PLANT classificator

Mattia Spazzoli

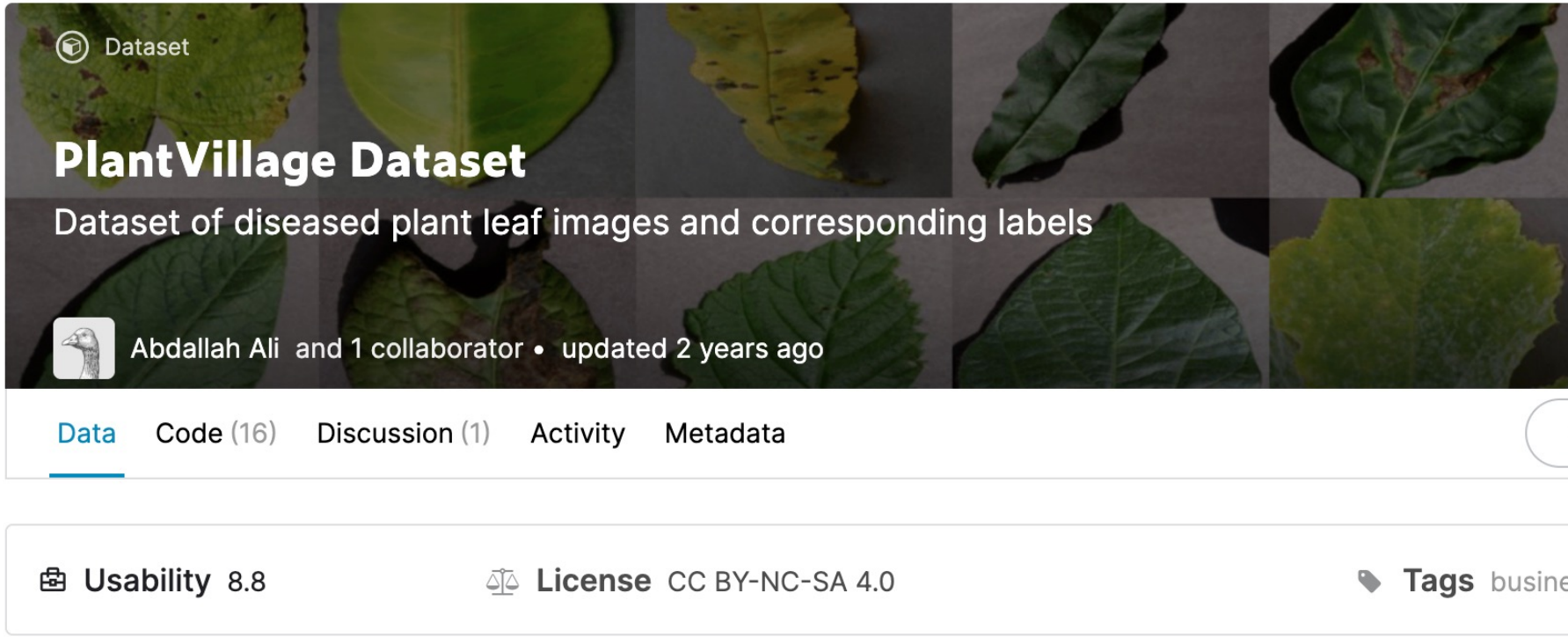
https://github.com/MattiaSpazzoli/PlantClassificator

