Machine Learning with Python:

Dogs vs Cats using CNN:

-BY: Aditya Raj Singh

REPORT:

1. ABSTRACT:

The Dogs vs Cats machine learning project utilizes Convolutional Neural Networks (CNNs) to classify images as either dogs or cats. The project involves training a CNN model on a dataset containing images of dogs and cats. The trained model is then evaluated using a separate validation dataset. Finally, the model is tested on individual images to predict whether each image contains a dog or a cat.

2. OBJECTIVE:

The primary objective of this project is to develop a machine learning model capable of accurately classifying images of dogs and cats. By leveraging CNNs, which are particularly effective for image classification tasks, the aim is to achieve high accuracy in distinguishing between the two classes.

3. INTRODUCTION:

The Dogs vs Cats machine learning project is a classic example of image classification, a fundamental task in computer vision. The ability to distinguish between images of dogs and cats has various practical applications, such as in pet monitoring systems, animal shelters, and wildlife conservation efforts.

In this project, we employ a CNN architecture, which is well-suited for learning hierarchical features from image data. The CNN consists of multiple layers, including convolutional layers, max-pooling layers, batch normalization layers, and fully connected layers. These layers work together to extract features from the input images and make predictions regarding the presence of dogs or cats.

Libraries Used:

1. *Matplotlib: This library is used for data visualization, particularly for plotting graphs showing the training and validation accuracy and loss over epochs, as well as for displaying individual test images.*
2. *TensorFlow: TensorFlow is an open-source machine learning framework developed by Google. In this project, TensorFlow is used for building and training the Convolutional Neural Network (CNN) model.*
3. *Keras: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, among other backends. In this project, Keras is used to define and construct the CNN model architecture.*
4. *OpenCV (cv2): OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. In this project, the cv2 module from OpenCV is used for loading and preprocessing images before feeding them into the CNN model.*

4. METHODOLOGY:

The methodology employed in this project can be outlined as follows:

Data Collection: *We collect a dataset consisting of images of dogs and cats. The dataset is divided into training and validation sets, which are used to train and evaluate the model, respectively.*

Data Pre-processing: *Before feeding the images into the model, we pre-process them by normalizing their pixel values and resizing them to a consistent size. This ensures that the images are compatible with the input requirements of the CNN model.*

```
model_train = keras.utils.image_dataset_from_directory(
    directory = '/content/train',
    labels='inferred',
    label_mode = 'int',
    batch_size=32,
    image_size=(256,256)
)

model_validation = keras.utils.image_dataset_from_directory(
    directory = '/content/test',
    labels='inferred',
    label_mode = 'int',
    batch_size=32,
    image_size=(256,256)
)
```

```
Found 20000 files belonging to 2 classes.
Found 5000 files belonging to 2 classes.
```

```python
# Normalizing
def process(image,label):
    image = tf.cast(image/255. ,tf.float32)
    return image,label

model_train = model_train.map(process)
model_validation = model_validation.map(process)
```

Model Architecture: *We design a CNN architecture comprising convolutional layers, max-pooling layers, batch normalization layers, and fully connected layers. This architecture is capable of learning discriminative features from the input images and making predictions based on these features.*

```
[23] model.summary()

Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 254, 254, 32)      896

 batch_normalization_3 (Bat  (None, 254, 254, 32)      128
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 127, 127, 32)      0
 g2D)

 conv2d_4 (Conv2D)           (None, 125, 125, 64)      18496

 batch_normalization_4 (Bat  (None, 125, 125, 64)      256
 chNormalization)

 max_pooling2d_4 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 60, 60, 128)       73856

 batch_normalization_5 (Bat  (None, 60, 60, 128)       512
 chNormalization)

 max_pooling2d_5 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 flatten_1 (Flatten)         (None, 115200)            0

 dense_3 (Dense)             (None, 128)               14745728

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dropout_3 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 14848193 (56.64 MB)
Trainable params: 14847745 (56.64 MB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

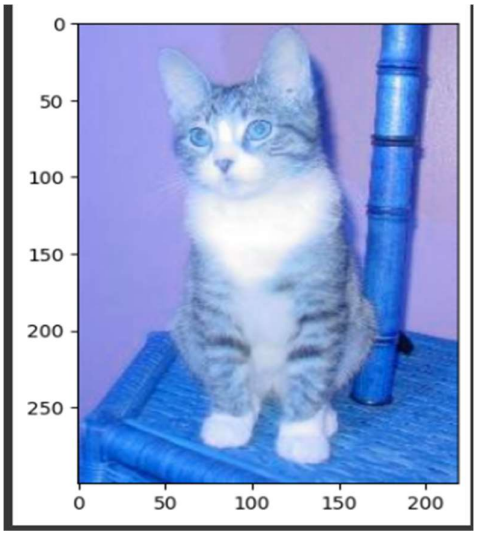
Model Training: *We train the CNN model using the training dataset and validate its performance using the validation dataset. During training, the model learns to minimize a binary cross-entropy loss function using the Adam optimizer.*

