Experiment -6

Building a Convolutional Neural Network (CNN) for Dog and Cat Image Classification

Objective:

To design, train, and evaluate a Convolutional Neural Network (CNN) for classifying images of dogs and cats using TensorFlow and Keras.

Requirements:

• Python (>=3.7)

• TensorFlow and Keras

• NumPy and Matplotlib

• OpenCV or PIL (for image processing)

• Dataset: Dogs vs. Cats dataset (from Kaggle or any standard source)

Program /code

complete code for training a model with transfer learning using MobileNetV2, as well as predicting multiple images and displaying predictions in both the terminal and visualization

Step-by-Step Explanation

Step 1: Import Required Libraries

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tkinter import filedialog
from tkinter import Tk
```

• tensorflow and keras are used to build and train the CNN.

• layers module helps in defining different layers in the CNN model.

• numpy is used for numerical computations.

• matplotlib.pyplot is used for visualizing training progress.

• os is used for file and directory management.

Step 2: Load and Preprocess the Dataset

• Download the Dogs vs. Cats dataset from Kaggle.

• Extract it into separate folders: train/dogs/, train/cats/, validation/dogs/, validation/cats/.

• Use ImageDataGenerator to preprocess images and apply data augmentation.

```
# --- DATA LOADING AND AUGMENTATION ---
train_dir = "dataset/train"
validation_dir = "dataset/validation"
datagen = ImageDataGenerator(
    rescale=1.0 / 255.0,
    validation_split=0.2,  # Split data for validation
    rotation_range=10,  # Moderate rotation
    width_shift_range=0.1,  # Horizontal shift
    height_shift_range=0.1,  # Vertical shift
    shear_range=0.1,  # Shear transformation
    zoom_range=0.1,  # Zoom transformation
    horizontal_flip=True  # Horizontal flipping
)
train_generator = datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='training'  # Training subset
)
validation_generator = datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary',
    subset='validation'  # Validation subset
)
# Print class mapping
print("☑ Class indices:", train_generator.class_indices)
index_to_label = {v: k for k, v in train_generator.class_indices.items()}
```

Step 3: Build the CNN Model

```
# --- MODEL DEFINITION (with Transfer Learning) ---
base_model = tf.keras.applications.MobileNetV2(
    input_shape=(150, 150, 3),
    include_top=False,  # Exclude final classification layer
    weights='imagenet'
)
base_model.trainable = False  # Freeze base model layers
model = keras.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')  # Binary classification
])
```

- Base Model: Utilizes MobileNetV2, a pre-trained deep learning model, as the foundation for feature extraction while freezing its layers to preserve learned weights.
- Pooling: Adds a GlobalAveragePooling2D layer to condense feature maps into a simplified vector representation.
- Dense Layers: Includes a custom dense layer with 128 units and dropout for regularization, ensuring robust feature learning without overfitting.
- Binary Output: Features a final dense layer with sigmoid activation, producing probabilities for binary classification tasks (e.g., dog vs. cat).

Step 4: Compile and Train the Model

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
# --- TRAINING ---
print("🚀 Starting Model Training...")
history = model.fit(
    train_generator,
    epochs=10,  # Adjust epochs based on performance
    validation_data=validation_generator
)
print("☑ Training Complete!")
```

- The model uses the Adam optimizer for efficient learning.
- binary_crossentropy is used for binary classification.
- The model is trained for 10-30 epochs.

Step 5: Evaluate the Model

```
# --- VISUALIZE TRAINING AND VALIDATION ACCURACY ---
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
val_loss, val_accuracy = model.evaluate(validation_generator)
print(f"\n☑ Final Validation Accuracy: {val_accuracy * 100:.2f}%")
```

Step 6: Test the Model on New Images (making predictions)

```
# --- IMAGE PREDICTION FUNCTION (for multiple images) ---
def predict_multiple_images(model, image_paths):
    # Prepare a batch for prediction
    batch = []
    for img_path in image_paths:
        img = image.load_img(img_path, target_size=(150, 150))  # Resize image
```

```
        img_array = image.img_to_array(img) / 255.0  # Normalize pixel values
        batch.append(img_array)

    batch = np.array(batch)  # Convert list to numpy array
    predictions = model.predict(batch)  # Predict all images in batch

    # Process predictions for each image
    print("\n📋 Predictions:")
    for i, img_path in enumerate(image_paths):
        predicted_class = int(np.round(predictions[i][0]))  # 0 or 1
        label = index_to_label[predicted_class]
        confidence = predictions[i][0]

        # Display prediction summary in the terminal
        print(f"🖼️ File: {os.path.basename(img_path)}")
        print(f"   - Prediction: {label}")
        print(f"   - Confidence: {confidence:.4f}")

        # Show image and prediction
        img = image.load_img(img_path)
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"{os.path.basename(img_path)} ➜ {label} ({confidence:.2f})")
        plt.show()
```