Addestramento del classificatore - DataSet

kaggle

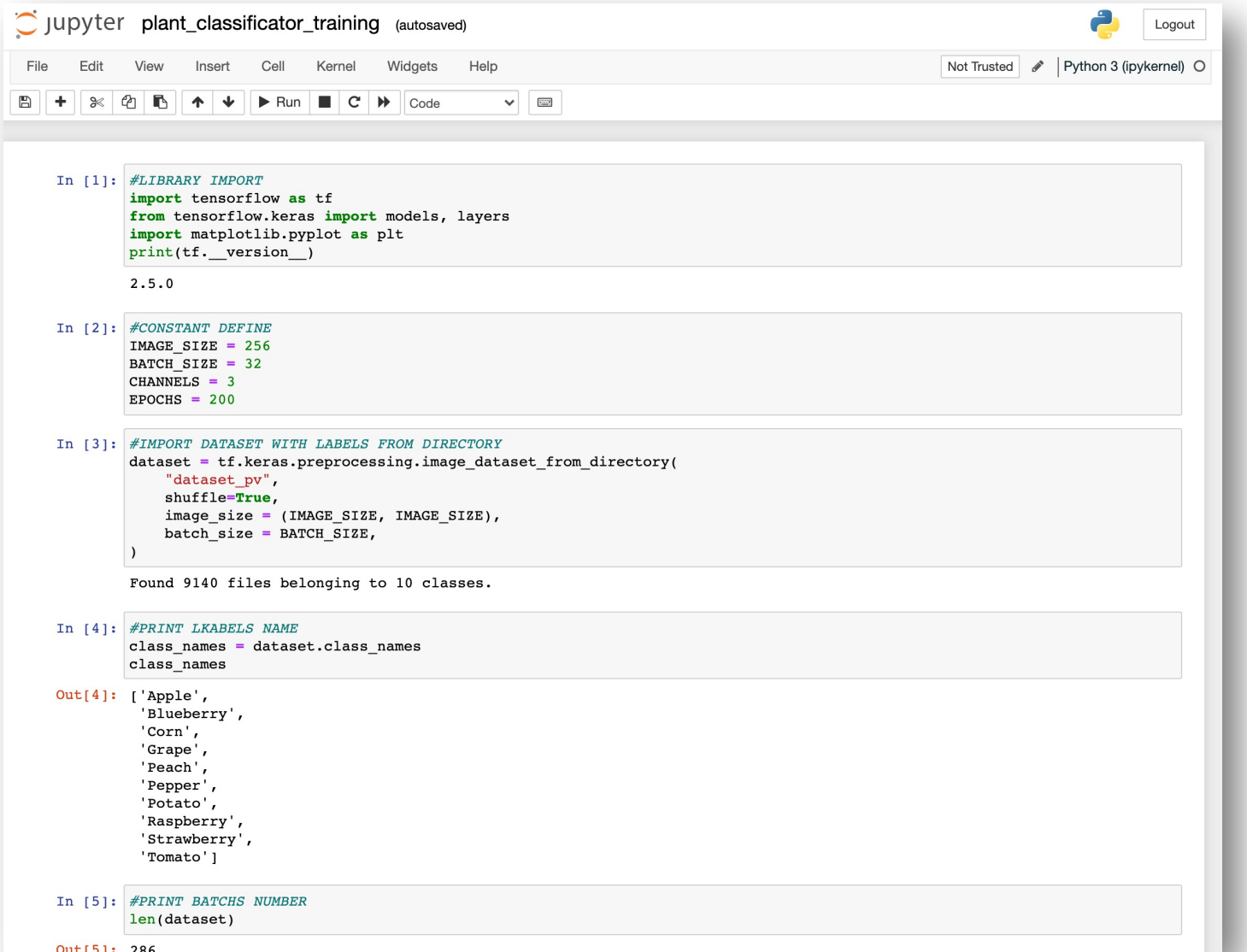
50000+ immagini

38 classi



https://www.kaggle.com/abdallahalidev/plantvillage-dataset

Addestramento del classificatore - Settings



python™

jupyter

Windows 10

intel CORE i7

Addestramento del classificatore - Preprocessing

```
#IMPORT DATASET WITH LABELS FROM DIRECTORY
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "dataset_pv",
    shuffle=True,
    image_size = (IMAGE_SIZE, IMAGE_SIZE),
    batch_size = BATCH_SIZE,
)
```

Caricamento del datset

```
#FETCH SOME FUNCTIONS TO USE THEM IN FOLLOW EPOCHS
train_ds=train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds=val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds=test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Prefetch

```
#SCALE RGB TENSOR TO 0-1 VALUES
resize_and_rescale= tf.keras.Sequential([layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
                                         layers.experimental.preprocessing.Rescaling(1.0/255)])
```

Resize and rescale

