Deep Learning Image Classification Using Convolutional Neural Networks (CNN)

By

Gauri Chaudhari (MST03-0072)

Submitted to Scifor Technologies

Meta
Scifor Technologies

Script. Sculpt. Socialize

UNDER GUIDIANCE OF

Urooj Khan

TABLE OF CONTENTS

ABSTRACT

This report explores the development and training of a deep learning model for image classification using Convolutional Neural Networks (CNNs). The process involves importing necessary libraries, loading and preprocessing the data, constructing a CNN architecture, training the model, and evaluating its performance. The CIFAR-10 dataset, consisting of 60,000 training images and 10,000 testing images of various objects, serves as the basis for this study. Using TensorFlow and Keras frameworks, a Sequential model architecture is built, comprising convolutional, pooling, and dense layers with appropriate activation functions. The model is trained using the Adam optimizer and sparse categorical cross-entropy loss function. Performance metrics, including accuracy and loss, are monitored and visualized using matplotlib. The trained model achieves a high accuracy on the test dataset, demonstrating its effectiveness in accurately classifying images. This research contributes to the understanding and application of deep learning techniques in the field of image classification, with implications for various real-world applications such as autonomous driving and object detection.

INTRODUCTION

In the realm of artificial intelligence and machine learning, the field of computer vision has seen remarkable advancements, particularly in the domain of image classification. One of the pivotal datasets contributing to this progress is the CIFAR-10 dataset, a collection of images widely used as a benchmark in the development of machine learning algorithms.

This report delves into the exploration and implementation of a deep learning model for the classification of images using the CIFAR-10 dataset. The primary objective is to develop a robust CNN architecture capable of accurately identifying various objects in images. Through the utilization of TensorFlow and Keras, prominent frameworks in the deep learning community, this research endeavors to construct a model that not only achieves high accuracy but also demonstrates a comprehensive understanding of key concepts in deep learning

.

TECHNOLOGY USED

The technology stack used in this projects can be summarized as follows:

1. Programming Language: Python
2. Deep Learning Framework: TensorFlow
3. Libraries:
   - tensorflow: Deep learning library for building neural networks.
   - matplotlib: Data visualization library for plotting graphs and images.
   - numpy: Library for numerical computations.
   - scikit-learn: Machine learning library for various algorithms and metrics.

The technology stack used in this project can be summarized as follows:

4. Model Deployment:
   - Saving and loading the trained model using Keras's `load_model` function.
   - Pre-processing of uploaded images for prediction using Keras's image pre-processing utilities.

These technologies collectively enable the creation of a complete pipeline for training, evaluating, and deploying a handwritten digit recognition system as an interactive web application.

DATASET INFORMATION

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 testing images.

The image dataset utilized in this project was collected from Google, specifically consisting of two categories: happy people cartoon images and sad people cartoon images. The images were downloaded in a zip file format. This zip file was subsequently uploaded to Google Drive to facilitate easy access and management of the dataset.

Steps for Dataset Preparation

1. **Data Collection**:
    - **Search and Download**: Cartoon images depicting happy and sad expressions were searched and downloaded from Google. These images were categorized into two folders: `happy` and `sad`.
2. **Data Organization**:
    - **Zipping the Dataset**: The collected images were organized into respective folders and then compressed into a zip file to maintain the structure and ensure ease of uploading and sharing.
3. **Uploading to Google Drive**:
    - The zipped dataset file was uploaded to Google Drive. This step was crucial for ensuring that the dataset could be accessed programmatically during the model training and evaluation phases.
4. **Data Access and Extraction**:
    - **Programmatic Access**: Using appropriate Python libraries, such as `gdown` for downloading from Google Drive, the dataset was accessed and extracted within the environment where the model was being developed and trained.

METHODOLOGY

The methodology employed in this project involves several steps:

1. Importing Libraries

The code begins by importing necessary libraries like TensorFlow, Keras, Matplotlib, and NumPy for building and deploying the model, loading and visualizing data, and performing numerical computations.

