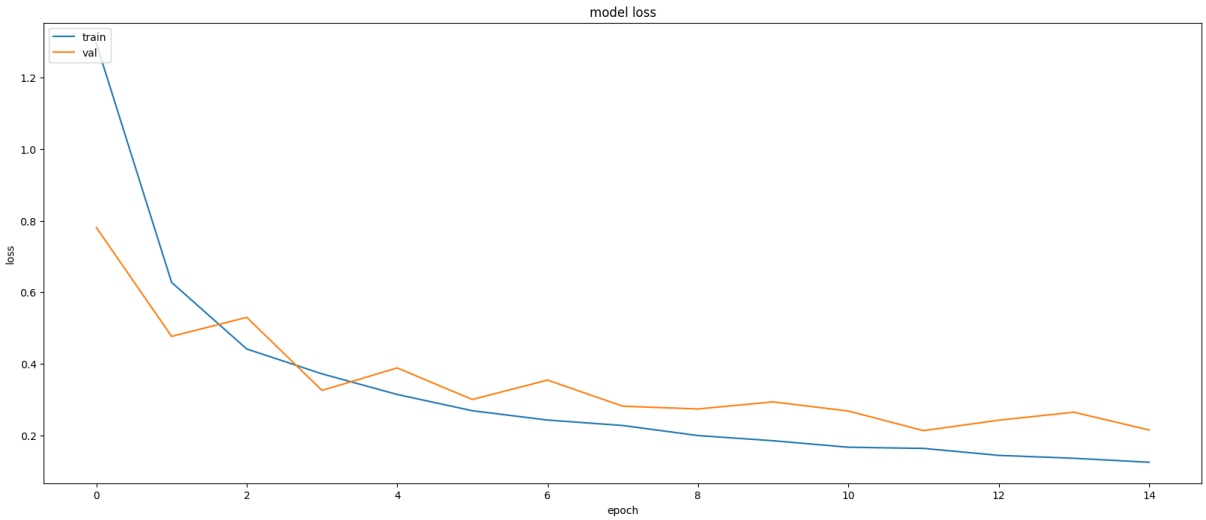
During the training process, both training and validation metrics were monitored to assess the model's performance:

- **Loss and Accuracy:**
  - The model's loss and accuracy were recorded for both training and validation datasets at the end of each epoch. This information is critical for diagnosing potential issues such as overfitting or underfitting.

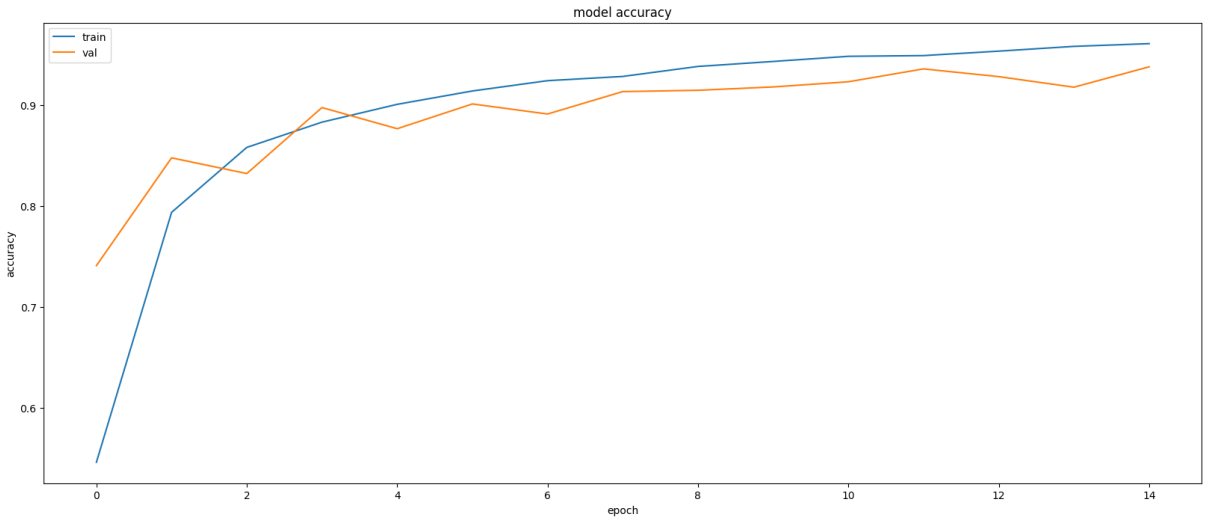
Performance Metrics

The following plots were generated to visualize the training process and evaluate model performance:

