Configuration Manual

MSc Research Project
Data Analytics

Dharavi Sorathiya
Student ID: x20173636

School of Computing
National College of Ireland

Supervisor: Dr. Bharathi Chakravarthi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Dharavi Rasikbhai Sorathiya		
Student ID:	x20173636		
Programme:	MSc. Data Analytics	**Year:**	2021-22
Module:	MSc. Research Project		
Lecturer:	Dr. Bharathi Chakravarthi		
Submission Due Date:	16th December, 2021		
Project Title:	Gujarati Handwritten Character Recognition using Convolution Neural Network		
Word Count: 723		**Page Count:** 11	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Dharavi R. Sorathiya

Date: 15th December, 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Dharavi Sorathiya
x20173636

1 Introduction

This document explains the prerequisites to implement the method that is created to recognize Gujarati handwritten characters through deep learning algorithms. Moreover, this manual includes all necessary requirements in terms of software and hardware, for project's successful completion.

2 Hardware and Software Configurations

This section covers all the hardware and software requirements which are needed to carry out the project.

2.1 Hardware Configuration:

Table 1 Hardware Configuration

Hardware	Specifications
Operating System	Windows 10 Pro (64-bit)
Processor	Intel(R) Core(TM) i3-7100U
RAM	8 GB
Hard Disk	1 TB

Figure 1 Device Specification

Operating system utilized for project is Windows-10 (64-bit).

2.2 Software Requirements

This poerton elaborates all the software requirements which need to be satisfied for the successful implementation of the project.

Table 2 Software requirements

Software/Library	Version
Python	3.6
Tensorflow	2.7.0
Numpy	1.19.4
OpenCV	4.4.0
Matplotlib	3.3.4
Sklearn	0.24.1

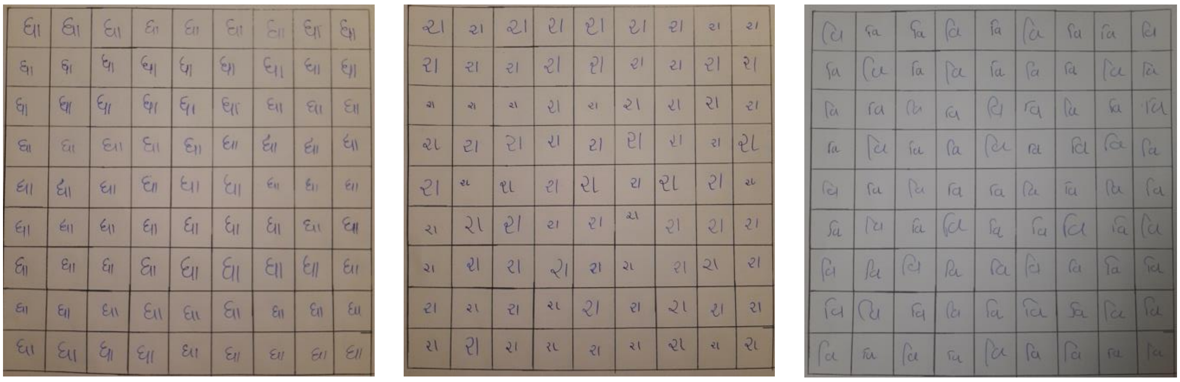
3 Project Implementation

3.1 Dataset Generation

Dataset generation process is implemented in four parts as follows.

3.1.1 Generate raw dataset

This research paper addresses such issue for which no such dataset is availble on any dataset provider such as kaggle. Therefore, a custom dataset is created by writing gujarati characters on papers followed by capturing it by mobile camera. A sample raw dataset is shown below. There are total 374 images including following three.



Dha Ra Vi

Figure 2 Sample raw dataset

Figure 3 Directories generated by script

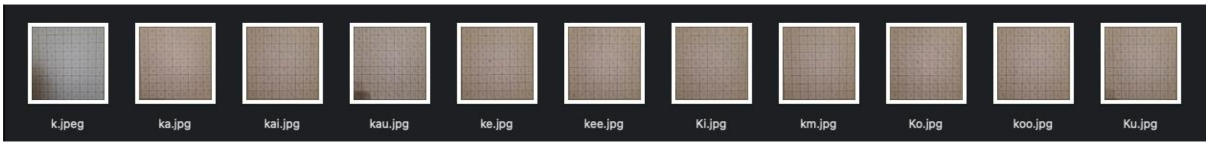


Figure 4 Images in each directory

3.1.2 Generate train dataset

Raw dataset is split into train, test and validation dataset. To perform such logic, a python script is used. A python script to genrate training dataset and its output is shown in the screenshots.