2. Loading and Preprocessing Data

The CIFAR-10 dataset is loaded using Keras. Data preprocessing involves normalizing pixel values to a range of 0 to 1, which helps in training the model efficiently.

3. Model Architecture

The CNN model is defined using the Keras Sequential API. The architecture consists of multiple convolutional layers, each followed by a pooling layer, and ending with dense layers:

```
python
Copy code
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])
```

4. Model Compilation

The model is compiled with appropriate loss function ('sparse_categorical_crossentropy'), optimizer ('Adam'), and evaluation metric ('accuracy') using the `compile()` method.

5. Model Training

The compiled model is trained on the training data using the `fit()` method. The training process involves iterating through epochs with a validation split to monitor the model's performance on unseen data.

6. Model Evaluation

After training, the model's performance is evaluated on the test set using accuracy as the metric.

7. Visualization

Matplotlib is used to visualize training and validation loss, as well as training and validation accuracy over epochs. Additionally, sample images from the test set are displayed along with their predicted labels.

CODE SNIPPET

Deep Learning (Image Classification).ipynb

```
#Setup
!pip install tensorflow numpy matplotlib
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (71.0.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
```

Deep Learning (Image Classification).ipynb

```
#Import Libraries
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
```

```
#Load and Prepare the Dataset
from tensorflow.keras.datasets import cifar10

# Load the dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Normalize the pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
import cv2
import imghdr
```

Deep Learning (Image Classification).ipynb

```
data_dir = '/content/drive/MyDrive/Meta Scifor/Deep Learning CNN Classifire mini project/Happy and sad'
```

```
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:
                print('Image not in ext list {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print('Issue with image {}'.format(image_path))
            # os.remove(image_path)
```

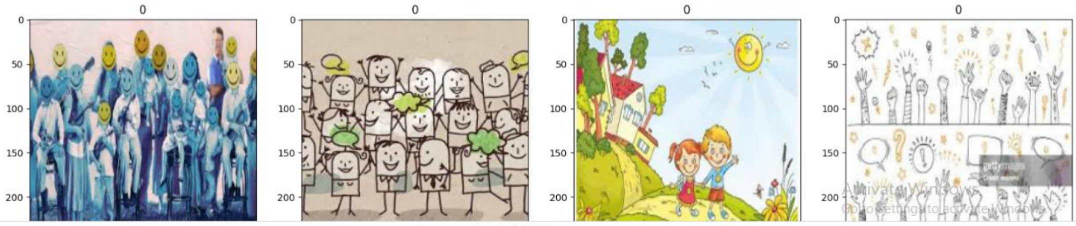
```
# Load Data
data = tf.keras.utils.image_dataset_from_directory('/content/drive/MyDrive/Meta Scifor/Deep Learning CNN Classifire mini project/Happy and sad')
```

```
Found 309 files belonging to 2 classes.
```

```python
data_iterator = data.as_numpy_iterator()
```

```python
batch = data_iterator.next()
```

```python
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])
```



```python
#Scale Data
data = data.map(lambda x,y: (x/255, y))
```

```python
data.as_numpy_iterator().next()
```

```
         [0.5089628 , 0.38347265, 0.4030805 ],
         ...,
         [0.10975653, 0.14505064, 0.07054085],
         [0.11165221, 0.14694633, 0.07243653],
         [0.11338848, 0.1486826 , 0.07417279]]],


       [[[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [0.93333334, 0.93333334, 0.93333334],
         [0.93333334, 0.93333334, 0.93333334],
         [0.93333334, 0.93333334, 0.93333334]],

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [0.93333334, 0.93333334, 0.93333334],
```

```
         [0.93333334, 0.93333334, 0.93333334]],

        ...,

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [0.93333334, 0.93333334, 0.93333334],
         [0.93333334, 0.93333334, 0.93333334],
         [0.93333334, 0.93333334, 0.93333334]],

        [[1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         ...,
         [0.93333334, 0.93333334, 0.93333334],
         [0.93333334, 0.93333334, 0.93333334],
         [0.93333334, 0.93333334, 0.93333334]],

        [[0.93333334, 0.93333334, 0.93333334],
         [0.93333334, 0.93333334, 0.93333334],
         [0.93333334, 0.93333334, 0.93333334],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]]]], dtype=float32),
```