- **Training Loss vs. Validation Loss:**
  - A decreasing training loss alongside a steady or slightly increasing validation loss may indicate overfitting, while similar trends in both curves suggest good generalization.



- **Training Accuracy vs. Validation Accuracy:**
  - The accuracy plots help in assessing how well the model is performing on both the training and validation datasets, providing insights into its learning capabilities.



Prediction and Performance Reporting

After training the model, predictions were made on the validation set to evaluate its classification performance comprehensively. This section outlines the metrics used for assessment and presents a classification report summarizing the model's effectiveness.

Prediction on Validation Set: The model made predictions on the images in the validation set. These predictions were then compared to the true labels for calculating accuracy.

Classification Report

The classification report presents key performance metrics for each animal class, including **precision**, **recall**, and **F1-score**. The following table summarizes the results:

Class	Precision	Recall	F1 Score	Support (Samples)
Cane (Dog)	0.88	0.96	0.92	729
Cavallo (Horse)	0.93	0.92	0.93	393
Elefante (Elephant)	0.91	0.95	0.93	216
Farfalla (Butterfly)	0.96	0.93	0.95	316
Gallina (Hen)	0.98	0.94	0.96	464
Gatto (Cat)	0.95	0.93	0.94	250
Mucca (Cow)	0.93	0.91	0.92	279
Pecora (Sheep)	0.91	0.82	0.86	273
Ragno (Spider)	0.96	0.98	0.97	723
Scoiattolo (Squirrel)	0.96	0.89	0.92	279
Cane (Dog)	0.88	0.96	0.92	729
Cavallo (Horse)	0.93	0.92	0.93	393

Implications of Metrics:

- **Precision:** This is the accuracy of positive predictions. High precision means the model makes few false positive errors.
- **Recall:** Measures the model's ability to pick up all the relevant examples (true positives). High recall means fewer false negatives.
- **F1-Score:** Harmonic mean of precision and recall, giving a single score that can be used to evaluate the model's performance. The higher the F1-score, the better the balance between precision and recall.

Analysis of Results

The classification report indicates strong performance across most classes, particularly with the **Gallina (Hen)** and **Farfalla (Butterfly)** categories, which exhibited precision and recall values nearing 1. However, classes such as **Pecora (Sheep)** show lower recall values, suggesting potential misclassifications or difficulties in distinguishing this class from others.

Visuals for Correctly and Incorrectly Classified Images

To gain further insight into the model's performance, we examined specific examples of images that were classified correctly and incorrectly. This qualitative analysis helps to understand the strengths and weaknesses of the model in real-world scenarios.

Correctly Classified Images

The following images illustrate examples where the model successfully predicted the correct animal classes. These instances indicate that the model effectively recognized distinguishing features of each class.

