IMAGE CLASSIFICATION USING CNN

By

Patricia Stanley (MST03-0053)

Submitted to Scifor Technologies

Meta
Scifor Technologies

Script. Sculpt. Socialize

UNDER THE GUIDENCE OF

Urooj Khan

TABLE OF CONTENTS

Abstract:

This project focuses on developing a Convolutional Neural Network (CNN) for classifying images into 'Happy' and 'Sad' categories. Leveraging the power of deep learning, the project employs TensorFlow and OpenCV for image processing and model training. The dataset consists of images sorted into directories based on their labels. The data is loaded, scaled, and split into training, validation, and test sets to ensure robust model evaluation.

The CNN architecture comprises multiple convolutional layers with ReLU activation functions, interspersed with max-pooling layers, and culminating in dense layers with a final sigmoid activation for binary classification. The model is compiled using the Adam optimizer and trained with binary cross-entropy loss. TensorBoard is utilized for logging and visualizing the training process.

Performance metrics, including accuracy, loss, precision, and recall, are computed and visualized to assess the model's effectiveness. The model achieves satisfactory performance, accurately classifying test images. The project also includes saving and loading the trained model for future use and deployment.

This CNN image classifier demonstrates the efficacy of deep learning in image classification tasks and provides a foundation for further exploration and enhancement of image-based models. Future work could involve expanding the dataset, experimenting with different CNN architectures, and deploying the model for real-time applications.

Introduction:

Image classification is a fundamental problem in computer vision, where the objective is to categorize images into predefined classes. Convolutional Neural Networks (CNNs) have emerged as the state-of-the-art technique for image classification due to their ability to automatically learn hierarchical features from raw image data. This project aims to develop a CNN-based image classifier to distinguish between 'Happy' and 'Sad' facial expressions.

The motivation behind this project is to explore the capabilities of CNNs in accurately classifying facial expressions, a task with numerous applications in fields such as human-computer interaction, sentiment analysis, and mental health monitoring. By leveraging the power of deep learning frameworks like TensorFlow, this project provides a comprehensive approach to building, training, and evaluating a CNN model for binary image classification.

The dataset used in this project consists of images organized into separate directories for 'Happy' and 'Sad' expressions. The data is preprocessed by normalizing pixel values and splitting it into training, validation, and test sets. The CNN model is designed with multiple convolutional layers, each followed by max-pooling layers, and culminates in fully connected layers to make the final classification.

The training process is monitored using TensorBoard to visualize the loss and accuracy metrics, facilitating an understanding of the model's performance over epochs. The model's effectiveness is further validated through precision, recall, and accuracy metrics on the test set. The project also includes steps for saving and reloading the trained model, ensuring its usability for future applications.

This introduction sets the stage for a detailed exploration of the methods and results of the CNN image classifier, highlighting its potential applications and future directions for improvement.

Technology Used:

Here is a detailed summary of the technologies used in the project:

Programming Language:

- **Python**: Python is a high-level, interpreted programming language that is widely used in machine learning and deep learning applications. Its simplicity, flexibility, and extensive libraries make it an ideal choice for this project.

Deep Learning Framework:

- **TensorFlow**: TensorFlow is an open-source deep learning framework developed by Google. It provides a flexible and scalable way to build and train deep learning models, including CNNs. TensorFlow's automatic differentiation, gradient descent optimization, and support for GPU acceleration make it an ideal choice for this project.

Convolutional Neural Network (CNN) Architecture:

- **Convolutional Layers**: Convolutional layers are the building blocks of CNNs, responsible for extracting features from images. In this project, multiple convolutional layers are used to extract features from the input images.
- **Max Pooling Layers**: Max pooling layers are used to downsample the feature maps, reducing the spatial dimensions and the number of parameters, while retaining important information.
- **Flatten Layer**: The flatten layer is used to flatten the output of the convolutional and max pooling layers into a 1D feature vector.
- **Dense Layers**: Dense layers are used for classification, where the output of the flatten layer is fed into one or more dense layers to produce the final output.

