Project Report: Cat vs Dog Image Classification Using CNN

---

1. Introduction

The goal of this project is to build a **Convolutional Neural Network (CNN)** model that can classify images of cats and dogs with high accuracy.
CNNs are particularly effective for image classification tasks because they can automatically extract and learn **features from images** such as edges, shapes, and textures.

Objective:

- Train a CNN on a dataset of cat and dog images
- Evaluate its performance on unseen test images
- Deploy a model that can predict the class of a new image

---

2. Dataset

- Dataset used: **Kaggle Cats and Dogs Dataset** (~25,000 images)
- Structure:
- PetImages/
- Cat/
- Dog/
- Images are in .jpg format with varying sizes
- The dataset contains a few corrupted images that need to be removed

Labels:

- Cat → 0
- Dog → 1

Train/Test Split:

- 80% training data
- 20% test data

---

3. Data Preprocessing

1. **Cleaning the dataset:**
    - Removed corrupted images and non-image files using PIL.Image.verify()
    - Ensured all paths exist and images are .jpg or .jpeg
2. **Data Augmentation (for training only):**
    - Rescaling: 1./255
    - Random rotations: rotation_range=30
    - Random shear: shear_range=0.2
    - Random zoom: zoom_range=0.2
    - Horizontal flipping: horizontal_flip=True
3. **Test/Validation data:**
    - Only rescaling (1./255)
    - No augmentation

---

4. Model Architecture

CNN Model using Keras Sequential API:

Layer	Output Shape	Parameters
Conv2D (32, 3x3)	(128,128,32)	896
MaxPooling2D (2x2)	(64,64,32)	0
Conv2D (64, 3x3)	(64,64,64)	18,496
MaxPooling2D (2x2)	(32,32,64)	0
Conv2D (128, 3x3)	(32,32,128)	73,856
MaxPooling2D (2x2)	(16,16,128)	0

Layer	Output Shape	Parameters
GlobalAveragePooling2D	(128,)	0
Dense (256, ReLU)	(256,)	32,896
Dropout (0.5)	(256,)	0
Dense (1, Sigmoid)	(1,)	257

Explanation:

- Convolutional layers extract features from images
- MaxPooling reduces spatial dimensions
- GlobalAveragePooling reduces parameters while preserving features
- Dense layer with 256 neurons learns high-level combinations
- Dropout prevents overfitting
- Output layer predicts probability of dog (sigmoid)

---

5. Model Compilation

```
model.compile(
  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
  loss='binary_crossentropy',
  metrics=['accuracy']
)
```

- **Optimizer:** Adam
- **Loss Function:** Binary Crossentropy
- **Metrics:** Accuracy

---

6. Callbacks

1. **EarlyStopping**

- Monitors validation loss
- Patience: 5 epochs
- Restores best weights

2. **ReduceLROnPlateau**
    - Monitors validation loss
    - Reduces learning rate by 0.5 if loss plateaus for 2 epochs

Purpose: Improve convergence and prevent overfitting

---

7. Training

- Epochs: 50 (EarlyStopping often stopped earlier, e.g., epoch 38)
- Batch size: 32
- Training & validation loss and accuracy were tracked

Example Training Results:

- Training Accuracy: ~0.89
- Validation Accuracy: ~0.88
- Loss decreased steadily

Behavior:

- Accuracy plateaued after a few epochs
- Learning rate reduced on plateau to improve convergence
- Early stopping prevented overfitting

Graphs:

1. **Accuracy vs Epochs** – shows training and validation accuracy over epochs
2. **Loss vs Epochs** – shows training and validation loss over epochs

---

8. Model Evaluation

test_loss, test_acc = model.evaluate(test_iterator)

```
print(f"Test Accuracy: {test_acc:.4f}, Test Loss: {test_loss:.4f}")
```

- Test Accuracy: 0.8878 (~88.8%)
- Test Loss: 0.2657
- Indicates **good generalization** on unseen data

---

9. Prediction on New Images

Code snippet for single-image prediction:

```
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np

img_path = "/content/my_dog.jpg"
img = image.load_img(img_path, target_size=(128,128))
img_array = np.expand_dims(image.img_to_array(img)/255.0, axis=0)
pred = model.predict(img_array)[0][0]

if pred > 0.5:
  print(f"DOG ({pred:.4f})")
else:
  print(f"CAT ({1-pred:.4f})")
plt.imshow(img)
plt.axis('off')
plt.show()
```

- Works for **any image uploaded to Colab**
- Outputs **class and confidence**

---

10. Model Export

- Saved the trained model as .h5:

```
model.save('cat_dog_model.h5')
```

- Can be **downloaded** and reused later:

```
from google.colab import files

files.download('cat_dog_model.h5')
```

- Loading in a new Colab session:

```
from google.colab import files

import tensorflow as tf


uploaded = files.upload()  # select cat_dog_model.h5

model = tf.keras.models.load_model('cat_dog_model.h5')
```

---

11. Conclusion

- Built a **CNN classifier** for cats vs dogs with ~88–89% accuracy
- Used **data cleaning, augmentation, and proper callbacks** to improve performance
- Model generalizes well to unseen test images
- Saved model as .h5 for easy **reuse and deployment**
- Can predict any new image with a simple **inference script**

Future Improvements:

- Use **more advanced architectures** like ResNet or MobileNet
- Increase dataset size or balance classes better
- Tune hyperparameters for better accuracy (>90%)