Chapter 9

Project Implementation

9.1 Required Libraries

```
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Dense, Activation,Dropout,Conv2D, MaxP
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model, load_model, Sequential
import numpy as np
import pandas as pd
import shutil
import time
import cv2 as cv2
from tqdm import tqdm
from sklearn.model_selection import train_test_split
```

```
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import seaborn as sns
import datetime
from datetime import datetime
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report
from IPython.core.display import display, HTML
import logging
```

9.2 Used Dataset

9.2.1 IP102 Dataset

9.2.2 Dangerous farm insect Dataset

9.3 Basic Functionalities for working

9.3.1 Display images samples

```
def show_image_samples(gen ):
t_dict=gen.class_indices
classes=list(t_dict.keys())
images,labels=next(gen) # get a sample batch from the generator
plt.figure(figsize=(20, 20))
length=len(labels)
if length<25:   #show maximum of 25 images
    r=length
else:
    r=25
for i in range(r):
    plt.subplot(5, 5, i + 1)
    image=images[i]/255
```

```
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
        plt.title(class_name, color='blue', fontsize=12)
        plt.axis('off')
    plt.show()
```

9.3.2 Display Image

```
    def show_images(tdir):
    classlist=os.listdir(tdir)
    length=len(classlist)
    columns=5
    rows=int(np.ceil(length/columns))
    plt.figure(figsize=(20, rows * 4))
    for i, klass in enumerate(classlist):
        classpath=os.path.join(tdir, klass)
        imgpath=os.path.join(classpath, '1.jpg')
        img=plt.imread(imgpath)
        plt.subplot(rows, columns, i+1)
        plt.axis('off')
        plt.title(klass, color='blue', fontsize=12)
        plt.imshow(img)
```

9.3.3 Border Text

```
    def print_in_color(txt_msg,fore_tupple,back_tupple,):
    #prints the text_msg in the foreground color specified by fore_tupple w
    #text_msg is the text, fore_tupple is foregroud color tupple (r,g,b), b
    rf,gf,bf=fore_tupple
    rb,gb,bb=back_tupple
    msg='{0}' + txt_msg
```

```
mat='\33[38;2;' + str(rf) +';' + str(gf) + ';' + str(bf) + ';48;2;' + s
print(msg .format(mat), flush=True)
print('\33[0m', flush=True) # returns default print color to back to bl
return
```

9.4 Algorithm

9.4.1 Initialization

```
def __init__(self,model, base_model, patience,stop_patience, threshold,
factor, dwell, batches, initial_epoch,epochs, ask_epoch, csv_path=None)
    super(LRA, self).__init__()
    self.model=model
    self.base_model=base_model
    self.patience=patience
    self.stop_patience=stop_patience
    self.threshold=threshold
    self.factor=factor
    self.dwell=dwell
    self.batches=batches
    self.initial_epoch=initial_epoch
    self.epochs=epochs
    self.ask_epoch=ask_epoch
    self.ask_epoch_initial=ask_epoch
    self.csv_path=csv_path
    self.count=0
    self.stop_count=0
    self.best_epoch=1
    self.initial_lr=float(tf.keras.backend.get_value(
    model.optimizer.lr))
    self.highest_tracc=0.0
```

```
        self.lowest_vloss=np.inf
        self.best_weights=self.model.get_weights()
        self.initial_weights=self.model.get_weights()
        self.data_dict={}
        for key in ['epoch','tr loss','tr acc','vloss','vacc','current lr',
        'next lr','monitor','% improv','duration']:
            self.data_dict[key]=[]
```

