Deep Learning-Based Binary Classification of Cats vs. Dogs

Qatar University  
Selected Topic in Computer Engineering L02  
Prof. Saeed Salem  
May 2025

Abdelrahman Abdelwahed 202207121

Amer Alzawawi 2205054

# Abstract

This project implements a deep learning model to automatically classify images of cats and dogs. It utilizes a Convolutional Neural Network (CNN) trained on a subset of the Kaggle Cats vs. Dogs dataset. By employing data preprocessing techniques including normalization, augmentation, and handling of corrupted images, the system ensures a robust and efficient training process. We evaluate the model using key metrics such as accuracy, precision, recall, and F1-score, and visualize its performance using loss curves and confusion matrices. The study also explores the impact of transfer learning with MobileNetV2 to compare and improve results.

# 1. Introduction

Binary image classification is a foundational problem in computer vision, with widespread applications in security, healthcare, and digital content management. The classification of cats and dogs presents a well-known benchmark problem. Traditional image recognition tasks required manual feature engineering, which is time-consuming and less accurate. With the advancement of deep learning, particularly CNNs, automated feature extraction has significantly improved classification accuracy.

This project aims to develop and evaluate a CNN-based system that distinguishes between cats and dogs using the Kaggle PetImages dataset. The process involves data preparation, CNN architecture design, model training, evaluation, and performance visualization.

# 2. Dataset Description

The project uses the Kaggle Cats vs. Dogs Dataset. This dataset contains 25,000 labeled images (12,500 cats and 12,500 dogs) of varying resolution and quality. To reduce training time and control overfitting, we use a subset: 1,000 valid images from each class.

The dataset is pre-organized into two folders:

/small\_dataset/

/Cat/

/Dog/

# 3. Preprocessing and Augmentation

We employed several preprocessing techniques to clean and enhance the dataset:

3.1. Corrupted Image Removal

Some images are unreadable or corrupted. These were detected using PIL’s verify() function. This ensures only valid images are used in training.

3.2. Normalization

Pixel values are rescaled to the range [0, 1] to standardize inputs for stable and faster learning.

3.3. Augmentation

To prevent overfitting and improve generalization, we applied:

- Rotation: up to ±20°

- Zoom: up to 20%

- Horizontal flip

# 4. Model Architecture

We developed two models:

4.1. Custom CNN

- 3 convolutional layers with ReLU activations

- Batch normalization

- Max pooling layers

- Dropout to reduce overfitting

- Final dense layer with sigmoid activation

4.2. Transfer Learning (MobileNetV2)

- Pretrained MobileNetV2 as base

- Custom classification head

- Base layers frozen during training

# 5. Training and Validation

- Train-validation split: 80/20

- Loss function: Binary Crossentropy

- Optimizer: Adam

- Batch size: 32

- Epochs: 24

Training and validation accuracy/loss were monitored across epochs.

# 6. Evaluation Metrics

After training, the model was evaluated on validation data using:

- Accuracy: Overall classification correctness

- Precision: Correct dog predictions / all dog predictions

- Recall: Correct dog predictions / actual dogs

- F1-Score: Harmonic mean of precision and recall

Accuracy: 0.91

Precision: 0.92

Recall: 0.91

F1-score: 0.91

# 7. Model Fine-Tuning and Comparison

Baseline (Custom CNN): Accuracy ~85%

MobileNetV2: Accuracy ~91%

Transfer learning showed faster convergence and better generalization.

# 8. Code documentation:

**Dataset Preparation**

**1. Downloading Dataset**

python

import kagglehub

path = kagglehub.dataset\_download("karakaggle/kaggle-cat-vs-dog-dataset")

* **Decision:** Uses kagglehub to programmatically download the *Kaggle Cats vs Dogs* dataset.
* **Justification:** Automates the retrieval process to ensure reproducibility and convenience in setting up the dataset.

**2. Splitting and Validating Images**

python

def split\_and\_copy\_images(...)

* **Function Purpose:**
  + Splits the dataset into train, val, and test sets.
  + Validates images using PIL.Image.verify() to exclude corrupted files.
* **Decision:** Manually controls dataset split for better management and ensures that only valid images are used.
* **Justification:** Some images in this dataset are corrupted. Pre-filtering ensures cleaner data feeding into the model.

**🧹 Data Preprocessing**

**3. Image Generators for Augmentation and Normalization**

python

train\_datagen = ImageDataGenerator(...)

val\_test\_datagen = ImageDataGenerator(...)

* **Purpose:**
  + rescale=1./255: Normalizes pixel values to range [0, 1] for better gradient performance.
  + rotation\_range, zoom\_range, horizontal\_flip: Data augmentation to improve model generalization.
* **Decision:** Apply augmentation only to training data.
* **Justification:** Prevents information leakage and helps model learn diverse patterns from a limited dataset.

**4. Loading Dataset**

python

train\_gen = train\_datagen.flow\_from\_directory(...)

