Task 1

Here is the full executable code for the solution to this task. I just did not have enough time to run it and present results. (I also came across some warning regarding the data-generator which needs furthur debugging)

In [ ]:
```
############
# Deep weeds is currently only available in tfds-nightly
############
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_datasets as tfds
from PIL import Image

# Loads deep weeds data.
# Can subsample the data with <fraction> parameter to prevent exhausting memory
def load_deepweeds(fraction=1.0):
  train_x = []
  train_y = []
  ds = tfds.load('deep_weeds', split='train', as_supervised=True)
  for image,label in tfds.as_numpy(ds):
    if np.random.uniform(0,1) < fraction:
      train_x.append(np.array(Image.fromarray(image).resize((224,224))))
      train_y.append(label)
  train_x = np.array(train_x)/255.0
  train_y = tf.keras.utils.to_categorical(np.array(train_y))
  return train_x, train_y


train_x, train_y = load_deepweeds(fraction=0.2)
```

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.models import Model
import numpy as np
```

```
import csv
from keras import backend as K
from sklearn.metrics import classification_report
from skimage.io import imread
from skimage.transform import resize
from keras.layers import Dense, GlobalAveragePooling2D, Flatten
from keras.models import Model
from keras.layers import Input, Conv2D, Conv2DTranspose, concatenate, Dropout, MaxPooling2D

# Global paths
OUTPUT_DIRECTORY = "./outputs/"
LABEL_DIRECTORY = "./labels/"
MODEL_DIRECTORY = "./models/"
IMG_DIRECTORY = "./images/"

# Global variables
RAW_IMG_SIZE = (256, 256)
IMG_SIZE = (224, 224)
INPUT_SHAPE = (IMG_SIZE[0], IMG_SIZE[1], 3)
MAX_EPOCH = 20
BATCH_SIZE = 32
FOLDS = 5
STOPPING_PATIENCE = 32
LR_PATIENCE = 16
INITIAL_LR = 0.0001
CLASSES = [0, 1, 2, 3, 4, 5, 6, 7, 8]
CLASS_NAMES = ['Chinee Apple',
               'Lantana',
               'Parkinsonia',
               'Parthenium',
               'Prickly Acacia',
               'Rubber Vine',
               'Siam Weed',
               'Snake Weed',
               'Negatives']



def crop(img, size):
    """
    Crop the image concentrically to the desired size.
```

```
    :param img: Input image
    :param size: Required crop image size
    :return:
    """
    (h, w, c) = img.shape
    x = int((w - size[0]) / 2)
    y = int((h - size[1]) / 2)
    return img[y:(y + size[1]), x:(x + size[0]), :]


def crop_generator(batches, size):
    """
    Take as input a Keras ImageGen (Iterator) and generate random
    crops from the image batches generated by the original iterator
    :param batches: Batches of images to be cropped
    :param size: Size to be cropped to
    :return:
    """
    while True:
        batch_x, batch_y = next(batches)
        (b, h, w, c) = batch_x.shape
        batch_crops = np.zeros((b, size[0], size[1], c))
        for i in range(b):
            batch_crops[i] = crop(batch_x[i], (size[0], size[1]))
        yield (batch_crops, batch_y)


def get_data():
    train_dataframe = None
    val_dataframe = None
    test_dataframe = None
    # K fold cross validation, saving outputs for each fold
    for k in range(FOLDS):

        # Prepare training, validation and testing labels for kth fold
        train_label_file = "{}train_subset{}.csv".format(LABEL_DIRECTORY, k)
        val_label_file = "{}val_subset{}.csv".format(LABEL_DIRECTORY, k)
        test_label_file = "{}test_subset{}.csv".format(LABEL_DIRECTORY, k)
        if k == 0:
            train_dataframe = pd.read_csv(train_label_file)
        else:
            train_dataframe = pd.concat([train_dataframe, pd.read_csv(train_label_file)], axis=0, ignore_index=True)
```