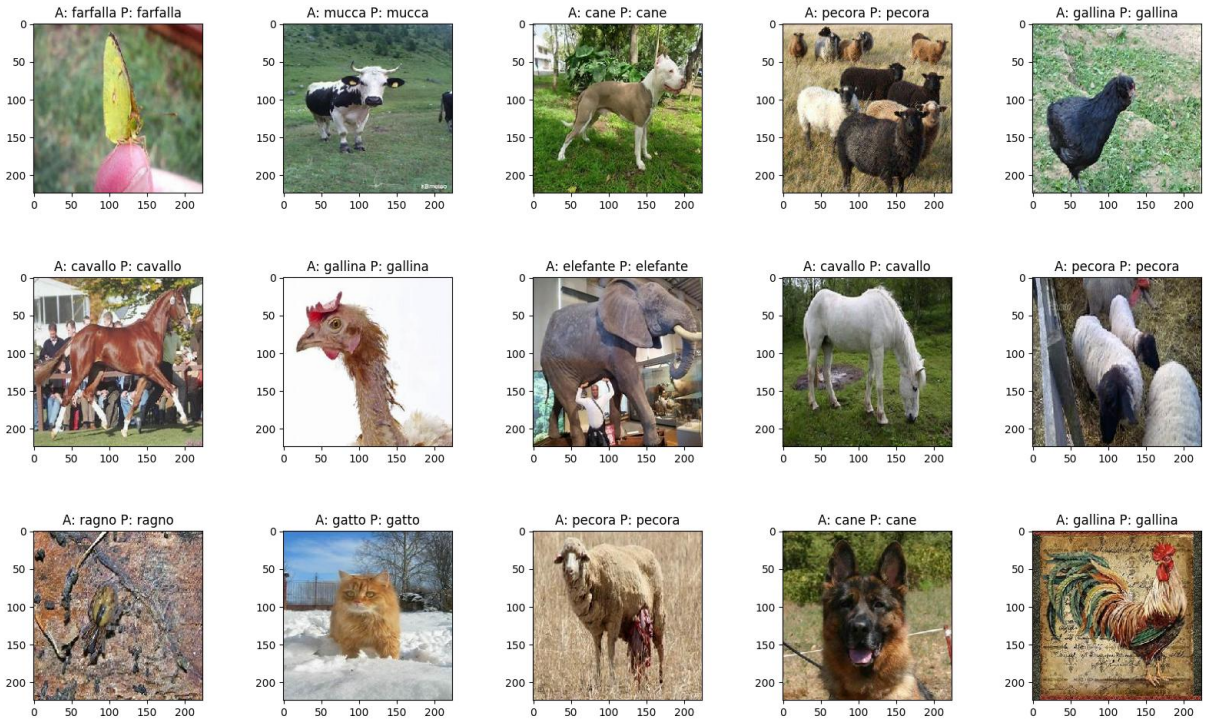


Figure X: Examples of correctly classified images.

Incorrectly Classified Images

In contrast, the following images show instances where the model misclassified the animal classes. Analyzing these examples reveals patterns that may indicate the model's limitations or challenges in distinguishing between similar classes or the influence of background elements.