9.4.2 Training the model

```
    def on_train_begin(self, logs=None):
        if self.base_model != None:
            status=base_model.trainable
            if status:
                msg=' initializing callback starting training with
                base_model trainable'
            else:
                msg='initializing callback starting training with
                base_model not trainable'
        else:
            msg='initialing callback and starting training'
        print_in_color (msg, (244, 252, 3), (55,65,80))
        msg='{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}{7:^10s}
        {8:10s}{9:^8s}'.format('Epoch', 'Loss', 'Accuracy', 'V_loss',
        'V_acc', 'LR', 'Next LR', 'Monitor','% Improv', 'Duration')
        print_in_color(msg, (244,252,3), (55,65,80))
        self.start_time= time.time()

    def on_train_end(self, logs=None):
        stop_time=time.time()
        tr_duration= stop_time- self.start_time
        hours = tr_duration // 3600
```

```
        minutes = (tr_duration - (hours * 3600)) // 60
        seconds = tr_duration - ((hours * 3600) + (minutes * 60))
        if self.csv_path !=None:
            df=pd.DataFrame.from_dict(self.data_dict)
            now = datetime.now()
            year = str(now.year)
            month=str(now.month)
            day=str(now.day)
            hour=str(now.hour)
            minute=str(now.minute)
            sec=str(now.second)
            label = month + '-'+ day + '-' + year + '-' + hour + '-' +
            minute + '-' + sec +'.csv'
            csv_path=self.csv_path + '-'+ label
            df.to_csv(csv_path, index=False)

    def on_train_batch_end(self, batch, logs=None):
        acc=logs.get('accuracy')* 100
        loss=logs.get('loss')
        msg='{0:20s}processing batch {1:4s} of {2:5s} accuracy= {3:8.3f}
loss: {4:8.5f}'.format(' ', str(batch), str(self.batches), acc, loss)
        print(msg, '\r', end='')
```

9.4.3 Epoch Training

```
    def on_epoch_begin(self,epoch, logs=None):
        self.now= time.time()

    def on_epoch_end(self, epoch, logs=None):
        later=time.time()
        duration=later-self.now
        lr=float(tf.keras.backend.get_value(self.model.optimizer.lr))
```

```
    current_lr=lr
    v_loss=logs.get('val_loss')
    acc=logs.get('accuracy')
    v_acc=logs.get('val_accuracy')
    loss=logs.get('loss')
    if acc < self.threshold:
        monitor='accuracy'
        if epoch ==0:
            pimprov=0.0
        else:
            pimprov= (acc-self.highest_tracc )*100/self.highest_tracc
        if acc>self.highest_tracc:
            self.highest_tracc=acc
            self.best_weights=self.model.get_weights()
            self.count=0
            self.stop_count=0
            if v_loss<self.lowest_vloss:
                self.lowest_vloss=v_loss
            color= (0,255,0)
            self.best_epoch=epoch + 1
        else:
            if self.count>=self.patience -1:
                color=(245, 170, 66)
                lr= lr* self.factor
                tf.keras.backend.set_value(self.model.optimizer.lr, lr)
                self.count=0
                self.stop_count=self.stop_count + 1
                self.count=0
                if self.dwell:
                    self.model.set_weights(self.best_weights)
```

```
                else:
                    if v_loss<self.lowest_vloss:
                        self.lowest_vloss=v_loss
            else:
                self.count=self.count +1
    else:
        monitor='val_loss'
        if epoch ==0:
            pimprov=0.0
        else:
            pimprov= (self.lowest_vloss- v_loss )*100/self.lowest_vloss
        if v_loss< self.lowest_vloss:
            self.lowest_vloss=v_loss
            self.best_weights=self.model.get_weights()
            self.count=0
            self.stop_count=0
            color=(0,255,0)
            self.best_epoch=epoch + 1
        else: # validation loss did not improve
            if self.count>=self.patience-1:
                color=(245, 170, 66)
                lr=lr * self.factor
                self.stop_count=self.stop_count + 1
                self.count=0
                tf.keras.backend.set_value(self.model.optimizer.lr, lr)
                if self.dwell:
                    self.model.set_weights(self.best_weights)
            else:
                self.count =self.count +1
            if acc>self.highest_tracc:
```

```
                    self.highest_tracc= acc
```