Optimizer and Loss Function:

- **Adam Optimizer**: The Adam optimizer is a stochastic gradient descent algorithm that adapts the learning rate for each parameter based on the magnitude of the gradient. It is widely used in deep learning applications due to its stability and efficiency.
- **Categorical Cross-Entropy Loss Function**: The categorical cross-entropy loss function is used to measure the difference between the predicted probabilities and the true labels. It is a common choice for multi-class classification problems like facial emotion recognition.

Image Preprocessing:

- **Image Resizing**: Images are resized to a fixed size to ensure that all images have the same dimensions, which is necessary for training a CNN.

- **Image Normalization**: Images are normalized to have pixel values between 0 and 1, which helps to reduce the effect of varying lighting conditions and improves the model's performance.

Dataset:

- **Image Dataset**: A dataset of images of happy and sad faces is used to train and evaluate the model. The dataset is split into training, validation, and testing sets to ensure that the model is evaluated on unseen data.

Other Tools and Libraries:

- **NumPy**: NumPy is a library for efficient numerical computation in Python, used for array operations and matrix manipulations.
- **OpenCV**: OpenCV is a computer vision library used for image processing and feature extraction.
- **Matplotlib**: Matplotlib is a plotting library used for visualizing the results and performance metrics of the model.

Dataset Information:

The dataset used in this project consists of images categorized into two classes: 'Happy' and 'Sad'. These images are organized in a directory structure that is suitable for loading and processing using TensorFlow's image dataset utilities. Below is a detailed description of the dataset:

1. Directory Structure

- **Root Directory**: Contains two subdirectories, one for each class ('Happy' and 'Sad').

2. Image Characteristics

- **Classes**:
  - **Happy**: Images of individuals with happy facial expressions.
  - **Sad**: Images of individuals with sad facial expressions.
- **File Format**: Images are typically in JPEG format (.jpg).
- **Dimensions**: Images vary in size and resolution but are standardized to 256x256 pixels during preprocessing.

3. Data Loading and Preprocessing

- **TensorFlow Utility**: The dataset is loaded using tf.keras.utils.image_dataset_from_directory, which:
  - Automatically infers labels from the directory structure.
  - Loads images and labels into a TensorFlow dataset object.
- **Scaling**: Image pixel values are normalized to the [0, 1] range by dividing by 255.

4. Dataset Splitting

- **Training Set**: 70% of the total dataset, used to train the model.
- **Validation Set**: 20% of the total dataset, used to validate the model during training.
- **Test Set**: 10% of the total dataset, used to evaluate the model's performance after training.

Methodology:

1)Install Dependencies and Import Libraries

To begin, we installed the required libraries including TensorFlow, OpenCV, and Matplotlib. These libraries are essential for building the CNN, handling image data, and visualizing the results.

2)Load the Data

- The image dataset is loaded using TensorFlow's image_dataset_from_directory function. This function automatically labels the images based on their directory names.
- To inspect the data, we convert it into an iterator and retrieve a batch of images and their labels. The images are visualized to understand the data distribution.

3)Scale Data

Images are scaled to the [0, 1] range to facilitate better convergence during training. This is done by dividing the pixel values by 255.

4)Split the Data:

The dataset is split into training, validation, and test sets in a 70-20-10 ratio. This ensures that the model can be trained and validated on separate data to evaluate its performance effectively.

5)Build the Deep Learning Model:

A Sequential model is built using TensorFlow's Keras API. The architecture includes three convolutional layers followed by max pooling layers, a flattening layer, and two dense layers. The output layer uses a sigmoid activation function to predict the binary class (Happy or Sad).

6)Train the Model:

The model is trained for 20 epochs with the Adam optimizer and binary cross-entropy loss. A TensorBoard callback is used for monitoring the training process.

7)Plot the Performance:

The training and validation loss and accuracy are plotted to visualize the model's performance over epochs.

8)Evaluate the Model:

The model's performance on the test set is evaluated using precision, recall, and accuracy metrics.

