DEEP LEARNING
FINAL PROJECT REPORT

CPS 584_ 01

Prof. Mehdi Zargham

TEAM MEMBERS :
Saumya Kothuri

INTRODUCTION

This deep learning project is focused on locating sculptures at the University of Dayton.The project requires us to train a model with CNN network to help classify an image class. The image is an on-campus statue.Initially, We created a dataset by taking images of the sculpture from differing viewpoints.We were then given the whole dataset of all the sculptures, which was then utilized to train the model.This dataset consists of 24 classes.

PURPOSE OF THE PROJECT

History of the Sculpture:
Marie Louie is the name of the sculpture that we chose.We drive through the Stuart Grounds every day on our route to college, and this has sparked our interest in its history. After doing some study, we discovered that Marie Louise is linked to the history of it, which was the motivation for us to choose this sculpture.

What we know today is a Top 10 Catholic University and Ohio's largest independent University Grew from a community called Nazareth. It started in 1850 in a farmhouse with four men: a priest, a teacher, a cook and a gardener, a catholic religious order that grew out of Lake communities working to revive the religious spirit in the aftermath of the French Revolution. Mary Louisa had died of Cholera and her father decided to sell his Dewberry Farm 120 Acres on the southern edge of the city limits, recognizing the Farms potential father Meijer bargain for the purchase of the farm the Stewart home on the caretaker's cottage for a price of $12,000 father Meyer was penniless and accepted it for the simple medal of st. Joseph as collateral

DETAILS OF IMPLEMENTATION

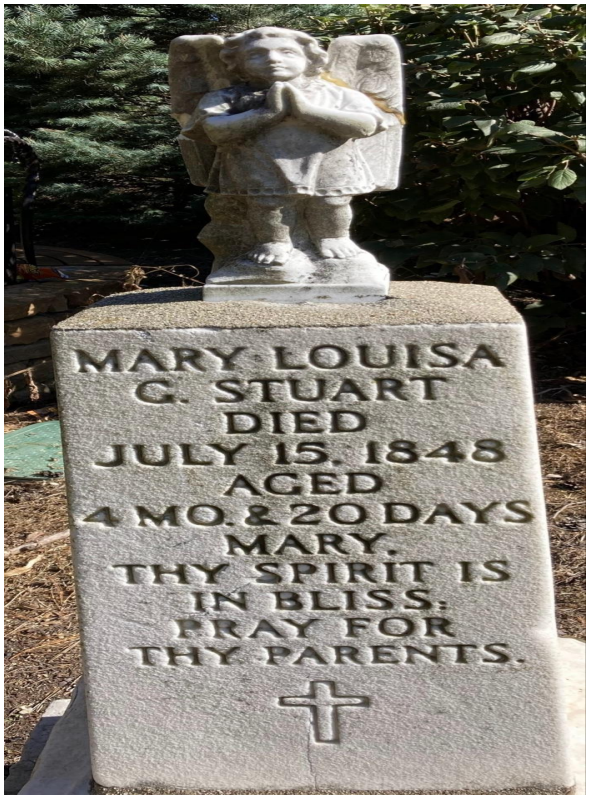
There are around four thousand pictures in the dataset. Among these four thousand photos 150 images are of our sculpture alone. We utilize the ImageDataGenerator, which first organizes our data in a specified directory

structure on our computer. ImageDataGenerator is a strong program that can be used to enrich photos and feed them into our model. We next inform the generators where to get the pictures by pointing them to these folders in the code.The augmentation happens in memory, and the generators make it simple to build up training and testing data without having to manually label the pictures.

Below are some photos from our Training Dataset:


IN BLISS:
PRAY FOR
THY PARENTS.


MARY LOUISA
C. STUART
DIED
JULY 15. 1848
AGED
4 MO. & 20 DAYS
MARY.
THY SPIRIT IS
IN BLISS:
PRAY FOR
THY PARENTS.


MARY LOUISA
C. STUART
DIED
JULY 15. 1848
AGED
4 MO. & 20 DAYS
MARY.
THY SPIRIT IS
IN BLISS:
PRAY FOR
THY PARENTS.

The Model is then defined. We are starting with an EfficientNet base model and adding custom layers. Now we'll construct and train our model on the dataset, which has been enhanced with ImageDataGenerators. Matplotlib is used to visualize the Model's performance graphically.