9.4.4 Working of Algorithm

```
msg=f'{str(epoch+1):^3s}/{str(self.epochs):4s} {loss:^9.3f}{acc*100:^9.3f}{
    print_in_color (msg,color, (55,65,80))
    key_list=['epoch','tr loss','tr acc','vloss','vacc','current lr','next
    val_list =[epoch + 1, loss, acc, v_loss, v_acc, current_lr, lr, monitor
    for key, value in zip(key_list, val_list):
        self.data_dict[key].append(value)

    if self.stop_count> self.stop_patience - 1:
        msg=f' training has been halted at epoch {epoch + 1} after {self.st
        print_in_color(msg, (0,255,255), (55,65,80))
        self.model.stop_training = True
    else:
        if self.ask_epoch !=None:
            if epoch + 1 >= self.ask_epoch:
                if base_model.trainable:
                    msg='enter H to halt training or an integer for number
                else:
                    msg='enter H to halt training ,F to fine tune model, or
                print_in_color(msg, (0,255,255), (55,65,80))
                ans=input('')
                if ans=='H' or ans=='h':
                    msg=f'training has been halted at epoch {epoch + 1} due
                    print_in_color(msg, (0,255,255), (55,65,80))
                    self.model.stop_training = True
                elif ans == 'T' or ans=='t':
                    if base_model.trainable:
                        msg='base_model is already set as trainable'
```

```
        else:
            msg='setting base_model as trainable for fine tunin
            self.base_model.trainable=True
        print_in_color(msg, (0, 255,255), (55,65,80))
        msg='Enter an integer for the number of epochs to run t
        print_in_color(msg, (0,2555,255), (55,65,80))
        ans=input()
        ans=int(ans)
        self.ask_epoch +=ans
        msg=f' training will continue until epoch ' + str(self.
        print_in_color(msg, (0, 255,255), (55,65,80))
        msg='{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}
        'V_loss','V_acc', 'LR', 'Next LR', 'Monitor','% Improv'
        print_in_color(msg, (244,252,3), (55,65,80))
        self.count=0
        self.stop_count=0
        self.ask_epoch = epoch + 1 + self.ask_epoch_initial
    else:
        ans=int(ans)
        self.ask_epoch +=ans
        msg=f' training will continue until epoch ' + str(self.
        print_in_color(msg, (0, 255,255), (55,65,80))
        msg='{0:^8s}{1:^10s}{2:^9s}{3:^9s}{4:^9s}{5:^9s}{6:^9s}
        'V_loss','V_acc', 'LR', 'Next LR', 'Monitor','% Improv'
        print_in_color(msg, (244,252,3), (55,65,80))
```

9.5 Function for Plotting the Accuracy and Loss

```
def tr_plot(tr_data, start_epoch):
#Plot the training and validation data
```

```
tacc=tr_data.history['accuracy']
tloss=tr_data.history['loss']
vacc=tr_data.history['val_accuracy']
vloss=tr_data.history['val_loss']
Epoch_count=len(tacc)+ start_epoch
Epochs=[]
for i in range (start_epoch ,Epoch_count):
    Epochs.append(i+1)
index_loss=np.argmin(vloss)#  this is the epoch with the lowest validat
val_lowest=vloss[index_loss]
index_acc=np.argmax(vacc)
acc_highest=vacc[index_acc]
plt.style.use('fivethirtyeight')
sc_label='best epoch= '+ str(index_loss+1 +start_epoch)
vc_label='best epoch= '+ str(index_acc + 1+ start_epoch)
fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(20,8))
axes[0].plot(Epochs,tloss, 'r', label='Training loss')
axes[0].plot(Epochs,vloss,'g',label='Validation loss' )
axes[0].scatter(index_loss+1 +start_epoch,val_lowest, s=150, c= 'blue',
axes[0].set_title('Training and Validation Loss')
axes[0].set_xlabel('Epochs')
axes[0].set_ylabel('Loss')
axes[0].legend()
axes[1].plot (Epochs,tacc,'r',label= 'Training Accuracy')
axes[1].plot (Epochs,vacc,'g',label= 'Validation Accuracy')
axes[1].scatter(index_acc+1 +start_epoch,acc_highest, s=150, c= 'blue',
axes[1].set_title('Training and Validation Accuracy')
axes[1].set_xlabel('Epochs')
axes[1].set_ylabel('Accuracy')
axes[1].legend()
```

```
plt.tight_layout
#plt.style.use('fivethirtyeight')
plt.show()
```

