KGiSL Institute Of Technology

NAAN MUDHALVAN

Problem Definition:

The project involves creating an image recognition system using IBM Cloud Visual Recognition. The goal is to develop a platform where users can upload images, and the system accurately classifies and describes the image contents. This will enable users to craft engaging visual stories with the help of AI-generated captions, enhancing their connection with the audience through captivating visuals and compelling narratives.

We collect a diverse dataset of images relevant to our project.We annotate,label and classify objects based on their appearance , colour , size and shape etc... This step is crucial for training the image recognition model.

Tools Used : LabelImg,OpenCV

Building The Image Recognition Model:

We use Python and TensorFlow's Keras API to create a basic image classification model using the famous MNIST dataset, which contains hand-written digits.

- TENSORFLOW : TensorFlow is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. TensorFlow was originally developed for large numerical computations without keeping deep learning in mind.

- KERAS API: Keras is the high-level API of the TensorFlow platform. It provides an approachable, highly-productive interface for solving machine learning (ML) problems, with a focus on modern deep learning.

- MNIST DATASET: The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.It was created by "re-mixing" the samples from NIST's original datasets. The MNIST database contains 60,000 training images and 10,000 testing images.

Code for developing a basic image classification model:

To use the tensor flow framework we must install it first .So the command to install tensorflow is given below:

```
pip install tensorflow
```

Training and testing the model:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers



# Step 1: Load and preprocess the dataset

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Normalize pixel values to the range [0, 1]
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
# Step 2: Define the model architecture
model = keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),  # Flatten the 28x28 input images
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.2),  # Add dropout to reduce overfitting
    layers.Dense(10, activation='softmax')  # 10 output classes (digits 0-9)
])

# Step 3: Compile the model
model.compile(optimizer='adam',  # You can use other optimizers like SGD, RMSprop,
etc.
              loss='sparse_categorical_crossentropy',  # Cross-entropy loss for classification
              metrics=['accuracy'])

# Step 4: Train the model
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# Step 5: Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f'\nTest accuracy: {test_accuracy * 100:.2f}%')

# Step 6: Make predictions
predictions = model.predict(x_test)

# You can use the model to predict on new images:
# new_predictions = model.predict(new_images)
```

For more complex image recognition tasks, we consider using pre-trained models like VGG, ResNet, or Inception, and fine-tuning them on our dataset.

Here's a high-level example of building an image recognition model using a pre-trained CNN (ResNet50) and fine-tuning it on the ImageNet dataset:

```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam
```

```
# Step 1: Load the pre-trained ResNet50 model (without top classification layer)

base_model = ResNet50(weights='imagenet', include_top=False)

# Step 2: Add custom classification layers for your specific task

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)  # Define the number of classes in your dataset

model = Model(inputs=base_model.input, outputs=predictions)

# Step 3: Freeze the layers of the pre-trained model (optional)

for layer in base_model.layers:
    layer.trainable = False

# Step 4: Compile the model

model.compile(optimizer=Adam(lr=0.0001),
 loss='categorical_crossentropy',
 metrics=['accuracy'])

# Step 5: Data preprocessing and augmentation
# Load your high-value dataset and perform data preprocessing and augmentation as needed.

# Step 6: Train the model

model.fit(train_data, train_labels, epochs=num_epochs, batch_size=batch_size, validation_data=(val_data, val_labels))

# Step 7: Evaluate the model

test_loss, test_accuracy = model.evaluate(test_data, test_labels, verbose=2)
print(f'\nTest accuracy: {test_accuracy * 100:.2f}%')

# Step 8: Make predictions

predictions = model.predict(test_data)

# You can use the model to predict on new images:
# new_predictions = model.predict(new_images)
```

In this code, we fine-tune a pre-trained ResNet50 model on your high-value dataset. Replace **num_classes**, **train_data**, **train_labels**, **val_data**, **val_labels**, **test_data**, and **test_labels** with your specific dataset and labels.

This is a basic outline of the process. Depending on the complexity of your dataset and the model's performance, you may need to explore techniques like data augmentation, different architectures, and hyperparameter tuning to achieve the best results. Additionally, for very large datasets, you may consider using distributed training on multiple GPUs or TPUs.

TEAM MEMBERS:

1.Chandini.D

2.Arthi.R

3.Sophia.S

4.Hemalatha.K