Followings functions are executed in sequence in order to generate train dataset.

- Setting up directories
- Read single raw image in grayscale
- Reshape to 2988 x 2988
- Horizontal split into 9 strips
- Each strips is split into 9 square images
- Margin of 10px is cut from every side of each smallest image.
- Removing extension from file name
- Place each image in corresponding directory.

```python
def parse_images(self):
    self.initialize_root()
    for each_dir in self.dir_list:
        if re.search("\W", each_dir):
            continue
        for each_file in os.listdir(self.raw_dataset + '/' + each_dir):
            if not (re.search('.jp', each_file)):continue
            file_path = self.raw_dataset + '/' + each_dir + '/' + each_file
            img = cv2.resize(cv2.imread(file_path, cv2.IMREAD_GRAYSCALE), self.reshape_dims)
            img = np.array(np.hsplit(img, 9))
            img = np.reshape(np.ravel(img), self.cell_dims)
            for each,i in zip(img, range(img.shape[0])):
                each = each[self.row_margin: self.cell_dims[1] - self.row_margin, self.column_margin: self.cell_dims[2] - self.column_margin]
                image_name = each_file.replace(".jpeg", '')
                image_name = image_name.replace('.jpg', '')
                cv2.imwrite(f'{self.parsed_dataset.lower()}/{image_name.lower()}/{image_name.lower()}_{i+1}.jpg', each)
    if os.path.exists(self.parsed_dataset+".ds_store"):
        shutil.rmtree(self.parsed_dataset+".ds_store")
    if os.path.exists(self.parsed_dataset+".Ds_Store"):
        shutil.rmtree(self.parsed_dataset+".Ds_Store")
```