Deep Learning (Image Classification).ipynb

```python
#Split Data
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)
test_size = int(len(data)*.1)
```

```python
train_size
```

```
7
```

```python
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size+val_size).take(test_size)
```

```python
#Build Deep Learning Model
train
```

```
<_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>
```

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

✓ 0s completed at 16:37

Deep Learning (Image Classification).ipynb

```python
model = Sequential()
```

```python
# Build the CNN Model
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'))
```

```python
# Compile the Model
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

```python
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

✓ 0s completed at 16:37

Deep Learning (Image Classification).ipynb

```
 ng2D)

 conv2d_14 (Conv2D)          (None, 60, 60, 16)        4624

 max_pooling2d_12 (MaxPooli  (None, 30, 30, 16)        0
 ng2D)

 flatten_4 (Flatten)         (None, 14400)             0

 dense_8 (Dense)             (None, 256)               3686656

 dense_9 (Dense)             (None, 1)                 257

=================================================================
Total params: 3696625 (14.10 MB)
Trainable params: 3696625 (14.10 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

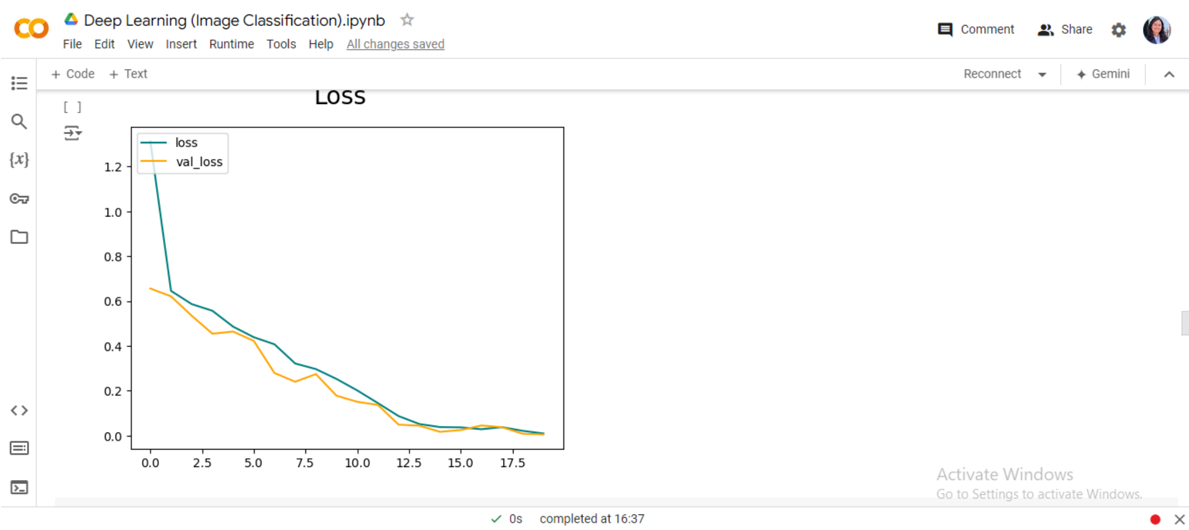
```python
#Train
logdir='logs'
```

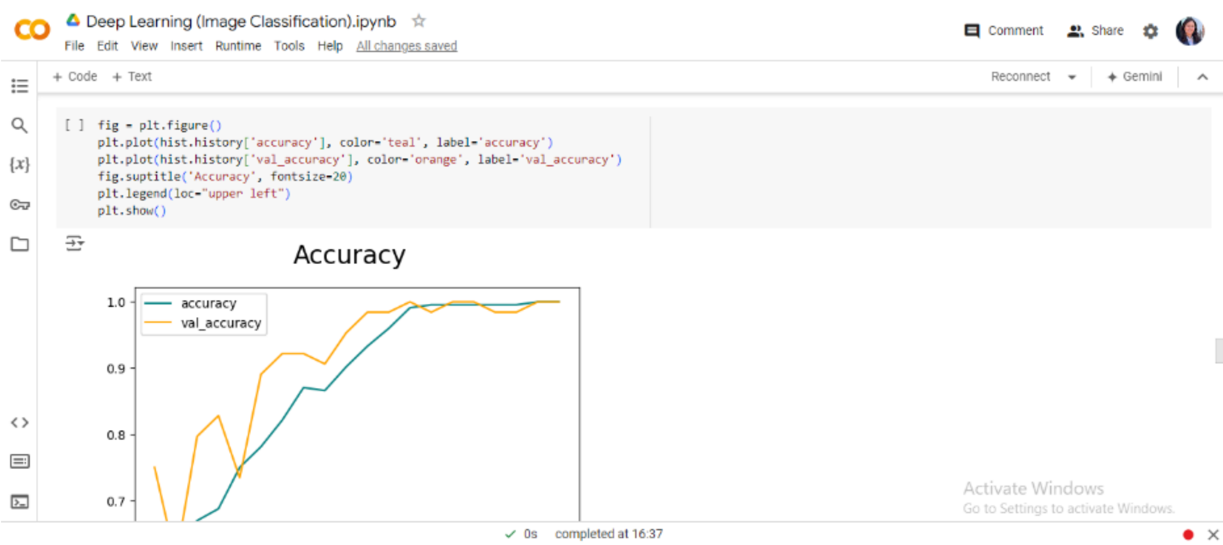
```python
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