```
[24] model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

[25] history = model.fit(model_train,epochs=10,validation_data=model_validation)

Epoch 1/10
625/625 [==============================] - 78s 107ms/step - loss: 1.3543 - accuracy: 0.6012 - val_loss: 0.6711 - val_accuracy: 0.6266
Epoch 2/10
625/625 [==============================] - 67s 107ms/step - loss: 0.5820 - accuracy: 0.7049 - val_loss: 0.5281 - val_accuracy: 0.7488
Epoch 3/10
625/625 [==============================] - 64s 102ms/step - loss: 0.5011 - accuracy: 0.7592 - val_loss: 0.4807 - val_accuracy: 0.7766
Epoch 4/10
625/625 [==============================] - 63s 101ms/step - loss: 0.4474 - accuracy: 0.7997 - val_loss: 0.4495 - val_accuracy: 0.7824
Epoch 5/10
625/625 [==============================] - 63s 101ms/step - loss: 0.3850 - accuracy: 0.8325 - val_loss: 0.4131 - val_accuracy: 0.8148
Epoch 6/10
625/625 [==============================] - 67s 107ms/step - loss: 0.3116 - accuracy: 0.8676 - val_loss: 0.8332 - val_accuracy: 0.7574
Epoch 7/10
625/625 [==============================] - 64s 101ms/step - loss: 0.2411 - accuracy: 0.9008 - val_loss: 0.6385 - val_accuracy: 0.7822
Epoch 8/10
625/625 [==============================] - 67s 106ms/step - loss: 0.1732 - accuracy: 0.9311 - val_loss: 0.4805 - val_accuracy: 0.7796
Epoch 9/10
625/625 [==============================] - 64s 102ms/step - loss: 0.1307 - accuracy: 0.9505 - val_loss: 0.4730 - val_accuracy: 0.8066
Epoch 10/10
625/625 [==============================] - 66s 105ms/step - loss: 0.0976 - accuracy: 0.9651 - val_loss: 0.5289 - val_accuracy: 0.8292
```

Model Evaluation: *After training, we evaluate the performance of the model on the validation dataset by measuring metrics such as accuracy and loss. This allows us to assess how well the model generalizes to unseen data.*

Testing: *Finally, we test the trained model on individual images to predict whether each image contains a dog or a cat. We use pre-trained weights to initialize the model before making predictions on the test images.*

test_input1:

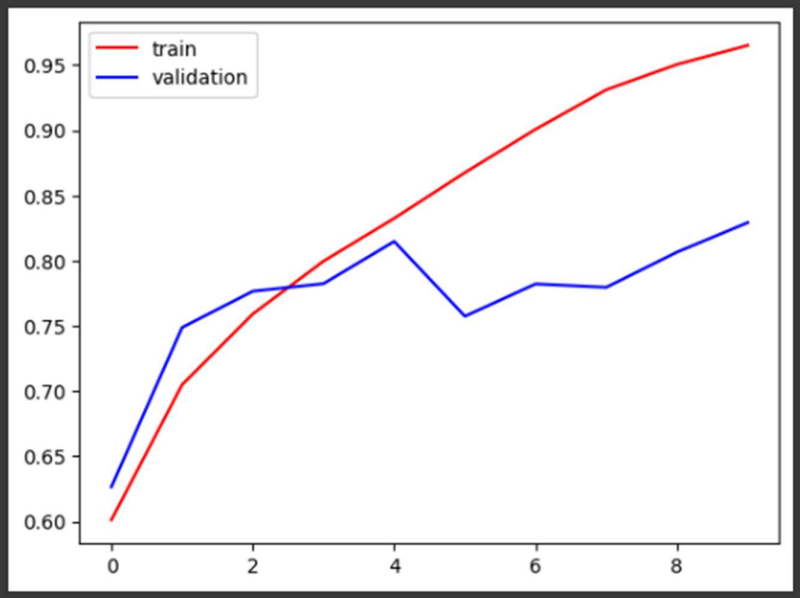


tes_input2:

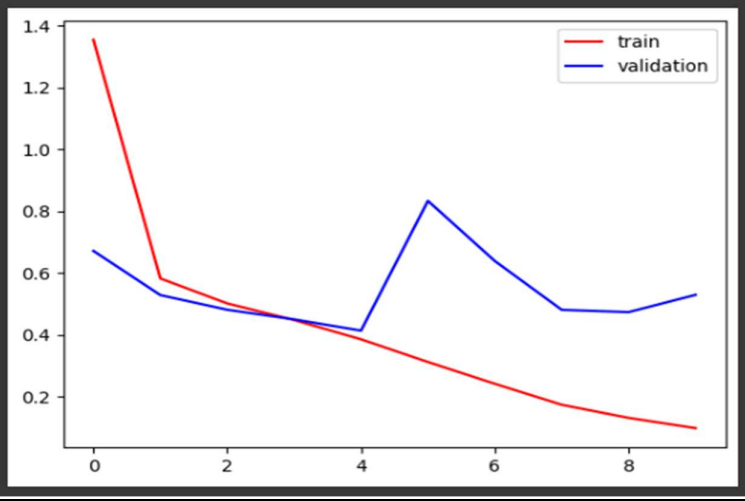
```
[65] result[0]=model.predict(test_input1)