Figure 5 Train dataset script

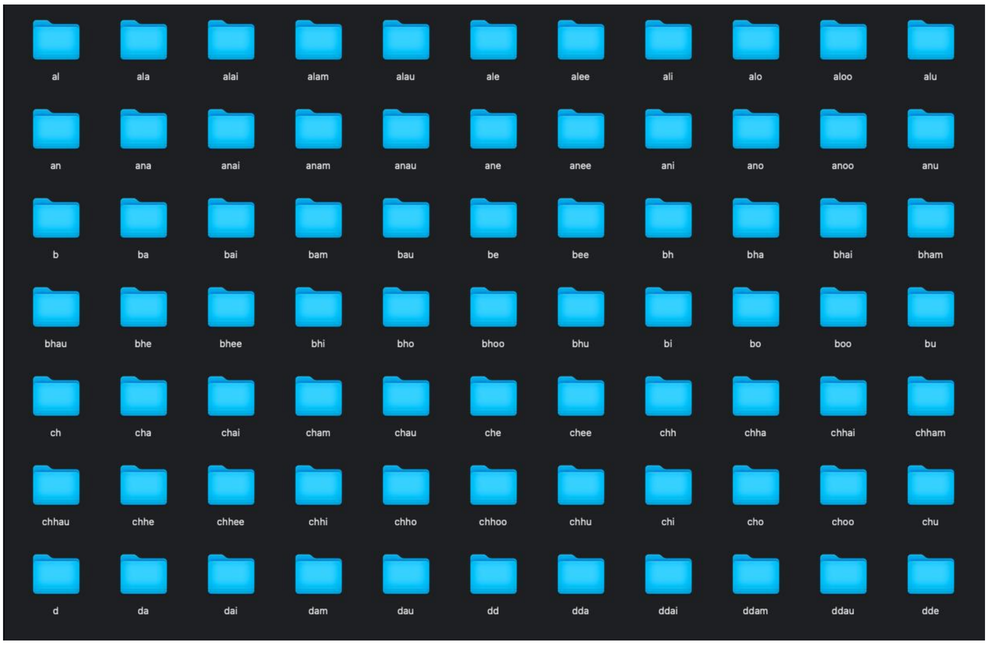


Figure 6 Directories generated by script

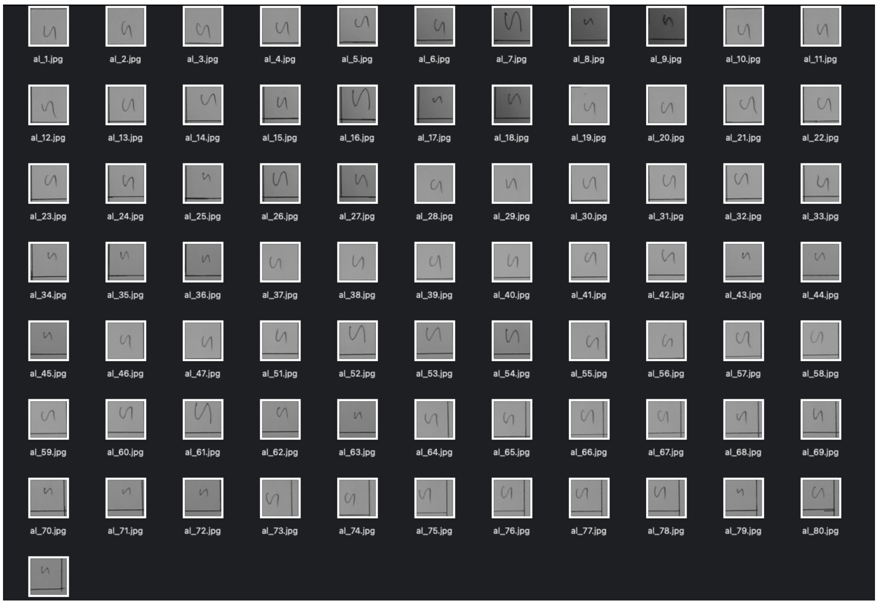


Figure 7 Images in each directory

3.1.3 Generate test dataset

A python script to generate testing dataset and its output is shown in screenshots. Following steps are followed in script in order to generate testing dataset. As output, It moves 374 images(single image from each class) from training dataset to testing dataset.

- Setting up directories
- Moving one image from taining to testing directory.

```
class CharacterParserWithoutDir:
    def __init__(self, split=10):
        self.split = split
        self.train_path = "/dataset/train"
        self.test_path = "/dataset/test"
    def ignore_files(self, dir, files):
        return [f for f in files if os.path.isfile(os.path.join(dir, f))]
    def initialize_root(self):
        if os.path.exists(self.test_path):
            shutil.rmtree(self.test_path)
        os.mkdir(self.test_path)
    def parse_images(self):
        self.initialize_root()
        for each_file in os.listdir(self.train_path):
            if(each_file == '.DS_Store' or each_file == '.ds_store'): continue
            shutil.move(f'{self.train_path}/{each_file}/{each_file}_50.jpg', f'{self.test_path}')
            os.rename(f'{self.test_path}/{each_file}_50.jpg', f'{self.test_path}/{each_file}.jpg')
```