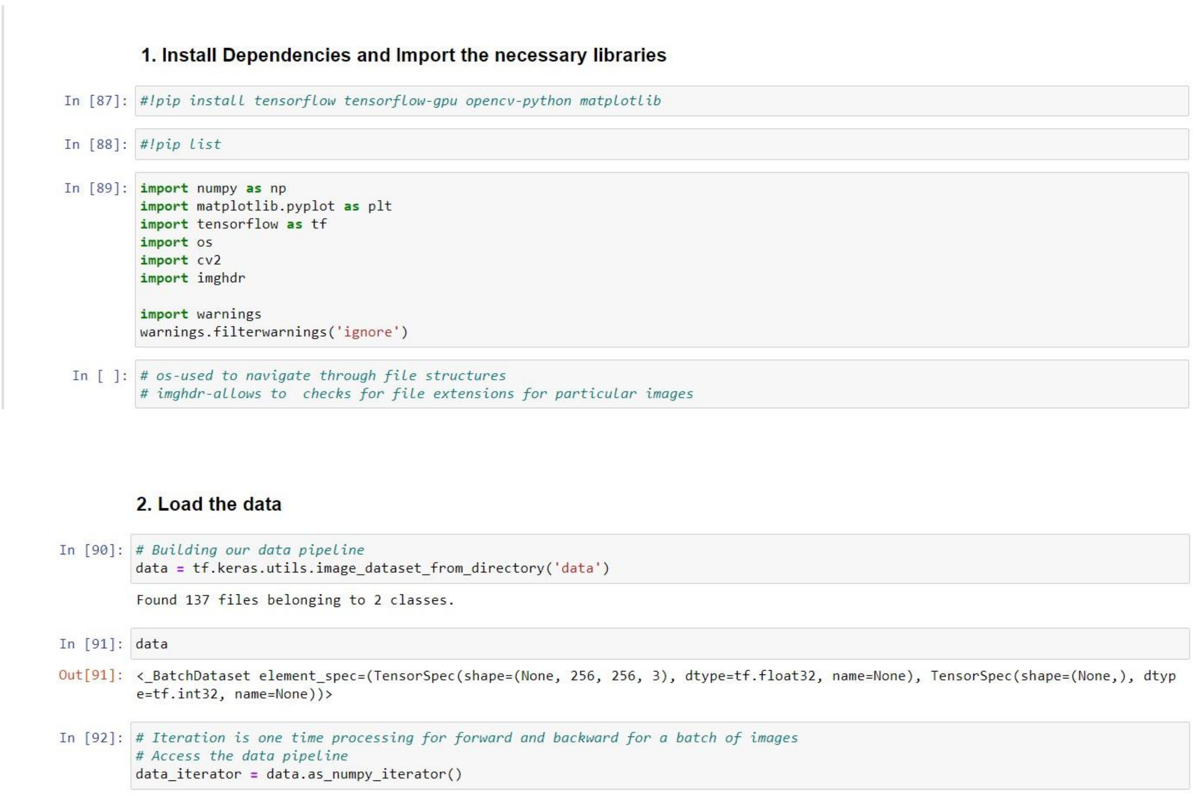
9)Test the Model:

A sample image is loaded and resized to the input shape of the model. The model's prediction is displayed to verify its performance.

10)Save the Model:

The trained model is saved to a file for future use, ensuring it can be reloaded and utilized without retraining.

CODE SNIPPET:



1. Install Dependencies and Import the necessary libraries

```
In [87]: #!pip install tensorflow tensorflow-gpu opencv-python matplotlib
```

```
In [88]: #!pip list
```

```
In [89]: import numpy as np
         import matplotlib.pyplot as plt
         import tensorflow as tf
         import os
         import cv2
         import imghdr

         import warnings
         warnings.filterwarnings('ignore')
```