9.6 Functions for creating Confusion Matrix and Classification Report

```
def print_info( test_gen, preds, print_code, save_dir, subject ):
class_dict=test_gen.class_indices
labels= test_gen.labels
file_names= test_gen.filenames
error_list=[]
true_class=[]
pred_class=[]
prob_list=[]
new_dict={}
error_indices=[]
y_pred=[]
for key,value in class_dict.items():
    new_dict[value]=key
classes=list(new_dict.values())
errors=0
for i, p in enumerate(preds):
    pred_index=np.argmax(p)
    true_index=labels[i]
    if pred_index != true_index:
        error_list.append(file_names[i])
        true_class.append(new_dict[true_index])
        pred_class.append(new_dict[pred_index])
        prob_list.append(p[pred_index])
```

```
            error_indices.append(true_index)
            errors=errors + 1
        y_pred.append(pred_index)
    tests=len(preds)
    acc= (1-errors/tests) *100
    msg= f'There were {errors} errors in {tests} test cases Model accuracy=
    print_in_color(msg,(0,255,255),(55,65,80))
    if print_code !=0:
        if errors>0:
            if print_code>errors:
                r=errors
            else:
                r=print_code
            msg='{0:^28s}{1:^28s}{2:^28s}{3:^16s}'.format('Filename', 'Pred
            print_in_color(msg, (0,255,0),(55,65,80))
            for i in range(r):
                split1=os.path.split(error_list[i])
                split2=os.path.split(split1[0])
                fname=split2[1] + '/' + split1[1]
                msg='{0:^28s}{1:^28s}{2:^28s}{3:4s}{4:^6.4f}'.format(fname,
                print_in_color(msg, (255,255,255), (55,65,60))
                #print(error_list[i]  , pred_class[i], true_class[i], prob
        else:
            msg='With accuracy of 100 % there are no errors to print'
            print_in_color(msg, (0,255,0),(55,65,80))
    if errors>0:
        plot_bar=[]
        plot_class=[]
        for  key, value in new_dict.items():
            count=error_indices.count(key)
```

```
        if count!=0:
            plot_bar.append(count)
            plot_class.append(value)
    fig=plt.figure()
    fig.set_figheight(len(plot_class)/3)
    fig.set_figwidth(10)
    plt.style.use('fivethirtyeight')
    for i in range(0, len(plot_class)):
        c=plot_class[i]
        x=plot_bar[i]
        plt.barh(c, x, )
        plt.title( ' Errors by Class on Test Set')
y_true= np.array(labels)
y_pred=np.array(y_pred)
if len(classes)<= 30:
    # create a confusion matrix
    cm = confusion_matrix(y_true, y_pred )
    length=len(classes)
    if length<8:
        fig_width=8
        fig_height=8
    else:
        fig_width= int(length * .5)
        fig_height= int(length * .5)
    plt.figure(figsize=(fig_width, fig_height))
    sns.heatmap(cm, annot=True, vmin=0, fmt='g', cmap='Blues', cbar=Fal
    plt.xticks(np.arange(length)+.5, classes, rotation= 90)
    plt.yticks(np.arange(length)+.5, classes, rotation=0)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
```

```
        plt.title("Confusion Matrix")
        plt.show()
    clr = classification_report(y_true, y_pred, target_names=classes, digit
    print("Classification Report:\n----------------------------------------\n", clr)
    return acc/100
```