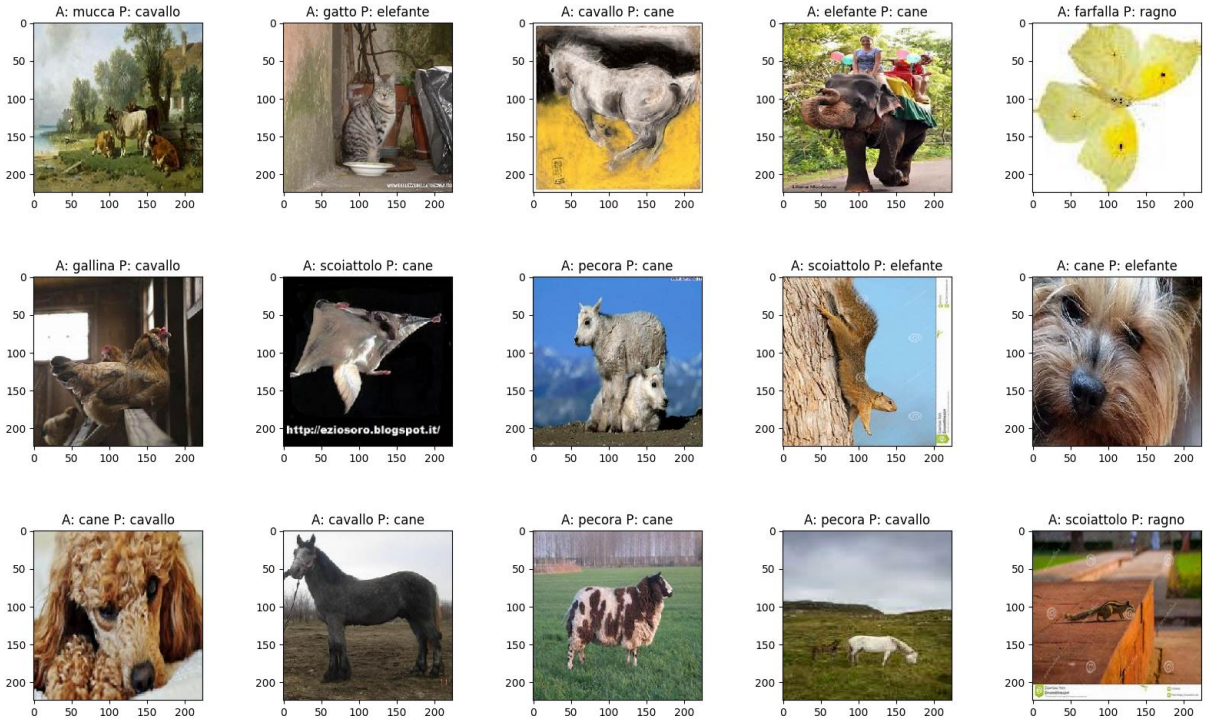


Figure Y: Examples of incorrectly classified images.

Analysis of Misclassifications

1. **Patterns in Misclassifications:**
    - Misclassifications often occur in classes with similar visual features, such as **Cane (Dog)** and **Gatto (Cat)**, where the model may struggle to differentiate between them due to similar shapes or colors.
    - Background noise or occlusions in the images can also contribute to incorrect predictions, as seen in some instances where the model focused on irrelevant features.
2. **Implications for Model Improvement:**
    - Insights gained from analyzing these misclassified images can inform future strategies for enhancing model accuracy. Techniques such as incorporating additional training data, refining data augmentation strategies, or utilizing more sophisticated model architectures may help address these challenges.

Model Interpretability Using Grad-CAM

Grad-CAM (Gradient-weighted Class Activation Mapping) is a powerful technique that provides insights into how deep learning models make predictions by highlighting the important regions in the input images. This section outlines the implementation of Grad-CAM in our animal image classification project, detailing the steps involved and the insights gained from visualizing model attention.

Implementation Steps for Grad-CAM

1. **Modified Model for Grad-CAM:**
    - To utilize Grad-CAM, we first modify our existing model to obtain both the feature maps from the last convolutional layer and the final predictions. This allows us to compute the gradients of the predicted class score concerning the output of the convolutional layer.
2. **Backpropagation of Gradients:**
    - Using backpropagation, we calculate the gradients of the output class score concerning the feature maps of the last convolutional layer. These gradients indicate the importance of each feature map in contributing to the model's final prediction.
3. **Computing Importance Weights:**
    - The gradients are pooled (averaged) to obtain importance weights, which are then applied to the feature maps. This process generates a superimposed map that highlights the regions that significantly influence the model's decision.
4. **Generation of Heatmap:**
    - The generated heatmap is colorized using a 'jet' colormap and resized to match the dimensions of the original input image. This heatmap visually represents the areas where the model has focused its attention while making predictions.
5. **Superimposing Heatmap on Original Image:**
    - Finally, the heatmap is superimposed on the original image, creating a composite visualization that indicates which regions of the input image were deemed important by the model.

Visualization of Grad-CAM for Correct and Incorrect Classifications

The following examples illustrate the Grad-CAM visualizations for both correctly classified and misclassified samples:

- **Correct Classifications:**
  - For correctly classified images, Grad-CAM heatmaps demonstrate that the model focuses on relevant parts of the animals, confirming that it utilizes the expected features for accurate predictions.