Figure 8 Test dataset script

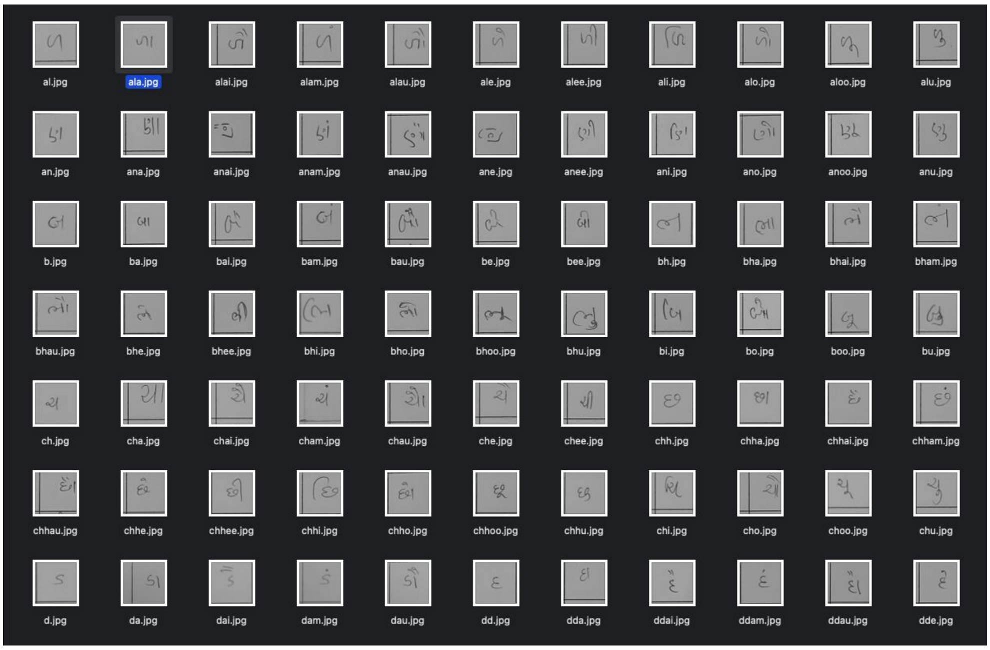


Figure 9 Images generated by testing script

3.1.4 Generate validation dataset

A python script to generate validation dataset and and its output is shown in screenshots. Following steps are followed in script in order to generate validation dataset. As output, It moves 748 images(two images from each class) from training dataset to validation dataset.

- Setting up directories
- Moving two image from taining to testing directory.

```python
class CharacterParser:
    def __init__(self, split=10):
        self.split = split
        self.train_path = "/dataset/train"
        self.validation_path = "/dataset/validation"
    def ignore_files(self, dir, files):
        return [f for f in files if os.path.isfile(os.path.join(dir, f))]
    def initialize_root(self):
        if os.path.exists(self.validation_path):
            shutil.rmtree(self.validation_path)
        shutil.copytree(self.train_path,self.validation_path,ignore=self.ignore_files)
    def parse_images(self):
        self.initialize_root()
        for each_file in os.listdir(self.train_path):
            if(each_file == '.DS_Store' or each_file == '.ds_store'): continue
            shutil.move(f'{self.train_path}/{each_file}/{each_file}_48.jpg', f'{self.validation_path}/{each_file}')
            shutil.move(f'{self.train_path}/{each_file}/{each_file}_49.jpg', f'{self.validation_path}/{each_file}')
```

Figure 10 Validation script

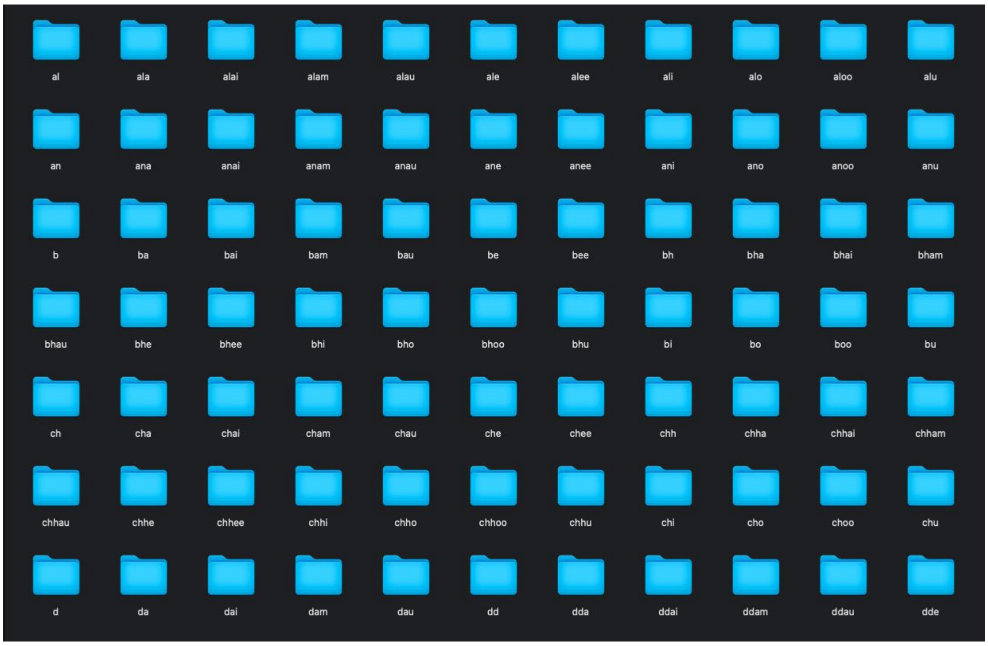


Figure 11 Directories generated validation script

Figure 12 Images in each directory

3.2 Modelling

Modelling involves training of three different convolution neural networks. For every training, there are few steps which are common for all as shown below.