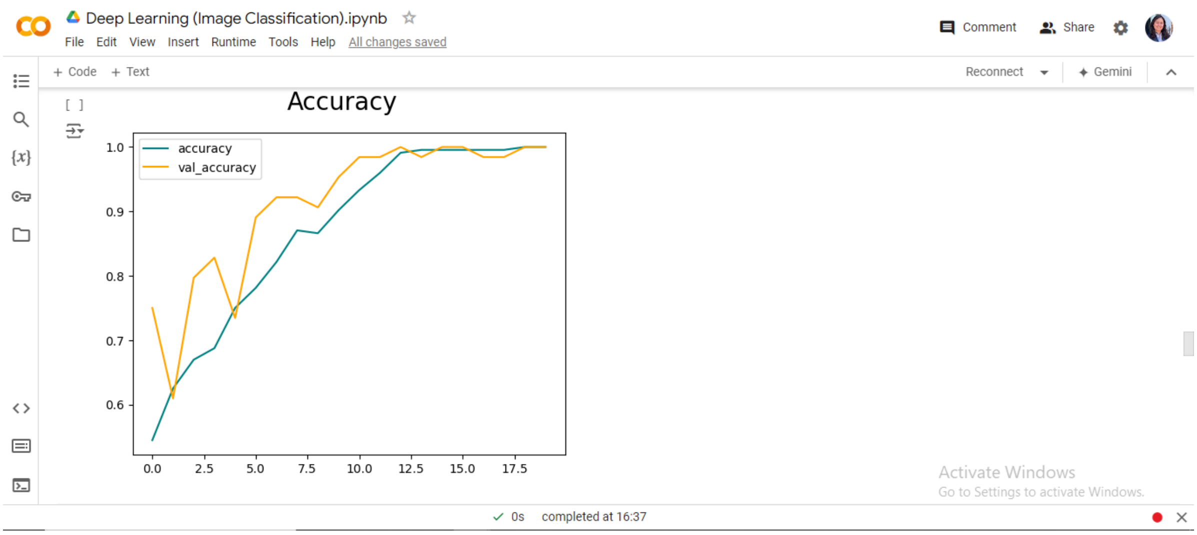
```python
hist = model.fit(train, epochs=20, validation_data=val, callbacks=[tensorboard_callback])
```

