AMITY UNIVERSITY
UTTAR PRADESH

AMITY INSTITUTE OF INFORMATION & TECHNOLOGY

SUMMER INTERNSHIP (ETSI100)
Report File

Program: BCA + MCA(DUAL)
Student Name: Bhupinder Pal Singh
Enrollment no.: A1049521015
Faculty Guide's Name: Dr Misbah Anjum

Contents

Abstract

A prominent area of research in computer vision is image classification using machine learning methods. It entails creating models that can instantly sort photos into predetermined classes according to their visual characteristics. By utilizing massive datasets and potent computer resources, machine learning approaches such as deep neural networks have demonstrated exceptional success in addressing picture categorization challenges.

An overview of machine learning algorithms for image classification is given in this report. It goes over important ideas like model training, feature extraction, and evaluation metrics. In the context of picture classification tasks, a number of machine learning methods are investigated, including support vector machines (SVM), random forests, and convolutional neural networks (CNN). Along with considerations for dataset preparation and preprocessing methods, these algorithms' benefits and drawbacks are examined.

Machine learning-based picture classification has a wide range of practical applications in fields like autonomous driving, medical imaging, surveillance, and social media content moderation. The study finishes by noting the current research in picture categorization and emphasizing the necessity for machine learning models to be interpretable, fair, and resilient in order to assure their reliable application in real-world scenarios.

Chapter 1 Introduction

The capacity of image classification models to automatically classify images into various groups based on their visual content has attracted a lot of interest in the field of computer vision. One such area of interest is emotion recognition, where image classification models can be used to discern between various emotional states depicted in photographs. The construction of a model for categorizing photographs into two emotional categories—happy and sad—is the main emphasis of this report.

Due to the complexity and subjectivity of emotions, it can be difficult to identify and comprehend human emotions from visual signals. However, the application of machine learning algorithms has yielded encouraging results in reliably predicting emotional states and extracting important information from photos. The model seeks to capture subtle visual signals including facial expressions, body language, and environmental information that contribute to the portrayal of these emotions when categorizing between joyful and sad feelings.

The suggested image classification algorithm uses supervised learning and trains on a labeled dataset of pictures that both express happiness and sadness. To create a powerful and reliable classification model, a variety of strategies including feature extraction, data preprocessing, and model selection will be explored.

The model will use deep learning techniques, especially convolutional neural networks (CNNs), which have proven to be incredibly effective in picture classification tasks, to accomplish accurate emotion classification. CNNs are skilled at learning and extracting pertinent information from images automatically, allowing them to recognize minute nuances and patterns that distinguish between joyful and sad emotions.

The overall goal of this project is to advance the field of emotion detection by creating an image classification model that can distinguish between happy and sad emotions with accuracy. The model aims to improve our comprehension of the emotional states represented in photographs and pave the way for useful applications in a variety of sectors by utilizing machine learning techniques and deep neural networks.

Chapter 2 Tensorflow and Keras

Two well-known open-source libraries that are extensively utilized in the deep learning and machine learning fields are TensorFlow and Keras.

TensorFlow is an end-to-end open-source machine learning platform that was created initially by scientists and engineers from the Google Brain team. It offers a whole ecosystem of resources, tools, and libraries for creating and implementing machine learning models. TensorFlow supports a variety of hardware platforms, including CPUs, GPUs, and specialized accelerators like TPUs (Tensor Processing Units), and is built to efficiently handle large-scale numerical computations.
A high-level neural network API called Keras was created in Python to offer a simple user interface for creating deep learning models. In 2017, Keras, which was first created as a separate library, was formally integrated into TensorFlow. Keras is a fantastic option for beginners for rapid prototyping since it places a strong emphasis on simplicity, usability, and modularity.

How Tensorflow and keras helps in creating a machine learning model
A user-friendly interface is provided for creating deep learning models using Keras, which is now TensorFlow's official high-level API. Keras makes developing, training, and assessing models easier thanks to its simple and intuitive syntax. It removes low-level implementation details so that users can concentrate on creating the model's architecture and defining its behavior. For both inexperienced and seasoned researchers, Keras offers a seamless and effective technique to prototype and experiment with various architectures and settings.

