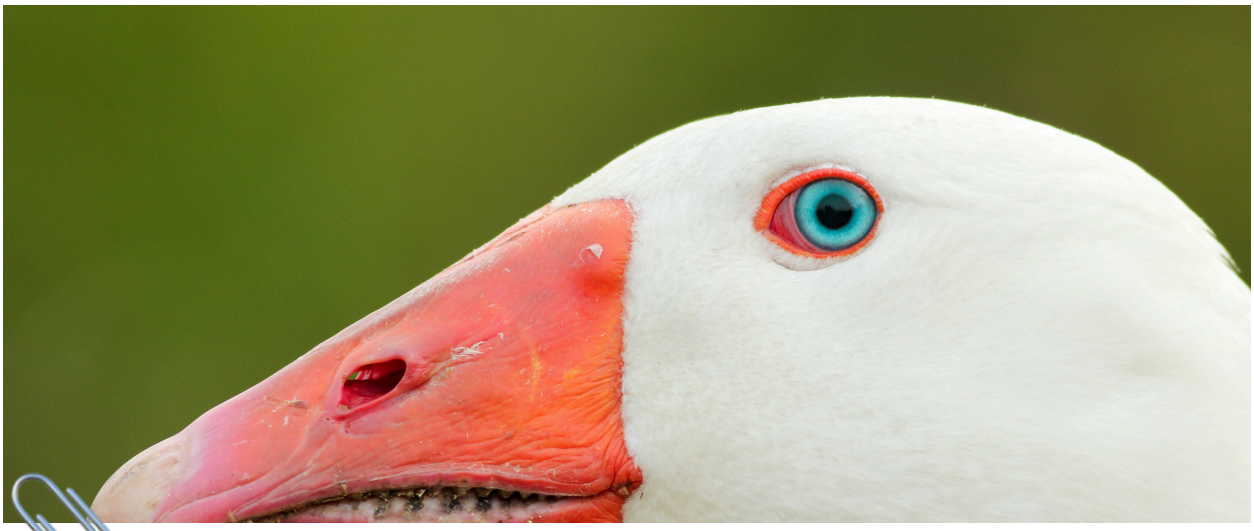


Report - Project I | Deep Learning: Image Classification with CNN

👥 Alex, Nico, Sam

📎 Template

Development CNN Model

Data Preprocessing

The dataset, stored on Google Drive, was split 80/20 into training and testing sets. Original Italian class labels were translated into English for clarity and used to rename folders. Images were normalized and augmented (e.g., horizontal flipping) to improve generalization. Sample images with labels verified preprocessing accuracy.

Basic Model Architecture

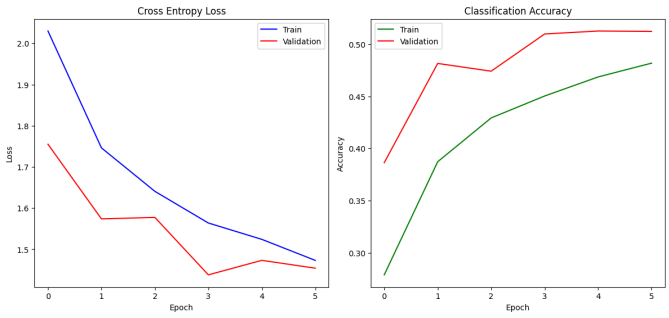
The model is a simple sequential image classifier. It starts with a convolutional layer to detect basic patterns, followed by pooling to reduce data size and emphasize key features. A second convolutional and pooling layer captures more complex patterns.

Data is then flattened and passed through a fully connected layer for high-level representation learning. A dropout layer reduces overfitting by randomly disabling some connections during training. The output layer uses softmax to produce class probabilities for predictions.

Model Training

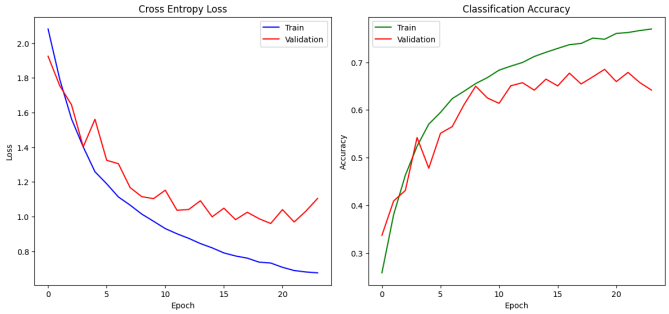
The model was compiled using the Adam optimizer and categorical crossentropy as the loss function, with accuracy as the evaluation metric. Early stopping was used to monitor validation loss and stop training if performance did not improve after two rounds. The model was trained for up to ten epochs using the training and validation data, then saved to the local directory.

The model achieves an accuracy of 0.48 with a loss of 1.47 on the training set, while validation accuracy is around 0.51 with a loss of 1.45. Both accuracy and loss indicate that there is room for improvement in model performance.