9.7 Pre-process of Data

```
    def preprocess (sdir, trsplit, vsplit):
    categories=['train', 'test', 'val']
    filepaths=[]
    labels=[]
    for category in categories:
        catpath=os.path.join(sdir, category)
        classlist=os.listdir(catpath)
        for klass in classlist:
            classpath=os.path.join(catpath,klass)
            flist=os.listdir(classpath)
            for f in flist:
                fpath=os.path.join(classpath,f)
                filepaths.append(fpath)
                labels.append(klass)
    Fseries=pd.Series(filepaths, name='filepaths')
    Lseries=pd.Series(labels, name='labels')
    df=pd.concat([Fseries, Lseries], axis=1)
    train_df, dummy_df=train_test_split(df, train_size=trsplit, shuffle=Tru
    dsplit=vsplit/(1-trsplit)
    valid_df, test_df=train_test_split(dummy_df, train_size=dsplit, shuffle
    print('train_df length: ', len(train_df), '  test_df length: ',len(test
valid_df length: ', len(valid_df))
```

```
trcount=len(train_df['labels'].unique())
tecount=len(test_df['labels'].unique())
vcount=len(valid_df['labels'].unique())
if trcount < tecount :
    msg='** WARNING ** number of classes in training set is less than t
    print_in_color(msg, (255,0,0), (55,65,80))
    msg='This will throw an error in either model.evaluate or model.pre
    print_in_color(msg, (255,0,0), (55,65,80))
if trcount != vcount:
    msg='** WARNING ** number of classes in training set not equal to n
    print_in_color(msg, (255,0,0), (55,65,80))
    msg=' this will throw an error in model.fit'
    print_in_color(msg, (255,0,0), (55,65,80))
    print ('train df class count: ', trcount, 'test df class count: ',
    ans=input('Enter C to continue execution or H to halt execution')
    if ans =='H' or ans == 'h':
        print_in_color('Halting Execution', (255,0,0), (55,65,80))
        import sys
        sys.exit('program halted by user')
msg='Below is image count per class to evaluate train_df balance'
print_in_color(msg, (0,255,255),(55,65,80))
print(list(train_df['labels'].value_counts()))
return train_df, test_df, valid_df
```

9.8 Balancing of data

The train data set is not balanced. To balance it use the balance function defined below. First limit maximum samples in a class to `max_samples`=300. Then for classes with less than 300 samples create augmented images and store the images in the aug directory. Then merge the current `train_dfwith` the `aud_df` to create a balanced `train_df`.

```
def balance(train_df,max_samples, min_samples, column, working_dir,
image_size):
train_df=train_df.copy()
train_df=trim (train_df, max_samples, min_samples, column)
# make directories to store augmented images
aug_dir=os.path.join(working_dir, 'aug')
if os.path.isdir(aug_dir):
    shutil.rmtree(aug_dir)
os.mkdir(aug_dir)
for label in train_df['labels'].unique():
    dir_path=os.path.join(aug_dir,label)
    os.mkdir(dir_path)
# create and store the augmented images
total=0
gen=ImageDataGenerator(horizontal_flip=True,  rotation_range=20,
width_shift_range=.2, height_shift_range=.2, zoom_range=.2)
groups=train_df.groupby('labels')
for label in train_df['labels'].unique():
    group=groups.get_group(label)
    sample_count=len(group)
    if sample_count< max_samples:
        aug_img_count=0
        delta=max_samples-sample_count
        target_dir=os.path.join(aug_dir, label)
        aug_gen=gen.flow_from_dataframe( group,  x_col='filepaths', y_c
        save_to_dir=target_dir, save_prefix='aug-', color_mode='rgb',
        save_format='jpg')
        while aug_img_count<delta:
            images=next(aug_gen)
            aug_img_count += len(images)
```

```
            total +=aug_img_count
    print('Total Augmented images created= ', total)
    # create aug_df and merge with train_df to create composite training se
    if total>0:
        aug_fpaths=[]
        aug_labels=[]
        classlist=os.listdir(aug_dir)
        for klass in classlist:
            classpath=os.path.join(aug_dir, klass)
            flist=os.listdir(classpath)
            for f in flist:
                fpath=os.path.join(classpath,f)
                aug_fpaths.append(fpath)
                aug_labels.append(klass)
        Fseries=pd.Series(aug_fpaths, name='filepaths')
        Lseries=pd.Series(aug_labels, name='labels')
        aug_df=pd.concat([Fseries, Lseries], axis=1)
        train_df=pd.concat([train_df,aug_df], axis=0).reset_index(drop=True

    print (list(train_df['labels'].value_counts()) )
    return train_df
```

Now To balance the data, we need to call the function mentioned above.

```
max_samples=300
min_samples= 0
column='labels'
working_dir = r'./'
img_size=(200,200)
```

```
train_df=balance(train_df, max_samples, min_samples, column, working_dir, i
```

