Signication- Learning Project for pre-trained models and sign language processing

Suhani Malik, Jahnvi Srivastav, Chahat Mukund

12.03.2023

Table of Contents

Introduction

- Through this project we aim to learn about pre-trained transfer learning models for image classification to develop our Machine Learning Skills.
- This aim has been achieved by developing a sign language recognition system using an ASL dataset from Kaggle.
- We learned about 2 of the existing pre-trained models and implemented them to predict certain ASL signs.
- We also learnt how to use various python libraries for Machine Learning.
- Through the pre-trained model we are now able to predict 40 basic signs.

Tech Stack

Language

Using Python on Google Colab as well as some Linux commands to connect Kaggle for dataset

Libraries

Tensorflow, Keras, Scikit-learn, Matplotlib Pyplot, Numpy, , OpenCV

Dataset

American Sign Language Recognition from Kaggle

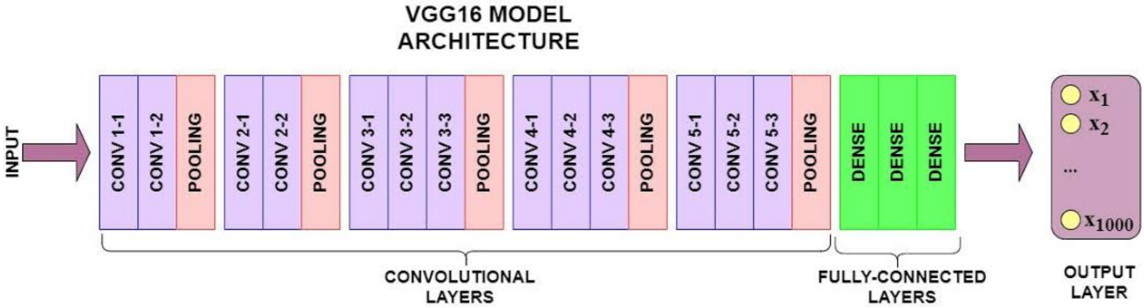
Pre-trained Models

VGG-16, ResNet50

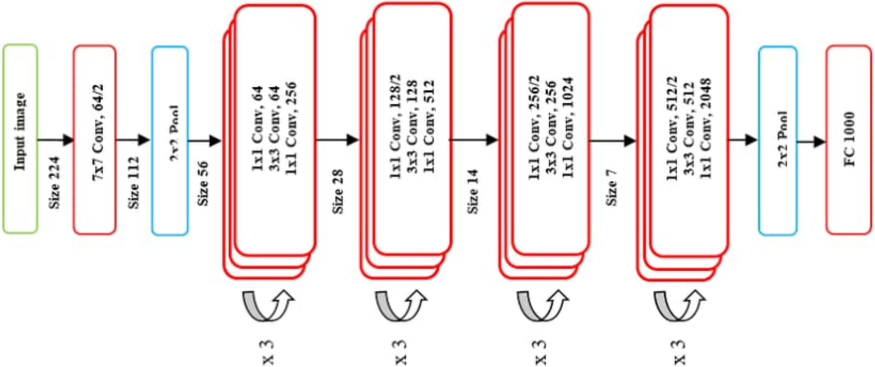
VGG-16

- VGG stands for Visual Geometry Group, a standard deep Convolutional Neural Network architecture with multiple layers.
- VGG16 is a convolutional neural network model proposed by the University of Oxford, which supports 16 layers.
- The VGG16 model was trained using Nvidia Titan Black GPUs for multiple weeks.
- VGG16 replaces the large kernel-sized filters with several 3×3 kernel-sized filters one after the other, making significant improvements.
- VGGNet-16 can classify images into 1000 object categories with an image input size of 224-by-224.