According to Kaggle homebrew models achieve an accuracy of 80%, therefore we knew that there is still room for improvement.

To further improve performance, we increased the image size to 128×128 up from 64×64. We added extra generalization methods to the ImageGenerator. Additionally we added convolutional layers in each block



to help the model learn more
complex patterns.

Dropout layers were added after each block to reduce overfitting. We also increased the size of the dense layer to 512 units for better representation learning and extended training to up to 50 epochs to allow more learning time. Since we are working with a slight imbalance in the dataset, we also made sure to balance it.

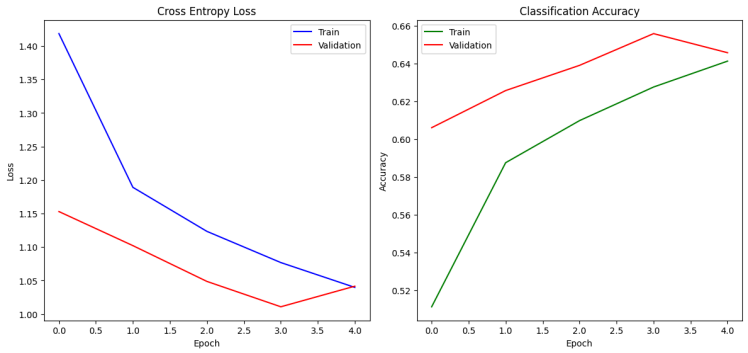
Transfer Learning

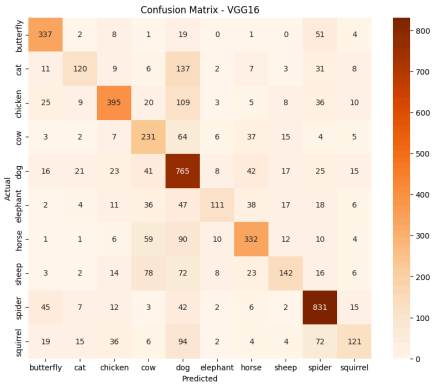
We chose VGG16 because it is a well established architecture that is easy to understand and modify, it has strong performance on many image classification tasks, and it is widely used as a feature extractor in transfer learning. Pretrained weights are readily available in TensorFlow, which makes implementation straightforward.

Transfer Learning Base Model

We trained the model for five epochs using the training and validation data generators. After training, we used the model to predict class probabilities on the validation set and converted those predictions into class labels.

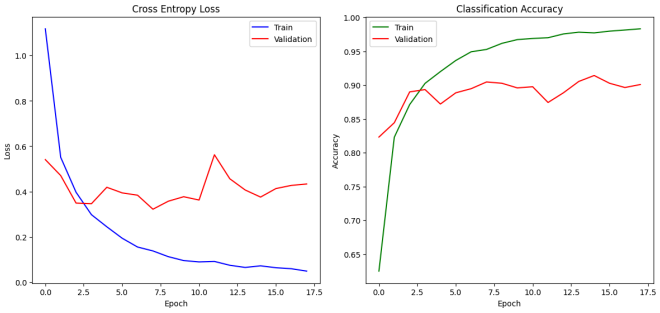
The model was configured with the top classification layers frozen to preserve pretrained weights and trained for 5 epochs. It achieved 65 percent accuracy on the validation set with an F1 score of 0.5966. While it performed well on certain classes, there was clear room for improvement in recall and balance across categories.

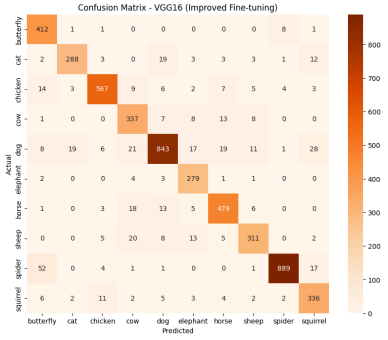




Transfer Learning Improved Model

The model was improved by unfreezing the last two convolutional blocks for fine tuning, reducing the dense layer from 256 to 128 units, lowering the learning rate to 1e-4, adding EarlyStopping with a patience of 10, applying class weights to address imbalance, and increasing the maximum epochs to 50. These changes raised validation accuracy from about 65 percent to about 90 percent, increased the F1 score from 0.60 to 0.90, boosted precision from 0.66 to 0.89, and improved recall from 0.58 to 0.91, though the results show a tendency toward overfitting.





Model deployment

The Animal Image Classifier uses two Keras models (VGG16 and a custom team model) to classify animal images into 10 categories. A Flask backend handles image uploads and routes predictions, while a clean frontend allows users to

preview and classify images using either model. JavaScript fetches predictions in real time, displaying results instantly. The result is a fully functional web-based image classification tool that allows users to:

- Upload animal images
- Choose between two prediction models
- View side-by-side classification results with confidence scores
- Access the app through the internet using Ngrok