```
#FLIP AND ROTATE RANDOMLY IMAGES
data_augmentation=tf.keras.Sequential([layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
                                       layers.experimental.preprocessing.RandomRotation(0.2)])
```

Flip and rotation

Addestramento del classificatore - Architettura

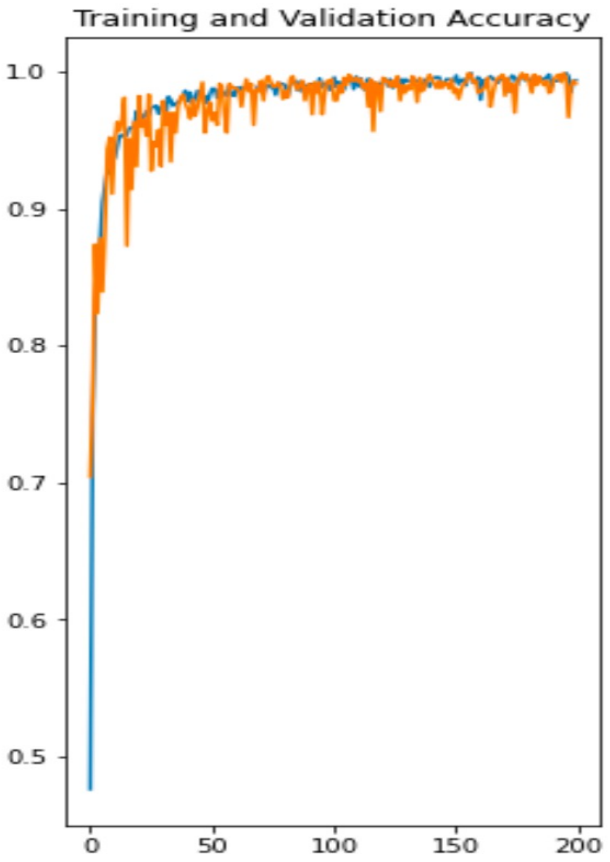
- 6 layer convoluzionali e 6 di pooling
- Funzione di loss: SparseCategoricalCrossentropy
- Batch da 32 immagini
- 10 classi di output

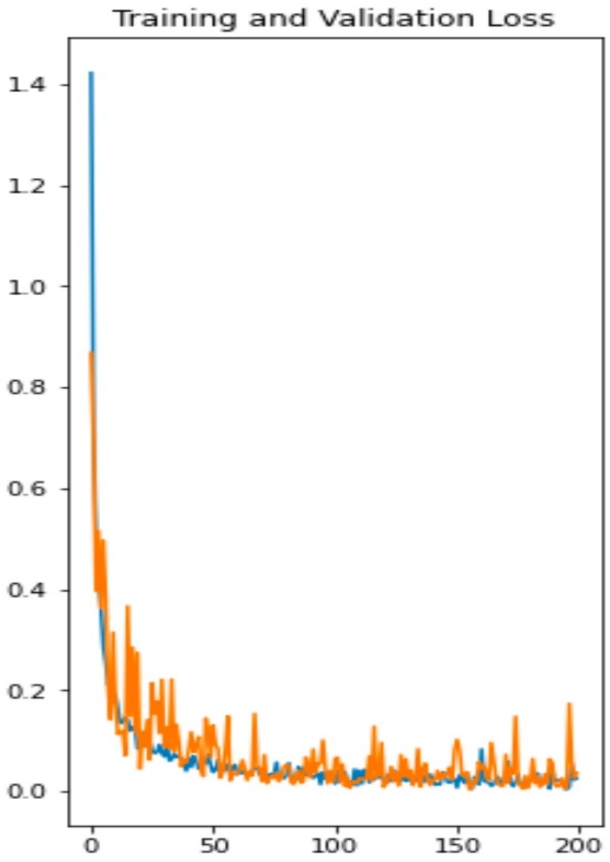
```
#CNN ARCHITECTURE CREATE
input_shape=(BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = len(class_names)
model = models.Sequential([
    #Applies preprocessing layers
    resize_and_rescale,

    #Applies CNN layers
    layers.Conv2D(32,(3,3), activation='relu', input_shape=input_shape), #Convolutional layer
    layers.MaxPooling2D((2,2)), #Pooling layer
    layers.Conv2D(64, kernel_size=(3,3), activation='relu'),  #Convolutional layer
    layers.MaxPooling2D((2,2)), #Pooling layer
    layers.Conv2D(64, kernel_size=(3,3), activation='relu'),  #Convolutional layer
    layers.MaxPooling2D((2,2)), #Pooling layer
    layers.Conv2D(64,(3,3), activation='relu'),  #Convolutional layer
    layers.MaxPooling2D((2,2)), #Pooling layer
    layers.Conv2D(64,(3,3), activation='relu'),  #Convolutional layer
    layers.MaxPooling2D((2,2)), #Pooling layer
    layers.Conv2D(64,(3,3), activation='relu'),  #Convolutional layer
    layers.MaxPooling2D((2,2)), #Pooling layer