VGG-16 Architecture



Resnet50 Architecture



Initial steps

1. Getting API Key from Kaggle
2. Through the key loading the Kaggle dataset into the colab notebook using linux commands
3. unzipping the file to obtain the training set and testing set

Importing Essential Libraries

```
# Importing the Keras libraries and packages
from keras.applications.vgg16 import VGG16
from keras.applications import ResNet50
from keras.applications.vgg19 import VGG19
from keras.models import Model
from keras.preprocessing import image
from tensorflow.keras.layers import Input, Lambda ,Dense ,Flatten ,Dropout
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
import os
import cv2

train_dir = "training_set"
eval_dir = "test_set"
```

Loading the Data

```
#Load Images/ Data from given directories
def load_images(directory):
    images = []
    labels = []
    for idx, label in enumerate(uniq_labels):
        for file in os.listdir(directory + "/" + label):
            filepath = directory + "/" + label + "/" + file
            image = cv2.resize(cv2.imread(filepath), (64, 64))
            images.append(image)
            labels.append(idx)
    images = np.array(images)
    labels = np.array(labels)
    return(images, labels)
```

```
import keras

uniq_labels = sorted(os.listdir(train_dir))
images, labels = load_images(directory = train_dir)

if uniq_labels == sorted(os.listdir(eval_dir)):
    X_eval, y_eval = load_images(directory = eval_dir)
```

Splitting the Training and Testing Data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size = 0.2, stratify = labels)

n = len(uniq_labels)
train_n = len(X_train)
test_n = len(X_test)

print("Total number of symbols: ", n)
print("Number of training images: " , train_n)
print("Number of testing images: ", test_n)

eval_n = len(X_eval)
print("Number of evaluation images: ", eval_n)
```

```
Total number of symbols:  40
Number of training images:  48281
Number of testing images:  12071
Number of evaluation images:  8000
```

Pre-processing the Data

```
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)
y_eval = keras.utils.to_categorical(y_eval)
```

```
print(y_train[0])
print(len(y_train[0]))
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
40
```

```
X_train = X_train.astype('float32')/255.0
X_test = X_test.astype('float32')/255.0
X_eval = X_eval.astype('float32')/255.0
```

Initialising and Fitting the Models

```
#Initialising vgg16
classifier_vgg16 = VGG16(input_shape= (64,64,3),include_top=False,weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

[40]
```
#initialising ResNet50
classifier_resnet = ResNet50(input_shape= (64,64,3),include_top=False,weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
```

[41]
```
#don't train existing weights for vgg16
for layer in classifier_vgg16.layers:
    layer.trainable = False

#don't train existing weights for resnet50
for layer in classifier_resnet.layers:
    layer.trainable = False
```

Initialising and Fitting the Models

```
classifier1 = classifier_vgg16.output#head mode
classifier1 = Flatten()(classifier1)#adding layer of flatten
classifier1 = Dense(units=256, activation='relu')(classifier1)
classifier1 = Dropout(0.6)(classifier1)
classifier1 = Dense(units=40, activation='softmax')(classifier1)

model = Model(inputs = classifier_vgg16.input , outputs = classifier1)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[43] classifier2 = classifier_resnet.output#head mode
classifier2 = Flatten()(classifier2)#adding layer of flatten
classifier2 = Dropout(0.6)(classifier2)
classifier2 = Dense(units=40, activation='softmax')(classifier2)

model2 = Model(inputs = classifier_resnet.input , outputs = classifier2)
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Initialising and Fitting the Models

```
[46] #fit the model
     #it will take some time to train
     history = model.fit(X_train, y_train, epochs =5, batch_size = 64,validation_data=(X_test,y_test))

     Epoch 1/5
     755/755 [==============================] - 1909s 3s/step - loss: 0.2429 - accuracy: 0.9498 - val_loss: 0.0026 - val_accuracy: 0.9998
     Epoch 2/5
     755/755 [==============================] - 1899s 3s/step - loss: 0.0121 - accuracy: 0.9989 - val_loss: 2.8155e-04 - val_accuracy: 1.0000
     Epoch 3/5
     755/755 [==============================] - 1933s 3s/step - loss: 0.0058 - accuracy: 0.9994 - val_loss: 1.5072e-04 - val_accuracy: 1.0000
     Epoch 4/5
     755/755 [==============================] - 2022s 3s/step - loss: 0.0049 - accuracy: 0.9992 - val_loss: 1.3345e-04 - val_accuracy: 1.0000
     Epoch 5/5
     755/755 [==============================] - 1965s 3s/step - loss: 0.0070 - accuracy: 0.9985 - val_loss: 1.2755e-05 - val_accuracy: 1.0000
```

Figure: Fitting Model 1