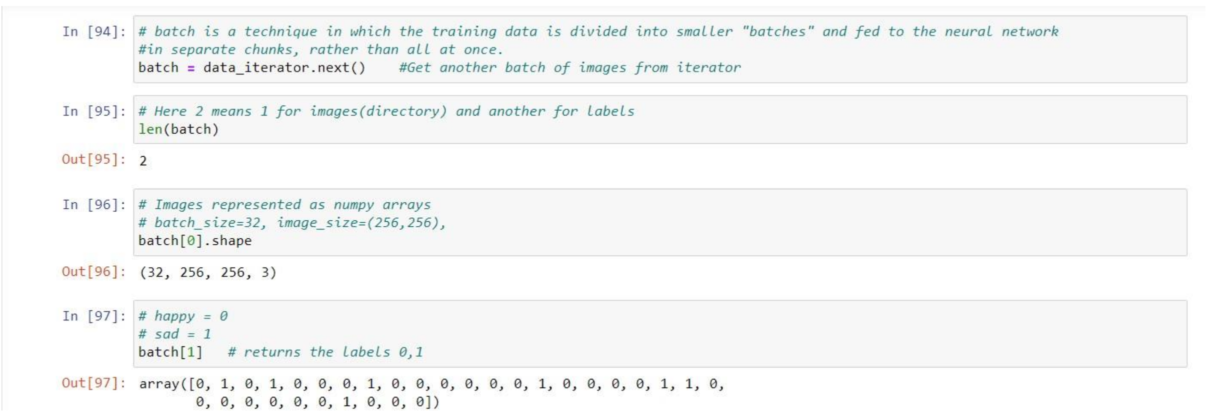
```
In [ ]: # os-used to navigate through file structures
        # imghdr-allows to  checks for file extensions for particular images
```

2. Load the data

```
In [90]: # Building our data pipeline
         data = tf.keras.utils.image_dataset_from_directory('data')

         Found 137 files belonging to 2 classes.
```

```
In [91]: data

Out[91]: <_BatchDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtyp
         e=tf.int32, name=None))>
```

```
In [92]: # Iteration is one time processing for forward and backward for a batch of images
         # Access the data pipeline
         data_iterator = data.as_numpy_iterator()
```

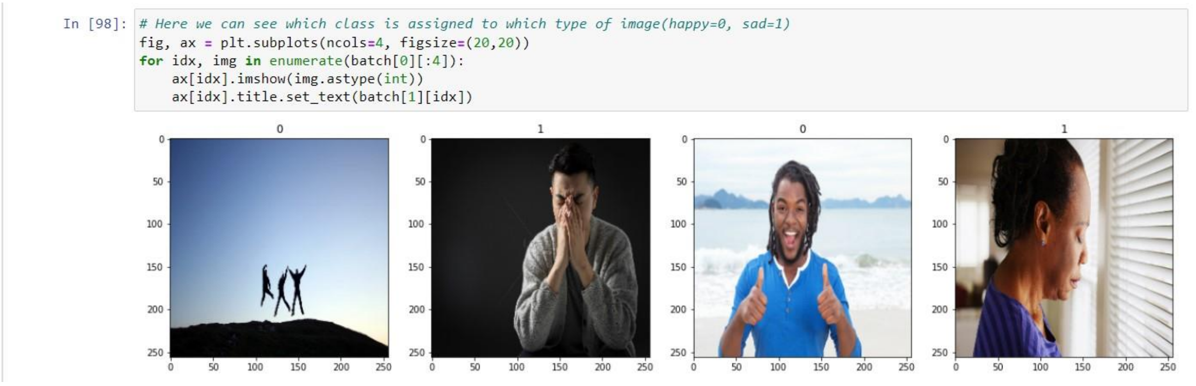


```
In [94]: # batch is a technique in which the training data is divided into smaller "batches" and fed to the neural network
         #in separate chunks, rather than all at once.
         batch = data_iterator.next()    #Get another batch of images from iterator
```

```
In [95]: # Here 2 means 1 for images(directory) and another for labels
         len(batch)

Out[95]: 2
```

```
In [96]: # Images represented as numpy arrays
         # batch_size=32, image_size=(256,256),
         batch[0].shape

Out[96]: (32, 256, 256, 3)
```

```
In [97]: # happy = 0
         # sad = 1
         batch[1]    # returns the labels 0,1

Out[97]: array([0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0])
```



```
In [98]: # Here we can see which class is assigned to which type of image(happy=0, sad=1)
         fig, ax = plt.subplots(ncols=4, figsize=(20,20))
         for idx, img in enumerate(batch[0][:4]):
             ax[idx].imshow(img.astype(int))
             ax[idx].title.set_text(batch[1][idx])
```