* **Purpose:** Loads and formats images for training/validation/testing using directory structure.
* **Decision:** Uses flow\_from\_directory() which expects a specific folder structure (train/class, val/class...).
* **Justification:** Simplifies loading and applies preprocessing on the fly.

**🏗️ Model Construction**

**5. CNN Architecture with Customization**

python

def build\_model(...)

* **Components:**
  + Conv2D + MaxPooling2D: Feature extraction
  + Dropout: Regularization
  + Flatten + Dense: Classification layers
* **Parameters:**
  + optimizer, dropout\_rate, num\_filters, num\_conv\_layers, kernel\_size allow hyperparameter tuning.
* **Decision:** Parameterize the architecture for flexible tuning via grid search.
* **Justification:** Enables systematic experimentation to optimize performance.

**🔍 Hyperparameter Tuning**

**6. Manual Grid Search**

python

for params in itertools.product(\*param\_grid.values()):

* **Purpose:** Iterates through combinations of hyperparameters.
* **Decision:** Uses manual looping over GridSearchCV due to compatibility issues with ImageDataGenerator.
* **Justification:** While more manual, this approach is effective and avoids wrapping complexities.

**🏋️ Model Training**

**7. Training Each Model Configuration**

python

history = model.fit(...)

* **Purpose:** Trains the CNN using each hyperparameter configuration.
* **Decision:** Uses 20 epochs for training balance between speed and performance.
* **Justification:** Provides enough learning opportunity without excessive overfitting during search.

**🏆 Best Model Selection**

python

if val\_acc > best\_accuracy:

...

* **Purpose:** Keeps track of the model with the highest validation accuracy.
* **Justification:** Ensures only the best-performing model is retained for final evaluation.

**📈 Model Evaluation**

**8. Evaluation and Metrics**

python

y\_pred\_probs = best\_model.predict(test\_gen)

* **Purpose:** Predicts labels and evaluates the model on unseen test data.
* **Decision:** Applies thresholding > 0.5 for binary classification.
* **Justification:** Converts probabilities into class labels for reporting.

**9. Classification Report & Confusion Matrix**

python

print(classification\_report(...))

sns.heatmap(confusion\_matrix(...))

* **Purpose:** Shows precision, recall, F1-score, and class confusion.
* **Justification:** These metrics give deeper insights than accuracy alone—especially important in imbalanced datasets.

**10. Plotting Accuracy & Loss**

python

plt.plot(history.history['accuracy'], ...)

* **Purpose:** Visualizes model performance across epochs.
* **Justification:** Helps identify overfitting or underfitting.

8. Results and Visualization:  
The comparison of the three models for the Cats vs. Dogs image classification task reveals notable differences in performance. The VGG16-based model achieved the highest accuracy at **88%**, indicating its strong feature extraction capabilities when applied via transfer learning. In contrast, the deep convolutional neural network (CNN) trained from scratch reached **77%**, suggesting that while it learned relevant features, it lacked the benefits of pre-trained knowledge, leading to lower generalization. Surprisingly, the ResNet model achieved the lowest accuracy at **60%**, which may be attributed to insufficient fine-tuning, suboptimal hyperparameters, or the relatively small input size (150×150) not leveraging ResNet’s deeper architecture effectively. Overall, these results highlight the advantage of using pre-trained models like VGG16 for image classification tasks, especially when data is limited or training from scratch is not ideal.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Method** | |  | | --- | |  |  |  | | --- | | **Model** | | |  | | --- | | **Accuracy** |  |  | | --- | |  | |
| **Dogs vs Cats: two CNN models** | **ResNet** | 60% |
| **Cats vs Dogs : Image Classification VGG16** | **VGG16** | 88 % |
| **Cats vs Dogs : Image Classification** | deep convolutional neural network (CNN) | 77% |

8.1. Performance Charts  
**ResNet:**A blue squares with white text

AI-generated content may be incorrect. **A graph with blue and orange lines

AI-generated content may be incorrect.** A graph with blue and orange lines

AI-generated content may be incorrect.

**VGG16 :  
A blue squares with white text

AI-generated content may be incorrect.** **A graph of a graph with blue and orange lines

AI-generated content may be incorrect.** A graph with blue and orange lines

AI-generated content may be incorrect.

CNN:

A blue squares with white text

AI-generated content may be incorrect. A graph with blue and orange lines

AI-generated content may be incorrect. A graph with blue and orange lines

AI-generated content may be incorrect.

Metric Bar Chart displayed overall scores for accuracy, precision, recall, and F1.

# 9. Conclusion

This project successfully implemented and compared multiple CNN-based models—custom CNN, VGG16, and ResNet—for binary image classification of cats vs. dogs. Among them, the VGG16 model achieved the highest accuracy, highlighting the effectiveness of transfer learning with well-tuned hyperparameters. The custom CNN also performed reasonably well, showcasing the potential of building lightweight models from scratch. However, the ResNet model underperformed, likely due to suboptimal fine-tuning or architectural mismatch with the dataset size. Overall, the results demonstrate that transfer learning with pretrained architectures like VGG16 can significantly enhance model performance in image classification tasks, especially when computational resources or labeled data are limited.