Machine Learning Image Recognizer

Jose Parra

Introduction

Its important to distinguish between species, we all know what a cat and dog looks like, but a machine doesn't. The one advantage of a of using a computer is its processing speed. In this project I will program in python an application that is able to tell the difference between a cat and dog using convolutional and pool layers. The model will be train by feeding it 25,000 picture of cats and dogs.

Method

The individual pictures were imported into the application and condensed into 128by128 pixels, Then those pixels where later translated in to RGB arrays. The RGB arrays were later normalized. The arrays were shuffled to insure randomness and avoid biases for every run. Then the data was split into sets for training.

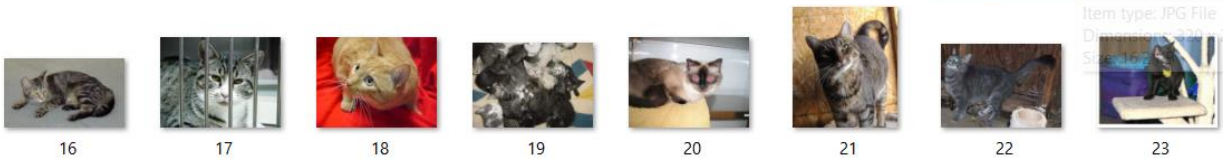
This is how the training CNN model was set up.

```
CNN model
n_model = keras.models.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(128, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
```

The Model was trained using 15 epochs with the batch sizes of 32


0
1
2
3
4
5
6
7


8
9
10
11
12
13
14
15


16
17
18
19
20
21
22
23


24
25
26
27
28
29
30
31


32
33
34
35
36
37
38
39

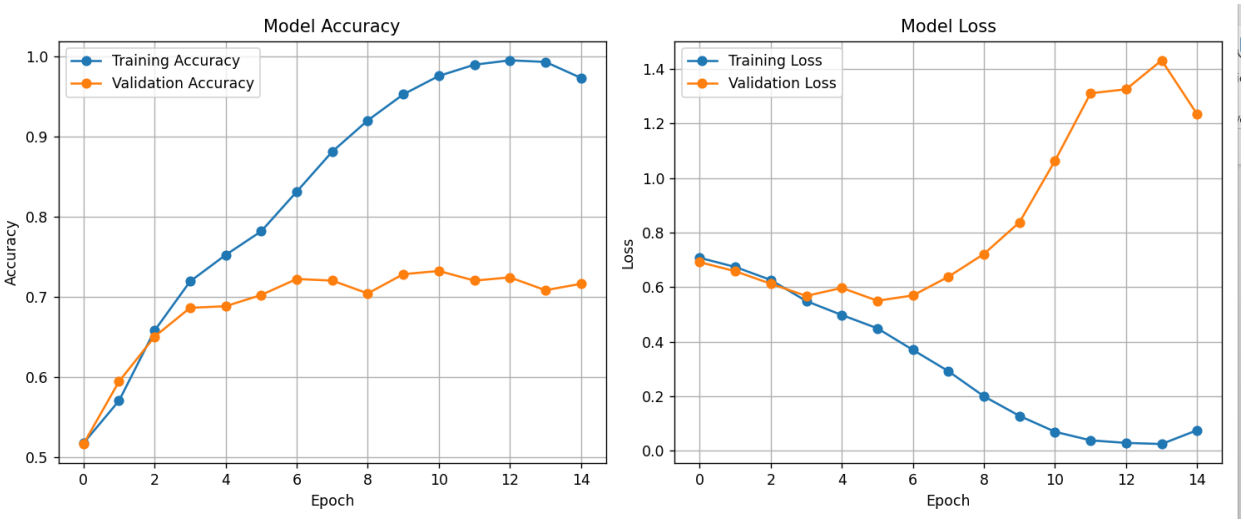

0
1
2
3
4
5
6
7


8
9
10
11
12
13
14
15


16
17
18
19
20
21
22
23


24
25
26
27
28
29
30
31

Results



Model Accuracy vs Model Loss

Appendix: Code

```
import os
import numpy as np
from PIL import Image
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt  # Import matplotlib for plotting

# Paths to the dataset directories
dog_dir = r'C:\ASU\grad sem 1\Machine learning\kagglecatsanddogs_5340\PetImages\Dog'
cat_dir = r'C:\ASU\grad sem 1\Machine learning\kagglecatsanddogs_5340\PetImages\Cat'

# Lists to hold image data
dog_images = []
cat_images = []
image_size = (128, 128)  # Resizing images to 128x128 pixels

def load_and_preprocess_image(image_path):
    """
    Load and preprocess an image: open, convert to RGB, resize, and convert to a
NumPy array.
    """
    try:
        img = Image.open(image_path).convert('RGB')
        img_resized = img.resize(image_size)
        return np.array(img_resized)
    except Exception as e:
        # Handle exceptions (e.g., corrupted images)
        print(f"Error processing image {image_path}: {e}")
        return None

# Valid file extensions to consider
valid_extensions = ('.jpg', '.jpeg', '.png', '.bmp', '.tiff')

# Maximum number of images to process per class (10% of 12,500 is 1,250)
max_images_per_class = 1250

# Process dog images
dog_image_files = [f for f in os.listdir(dog_dir) if
f.lower().endswith(valid_extensions)]
for filename in dog_image_files[:max_images_per_class]:
    image_path = os.path.join(dog_dir, filename)
    img_array = load_and_preprocess_image(image_path)
    if img_array is not None:
        dog_images.append(img_array)

# Process cat images
cat_image_files = [f for f in os.listdir(cat_dir) if
f.lower().endswith(valid_extensions)]
for filename in cat_image_files[:max_images_per_class]:
    image_path = os.path.join(cat_dir, filename)
    img_array = load_and_preprocess_image(image_path)
    if img_array is not None:
        cat_images.append(img_array)

# Convert lists to NumPy arrays
dog_dataset = np.array(dog_images)
```

```
cat_dataset = np.array(cat_images)

# labels
dog_labels = np.ones(len(dog_dataset))
cat_labels = np.zeros(len(cat_dataset))

# Combine data and labels
images = np.concatenate((dog_dataset, cat_dataset), axis=0)
labels = np.concatenate((dog_labels, cat_labels), axis=0)

(We normalized the data to avoid complications)

# Normalize pixel values to [0, 1]
images = images / 255.0
(I shuffled the images and labels to avoid any bias when rerunning the test)
# Shuffle the data
shuffle_indices = np.arange(len(images))
np.random.shuffle(shuffle_indices)
images = images[shuffle_indices]
labels = labels[shuffle_indices]

# Split data into training and testing sets (80% train, 20% test)
train_test_split_index = int(0.8 * len(images))
train_images, test_images = images[:train_test_split_index],
images[train_test_split_index:]
train_labels, test_labels = labels[:train_test_split_index],
labels[train_test_split_index:]

(Made A sequential model where layers are stacked sequentially)
# Building the model
cnn_model = keras.models.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(128, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')  # Binary classification
])


cnn_model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

# Train the model and store the training history
history = cnn_model.fit(
    train_images, train_labels,
    epochs=15,
    batch_size=32,
    validation_data=(test_images, test_labels)
)

# Evaluate the model on the test set
test_loss, test_acc = cnn_model.evaluate(test_images, test_labels)
print(f"Test accuracy: {test_acc}")
```

```
# Plot training & validation accuracy and loss values
plt.figure(figsize=(12, 5))

# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', marker='o')
plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```