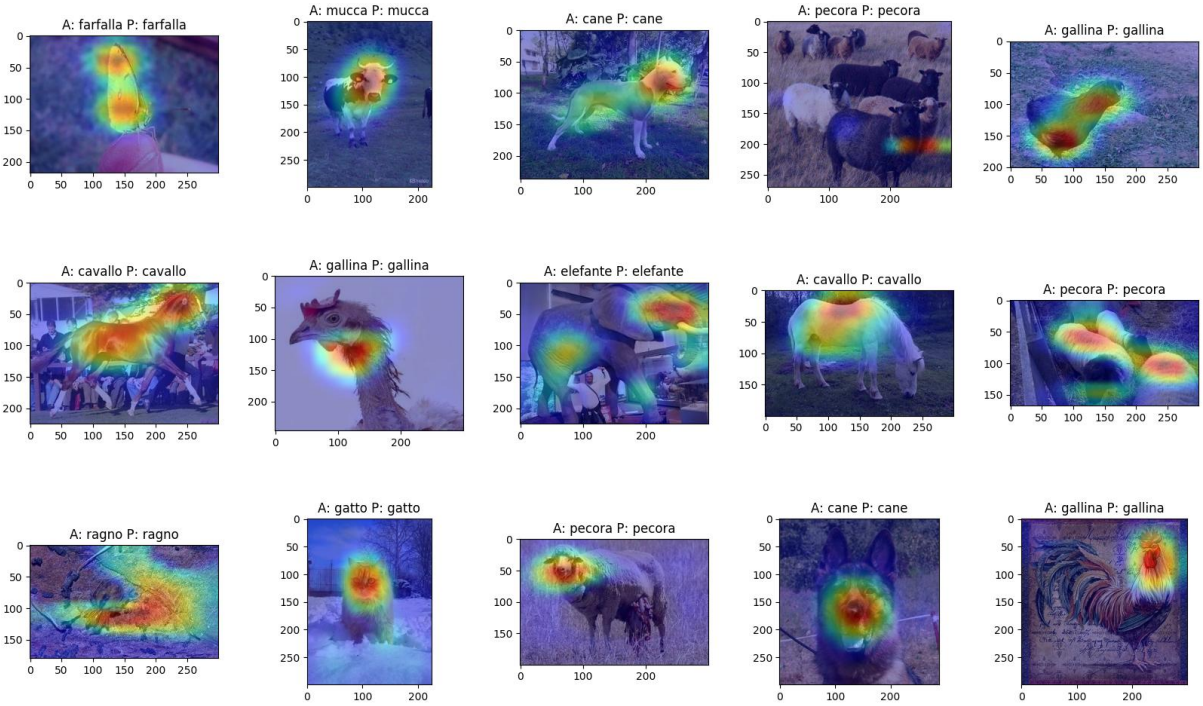


Figure X: Grad-CAM visualization for correctly classified images, highlighting relevant features.

- **Incorrect Classifications:**
  - In misclassified examples, Grad-CAM visualizations can reveal whether the model was misled by background elements or non-informative features, providing insights into the model's weaknesses.