    #Flat neurons
    layers.Flatten(),
    #Applies the rectified linear unit activation functio
    layers.Dense(64, activation="relu"),
    #Converts a vector of values to a probability distribution
    layers.Dense(n_classes, activation='softmax')
])
```

Addestramento del classificatore - Risultati





Configurazioni di training

200 epoche

30 ore di addestramento

Prestazioni del modello sul DataSet di Test

Loss: 0.008

Accuracy: 99.7 %

Conversione del modello Tensorflow

- Tf.lite.TFLiteConverter.from_keras_model()
- tf.lite.Interpreter()
- Input: Immagine a tre canali
- Output: 10 probailità corrispondenti alle labels

```
tf_lite_converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model=tf_lite_converter.convert()
```

```
tflite_model_name=TF_LITE_MODEL_FILE_NAME
open(tflite_model_name,"wb").write(tflite_model)
```

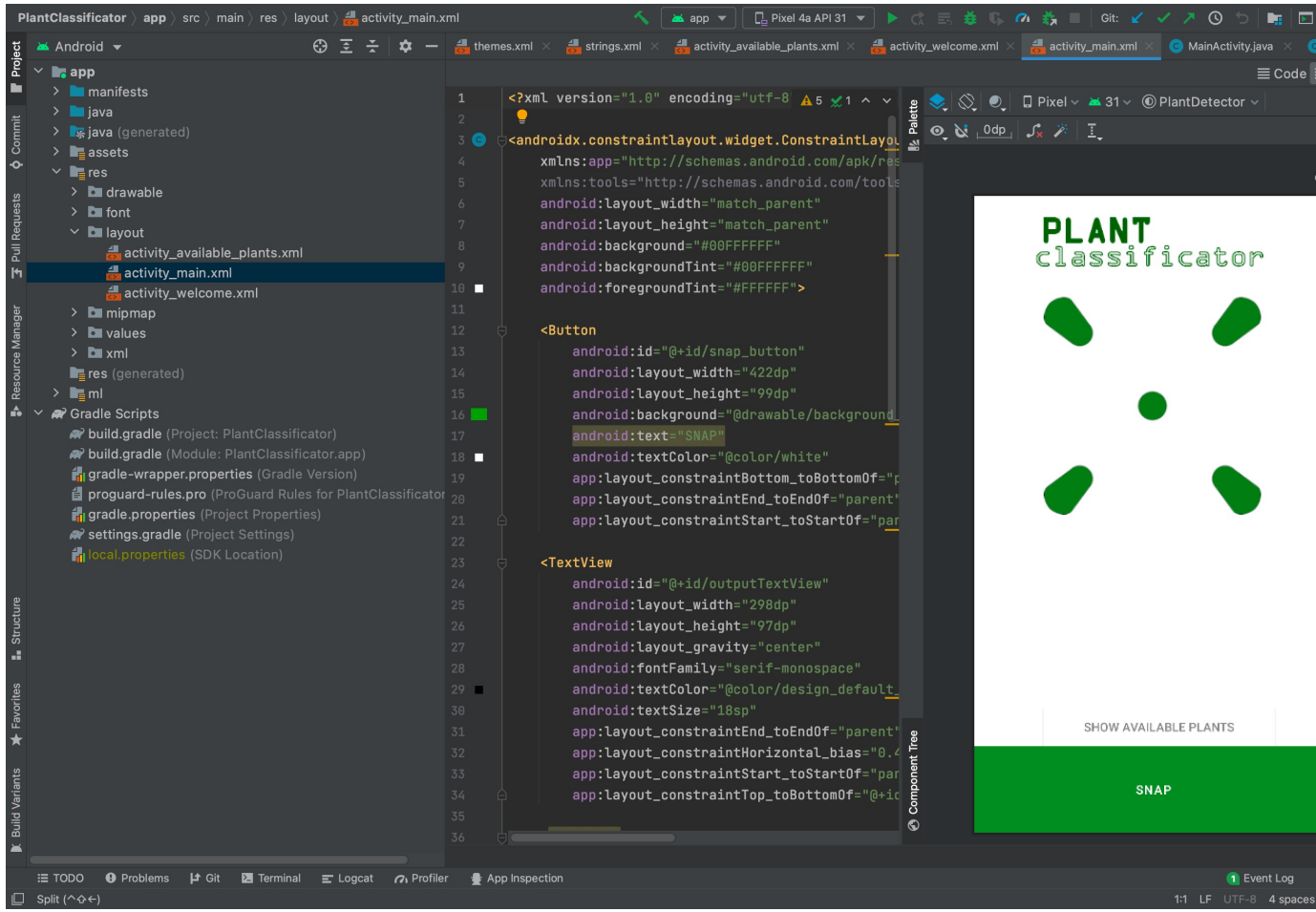
```
    interpreter = tf.lite.Interpreter(model_path = TF_LITE_MODEL_FILE_NAME)
    input_details = interpreter.get_input_details()
    output_details= interpreter.get_output_details()
    print("Input Shape:", input_details[0]['shape'])
    print("Input Type:", input_details[0]['dtype'])