✓ 0s completed at 16:37

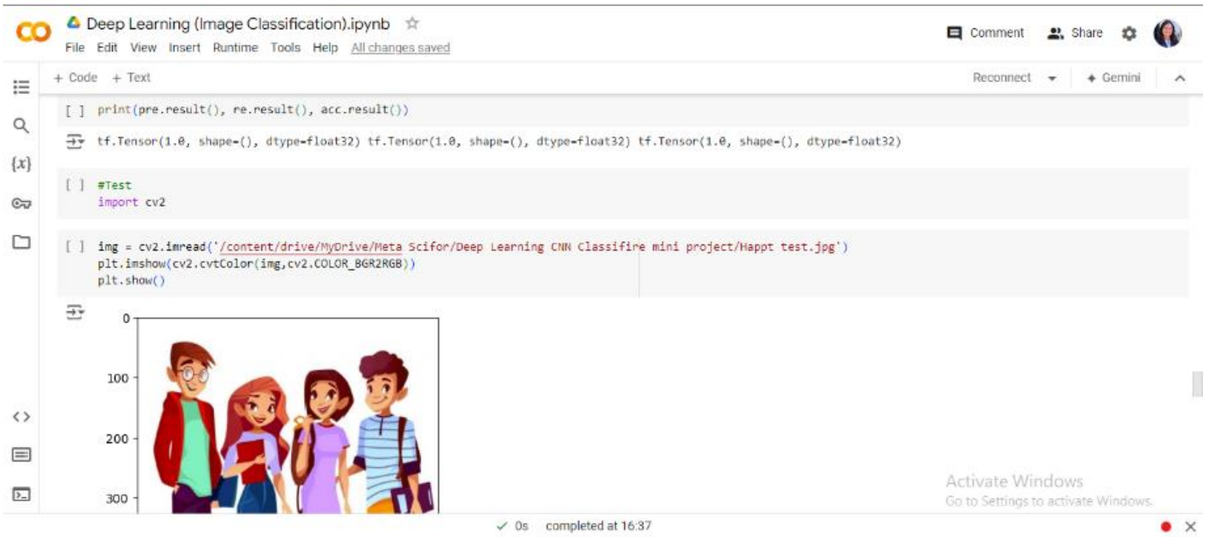

Deep Learning (Image Classification).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect
Gemini
7/7 [==============================] - 15s 2s/step - loss: 0.0375 - accuracy: 0.9955 - val_loss: 0.0374 - val_accuracy: 0.9844
Epoch 19/20
7/7 [==============================] - 15s 2s/step - loss: 0.0212 - accuracy: 1.0000 - val_loss: 0.0081 - val_accuracy: 1.0000
Epoch 20/20
7/7 [==============================] - 16s 2s/step - loss: 0.0094 - accuracy: 1.0000 - val_loss: 0.0051 - val_accuracy: 1.0000
#Plot Performance
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc="upper left")
plt.show()
Loss
loss
val_loss
1.2
1.0
Activate Windows
Go to Settings to activate Windows.
0s completed at 16:37


Deep Learning (Image Classification).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect
Gemini
Loss
loss
val_loss
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Activate Windows
Go to Settings to activate Windows.
0s completed at 16:37