9.9 Train, Test and Validation generators

```
channels=3
batch_size=30
img_shape=(img_size[0], img_size[1], channels)
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length
test_steps=int(length/test_batch_size)
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps)
def scalar(img):
    return img
trgen=ImageDataGenerator(preprocessing_function=scalar, horizontal_flip=Tru
tvgen=ImageDataGenerator(preprocessing_function=scalar)
msg='                              for the train generator'
print(msg, '\r', end='')
train_gen=trgen.flow_from_dataframe( train_df, x_col='filepaths', y_col='la
color_mode='rgb', shuffle=True, batch_size=batch_size)
msg='                              for the test generator'
print(msg, '\r', end='')
test_gen=tvgen.flow_from_dataframe( test_df, x_col='filepaths', y_col='labe
color_mode='rgb', shuffle=False, batch_size=test_batch_size)
msg='                              for the validation generator'
print(msg, '\r', end='')
valid_gen=tvgen.flow_from_dataframe( valid_df, x_col='filepaths', y_col='la
color_mode='rgb', shuffle=True, batch_size=batch_size)
classes=list(train_gen.class_indices.keys())
class_count=len(classes)
train_steps=int(np.ceil(len(train_gen.labels)/batch_size))
```

```
labels=test_gen.labels
```

```
test batch size:  80   test steps:  1
Found 4488 validated image filenames belonging to 15 classes.   for the train generator
Found 80 validated image filenames belonging to 15 classes.    for the test generator
Found 80 validated image filenames belonging to 15 classes.    for the validation generator
```

9.10 Create and Compile the Model

```
    model_name='EfficientNetB4'
base_model=tf.keras.applications.efficientnet.EfficientNetB4(include_top=Fa
x=base_model.output
x=keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x
x = Dense(512, kernel_regularizer = regularizers.l2(l = 0.016),activity_reg
            bias_regularizer=regularizers.l1(0.006) ,activation='relu')
x=Dropout(rate=.45, seed=123)(x)
output=Dense(class_count , activation='softmax')(x)
model=Model(inputs=base_model.input , outputs=output)
model.compile(Adamax(learning_rate=.001), loss='categorical_crossentropy',
```

9.11 Instantiate the Custom Callback and train the model

```
epochs =40
patience= 1
stop_patience =3
threshold=.9
factor=.5
dwell=True
freeze=False
ask_epoch=10
batches=train_steps
```

```
csv_path=os.path.join(working_dir,'my_csv')
callbacks=[LRA(model=model,base_model= base_model,patience=patience,stop_pa
factor=factor,dwell=dwell, batches=batches,initial_epoch=0,epochs=epochs, a

history=model.fit(x=train_gen, epochs=epochs, verbose=0,
callbacks=callbacks, validation_data=valid_gen,
validation_steps=None, shuffle=False, initial_epoch=0)
```