    print("Output Shape:", output_details[0]['shape'])
    print("Output Type:", output_details[0]['dtype'])
```

```
Input Shape: [  1 256 256   3]
Input Type: <class 'numpy.float32'>
Output Shape: [ 1 10]
Output Type: <class 'numpy.float32'>
```

Applicazione Android - Introduzione

- Android Studio Artic Fox 2020.3.1
- Activity class .java
- Activity layout .xml
- AndroidManifest.xml e build.gradle



Applicazione Android – Apertura

Looper da 1 secondo

Permission per fotocamera

SNAP

SHOW AVAILABLE PLANTS

Applicazione Android - SNAP

- Apertura fotocamera
- Acquisizione dell'immagine
- Inferenza sul modello TensorflowLite
- Match tra probabilità e labels
- Setting dell'ImageView e della TextView con i risultati ottenuti

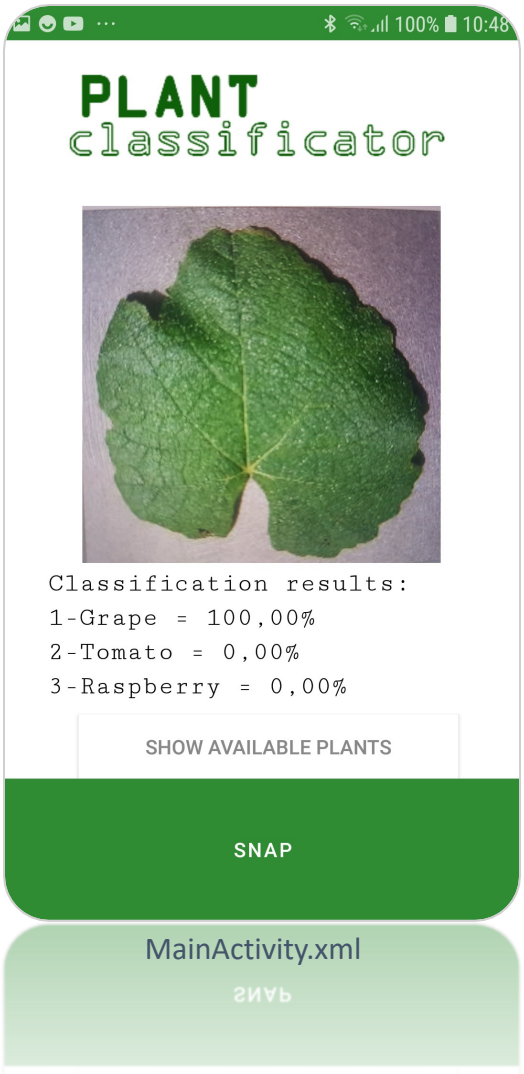
```
// Create Buffer di input
TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1, 256, 256, 3}, DataType.FLOAT32);
inputFeature0.loadBuffer(tensorImage.getBuffer());

// Create Buffer di output
float[][] result = new float[1][10];

MappedByteBuffer tfliteModel
        = FileUtil.loadMappedFile( context: this,
         filePath: "tf_lite_model.tflite");
InterpreterApi tflite = new InterpreterFactory().create(
        tfliteModel, new InterpreterApi.Options());

// Run inference
if (null != tflite) {
    tflite.run(inputFeature0.getBuffer(), result);
}
```

Tempo dell'inferenza: 147 ms



MainActivity.xml

Applicazione Android – Show available plants

AvailablePlants.xml