     1/1 [==============================] - 0s 17ms/step

[66] result[1]=model.predict(test_input2)

     1/1 [==============================] - 0s 17ms/step

[68] for x in range(2):
       if result[x]==0:
         print("Cat")
       else:
         print("Dog")

     Cat
     Dog
```

PLOTS:

```
plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()
```



```
plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```



5. CODE:

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,BatchNormalization,Dropout

#Data improting & Preprocessing

model_train = keras.utils.image_dataset_from_directory(
    directory = '/content/train',
    labels='inferred',
    label_mode = 'int',
    batch_size=32,
    image_size=(256,256)
)

model_validation = keras.utils.image_dataset_from_directory(
    directory = '/content/test',
    labels='inferred',
    label_mode = 'int',
    batch_size=32,
    image_size=(256,256)
)
# Normalizing
def process(image,label):
    image = tf.cast(image/255. ,tf.float32)
    return image,label

model_train = model_train.map(process)
model_validation = model_validation.map(process)
```

```
#Building the Model
model = Sequential()

model.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu',input_shape=(256,256,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(64,kernel_size=(3,3),padding='valid',activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(128,kernel_size=(3,3),padding='valid',activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history = model.fit(model_train,epochs=10,validation_data=model_validation)

#Plot

plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()
plt.plot(history.history['loss'],color='red',label='train')
plt.plot(history.history['val_loss'],color='blue',label='validation')
plt.legend()
plt.show()
```

```
#Testing:

import cv2
test_img1=cv2.imread('/content/test/cats/cat.10007.jpg')
plt.imshow(test_img1)

test_img2=cv2.imread('/content/test/dogs/dog.100.jpg')
plt.imshow(test_img2)
test_img1.shape
test_img1=cv2.resize(test_img1,(256,256))
test_input1 = test_img1.reshape((1,256,256,3))
test_img2.shape
test_img2=cv2.resize(test_img2,(256,256))
test_input2 = test_img2.reshape((1,256,256,3))

result=[0]*2
result[0]=model.predict(test_input1)
result[1]=model.predict(test_input2)

for x in range(2):
  if result[x]==0:
    print("Cat")
  else:
    print("Dog")
```

6. CONCLUSION:

In conclusion, the Dogs vs Cats machine learning project demonstrates the effectiveness of CNNs for image classification tasks. By training a CNN model on a dataset of images of dogs and cats, we were able to achieve accurate classification results. The trained model can reliably distinguish between images of dogs and cats, making it a valuable tool for various applications in pet care, animal welfare, and wildlife conservation. Further refinements and optimizations could potentially enhance the model's performance, paving the way for future advancements in image classification technology.

Thank You