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

A broad variety of pre-defined layers are offered by TensorFlow and Keras and can be quickly added to the model architecture. These layers include normalization layers, recurrent layers, dense layers, convolutional layers, and more. Users can create intricate neural network designs by merely stacking these layers. The availability of pre-defined layers eliminates the need to create these layers from scratch, which saves a lot of time and work. It is also simple to create models with the appropriate features and behavior

because these layers have built-in functionalities like regularization, parameter initialization, and activation functions.

```
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(5, activation='sigmoid'))
```

TensorFlow is designed to take advantage of the GPU and TPU's processing capability, speeding up the training and inference phases of large-scale models.

```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

Users can select from a variety of built-in optimization techniques and loss functions in TensorFlow and apply them to the training of their models.

```
model.compile('adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'
```

Chapter 3 Process of Creating a Machine Learning Model

Step 1.

Data collection: Compile a set of pictures or audio clips that illustrate the emotions of happiness and sadness. Make sure that each sample's matching emotion is accurately categorized in the dataset.

Step 2.

Data Preprocessing: Clean up and preprocess the data that was gathered. This may entail scaling, normalizing, and transforming photos into a format that is appropriate for training.

```
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)
```

Step 3.

Data Split: The dataset should be divided into training and testing sets. The testing set will be used to assess the model's performance once it has been trained using the training set.

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

Step 4.

Model Selection: Choose an appropriate machine learning method or model architecture for the purpose of classifying emotions. Recurrent neural networks (RNNs) for audio-based classification or convolutional neural networks (CNNs) for image-based classification are two popular options. As

an alternative, you can modify pre-trained models for your particular task.

Step 5.

Model training: Use the training dataset to train the chosen model. The model gains the ability to identify patterns and characteristics that distinguish between joyful and sad emotions during training. The model is fed input samples during the training phase, and its internal parameters are changed based on the labeled data's feedback.

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

Step 6.

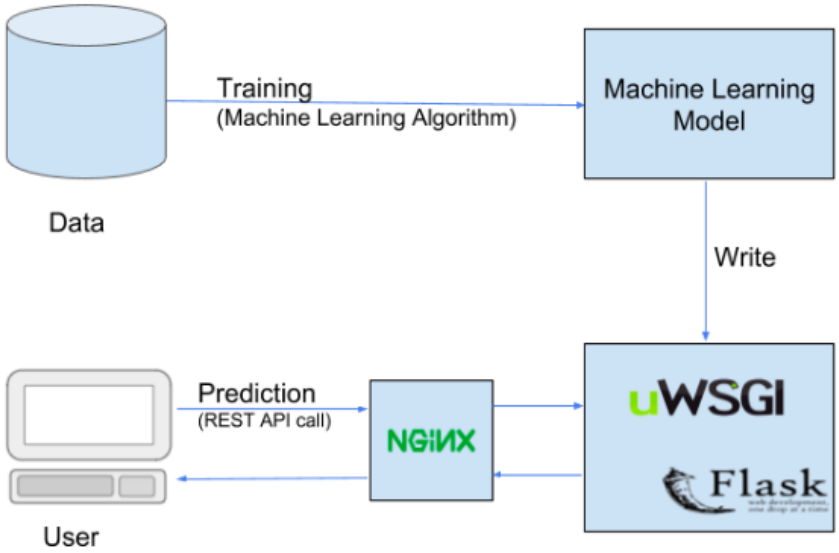
Model Evaluation: Assess the trained model's performance and accuracy in classifying happy and sad emotions using the testing dataset. To gauge the effectiveness of the model, use appropriate evaluation metrics like accuracy, precision, recall, or F1 score.