```
#fit the model

history2 = model2.fit(X_train, y_train, epochs =5, batch_size = 64,validation_data=(X_test,y_test))

Epoch 1/5
755/755 [==============================] - 618s 814ms/step - loss: 0.4293 - accuracy: 0.9163 - val_loss: 0.0460 - val_accuracy: 0.9979
Epoch 2/5
755/755 [==============================] - 614s 814ms/step - loss: 0.0772 - accuracy: 0.9873 - val_loss: 0.0177 - val_accuracy: 0.9992
Epoch 3/5
755/755 [==============================] - 614s 813ms/step - loss: 0.0499 - accuracy: 0.9908 - val_loss: 0.0096 - val_accuracy: 0.9997
Epoch 4/5
755/755 [==============================] - 614s 814ms/step - loss: 0.0364 - accuracy: 0.9929 - val_loss: 0.0052 - val_accuracy: 0.9998
Epoch 5/5
755/755 [==============================] - 615s 815ms/step - loss: 0.0305 - accuracy: 0.9933 - val_loss: 0.0037 - val_accuracy: 0.9998
```

Figure: Fitting Model 2

Getting Accuracy of the Models

```
score = model.evaluate(x = X_test, y = y_test, verbose = 0)
print('Accuracy for test images:', round(score[1]*100, 3), '%')
score = model.evaluate(x = X_eval, y = y_eval, verbose = 0)
print('Accuracy for evaluation images:', round(score[1]*100, 3), '%')
```

```
Accuracy for test images: 100.0 %
Accuracy for evaluation images: 100.0 %
```

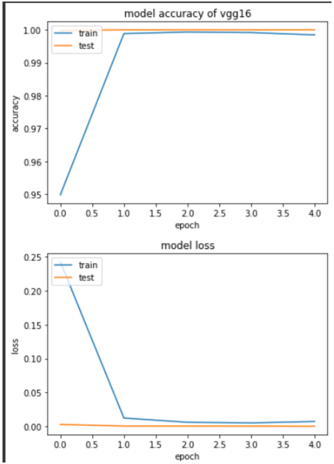
Figure: Accuracy of Model 1

```
score = model.evaluate(x = X_test, y = y_test, verbose = 0)
print('Accuracy for test images:', round(score[1]*100, 3), '%')
score = model.evaluate(x = X_eval, y = y_eval, verbose = 0)
print('Accuracy for evaluation images:', round(score[1]*100, 3), '%')
```

```
Accuracy for test images: 100.0 %
Accuracy for evaluation images: 100.0 %
```

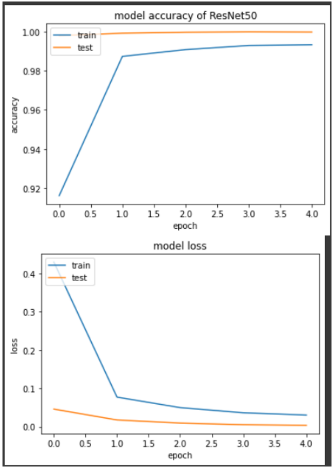
Figure: Accuracy of Model 2

Accuracy and Loss Plots of the Models



Plot of Model 1

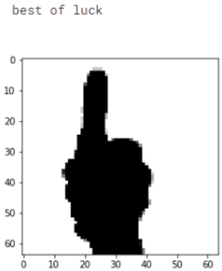
Accuracy and Loss Plots of the Models



Plot of Model 2

Prediction

We obtain the prediction using any image from the test set. Below is the example of predicting Best of Luck



Future Goals

- Learn the functioning of more pre-trained models like VGG19, InceptionV3, MobileNet etc.
- Try using heavier datasets with more images
- Integrate the algorithm with a front end to create a real-time sign language dictionary application

References

- Dataset – kaggle.com
- VGG16 – Builtin.com
- VGG16 Architecture – Idiotdeveloper.com
- Resnet50 Architecture – datagen.tech
- Resnet50 Architecture – towardsdatascience.com

Thank You