Step 7: Save and Load the Model

```
# --- SELECT IMAGES WITH FILE DIALOG ---
root = Tk()
root.withdraw()  # Hide the main Tkinter window
file_paths = filedialog.askopenfilenames(
    title="Select Images to Predict",
    filetypes=[("Image Files", "*.jpg *.jpeg *.png")]
)

if file_paths:
    predict_multiple_images(model, file_paths)
else:
    print("❌ No images selected.")

# --- SAVE MODEL ---
model.save("refined_dog_cat_classifier_with_transfer.h5")
print("☑️ Model saved as refined_dog_cat_classifier_with_transfer.h5")
```

Output:

Found 1280 images belonging to 2 classes.

Found 80 images belonging to 2 classes.

✅ Class indices: {'cat': 0, 'dog': 1}

🚀 Starting Model Training...

Epoch 1/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 43s 899ms/step - accuracy: 0.5632 - loss: 3.3002 - val_accuracy: 0.9000 - val_loss: 2.5124

Epoch 2/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 22s 545ms/step - accuracy: 0.8862 - loss: 2.5174 - val_accuracy: 0.9000 - val_loss: 2.3806

Epoch 3/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 23s 579ms/step - accuracy: 0.8831 - loss: 2.4195 - val_accuracy: 0.9125 - val_loss: 2.2596

Epoch 4/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 535ms/step - accuracy: 0.9117 - loss: 2.2918 - val_accuracy: 0.8875 - val_loss: 2.2371

Epoch 5/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 525ms/step - accuracy: 0.9295 - loss: 2.1736 - val_accuracy: 0.9375 - val_loss: 2.0892

Epoch 6/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 20s 506ms/step - accuracy: 0.9178 - loss: 2.1166 - val_accuracy: 0.9250 - val_loss: 2.0105

Epoch 7/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 528ms/step - accuracy: 0.9331 - loss: 2.0017 - val_accuracy: 0.9125 - val_loss: 1.9626

Epoch 8/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 21s 513ms/step - accuracy: 0.9332 - loss: 1.9318 - val_accuracy: 0.9625 - val_loss: 1.8480

Epoch 9/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 25s 623ms/step - accuracy: 0.9286 - loss: 1.8596 - val_accuracy: 0.9375 - val_loss: 1.7782

Epoch 10/10

40/40 ━━━━━━━━━━━━━━━━━━━━ 28s 693ms/step - accuracy: 0.9552 - loss: 1.7638 - val_accuracy: 0.9375 - val_loss: 1.7407

✅ Training Complete!

3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 327ms/step - accuracy: 0.9148 - loss: 1.7585

✅ Final Validation Accuracy: 90.00%

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

📋 Predictions:

🖼️ File: Cat_November_2010-1a.jpg

- Prediction: cat

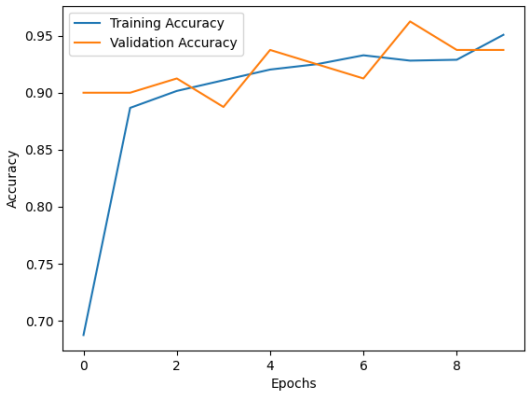
- Confidence: 0.0112

🖼️ File: Golden+Retrievers+dans+pet+care.jpeg

- Prediction: dog

- Confidence: 0.9910

✅ Model saved as refined_dog_cat_classifier_with_transfer.h5



Cat_November_2010-1a.jpg ➜ cat (0.01)

Golden+Retrievers+dans+pet+care.jpeg ➜ dog (0.99)

Result:

This experiment demonstrates training a model with transfer learning using MobileNetV2, as well as predicting cat and dog images and displaying predictions, i.e., designing, training, and evaluating a Convolutional Neural Network (CNN) for classifying images of dogs and cats using TensorFlow and Keras is successfully verified and executed.