```
from tensorflow.keras.metrics import Precision, Recall, BinaryAccuracy
```

```
pre = Precision()
re = Recall()
acc = BinaryAccuracy()
```

Step 7.

Model Deployment: Deploy the model after being satisfied with its ability to categorize emotions in actual situations. This could refer to implementing the model into a web or mobile application that would enable users to input photos or sounds and receive predictions of joyful or sad feelings.



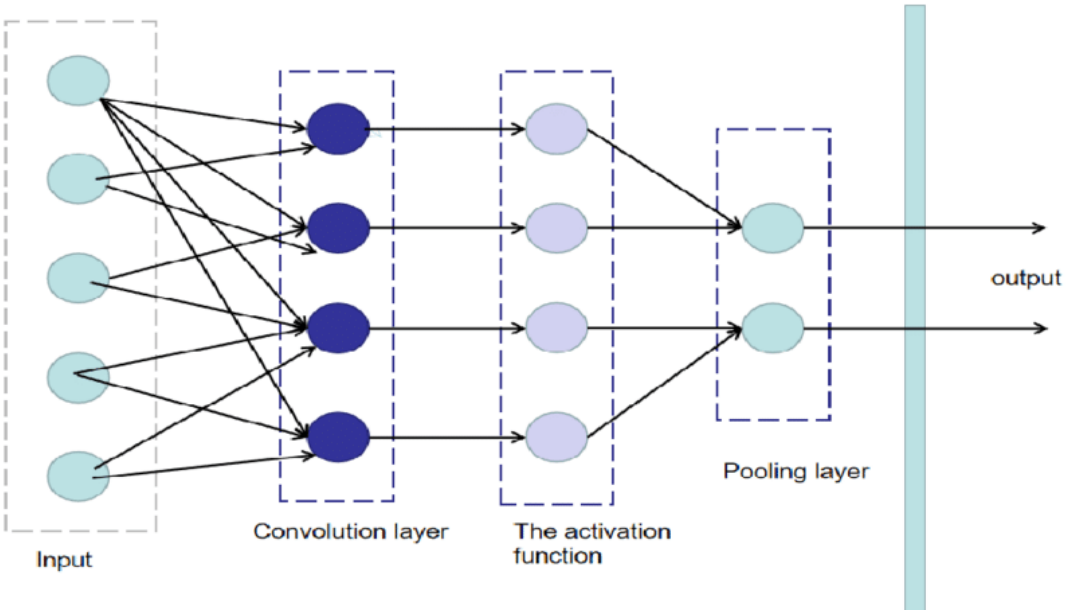
Chapter 4 How a Machine Learning Model Actually Learns?

Step 1.

Initialization: The weights and biases in the model are set at random or with predetermined values. These parameters make up the trainable parts that the model will change.

Step 2.

Forward Propagation: Using the input data and the current parameter values, the model computes a prediction during forward propagation. The layers of the model process the input data as it moves through them, and each layer transforms the input to produce an output. The model then propagates this process until the final forecast is made.

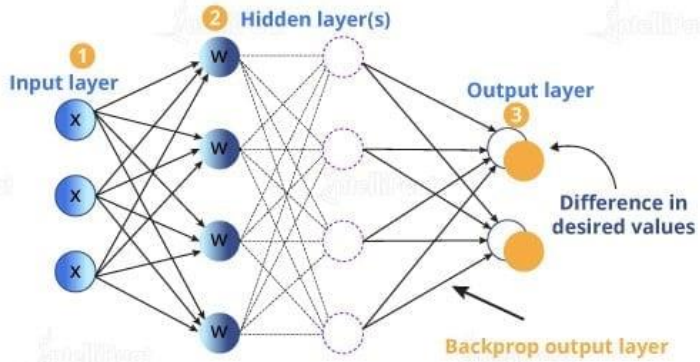


Step 3.

Loss Calculation: The model compares its prediction with the real labels from the training data after forward propagation. The difference between the output that is expected and the actual production is measured by the loss function, sometimes referred to as the objective or cost function. Depending on the particular learning task, a loss function may be selected, such as cross-entropy for classification or mean squared error for regression.

Step 4.

Backpropagation: This phase of training is essential. It determines the gradient of the loss function in relation to each model parameter. This gradient shows the amount and size of the adjustment needed to reduce the loss. The gradients are propagated backward through the model levels using the chain rule of calculus, starting from the top layer.



Step 5

Parameter Update: After computing the gradients, the model uses an optimization procedure to adjust its parameters. Stochastic Gradient Descent (SGD) is the most widely used algorithm, however there are also other variations like Adam, RMSprop, and others. The learning rate, which regulates the step size in the parameter update, and the gradients are used by these algorithms to alter the parameters.