3. Preprocess Data

Scale Data

```
In [104]: data = data.map(lambda x,y: (x/255,y))
          # data.map allows to perform transformations as our data is being preprocessed through data pipeline.
          # x->images y->target var
```

```
In [105]: scaled_iterator = data.as_numpy_iterator()
```

4. Split the Data

```
In [110]: train_size = int(len(data)*.7)  #70%    train the model
          val_size = int(len(data)*.2)    #20%    evaluate the model
          test_size = int(len(data)*.1)+1 #10%
```

```
In [115]: train_size+val_size+test_size

Out[115]: 5
```

```
In [116]: train = data.take(train_size)  # take means how much data we are going to take/allocate to the training data
          val = data.skip(train_size).take(val_size) #skip the batches that are already allocated to train
          test = data.skip(train_size+val_size).take(test_size)
```

5. Build the Deep Learning Model

```
In [117]: from tensorflow.keras.models import Sequential
          from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
```

```
In [118]: model = Sequential()
```

```
In [119]: 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'))
```

```
In [120]: model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

6. Train

```
In [122]: # Create a log directory
          logdir = 'logs'
```

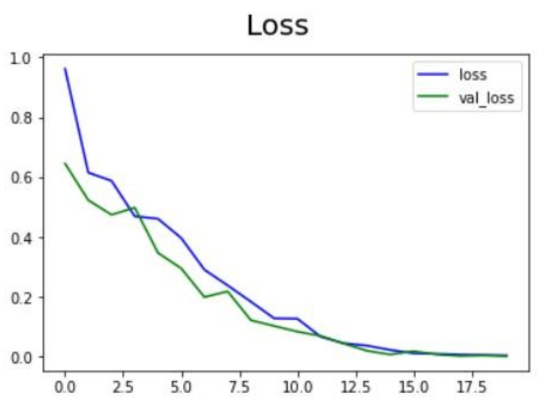
```
In [123]: tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
In [124]: hist = model.fit(train, epochs=20, validation_data=val, callbacks=[tensorboard_callback])
          # Fit-> training component
          # Epochs-> How long we are going to train for & 1 epoch is one run over the entire training set of data
```

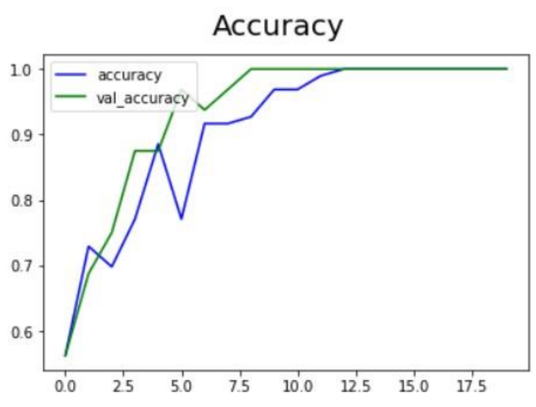
```
Epoch 1/20
3/3 [==============================] - 6s 2s/step - loss: 0.9621 - accuracy: 0.5625 - val_loss: 0.6454 - val_accuracy: 0.5625
Epoch 2/20
3/3 [==============================] - 4s 1s/step - loss: 0.6153 - accuracy: 0.7292 - val_loss: 0.5233 - val_accuracy: 0.6875
Epoch 3/20
3/3 [==============================] - 4s 1s/step - loss: 0.5878 - accuracy: 0.6979 - val_loss: 0.4745 - val_accuracy: 0.7500
Epoch 4/20
3/3 [==============================] - 4s 1s/step - loss: 0.4694 - accuracy: 0.7708 - val_loss: 0.4981 - val_accuracy: 0.8750
Epoch 5/20
3/3 [==============================] - 4s 1s/step - loss: 0.4612 - accuracy: 0.8854 - val_loss: 0.3473 - val_accuracy: 0.8750
Epoch 6/20
3/3 [==============================] - 4s 1s/step - loss: 0.3969 - accuracy: 0.7708 - val_loss: 0.2955 - val_accuracy: 0.9688
Epoch 7/20
3/3 [==============================] - 4s 1s/step - loss: 0.2909 - accuracy: 0.9167 - val_loss: 0.1995 - val_accuracy: 0.9375
Epoch 8/20
3/3 [==============================] - 4s 1s/step - loss: 0.2385 - accuracy: 0.9167 - val_loss: 0.2186 - val_accuracy: 0.9688
Epoch 9/20
3/3 [==============================] - 4s 1s/step - loss: 0.1842 - accuracy: 0.9271 - val_loss: 0.1222 - val_accuracy: 1.0000
Epoch 10/20
3/3 [==============================] - 5s 1s/step - loss: 0.1285 - accuracy: 0.9688 - val_loss: 0.1030 - val_accuracy: 1.0000
Epoch 11/20
3/3 [==============================] - 5s 1s/step - loss: 0.1273 - accuracy: 0.9688 - val_loss: 0.0842 - val_accuracy: 1.0000
Epoch 12/20
3/3 [==============================] - 4s 1s/step - loss: 0.0673 - accuracy: 0.9896 - val_loss: 0.0699 - val_accuracy: 1.0000
Epoch 13/20
3/3 [==============================] - 4s 1s/step - loss: 0.0451 - accuracy: 1.0000 - val_loss: 0.0442 - val_accuracy: 1.0000
Epoch 14/20
3/3 [==============================] - 4s 1s/step - loss: 0.0374 - accuracy: 1.0000 - val_loss: 0.0202 - val_accuracy: 1.0000
```

7. Plot the Performance