```
        if k == 0:
            val_dataframe = pd.read_csv(val_label_file)
        else:
            val_dataframe = pd.concat([val_dataframe, pd.read_csv(val_label_file)], axis=0, ignore_index=True)
        if k == 0:
            test_dataframe = pd.read_csv(test_label_file)
        else:
            test_dataframe = pd.concat([test_dataframe, pd.read_csv(test_label_file)], axis=0, ignore_index=True)

    train_image_count = train_dataframe.shape[0]
    val_image_count = val_dataframe.shape[0]
    test_image_count = test_dataframe.shape[0]

    train_dataframe['Label'] = train_dataframe['Label'].astype(str)
    val_dataframe['Label'] = val_dataframe['Label'].astype(str)
    test_dataframe['Label'] = test_dataframe['Label'].astype(str)

    train_data_generator = ImageDataGenerator(
        rescale=1. / 255,
        fill_mode="constant",
        shear_range=0.2,
        zoom_range=(0.5, 1),
        horizontal_flip=True,
        rotation_range=360,
        channel_shift_range=25,
        brightness_range=(0.75, 1.25)).flow_from_dataframe(
        dataframe=train_dataframe,
        directory=IMG_DIRECTORY,
        x_col='Filename',
        y_col='Label',
        target_size=RAW_IMG_SIZE,
        batch_size=BATCH_SIZE,
        has_ext=True,
        classes=CLASSES,
        class_mode='categorical')

    # Load validation images in batches from directory and apply rescaling
    val_data_generator =  ImageDataGenerator(
        rescale=1. / 255,
        fill_mode="constant",
        shear_range=0.2,
        zoom_range=(0.5, 1),
```

```
        horizontal_flip=True,
        rotation_range=360,
        channel_shift_range=25,
        brightness_range=(0.75, 1.25)).flow_from_dataframe(
        val_dataframe,
        IMG_DIRECTORY,
        x_col="Filename",
        y_col="Label",
        target_size=RAW_IMG_SIZE,
        batch_size=BATCH_SIZE,
        classes=CLASSES,
        class_mode='categorical',
        has_ext=True)

    # Load test images in batches from directory and apply rescaling
    test_data_generator = ImageDataGenerator(rescale=1. / 255).flow_from_dataframe(
        test_dataframe,
        IMG_DIRECTORY,
        x_col="Filename",
        y_col="Label",
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        has_ext=True,
        shuffle=False,
        classes=CLASSES,
        class_mode='categorical')

    # Crop augmented images from 256x256 to 224x224
    train_data_generator = crop_generator(train_data_generator, IMG_SIZE)
    val_data_generator = crop_generator(val_data_generator, IMG_SIZE)


    return train_data_generator, val_data_generator, test_data_generator , train_image_count,\
        val_image_count, test_image_count


def get_left_UNET():
    inputs = tf.keras.Input(shape=INPUT_SHAPE)

    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
```

```
    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool3)
    conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    # Bottom Layer
    conv5 = Conv2D(1024, (3, 3), activation='relu', padding='same')(pool4)
    conv5 = Conv2D(1024, (3, 3), activation='relu', padding='same', name="last_conv_layer")(conv5)
    x = GlobalAveragePooling2D(name='avg_pool')(conv5)
    x = Dense(1024, activation='relu', name="fully_connected")(x)
    outputs = Dense(len(CLASSES), activation='softmax', name='output')(x)
    model = Model(inputs=inputs, outputs=outputs)

    return model


def train_model(train_data_generator, val_data_generator, test_data_generator , train_image_count,\
        val_image_count, test_image_count, model, show_results=False, evaluate=False):
    model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=INITIAL_LR), metrics=['accuracy'])
    history = model.fit(
                x=train_data_generator,
                batch_size = BATCH_SIZE,
                steps_per_epoch=train_image_count // BATCH_SIZE,
                epochs=MAX_EPOCH,
                validation_data=val_data_generator,
                validation_steps=val_image_count // BATCH_SIZE,
                shuffle=False)
    if show_results:
        plt.plot(history.history['loss'], label='Training Loss')
        plt.plot(history.history['val_loss'], label='Validation Loss')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.title('Training and Validation Losses')
        plt.legend()
```