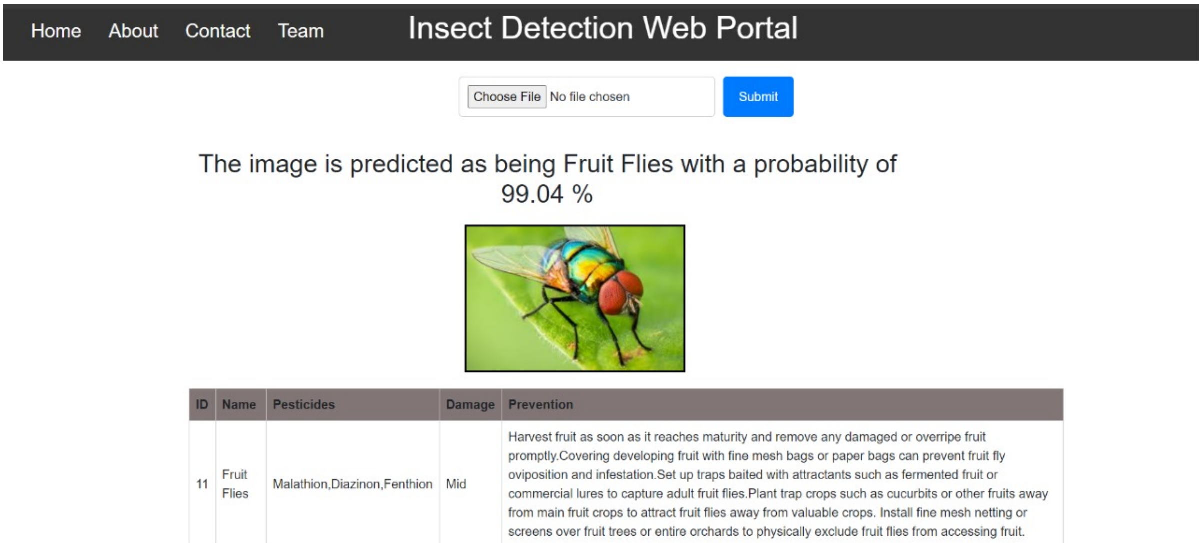
```
 1 /40    13.217   58.155  10.83506  71.250   0.00100  0.00100  accuracy     0.00    247.32
 2 /40     8.200   88.547   7.19621  72.500   0.00100  0.00100  accuracy    52.26    227.99
 3 /40     5.357   95.811   4.90405  73.750   0.00100  0.00100  val_loss    31.85    228.34
 4 /40     3.518   97.393   3.44893  72.500   0.00100  0.00100  val_loss    29.67    228.08
 5 /40     2.298   98.084   2.55360  71.250   0.00100  0.00100  val_loss    25.96    230.05
 6 /40     1.526   98.329   1.92741  77.500   0.00100  0.00100  val_loss    24.52    228.28
 7 /40     1.044   98.730   1.56948  73.750   0.00100  0.00100  val_loss    18.57    228.05
 8 /40     0.754   98.730   1.37530  78.750   0.00100  0.00100  val_loss    12.37    228.32
 9 /40     0.593   98.418   1.28600  73.750   0.00100  0.00100  val_loss     6.49    228.61
10 /40     0.495   98.663   1.22317  75.000   0.00100  0.00100  val_loss     4.89    228.65
enter H to halt training or an integer for number of epochs to run then ask again
10
 training will continue until epoch 20
...
Training is completed - model is set with weights from epoch 15
training elapsed time was 1.0 hours,  9.0 minutes, 18.55 seconds)
```