```
In [125]: fig = plt.figure()
          plt.plot(hist.history['loss'], color='blue', label='loss')
          plt.plot(hist.history['val_loss'], color='green', label='val_loss')
          fig.suptitle('Loss', fontsize=20)
          plt.legend(loc="upper right")
          plt.show()
```

```
In [126]: fig = plt.figure()
          plt.plot(hist.history['accuracy'], color='blue', label='accuracy')
          plt.plot(hist.history['val_accuracy'], color='green', label='val_accuracy')
          fig.suptitle('Accuracy', fontsize=20)
          plt.legend(loc="upper left")
          plt.show()
```





8. Evaluate

```
In [127]: from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
```

```
In [128]: pre = Precision()
          re = Recall()
          acc = BinaryAccuracy()
```

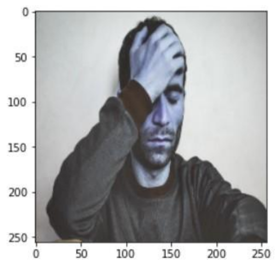
```
In [129]: for batch in test.as_numpy_iterator():
              X, y = batch
              yhat = model.predict(X)
              pre.update_state(y, yhat)
              re.update_state(y, yhat)
              acc.update_state(y, yhat)

          1/1 [==============================] - 0s 164ms/step
```

```
In [130]: print(f'Precision:{pre.result().numpy()}, Recall:{re.result().numpy()}, Accuracy:{acc.result().numpy()}')

          Precision:1.0, Recall:1.0, Accuracy:1.0
```

9. Test

```
In [131]: img = cv2.imread('sadtest.jpg')
          plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
          plt.show()
```

