Appendix C: Script for splitting dataset into train and test

```
import splitfolders

input_folder = "path_to_input"

output = "path_to_output”

splitfolders.ratio(input_folder, output=output, seed=42, ratio=(0.8,0.2))
```

Appendix D Image Recognition Model

```
# Extacting data
import numpy as np
import zipfile

# Unzip the downloaded file
zip_ref = zipfile.ZipFile("/content/drive/MyDrive/Multiphase_project/dataset_2.zip")
zip_ref.extractall()
zip_ref.close()
# How many images in each folder?
import os

# Walk through 10 percent data directory and list number of files
for dirpath, dirnames, filenames in os.walk("dataset_2"):
  print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
# Create training and test directory paths
train_dir_2 = "dataset_2/train"
test_dir_2 = "dataset_2/test"
# Setup data inputs
import tensorflow as tf
IMG_SIZE = (224, 224)
train_data_4 = tf.keras.utils.image_dataset_from_directory(train_dir_2,
                                                            label_mode="categorical",
                                                           image_size=IMG_SIZE,
                                                            batch_size=2)
test_data_4 = tf.keras.utils.image_dataset_from_directory(test_dir_2,
                                                           label_mode="categorical",
                                                           image_size=IMG_SIZE,
                                                           shuffle=False,
                                                        batch_size=2)
class_names = test_data_4.class_names
class_names
# Create early stopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_accuracy",
                                                  patience=5)
# Creating data augmentation layer
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.models import Sequential

# Setup data augmentation
data_augmentation = Sequential([
    preprocessing.RandomFlip("horizontal"),
    preprocessing.RandomRotation(0.2),
    preprocessing.RandomZoom(0.2),
        preprocessing.RandomContrast(0.2),
], name ="data_augmentation")
# Function to plot loss cureves
import matplotlib.pyplot as plt

# Plot the validation and training curves
def plot_loss_curves(history):
  """
  Returns seperate loss curves for training and validation metrics.

  Args:
    history: Tensorflow History object.

  Returns:
    Plot of training/validation loss and accuracy metrics
  """
  loss = history.history["loss"]
  val_loss = history.history["val_loss"]

  accuracy = history.history["accuracy"]
  val_accuracy= history.history["val_accuracy"]

  epochs = range(len(history.history["loss"]))

  # Plot loss
  plt.plot(epochs, loss, label="training_loss")
  plt.plot(epochs, val_loss, label="val_loss")
  plt.title("Loss")
  plt.xlabel("Epochs")
  plt.legend()

  # Plot accuracy
  plt.figure()
  plt.plot(epochs, accuracy, label="training_accuracy")
  plt.plot(epochs, val_accuracy, label="val_accuracy")
  plt.title("Accuracy")
  plt.xlabel("Epochs")
  plt.legend()
# Create a confusion metrics

import itertools
from sklearn.metrics import confusion_matrix


def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(10, 10), text_size=15):


  # Create the confusion matrix
  cm = confusion_matrix(y_true, y_preds)
  cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
  n_classes = cm.shape[0]

  # Let's prettify it
  fig, ax = plt.subplots(figsize=figsize)
  # Create a matrix plot
  cax = ax.matshow(cm, cmap=plt.cm.Blues)
  fig.colorbar(cax)


  # Set labels to be claseses
  if classes:
    labels = classes
  else:
    labels = np.arange(cm.shape[0])

  # Label the axes
  ax.set(title="Confusion Matrix",
         xlabel="Predicted label",
         ylabel="True label",
         xticks=np.arange(n_classes),
         yticks=np.arange(n_classes),
         xticklabels=labels,
         yticklabels=labels)

  # Set x-axis labels to bottom
  ax.xaxis.set_label_position("bottom")
  ax.xaxis.tick_bottom()

  # Adjust label size
  ax.xaxis.label.set_size(text_size)
  ax.yaxis.label.set_size(text_size)
  ax.title.set_size(text_size)

  # Set threshold for different colors
  threshold = (cm.max() + cm.min()) / 2.

  # Plot the text on each cell
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, f"{cm[i, j]} ({cm_norm[i, j]*100:.1f}%)",
             horizontalalignment="center",
             color="white" if cm[i, j] > threshold else "black",
             size=text_size)
# Using EfficientNetB2
# Setup the base model
base_model_22 = tf.keras.applications.EfficientNetB2(include_top=False)
base_model_22.trainable = False

# Setup model architecture with trainable top layers
inputs = layers.Input(shape=(224, 224, 3), name="input_layer")
x = data_augmentation(inputs)
x = base_model_22(x, training=False)
x = layers.GlobalAveragePooling2D(name="global_avg_pool_layer")(x)
outputs = layers.Dense(5, activation="softmax", name="output_layer")(x)
model_22 = tf.keras.Model(inputs, outputs)

# Compile
model_22.compile(loss="categorical_crossentropy",
                 optimizer=tf.optimizers.Adam(),
                 metrics=["accuracy"])

# Fit
history_model_22 = model_22.fit(train_data_4,
                     epochs=100,
                     validation_data=test_data_4,
                     validation_steps=len(test_data_4),
                     callbacks=[early_stopping])


# Making predictions with our best model so far
y_probs = model_22.predict(test_data_4)

# View the first predcitons
y_probs[:5]
# unbatching test data batch to get y_labels
y_labels = []
for images, labels in test_data_4.unbatch():
  y_labels.append(labels.numpy().argmax())
y_labels[:10]

# make a good confusion matrix
make_confusion_matrix(y_true=y_labels,
                      y_pred=y_preds,
                      classes=class_names,
                      figsize=(15, 15),
                      text_size=10)
# Saving model to Goodgle Drive
save_dir = "/content/drive/MyDrive/Multiphase_project/"
model_22.save(save_dir)
loaded_model.evaluate(test_data_4)
# Making predictions with our best model so far
y_probs_loaded = loaded_model.predict(test_data_4)

# View the first predcitons
y_probs_loaded[:5]
y_preds_loaded = y_probs_loaded.argmax(axis=1)
y_preds_loaded
# Get all of the image file paths in the test dataset
filepaths = []
for filepath in test_data_4.list_files("/content/dataset_2/test/*/*.png",
                                      shuffle=False):
  filepaths.append(filepath.numpy())
filepaths[:10]
#Create a DataFrame of different parameters for each of the test images
import pandas as pd
pred_df = pd.DataFrame({"img_path": filepaths,
                        "y_true": y_labels,
                        "y_pred": y_preds_loaded,
                        "pred_conf": y_probs_loaded.max(axis=1),
                        "y_true_classname": [class_names[i] for i in y_labels],
                        "y_pred_classname": [class_names[i] for i in y_preds_loaded]})
pred_df
# Saving file
pred_df.to_excel("pred_df.xlsx", index=False)
```