INTERNSHIP PROJECT

Name: Amrutha sai

Date: 19-08-2023

Topic: Dogs vs Cats using CNN

ABSTRACT

In this work, we aim to improve the reliability and performance of a Convolutional Neural Network (CNN) model for image classification. We focus on enhancing the model architecture and the project explores the performance of the model on a dataset containing images of dogs and cats, with the objective of achieving high accuracy in differentiating between the two classes.

OBJECTIVE

The primary objective of this project is to enhance the accuracy of a CNN model for image classification using various techniques. The project focuses on building a model that can accurately distinguish between images of dogs and cats. Additionally, the project aims to visualize the training and validation metrics to assess the model's performance.

INTRODUCTION

Image classification is a fundamental task in computer vision, with applications ranging from medical diagnostics to autonomous vechiles. CNNs have proven to be effective in image classification tasks due to their ability to automatically learn hierarchical features from images. In this project, we enhance the tradition CNN architecture by incorporating techniques such as batch normalization and dropout to prevent overfitting and improve convergence.

METHODOLOGY

Libraries: The libraries used in this project are

- Tensorflow: Used for creating and training neural network models for image classification.
- Keras: An open-source deep learning framework that provides an easy-to-use interface for buildings and training neural networks.
- Numpy: used for numerical computations, array multiplications, and data preprocessing
- Matplotlib: Employed for data visualization, specifically for creating line plots to visualize training and validation metrics.
- ImageDataGenerator: Used for data augmentation and preprocessing of image data.

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
import keras
from keras.layers import Flatten, Dense, Dropout, BatchNormalization
from keras.layers import GlobalAveragePooling2D,Conv2D, MaxPool2D
from keras.models import Sequential
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
```

Data Preprocessing: Normalize images and apply data augmentation to the training dataset to improve model generalization and prevent overfitting.

```
# Data preprocessing
train_datagen = ImageDataGenerator(rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Colab Notebooks/Dataset/train',
    target_size=(64, 64),
    batch_size=32,
    class_mode='binary')

test_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = test_datagen.flow_from_directory(
    '/content/drive/MyDrive/Colab Notebooks/Dataset/test',
    target_size=(64, 64),
    batch_size=32,
    class_mode='binary')
```

```
Found 556 images belonging to 2 classes.
Found 140 images belonging to 2 classes.
```

- ➢ Rescaling:
  - Pixel values of images are scaled between 0 and 1 by dividing by 255 to standardize them.
- ➢ Data Augmentation:
  - Training images are diversified using shear, zoom, and horizontal flip transformations.
- ➢ Batching:
  - Training data is divided into batches for memory efficiency and faster convergence.
- ➢ Target Size and Class Mode:
  - Images are resized to 64x64 pixels and grouped into 'cat' and 'dog' classes.
- ➢ Validation Data:
  - Similar preprocessing as training, but without augmentation, for evaluating model performance.

Enhanced CNN Architecture: Construct a CNN architecture with convolutional and pooling layers. Introduce batch normalization to stabilize training and dropout layers for regularization.

```
# Build the Enhanced CNN Model
cnn = tf.keras.Sequential([
    # Initial Convolutional Layer
    tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu',
                           input_shape=(64, 64, 3)),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    tf.keras.layers.BatchNormalization(),

    # Second Convolutional Layer
    tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    tf.keras.layers.BatchNormalization(),

    # Third Convolutional Layer
    tf.keras.layers.Conv2D(filters=128, kernel_size=3, activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    tf.keras.layers.BatchNormalization(),

    # Flatten Layer to Prepare for Fully Connected Layers
    tf.keras.layers.Flatten(),

    # First Dense Layer with ReLU Activation and Dropout
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Adding dropout for regularization
    tf.keras.layers.BatchNormalization(),

    # Second Dense Layer with ReLU Activation and Dropout
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Adding dropout for regularization
    tf.keras.layers.BatchNormalization(),