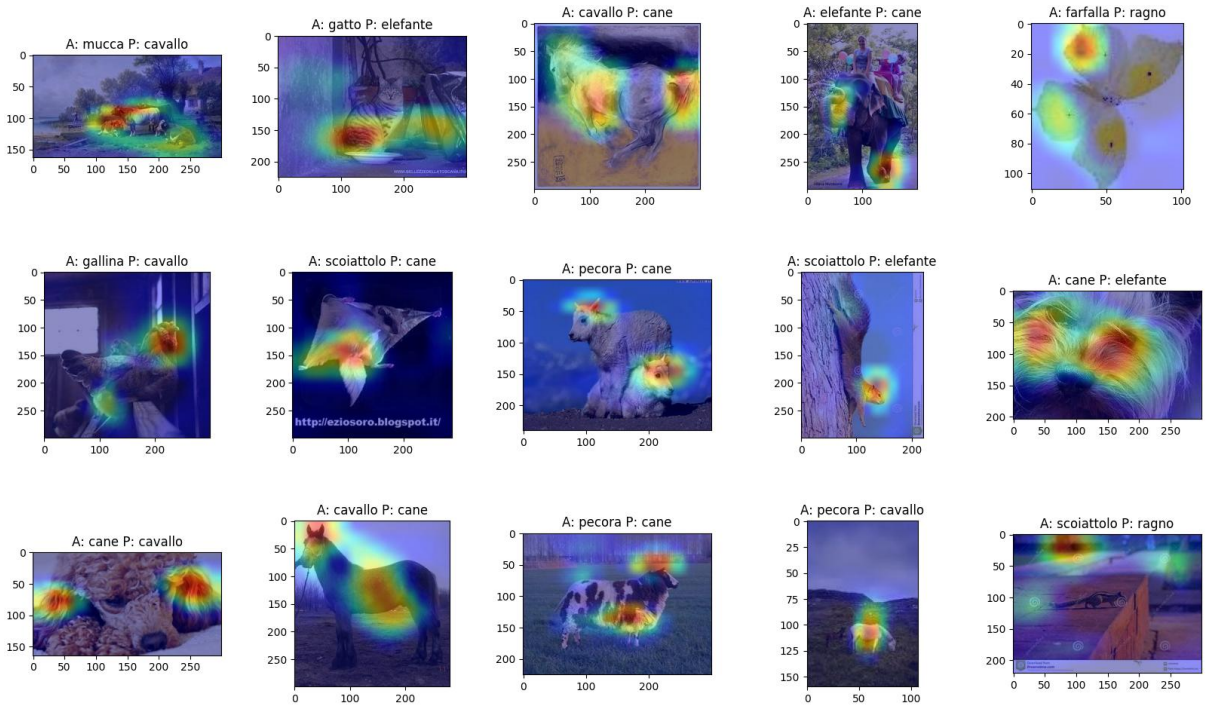


Figure Y: Grad-CAM visualization for incorrectly classified images, indicating model focus areas.

Conclusion of Grad-CAM Analysis

The implementation of Grad-CAM significantly enhances model interpretability by providing visual feedback on how the model arrives at its predictions. This understanding can guide further refinement of the model and data strategies, ultimately leading to improved classification accuracy and robustness.

Conclusion

This project effectively demonstrates the application of transfer learning using the VGG16 architecture for classifying animal images. By leveraging a pre-trained model on the ImageNet dataset, we achieved significant improvements in classification performance while minimizing the need for extensive computational resources and large datasets.

Key findings from the project include:

1. **Performance Metrics:**
    - The model achieved high accuracy across various animal classes, as reflected in the detailed classification report. Metrics such as precision, recall, and F1 scores indicate the model's reliability and efficiency in distinguishing between different animal types. Notably, classes such as dogs, horses, and butterflies exhibited excellent performance, while the sheep and squirrel categories identified areas for potential improvement.
2. **Robustness through Data Augmentation:**
    - The extensive use of data augmentation techniques not only enhanced the model's robustness against overfitting but also improved its generalization capabilities. Techniques like random rotation, translation, and flipping introduced variability, enabling the model to learn diverse features from the training data effectively.
3. **Insights from Grad-CAM:**
    - Grad-CAM visualizations provided valuable insights into the model's decision-making process. By highlighting important regions in the images, we gained a clearer understanding of which features the model relied on for its predictions. This interpretability is crucial for identifying potential areas of improvement, particularly in classes where performance metrics fell short.
4. **Future Work:**
    - Building on the current findings, future work may involve experimenting with more advanced architectures, such as EfficientNet or ResNet, to further enhance classification accuracy. Additionally, integrating more diverse datasets could improve the model's performance across less-represented animal classes, especially for sheep and squirrels, which exhibited lower recall rates.

In summary, this project not only showcases the effectiveness of transfer learning in image classification tasks but also emphasizes the importance of model interpretability through techniques like Grad-CAM. By providing insights into model behavior, we can continue to refine and improve our classification models, contributing to advancements in the field of deep learning and computer vision.