- Importing dependent libraries

```
import tensorflow as tf
import os
import time
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Figure 13 Imported Libraries

- Initializing global variables

```
TRAIN_DATASET_PATH = '../../dataset/train'
VALIDATION_DATASET_PATH = '../../dataset/validation'
MODEL_SAVE_PATH = '../../saved_models/custom_model'
```

Figure 14 Global variables

- Initializing image generator for continuous flow of images to model while training

```
train_datagen = ImageDataGenerator(rescale = 1./255)
val_datagen = ImageDataGenerator(rescale = 1./255)
train_generator = train_datagen.flow_from_directory(train_path,
                                                    target_size = (INPUT_SHAPE[0], INPUT_SHAPE[1]),
                                                    batch_size = 64,
                                                    class_mode = 'sparse')
validation_generator = val_datagen.flow_from_directory(val_path,
                                                  target_size = (INPUT_SHAPE[0], INPUT_SHAPE[1]),
                                                  batch_size = 64,
                                                  class_mode = 'sparse')

Found 29172 images belonging to 375 classes.
Found 748 images belonging to 375 classes.
```

Figure 15 Initialize Image Generator

3.2.1 Custom CNN model

- Global configuration for model

```
INPUT_SHAPE = (30, 30, 1)
CLASSES = 374
```

Figure 15 Global Configurations

- Model structure

```
model = tf.keras.models.Sequential([

    # The first convolution
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=INPUT_SHAPE),
    tf.keras.layers.MaxPooling2D(2, 2),

    # The second convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # The third convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(CLASSES, activation='softmax')
])
model.compile(loss='categorical_crossentropy',
              optimizer = 'adam',
              metrics=['accuracy'])

model.summary()
```

Figure 16 Model Architecture

- Call-back to save model after each epoch

```
class callback(tf.keras.callbacks.Callback):
    def __init__(self):
        super().__init__()
        self.acc = 0
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy') > self.acc):
            model.save(MODEL_SAVE_PATH)
            self.acc = logs.get('accuracy')
        if(logs.get('accuracy')>0.99):
            print("\nReached 99% accuracy so cancelling training!")
            model.save(MODEL_SAVE_PATH)
            self.model.stop_training = True

```

Figure 17 Callback Method to Save Model

- Training the model

```
start_time = time.time()
history = model.fit(
      train_generator,
      epochs=35,
      verbose=1,
      shuffle=True,
      steps_per_epoch = None,
      validation_data = validation_generator,
      callbacks=[callback()])
print(f'Consumed seconds: {time.time() - start_time}')
```

Figure 18 Train CNN

- Saving history for evaluation

```
np.save('../../custom_model_history.npy',history.history)
history=np.load('../../custom_model_history.npy',allow_pickle='TRUE').item()
```

Figure 19 Saving History

3.2.2 Inception v3 model

- Global configuration for model

```
INPUT_SHAPE = (75, 75, 3)
CLASSES = 374
EPOCHS = 20
```

Figure 20 Global Configuration of InceptionV3

- Model structure

```
inception_layers=tf.keras.applications.inception_v3.InceptionV3(input_shape=INPUT_SHAPE, include_top=False, weights
inception_layers.trainable=False
flatten_layer = tf.keras.layers.Flatten()
dense_1 = tf.keras.layers.Dense(512, activation="relu")
prediction_layer = tf.keras.layers.Dense(CLASSES,activation='softmax')

model = tf.keras.Sequential([
  inception_layers,
  flatten_layer,
  dense_1,
  prediction_layer
])
model.compile(optimizer='Adam',
              loss=tf.keras.losses.sparse_categorical_crossentropy,
              metrics=["accuracy"])
model.summary()
```

Figure 21 Model Design

- Callback() to save model after each epoch

```
class callback(tf.keras.callbacks.Callback):
    def __init__(self):
        super().__init__()
        self.acc = 0
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy') > self.acc):
            model.save('../../saved_models/inception_v3')
            self.acc = logs.get('accuracy')
        if(logs.get('accuracy')>0.99):
            print("\nReached 99% accuracy so cancelling training!")
            model.save('../../saved_models/inception_v3')
            self.model.stop_training = True
```

Figure 22 Callback() to Save Model

- Training the model

```
start_time = time.time()
history = model.fit(
      train_generator,
      epochs=EPOCHS,
      verbose=1,
      shuffle=True,
      steps_per_epoch = None,
      validation_data = validation_generator,
      callbacks=[callback()])
print(f'Consumed seconds: {time.time() - start_time}')
```

Figure 23 Model Training

- Saving history for evaluation