Chapter 10

Experimental Results

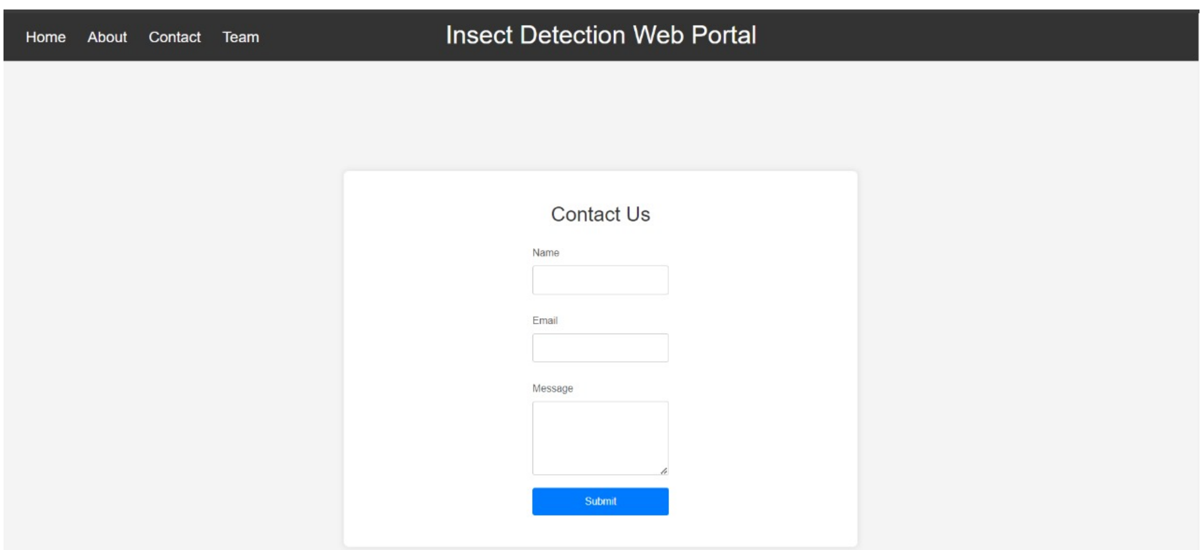
10.1 Web Portal

10.2 Results

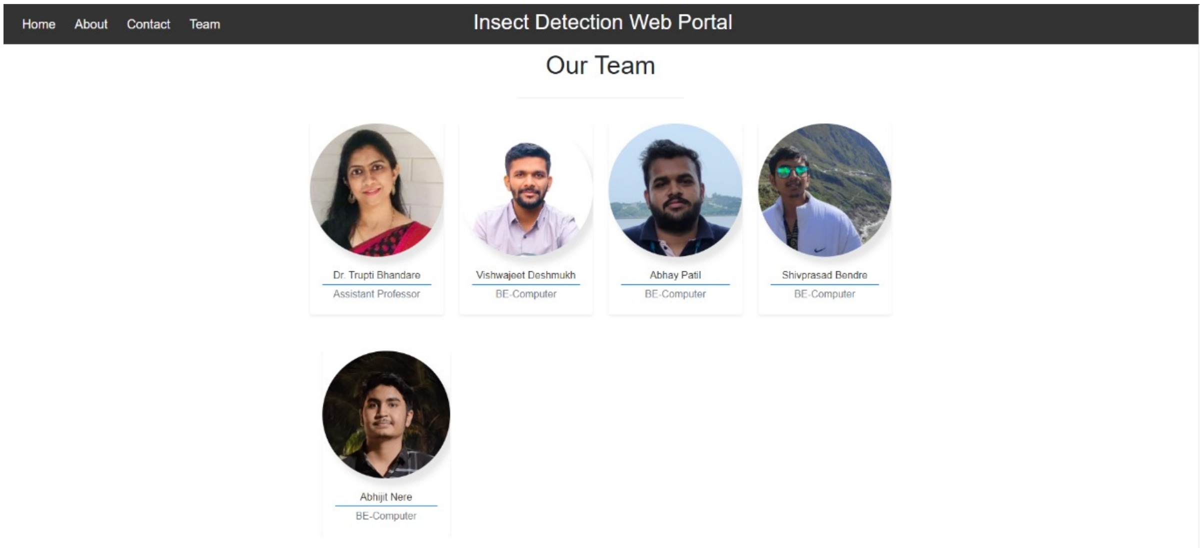


ID	Name	Pesticides	Damage	Prevention
11	Fruit Flies	Malathion,Diazinon,Fenthion	Mid	Harvest fruit as soon as it reaches maturity and remove any damaged or overripe fruit promptly.Covering developing fruit with fine mesh bags or paper bags can prevent fruit fly oviposition and infestation.Set up traps baited with attractants such as fermented fruit or commercial lures to capture adult fruit flies.Plant trap crops such as cucurbits or other fruits away from main fruit crops to attract fruit flies away from valuable crops. Install fine mesh netting or screens over fruit trees or entire orchards to physically exclude fruit flies from accessing fruit.

10.3 Contact Us



10.4 Our Team


Home
About
Contact
Team
Insect Detection Web Portal
Our Team
Dr. Trupti Bhandare
Assistant Professor
Vishwajeet Deshmukh
BE-Computer
Abhay Patil
BE-Computer
Shivprasad Bendre
BE-Computer
Abhijit Nere
BE-Computer

Chapter 11

Conclusion

Deep learning and machine learning algorithms were employed to develop robust insect detection models. These models can differentiate between various insect species and identify pest hotspots. Our solution promotes sustainable farming by reducing pesticide use and optimizing resource allocation. The project exemplifies the potential of interdisciplinary collaboration and technology-driven solutions in agriculture. Future efforts will involve refining the system based on feedback from farmers and stakeholders to cater to specific regional and crop needs. This project signifies the transformative power of technology in agriculture, paving the way for a more sustainable and resilient future in farming practices.