Step 6

Iterative Process: The steps from 2 through 5 are repeated over the course of several iterations, also known as epochs. A forward pass, loss computation, backpropagation, and parameter update are the components of each iteration. The model eventually gains the ability to reduce the loss and enhance its predictions by going over the training data repeatedly and tweaking the parameters.

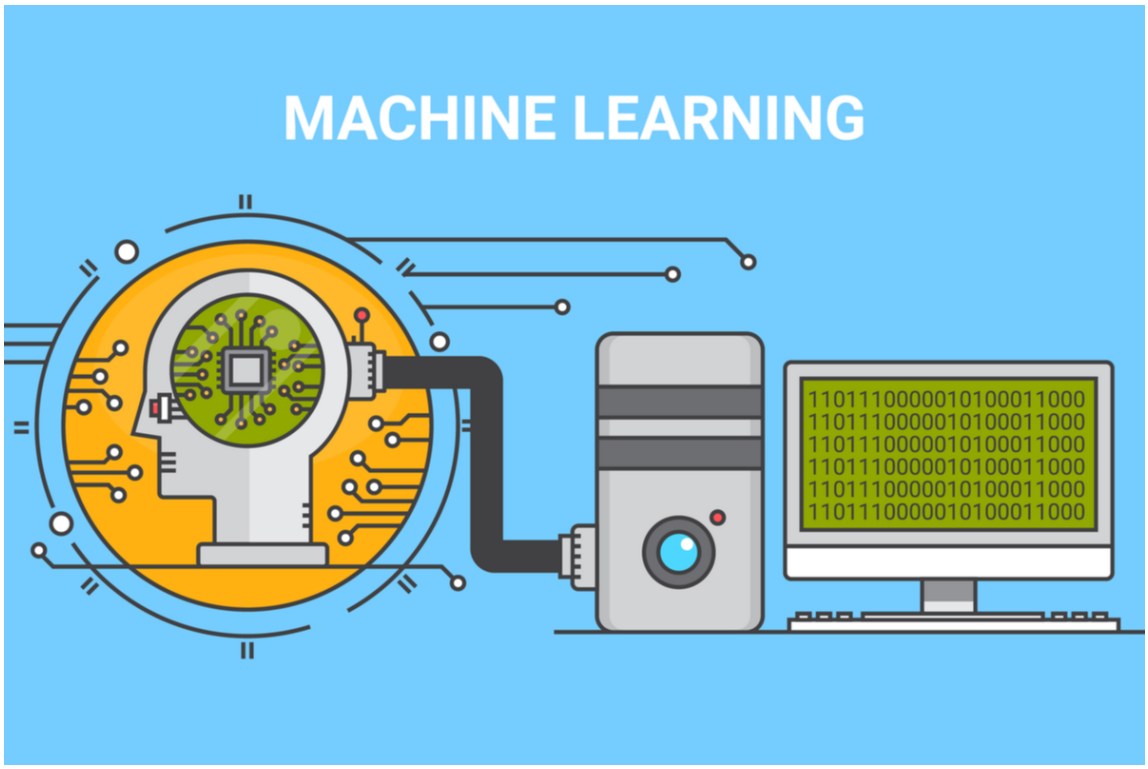
Things that happen in the model training process

Batch Training: Rather than using the complete dataset at once, training is frequently carried out on mini-batches of data in practice. The batch training method increases the effectiveness of the training procedure. A tiny batch of data is processed by the model, which then calculates the loss and

gradients and adjusts the parameters. Up until all batches from the training dataset have been processed, this procedure is repeated like in this example I used a batch size of 16.

Validation: A distinct validation dataset is used to periodically assess the model's performance during training. This aids in tracking the model's generalizability and identifying overfitting. Adjustments can be made to prevent the model from memorizing the training data and failing to perform effectively on unobserved data by paying attention to the validation measures, such as accuracy or loss.

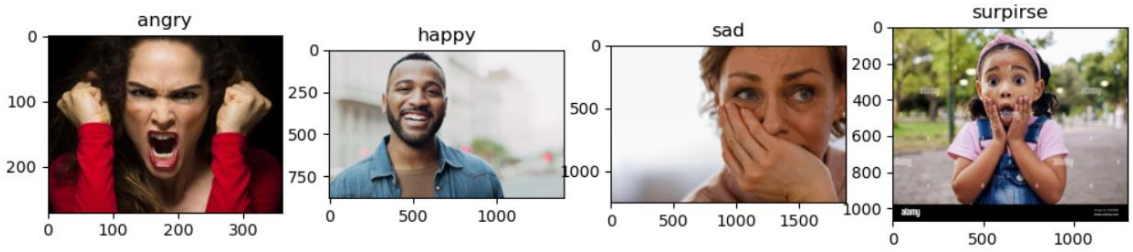
Training Termination: Training is carried out up until a predetermined ending point is reached. This condition can be reached after a predetermined number of epochs, once the system performs to the intended standard, or after the validation metrics stop improving significantly.



Chapter 5 Implementation

Step 1.

Data Collection: For data I collect a bunch of images of human facial expression like happy, sad, angry, surprise.



Step 2.

Filter the Data: During the data collection process there was a lot of wrong format file that need to be removed otherwise the model will pick on wrong patterns.

