Aim

To develop an image classification model that can classify images into two categories: "Happy" and "Sad" using a Convolutional Neural Network (CNN) with TensorFlow and Keras.

Objective

1. Preprocess and clean image data by unzipping files and filtering images based on their format.
2. Create a dataset from the cleaned images, split it into training, validation, and test sets.
3. Design and train a CNN model to classify images into two classes.
4. Evaluate the model's performance and save it for future use.

Summary

This project involves preprocessing image data, creating a CNN model, training it on the data, and evaluating its performance. The model is designed to classify images into two categories. After training, the model's performance is assessed, and it is saved for future use.

Tools and Libraries Used

- **Python**: Programming language used for the project.
- **TensorFlow**: For building and training the CNN model.
- **Keras**: High-level API for TensorFlow to build and train the model.
- **OpenCV**: For reading and processing images.
- **Matplotlib**: For visualizing training progress and images.
- **Zipfile**: For unzipping image files.

Procedure

1. **Setup and Dependencies:**

CODE:

```
pip install opencv-python matplotlib

import tensorflow as tf

import cv2

import imghdr

import os

import zipfile
```

2. **GPU Configuration:**

CODE:

```
physical_devices = tf.config.list_physical_devices('GPU')
try:
  tf.config.experimental.set_memory_growth(physical_devices[0], True)
except:
  pass
```

3. **Unzipping Images:**

CODE:

```
def unzip_images(directory):
  for root, dirs, files in os.walk(directory):
    for file in files:
      if file.endswith('.zip'):
        zip_filepath = os.path.join(root, file)
        with zipfile.ZipFile(zip_filepath, 'r') as zip_ref:
          zip_ref.extractall(root)
```

4. **Image Validation:**

CODE:

```
data_dir = 'data'
image_exts = ['jpeg','jpg', 'bmp', 'png']
for image_class in os.listdir(data_dir):
  for image in os.listdir(os.path.join(data_dir, image_class)):
    image_path = os.path.join(data_dir, image_class, image)
    try:
      img = cv2.imread(image_path)
      tip = imghdr.what(image_path)
      if tip not in image_exts:
        os.remove(image_path)
    except Exception as e:
      print('Issue with image {}'.format(image_path))
```

5. **Dataset Preparation:**

CODE:

```
data = tf.keras.utils.image_dataset_from_directory('data')
data = data.map(lambda x,y: (x/255, y))
```

6. **Dataset Splitting:**

CODE:

```
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)
test_size = int(len(data)*.1)
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

7. **Model Design:**

CODE:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout

model = Sequential()
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,3)))
model.add(MaxPooling2D())
model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

8. **Model Training:**

CODE:

```
logdir='logs'
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
hist = model.fit(train, epochs=20, validation_data=val, callbacks=[tensorboard_callback])
```

9. **Performance Visualization:**

CODE:

```
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
plt.legend(loc="upper left")
plt.show()

fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
plt.legend(loc="upper left")
plt.show()
```

10. **Evaluation:**

CODE:

```
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
for batch in test.as_numpy_iterator():
   X, y = batch
   yhat = model.predict(X)
   pre.update_state(y, yhat)
   re.update_state(y, yhat)
   acc.update_state(y, yhat)
print(pre.result(), re.result(), acc.result())
```

11. **Prediction and Saving Model:**

CODE:

```
img = cv2.imread('8iAb9k4aT.jpg')

resize = tf.image.resize(img, (256,256))

yhat = model.predict(np.expand_dims(resize/255, 0))

if yhat > 0.5:

   print(f'Predicted class is Sad')

else:

   print(f'Predicted class is Happy')


model.save(os.path.join('models','imageclassifier.h5'))

new_model = load_model('imageclassifier.h5')
```

Highlights

- **Model Architecture**: A simple CNN with three convolutional layers, each followed by max pooling, which helps in extracting features from images and reducing their dimensions.
- **Data Preprocessing**: Includes unzipping, filtering image formats, and normalizing pixel values.
- **Performance Visualization**: Plots of loss and accuracy curves for both training and validation sets to monitor the model's learning progress.

Conclusion

The image classifier successfully categorizes images into "Happy" or "Sad" based on the trained CNN model. The preprocessing steps ensure that only valid images are used for training, and performance metrics provide insights into the model's effectiveness. The trained model is saved and can be reloaded for future predictions.