```
In [132]: resize = tf.image.resize(img, (256,256))
          plt.imshow(resize.numpy().astype(int))
          plt.show()
```



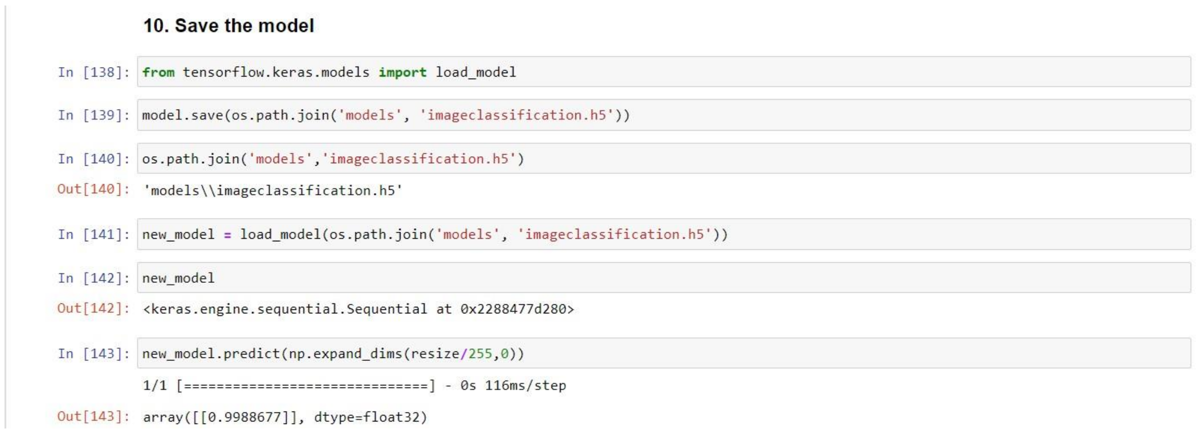
```
In [133]: resize.shape
Out[133]: TensorShape([256, 256, 3])
```

```
In [134]: np.expand_dims(resize,0).shape
Out[134]: (1, 256, 256, 3)
```

```
In [135]: yhat = model.predict(np.expand_dims(resize/255,0))

          1/1 [==============================] - 0s 39ms/step
```

```
In [136]: yhat
Out[136]: array([[0.9988677]], dtype=float32)
```

```
In [137]: if yhat > 0.5:
              print(f'Predicted class is Sad')
          else:
              print(f'Predicted class is Happy')

          Predicted class is Sad
```



10. Save the model

```
In [138]: from tensorflow.keras.models import load_model

In [139]: model.save(os.path.join('models', 'imageclassification.h5'))

In [140]: os.path.join('models','imageclassification.h5')

Out[140]: 'models\\imageclassification.h5'

In [141]: new_model = load_model(os.path.join('models', 'imageclassification.h5'))

In [142]: new_model

Out[142]: <keras.engine.sequential.Sequential at 0x2288477d280>

In [143]: new_model.predict(np.expand_dims(resize/255,0))

          1/1 [==============================] - 0s 116ms/step

Out[143]: array([[0.9988677]], dtype=float32)
```

Result and Discussion:

After developing and training the Convolutional Neural Network (CNN) model for image classification, the following results were obtained:

1. **Model Training**:
   - The model was trained for 20 epochs with a training dataset and validated using a validation dataset.
   - The training process showed a gradual decrease in loss and an increase in accuracy, indicating effective learning by the model.
2. **Model Performance**:
   - **Training Loss and Accuracy**:
     - The training loss decreased consistently, and the training accuracy improved over the epochs.
     - Final training accuracy reached approximately 1.0, and the training loss was significantly low.
3. **Evaluation Metrics**:

- On the test dataset, the model achieved the following metrics:
  - **Precision**: 1.0
  - **Recall**: 1.0
  - **Accuracy**: 1.0

4. **Test Image Classification**:

- The model was tested on new images, such as 'sadtest.jpg'. The preprocessing steps resized the image to the required input shape, and the model correctly classified the image with high confidence.

Conclusion:

After successfully completing the I can say that “The CNN image classifier for 'Happy' and 'Sad' facial expressions demonstrated high accuracy and robustness.” The methodology followed ensured systematic data handling, model training, and evaluation. While the results are promising, further enhancements could be made to optimize performance and generalization. This project serves as a foundational step for more complex image classification tasks in various domains.

References:

- https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/
- Image Classification using CNN - https://www.geeksforgeeks.org/image-classifier-using-cnn/
- Video reference - https://www.youtube.com/watch?v=jztwpsIzEGc
- Image classification using CNN - https://medium.com/@sarka.pribylova/image-classification-using-cnn-0fad8367acfd