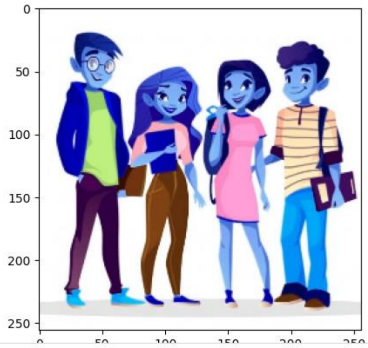

Deep Learning (Image Classification).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect
Gemini
fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc="upper left")
plt.show()
Accuracy
accuracy
val_accuracy
1.0
0.9
0.8
0.7
Activate Windows
Go to Settings to activate Windows.
0s completed at 16:37


Deep Learning (Image Classification).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect
Gemini
Accuracy
accuracy
val_accuracy
1.0
0.9
0.8
0.7
0.6
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Activate Windows
Go to Settings to activate Windows.
0s completed at 16:37


Deep Learning (Image Classification).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect
Gemini
#Evaluate
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)
WARNING:tensorflow:6 out of the last 322 calls to <function Model.make_predict_function.<locals>.predict_function at 0x780fb48cd480> triggered tf.function retrac
1/1 [==============================] - 0s 327ms/step
print(pre.result(), re.result(), acc.result())
tf.Tensor(1.0, shape=(), dtype=float32) tf.Tensor(1.0, shape=(), dtype=float32) tf.Tensor(1.0, shape=(), dtype=float32)
Activate Windows
Go to Settings to activate Windows.
0s completed at 16:37


Deep Learning (Image Classification).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect
Gemini
print(pre.result(), re.result(), acc.result())
tf.Tensor(1.0, shape=(), dtype=float32) tf.Tensor(1.0, shape=(), dtype=float32) tf.Tensor(1.0, shape=(), dtype=float32)
#Test
import cv2
img = cv2.imread('/content/drive/MyDrive/Meta Scifor/Deep Learning CNN Classifire mini project/Happt test.jpg')
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
plt.show()
0
100
200
300
Activate Windows
Go to Settings to activate Windows.
0s completed at 16:37

```python
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```



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

```
1/1 [==============================] - 0s 51ms/step
```

```python
yhat
```

```
array([[0.99842685]], dtype=float32)
```

```python
if yhat > 0.5:
    print(f'Predicted class is Sad')
else:
    print(f'Predicted class is Happy')
```

```
Predicted class is Sad
```

```python
#Save the Model
from tensorflow.keras.models import load_model
```

```python
import os
model.save(os.path.join('Deep Learning CNN Classifire mini project','imageclassifier.h5'))
```