```
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)
```

```
Image not in ext list data\angry\vector.svg
Image not in ext list data\angry\vector10.svg
Image not in ext list data\angry\vector2.svg
Image not in ext list data\angry\vector3.svg
Image not in ext list data\angry\vector4.svg
Image not in ext list data\angry\vector5.svg
Image not in ext list data\angry\vector6.svg
Image not in ext list data\angry\vector7.svg
Image not in ext list data\angry\vector8.svg
Image not in ext list data\angry\vector9.svg
```

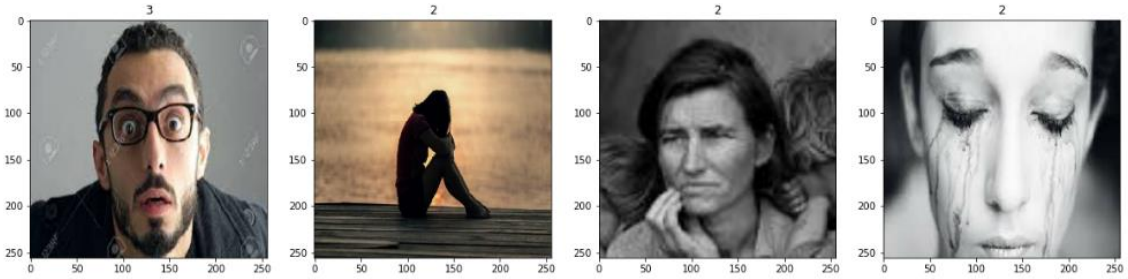
Step 3.

Load Data: Load the collect data into Keras, for that Keras provide a function called “image_dataset_from_directory”

```
import tensorflow as tf
```

```
data = tf.keras.utils.image_dataset_from_directory('data')
```

The function will categorize the data with directory name as labels



The above picture means that label 3 is surprise and label 2 is sad

Step 4.

Scaling and Splitting Data: The scaling of data means to scale image to 0 – 1 because it is faster to do operation on 0 to 1 compared to 0 to 255. After the scaling the data now split the data.