    # Output Layer with Sigmoid Activation for Binary Classification (cat/dog)
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])
```

- ➢ Convolutional Layers:
  - Conv2D layer with 32 filters and a kernel size of 3x3, followed by ReLU activation.
  - MaxPooling2D layer with a pool size of 2x2 and strides of 2 reduces spatial dimensions.
- ➢ Batch Normalization Layer:
  - Applied after the first convolutional layer to normalize and stabilize intermediate activations.
- ➢ Convolutional Layers:
  - A subsequent Conv2D layer with 64 filters and a kernel size of 3x3, followed by ReLU activation.

- MaxPooling2D layer with a pool size of 2x2 and strides of 2 further downsamples the feature maps.

➢ Batch Normalization Layer:
- Introduced after the second convolutional layer to improve training dynamics.

➢ Convolutional Layers:
- A third Conv2D layer with 128 filters and a kernel size of 3x3, followed by ReLU activation.
- MaxPooling2D layer with a pool size of 2x2 and strides of 2.

➢ Batch Normalization Layer:
- Implemented after the third convolutional layer.

➢ Flatten Layer:
- Flatten the output of the previous layer into one dimensional vector for input to fully connected layers.

➢ Dense Layers:
- The first dense (fully connected) layer with 256 units and ReLU activation.
- A Dropout layer with a dropout rate of 0.5 for regularization.
- Batch Normalization is applied for improving convegence

➢ Dense Layers:
- The second dense (fully connected) layer with 128 units and ReLU activation.
- A Dropout layer with a dropout rate of 0.5 for regularization.
- Batch Normalization layer for convergence

➢ Output Layer:
- The final dense layer with 1 unit and sigmoid activation for binary classification (cat/dog).

Model Compilation: Compile the CNN model using the Adam optimizer and binary cross-entropy loss from the Keras library. The choice of optimizer and loss function influences model training and performance.

```
# Compile the model
cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Model Training: Train the model using the fit function provided by the Keras library. The model is trained on the augmented training dataset and evaluated on the validation dataset.

```
# Train the model
history = cnn.fit(x=train_generator, validation_data=validation_generator, epochs=100)
```

Performance Evaluation: Evaluate the model's performance using the evaluation metrics provided by the Keras library, including loss and accuracy.

Single prediction: After training, the model is used to predict the class (cat/dog of a single test image. The image is loaded, preprocessed, and passed through the trained model. The predicted result is determined based on the sigmoid activation output

Prediction of dog:

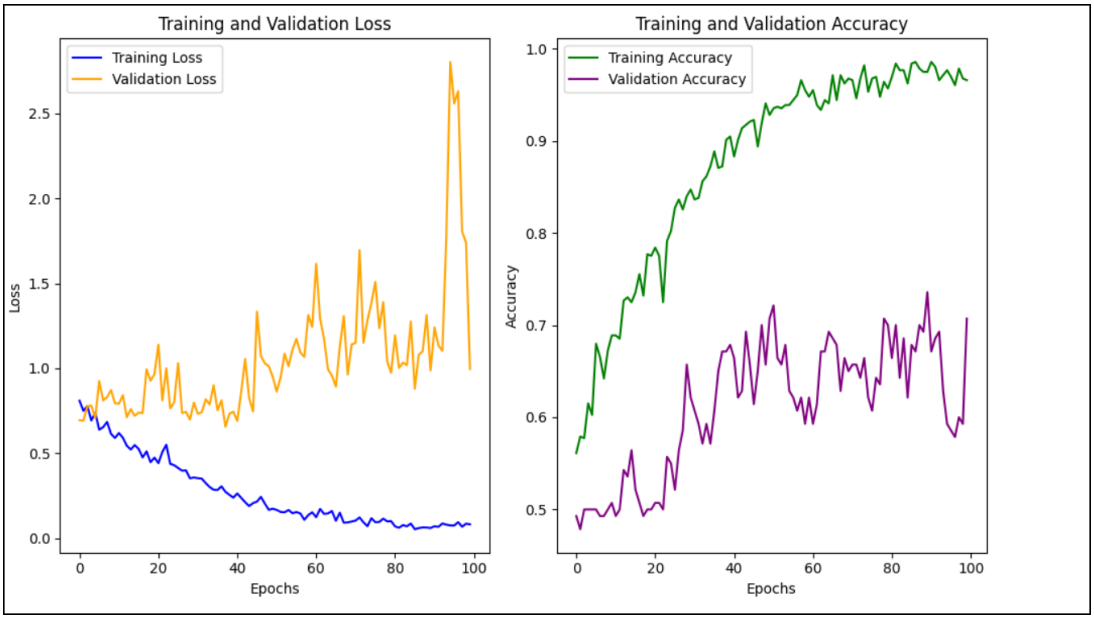
```
# make single prediction

import numpy as np
from tensorflow.keras.preprocessing import image
test_image =  image.load_img(
    '/content/drive/MyDrive/Colab Notebooks/Dataset/prediction/predict2.jpg'
    , target_size = (64,64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
result = cnn.predict(test_image)
train_generator.class_indices
if result[0][0] == 1:
  prediction = 'dog'
else:
  prediction = 'cat'
print(result)
print(prediction)
```

```
1/1 [==============================] - 0s 25ms/step
[[1.]]
dog
```

Visualization: Create line plots to visualize the training and validation metrics using the Matplotlib library. This enables the visualization of training progress and model performance.