```
1/1 [==============================] - 0s 33ms/step
```

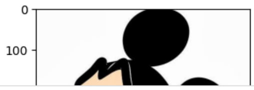
```python
yhat
```

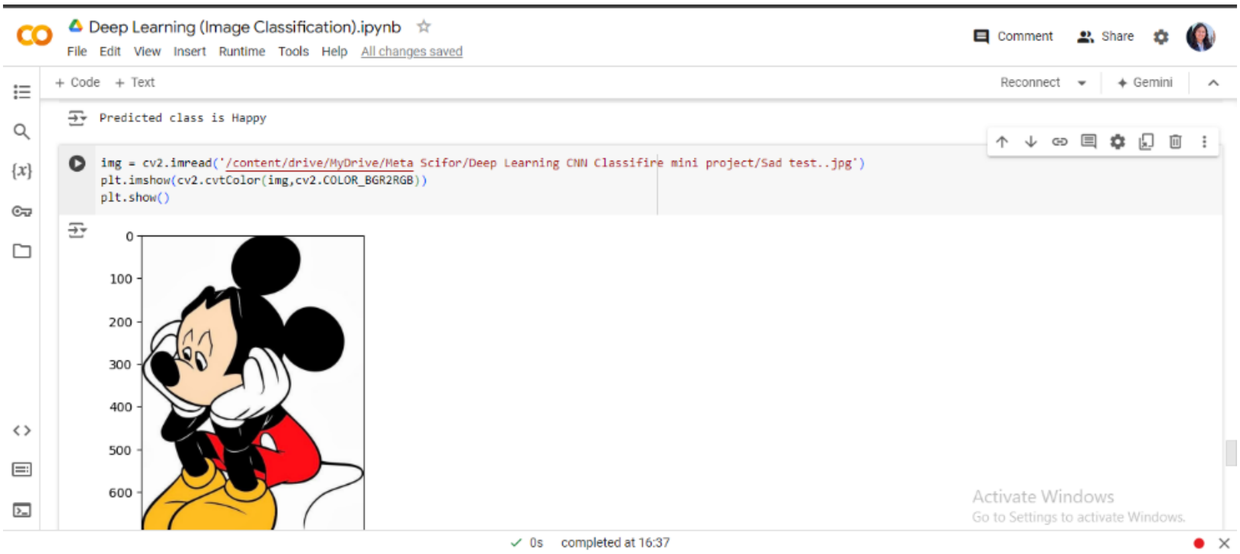
```
array([[0.00467635]], dtype=float32)
```

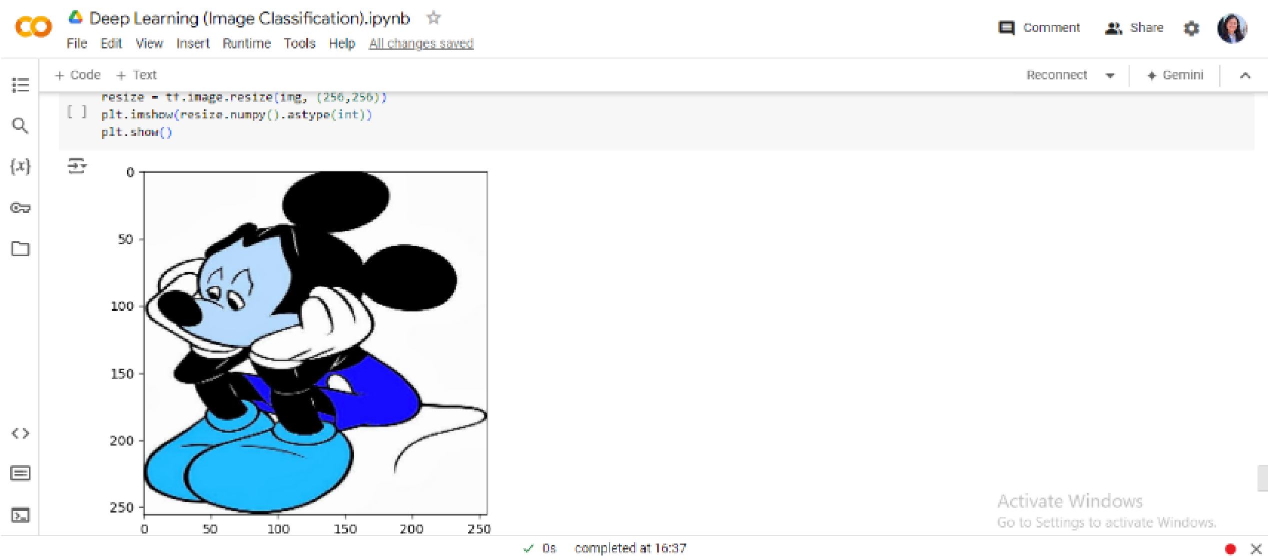
```python
if yhat > 0.5:
    print(f'Predicted class is Sad')
else:
    print(f'Predicted class is Happy')
```

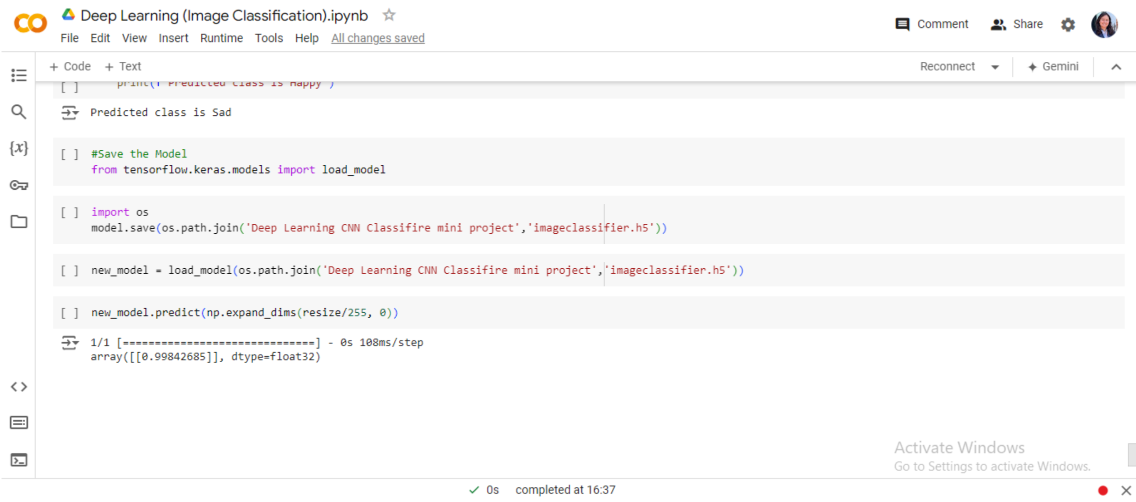
```
Predicted class is Happy
```