```
data = data.map(lambda x,y: (x/255, y))
```

```
train_size = int(len(data)*.7)
val_size = int(len(data)*.2)+1
test_size = int(len(data)*.1)+1
```

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

Step 5.

Create Model: By using keras layer and model we can create model. For this example, we are using a sequential model which means the layers will be in sequence. The model consists of 9 layer- 1 input layer, 1 output layer, and 7 hidden layer and for the optimizer we are using “adam”. The loss function is “sparse_categorical_crossentropy” and metrics is accuracy

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

```
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(5, activation='sigmoid'))
```

```
model.compile('adam', loss='sparse_categorical_crossentropy', metrics=['accuracy
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 254, 254, 16)      448

 max_pooling2d_6 (MaxPooling  (None, 127, 127, 16)     0
 2D)

 conv2d_7 (Conv2D)           (None, 125, 125, 32)      4640

 max_pooling2d_7 (MaxPooling  (None, 62, 62, 32)       0
 2D)

 conv2d_8 (Conv2D)           (None, 60, 60, 16)        4624

 max_pooling2d_8 (MaxPooling  (None, 30, 30, 16)       0
 2D)

 flatten_2 (Flatten)         (None, 14400)             0

 dense_4 (Dense)             (None, 256)               3686656

 dense_5 (Dense)             (None, 5)                 1285

=================================================================
Total params: 3,697,653
Trainable params: 3,697,653
Non-trainable params: 0
_________________________________________________________________
```

Step 6.

Train Model: Before we start training our model, we must create a logs directory where the model will store its checkpoints. With logs directory, the model can access the checkpoint where it performs better. For Training the model we must call the function fit and provide training data and

validation data that we have created.

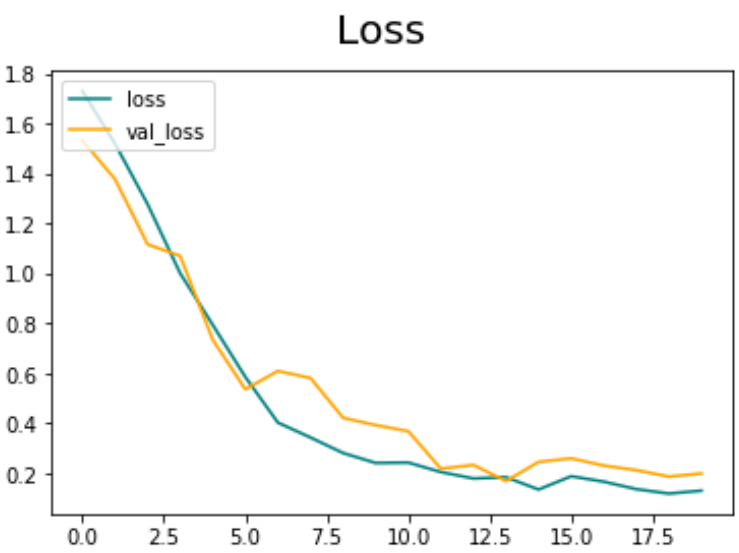
```
logdir='logs'
```

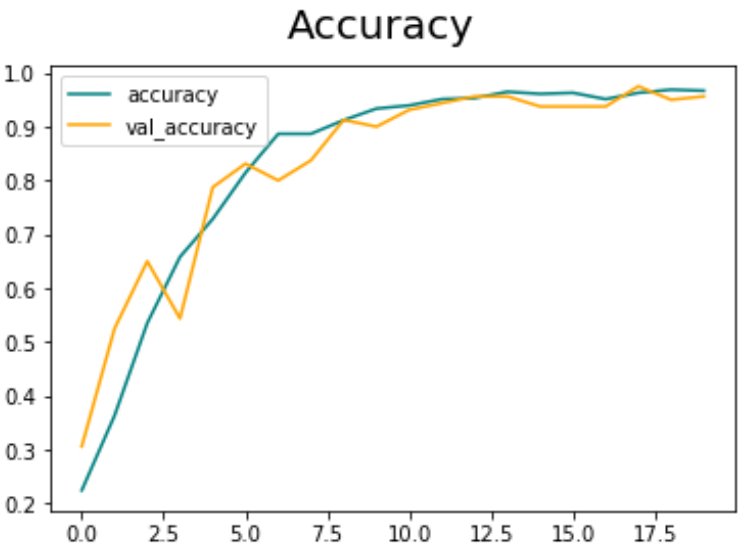
```
logs = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