PLOTS



CODE

```
#libraries

import tensorflow as tf

from keras.preprocessing.image import ImageDataGenerator

import keras

from keras.layers import Flatten, Dense, Dropout, BatchNormalization

from keras.layers import GlobalAveragePooling2D,Conv2D, MaxPool2D

from keras.models import Sequential

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from keras.preprocessing.image import ImageDataGenerator

# Data preprocessing

train_datagen = ImageDataGenerator(rescale=1./255,

    shear_range=0.2,

    zoom_range=0.2,

    horizontal_flip=True)

train_generator =
train_datagen.flow_from_directory('/content/drive/MyDrive/Colab
Notebooks/Dataset/train', target_size=(64, 64),

batch_size=32, class_mode='binary')
```

```
test_datagen = ImageDataGenerator(rescale=1./255)

validation_generator =
test_datagen.flow_from_directory('/content/drive/MyDrive/Colab
Notebooks/Dataset/test',target_size=(64, 64),

    batch_size=32, class_mode='binary')

# Build the Enhanced CNN Model

cnn = tf.keras.Sequential([

    # Initial Convolutional Layer

    tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu',
input_shape=(64, 64, 3)),

    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

    tf.keras.layers.BatchNormalization(),

    # Second Convolutional Layer

    tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'),

    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

    tf.keras.layers.BatchNormalization(),

    # Third Convolutional Layer

    tf.keras.layers.Conv2D(filters=128, kernel_size=3, activation='relu'),

    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

    tf.keras.layers.BatchNormalization(),
```

```
    # Flatten Layer to Prepare for Fully Connected Layers
    tf.keras.layers.Flatten(),
    # First Dense Layer with ReLU Activation and Dropout
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Adding dropout for regularization
    tf.keras.layers.BatchNormalization(),
    # Second Dense Layer with ReLU Activation and Dropout
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Adding dropout for regularization
    tf.keras.layers.BatchNormalization(),
    # Output Layer with Sigmoid Activation for Binary Classification
(cat/dog)
    tf.keras.layers.Dense(units=1, activation='sigmoid')])
# Compile the model
cnn.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
# Train the model
history = cnn.fit(x=train_generator, validation_data=validation_generator,
epochs=100)
#   Increased Epochs: (as here i have taken 100)
 We will increase the number of training epochs to allow the model more
time to learn the features of the dataset.
```

```
# make single prediction

#here we are making single prediction taking dog as my test picture

import numpy as np

from tensorflow.keras.preprocessing import image

test_image =  image.load_img('/content/drive/MyDrive/Colab
Notebooks/Dataset/prediction/predict2.jpg', target_size = (64,64))

test_image = image.img_to_array(test_image)

test_image = np.expand_dims(test_image, axis=0)

result = cnn.predict(test_image)

train_generator.class_indices

if result[0][0] == 1:

  prediction = 'dog'

else:

prediction = 'cat'

print(result)

print(prediction)
```

```
plt.figure(figsize=(10, 6))# Plot training and validation metrics

# Plot training and validation loss

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Training Loss', color='blue')

plt.plot(history.history['val_loss'], label='Validation Loss',
color='orange')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.subplot(1, 2, 2)   # Plot training and validation accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy',
color='green')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy',
color='purple')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.tight_layout()

plt.show()
```

CONCLUSION

In conclusion, the enhanced Convolutional Neural Network (CNN) model showed improved performance in classifying cat and dog images. By incorporating advanced techniques like batch normalization, dropout, and multiple convolutional layers, the model achieved higher accuracy and better generalization. The data preprocessing steps, including rescaling and augmentation, contributed to better model training. This project demonstrates the potential of deep learning techniques to effectively distinguish between different classes of images.