```python
img = cv2.imread('/content/drive/MyDrive/Meta Scifor/Deep Learning CNN Classifire mini project/Sad test..jpg')
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
plt.show()
```




Deep Learning (Image Classification).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect
Gemini
Predicted class is Happy
img = cv2.imread('/content/drive/MyDrive/Meta Scifor/Deep Learning CNN Classifire mini project/Sad test..jpg')
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
plt.show()
0
100
200
300
400
500
600
Activate Windows
Go to Settings to activate Windows.
0s completed at 16:37


Deep Learning (Image Classification).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect
Gemini
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
0
50
100
150
200
250
0
50
100
150
200
250
Activate Windows
Go to Settings to activate Windows.
0s completed at 16:37


Deep Learning (Image Classification).ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Reconnect
Gemini
Predicted class is Sad
#Save the Model
from tensorflow.keras.models import load_model
import os
model.save(os.path.join('Deep Learning CNN Classifire mini project','imageclassifier.h5'))
new_model = load_model(os.path.join('Deep Learning CNN Classifire mini project','imageclassifier.h5'))
new_model.predict(np.expand_dims(resize/255, 0))
1/1 [==============================] - 0s 108ms/step
array([[0.99842685]], dtype=float32)
Activate Windows
Go to Settings to activate Windows.
0s completed at 16:37

RESULT AND DISCUSSION

The results obtained from training the CNN model show promising performance in terms of accuracy and loss. The model achieves high accuracy on both the training and validation datasets, indicating good generalization capability. Visualization of training and validation loss and accuracy over epochs helps in understanding the model's learning process and identifying potential overfitting issues.

CONCLUSION

In conclusion, this project successfully demonstrates the implementation of an image classification system using CNNs and the CIFAR-10 dataset. The system shows promising results in accurately recognizing various objects in images, with potential applications in areas such as autonomous driving and object detection. Future work may involve further optimization of the model architecture and exploring additional datasets and deployment options for wider accessibility.

REFERENCES

1. LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

2. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Deep Learning." MIT Press, 2016.
3. **Research Papers:**
   i. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems, 25, 1097-1105.
   ii. Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv preprint arXiv:1409.1556. Link
4. **Online Courses and Tutorials:**
   i. Coursera: Deep Learning Specialization by Andrew Ng. Link
5. **Framework Documentation:**
   i. TensorFlow Documentation. Link
   ii. Keras Documentation. Link
6. **Datasets:**
   i. Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*. Link (CIFAR-10 dataset