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

```
Epoch 15/20
16/16 [==============================] - 4s 199ms/step - loss: 0.1358 - accur
acy: 0.9609 - val_loss: 0.2464 - val_accuracy: 0.9375
Epoch 16/20
16/16 [==============================] - 4s 199ms/step - loss: 0.1889 - accur
acy: 0.9629 - val_loss: 0.2605 - val_accuracy: 0.9375
Epoch 17/20
16/16 [==============================] - 4s 196ms/step - loss: 0.1673 - accur
acy: 0.9512 - val_loss: 0.2317 - val_accuracy: 0.9375
Epoch 18/20
16/16 [==============================] - 4s 199ms/step - loss: 0.1372 - accur
acy: 0.9629 - val_loss: 0.2134 - val_accuracy: 0.9750
Epoch 19/20
16/16 [==============================] - 4s 198ms/step - loss: 0.1195 - accur
acy: 0.9688 - val_loss: 0.1876 - val_accuracy: 0.9500
Epoch 20/20
16/16 [==============================] - 4s 200ms/step - loss: 0.1312 - accur
acy: 0.9668 - val_loss: 0.2001 - val_accuracy: 0.9563
```

The model will train for 20 epochs with a batch of 16 images. As you can see the loss of the model decreases and the accuracy increases





Step 7.

Test the Model: We call the predict function of the model and the model give 4 values representing the weightage of each class. The class which has the higher value is the model prediction.

```
import tensorflow as tf
import cv2
import numpy as np
import matplotlib.pyplot as plt

emotions = {"Angry":0, "Happy": 0, "Sad":0, "Surpirse":0}

image_name = "Angry.jpg"
image = cv2.imread(image_name)
resize = tf.image.resize(image, (256, 256))

model_prediction = model.predict(np.expand_dims(resize/255, 0))[0]
for index, emotion in enumerate(emotions):
    emotions[emotion] = model_prediction[index]

predicted_emotion = max(zip(emotions.values(), emotions.keys()))[1]
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image.astype(int))
plt.show()
print(predicted_emotion)
```

```
1/1 [==============================] - 0s 16ms/step
```


```
Angry
```

Chapter 6 Conclusion

With a focus on differentiating between Angry, Happy, Sad, and Surprise, the machine learning project attempted to categorize human emotions. The task included gathering a wide range of samples with labels that represented these feelings. The model trained to recognize the patterns and characteristics connected to each emotion through preprocessing and model building using methods like convolutional neural networks (CNNs) or recurrent neural networks (RNNs).

The model iteratively changed its parameters during training by minimizing a selected loss function. The loss was calculated by comparing the predictions it made through forward propagation to the real labels. Using optimization methods like adam, backpropagation allowed the model to compute gradients and modify its parameters.

The model's efficiency at properly classifying emotions steadily increased over the course of several training epochs. The model's generalization and performance on unobserved cases were evaluated using validation and evaluation on various datasets.

The model was successfully trained so that it could distinguish between the human emotions of Angry, Happy, Sad, and Surprise. The project's findings offer insightful information on emotion recognition and may be used in real-world contexts in a variety of fields, including sentiment analysis, human-computer interaction, and affective computing.

It's crucial to remember that the model's performance should be thoroughly assessed, considering elements like dataset quality, variety, and potential biases. To improve the model's accuracy and resilience in real-world circumstances, more adjustments and fine-tuning may be required.