```
        plt.show()
    if evaluate:
        predictions = model.predict_generator(test_data_generator, test_image_count // BATCH_SIZE + 1)
        y_true = test_data_generator.classes
        y_pred = np.argmax(predictions, axis=1)
        y_pred[np.max(predictions, axis=1) < 1 / 9] = 8  # Assign predictions worse than random guess to negative cla
        report = classification_report(y_true, y_pred, labels=CLASSES, target_names=CLASS_NAMES)
        print(report)
```

In [ ]:
```
train_data_generator, val_data_generator, test_data_generator , train_image_count,\
        val_image_count, test_image_count = get_data()

model = get_left_UNET()
```

```
Found 0 validated image filenames belonging to 9 classes.
h:\Uni\WiSe 2024\ML LAB\ml_lab_venv\Lib\site-packages\keras\src\preprocessing\image.py:1137: UserWarning: Found 52525
invalid image filename(s) in x_col="Filename". These filename(s) will be ignored.
  warnings.warn(
Found 0 validated image filenames belonging to 9 classes.
h:\Uni\WiSe 2024\ML LAB\ml_lab_venv\Lib\site-packages\keras\src\preprocessing\image.py:1137: UserWarning: Found 17511
invalid image filename(s) in x_col="Filename". These filename(s) will be ignored.
  warnings.warn(
Found 0 validated image filenames belonging to 9 classes.
h:\Uni\WiSe 2024\ML LAB\ml_lab_venv\Lib\site-packages\keras\src\preprocessing\image.py:1137: UserWarning: Found 17509
invalid image filename(s) in x_col="Filename". These filename(s) will be ignored.
  warnings.warn(
```

In [ ]:
```
train_model(train_data_generator, val_data_generator, test_data_generator , train_image_count,\
        val_image_count, test_image_count, model, show_results=True)
```

In [ ]:
```
def original_Unet():
    inputs = tf.keras.Input(shape=(224, 224, 3))

    # Encoder
    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
```

```
    conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool3)
    conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(conv4)

    # Bottom Layer
    conv5 = Conv2D(1024, (3, 3), activation='relu', padding='same')(pool4)
    conv5 = Conv2D(1024, (3, 3), activation='relu', padding='same', name="last_conv_layer")(conv5)

    # Decoder
    up6 = Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same')(conv5)
    merge6 = concatenate([conv4, up6], axis=-1)
    conv6 = Conv2D(512, (3, 3), activation='relu', padding='same')(merge6)
    conv6 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv6)

    up7 = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(conv6)
    merge7 = concatenate([conv3, up7], axis=-1)
    conv7 = Conv2D(256, (3, 3), activation='relu', padding='same')(merge7)
    conv7 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv7)

    up8 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(conv7)
    merge8 = concatenate([conv2, up8], axis=-1)
    conv8 = Conv2D(128, (3, 3), activation='relu', padding='same')(merge8)
    conv8 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv8)

    up9 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(conv8)
    merge9 = concatenate([conv1, up9], axis=-1)
    conv9 = Conv2D(64, (3, 3), activation='relu', padding='same')(merge9)
    conv9 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv9)

    # Output Layer
    outputs = Conv2D(3, (1, 1), activation='softmax')(conv9)

    model = Model(inputs=inputs, outputs=outputs)
    return model
```

In [ ]:

```
def load_segmentation_data():
    data = np.load('segmentation_data.npz')
```

```
    train_x = data['train_x']
    train_y = data['train_y']
    test_x = data['test_x']
    test_y = data['test_y']
    return train_x, train_y, test_x, test_y


train_x, train_y, test_x, test_y = load_segmentation_data()
```

In [ ]:
```
segment_model = original_Unet()
for layer_source, layer_dest in zip(model.layers, segment_model.layers):
    layer_dest.set_weights(layer_source.get_weights())
    layer_dest.trainable = False
    if layer_source.name == "last_conv_layer":
        break

segment_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
segment_model.fit(train_x, train_y, epochs=20, batch_size=8)

for layer in segment_model.layers:
    layer_dest.trainable = True

segment_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
segment_model.fit(train_x, train_y, epochs=20, batch_size=8)
```

In [ ]:
```
# Evaluate on test data
test_loss, test_accuracy = model.evaluate(test_x, test_y)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

I think that it would perform better