```
np.save('../../inception_v3_history.npy',history.history)
history=np.load('../../inception_v3_history.npy',allow_pickle='TRUE').item()
```

Figure 24 Saving History

3.2.3 Xception model

- Global configuration for model

```
INPUT_SHAPE = (71, 71, 3)
CLASSES = 374
EPOCHS = 15
```

Figure 25 Global Configuration of Xception

- Model structure

```
xception_layers=tf.keras.applications.xception.Xception(input_shape=INPUT_SHAPE, include_top=False, weights='imager

xception_layers.trainable=False

flatten_layer = tf.keras.layers.Flatten()

dense_1 = tf.keras.layers.Dense(512, activation="relu")

prediction_layer = tf.keras.layers.Dense(CLASSES,activation='softmax')

model = tf.keras.Sequential([
  xception_layers,
  flatten_layer,
  dense_1,
  prediction_layer
])

model.summary()
```

Figure 26 Model Design

- Call-back to save model after each epoch

```
class callback(tf.keras.callbacks.Callback):
    def __init__(self):
        super().__init__()
        self.acc = 0
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy') > self.acc):
            model.save('../../saved_models/xception')
            self.acc = logs.get('accuracy')
        if(logs.get('accuracy')>0.99):
            print("\nReached 99% accuracy so cancelling training!")
            model.save('../../saved_models/xception')
            self.model.stop_training = True
```

Figure 27 Callback to Save Model

- Training the model

```
start_time = time.time()
history = model.fit(
      train_generator,
      epochs=EPOCHS,
      verbose=1,
      shuffle=True,
      steps_per_epoch = None,
      validation_data = validation_generator,
      callbacks=[callback()])
print(f'Consumed seconds: {time.time() - start_time}')
```

Figure 28 Training Xception

- Saving history for evaluation

```
np.save('../../xception_history.npy',history.history)
history=np.load('../../xception_history.npy',allow_pickle='TRUE').item()
```

Figure 29 Saving History

3.3 Testing and Evaluation

Evaluation includes visualization graphs and statistics.

- Graphs

```python
import matplotlib.pyplot as plt
import numpy as np

figure, axis = plt.subplots(1, 2)

loss_train = history['loss']
loss_val = history['val_loss']
epochs = range(1,len(history['loss']) + 1)
axis[0].plot(epochs, loss_train, 'g', label='Training loss')
axis[0].plot(epochs, loss_val, 'b', label='validation loss')
axis[0].set_title('T_loss vs V_loss')
axis[0].set_xlabel('Epochs')
axis[0].set_ylabel('Loss')
axis[0].legend()

acc_train = history['accuracy']
acc_val = history['val_accuracy']
epochs = range(1,len(history['accuracy']) + 1)
axis[1].plot(epochs, acc_train, 'g', label='Training accuracy')
axis[1].plot(epochs, acc_val, 'b', label='validation accuracy')
axis[1].set_title('T_acc vs V_acc')
axis[1].set_xlabel('Epochs')
axis[1].set_ylabel('Accuracy')
axis[1].legend()
plt.show()
```

Figure 30 Plotting Loss-Accuracy Graphs

- Testing dataset and calculating statistics

```python
import cv2
import os
import numpy as np
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusi

model = tf.keras.models.load_model(MODEL_SAVE_PATH)
d = train_generator.class_indices
d = {value:key for key, value in d.items()}
base_url = '../../dataset/test/'
prediction = []
lables = []
img_list = []
for name in os.listdir('../../dataset/test'):
    lables.append(name.split('.')[0])
    img_list.append(base_url + name)
for each in img_list:
    img = cv2.resize(cv2.imread(each, cv2.IMREAD_GRAYSCALE), (INPUT_SHAPE[0], INPUT_SHAPE[1]))
    img = img.reshape(1, INPUT_SHAPE[0],INPUT_SHAPE[1],INPUT_SHAPE[2]) / 255
    predicted_class = model.predict(img)
    max_val = (np.amax(predicted_class[0]))
    index = list(predicted_class[0]).index(max_val)
    prediction.append(d[index])

print("accuracy_score: ",accuracy_score(lables, prediction))
print("precision_score: ",precision_score(lables, prediction, average='micro'))
print("recall_score: ", recall_score(lables, prediction, average='micro'))
print("classification_report: \n", classification_report(lables, prediction))
print("f1_score: ", f1_score(lables, prediction, average='micro'))

```

Figure 31 Model Testing