The Open Neural Network Exchange (ONNX) is a project that intends to connect deep learning frameworks. TF2ONNX was created to convert TensorFlow models to ONNX, so that TensorFlow features may be used by other deep learning systems. TF2ONNX, on the other hand, does not yet support quantization. Some models are only available in TensorFlow Lite (TFLite) format. This page describes TFLite2ONNX, a program that converts TFLite models to ONNX with quantization semantic translation.

```
from google.colab import drive
drive.mount('/content/drive')


#importing the required libraries
import os
```

```
import tensorflow as tf
from keras.models import Sequential, load_model
from keras.layers import Dropout, Dense,Flatten
local_path="/content/drive/MyDrive/UD SculptureDataset"
train = os.path.join(local_path,"TRAIN")
test = os.path.join(local_path,"VALIDATION")

#
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator( rescale = 1.0/255. ,
horizontal_flip=True )
test_datagen  = ImageDataGenerator( rescale = 1.0/255.,
horizontal_flip=True )

train_generator      =
train_datagen.flow_from_directory(train,batch_size=20,target_size=(224,
224))
validation_generator =
test_datagen.flow_from_directory(test,batch_size=20 , target_size=(224,
224))

#using EfficientNet
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import SGD
import tensorflow_hub as hub
base_model =
hub.KerasLayer("https://tfhub.dev/tensorflow/efficientnet/lite4/feature-
vector/2",
                               trainable=False)

model = tf.keras.Sequential([
       base_model,
       tf.keras.layers.Dense(24, activation='softmax')
])
model.build([1, 224, 224, 3])
model.summary()

model.compile(optimizer='adam',
```

```
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

#
checkpoint_filepath = "/content/drive/MyDrive/UD SculptureDataset"
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_loss',
    mode='min',
    save_best_only=True)

#
history = model.fit(train_generator,
validation_data=validation_generator,
epochs=25,callbacks=model_checkpoint_callback)

#plotting the training and validation accuracy
import matplotlib.pyplot as plt
acc = history.history[ 'accuracy' ]
val_acc = history.history[ 'val_accuracy' ]
loss = history.history[ 'loss' ]
val_loss = history.history['val_loss' ]
epochs = range(len(acc))

plt.plot ( epochs, acc )
plt.plot ( epochs, val_acc )
plt.title ('Training and validation accuracy')
plt.figure()

plt.plot ( epochs, loss )
plt.plot ( epochs, val_loss )
plt.title ('Training and validation loss' )

# generating a tflite model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model.
with open('model.tflite', 'wb') as f:
 f.write(tflite_model)
```

```
!pip install onnxruntime

!pip install -U tf2onnx

!pip install git+https://github.com/onnx/tensorflow-onnx

!git clone https://github.com/onnx/tensorflow-onnx

!python /content/tensorflow-onnx/setup.py bdist_wheel

# converting the tflite model to onnx model
!python -m tf2onnx.convert --opset 11 --tflite /content/model.tflite
--output model8.onnx
```

We deployed the model in Unity using Barracuda. Barracuda is a cross-platform network interface engine. Barracuda allows you to import and execute neural networks that have been pre-trained in a library of your choice and stored to disk. Barracuda uses ONNX (Open Neural Network Exchange), an open standard that most Deep Learning frameworks can export to. Importing is as simple as dragging and dropping the file into your Unity project for the user.

For this project we deployed the project on an iPhone because we had access to a MacBook. We had to join the Apple Developer Program**.** One of the plugins of the example Project is an external Xcode project manipulation DLL. The DLL is the build product of the source available in Unity's Bitbucket repository. A preferred way to include Xcode project manipulation functionality is to copy the C# source code files to the Assets/Editor folder in your Project.

PROBELMS ENCOUNTERED DURING THE PROJECT IMPLEMENTATION:

We had to train this model quite a few times as our sculpture was surrounded with trees. Because of the trees our sculpture wasn't trained properly. As the background was green and more vibrant than the sculpture the model kept

identifying the background. To resolve this problem we took a few more closeup photos of our sculpture and trained the model again.

OBSERVATIONS:

The sculptures with brilliant color showed greater training accuracy, however statues like William Shakespeare and Dante had detection issues because they looked almost identical. More epochs had to be completed.

RESULTS:

Plot of training/testing accuracy:

```
Text(0.5, 1.0, 'Training and validation loss')
```