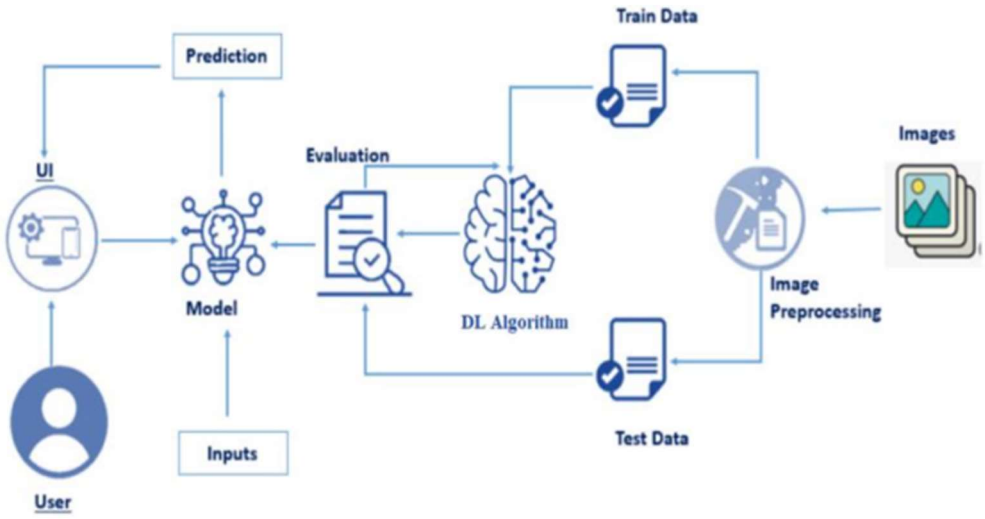
ASL- Alphabet Image Recognition

Project Description:

The objective of the ASL Alphabet Image Recognition project is to use image processing and machine learning techniques to create a reliable and precise system for identifying American Sign Language (ASL) alphabet movements. It is a visual language that conveys meaning through a combination of body language, facial emotions, and hand gestures. The development of technology to aid in closing the communication gap between the hearing and deaf communities has drawn more attention in recent years.

This project aims to improve communication and accessibility for this population. This project involves training a machine learning model to classify images of hand signs corresponding to the 26 letters of the English alphabet, as well as three additional classes for the signs for "space", "delete", and "nothing".

Technical Architecture:



Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.
- The Model analyzes the image, then the prediction is showcased on the Flask UI. To accomplish this, we have to complete all the activities and tasks listed below

Image Dataset Collection:

Acquire a diverse dataset of images featuring ASL alphabet gestures. The dataset should include variations in lighting conditions, backgrounds, and hand orientations to ensure the model's robustness.

Data Preprocessing:

Clean and preprocess the collected images to standardize them for training. This may involve resizing, normalization, and other image processing techniques to enhance the quality of the dataset.

Model Architecture:

Design a Convolutional Neural Network (CNN) or a suitable deep learning architecture for image recognition. The model should be capable of learning and distinguishing the unique hand configurations corresponding to each ASL alphabet gesture.

Training the Model:

Train the model using the preprocessed dataset. Implement techniques such as transfer learning if applicable, to leverage pre-trained models and improve the efficiency of the training process.

a. Import the necessary libraries for building the CNN model

b. Define the input shape of the image data

c. Add layers to the model:

Convolutional Layers: Apply filters to the input image to create feature maps

Pooling Layers: Reduce the spatial dimensions of the feature maps

Fully Connected Layers: Flatten the output of the convolutional layers and apply fully

connected layers to classify the images

d. Compile the model by specifying the optimizer, loss function, and metrics to be used

during training

Validation and Testing:

Validate the trained model using a separate dataset to assess its accuracy, precision, and recall. Fine-tune the model based on validation results and perform comprehensive testing to evaluate its real-world performance.

User Interface:

Develop a user-friendly interface to allow users to interact with the system. This interface could be a web or mobile application where users can input images of ASL alphabet gestures for recognition.

Real-Time Recognition:

Implement real-time recognition capabilities to make the system practical for dynamic communication scenarios. This involves optimizing the model for quick and accurate predictions.

Accessibility Features:

Integrate accessibility features to ensure the application is usable by individuals with varying levels of technical expertise and diverse needs.

Documentation and User Guide:

Create comprehensive documentation and a user guide to assist developers and end-users in understanding the system, its functionalities, and how to integrate it into different applications.

Deployment:

Deploy the ASL Alphabet Image Recognition system on a suitable platform, considering scalability and accessibility. This could involve cloud deployment for broader accessibility.

Prior Knowledge:

You must have prior knowledge of following topics to complete this project

- .Deep Learning Concepts of CNN:

Link: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

- VGG19:

VGG-19 convolutional neural network - MATLAB vgg19 - MathWorks India

- ResNet-50V2:

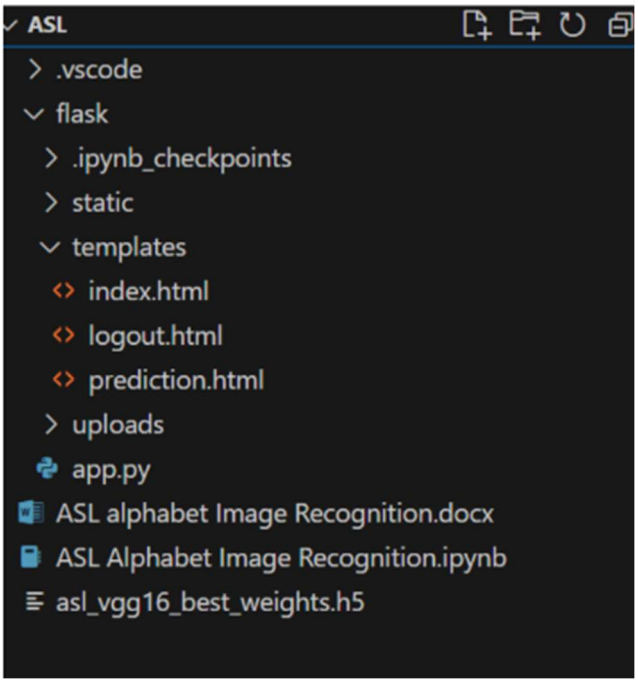
Link: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

- Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

Link: https://www.youtube.com/watch?v=lj4I_CvBnt0

Build Python Code:

Project Structure:



For building a Flask Application we needs HTML pages stored in the templates folder, CSS for styling the pages stored in the static folder and a python script app1.py for server side scripting.

MILE STONE 1: DATA COLLECTION

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

Activity 1: Download the dataset

Collect images of ASL alphabet signs then organise based on the alphabets as shown in project structure. Create folders of types of alphabets that need to be recognised.

You can download the dataset used in this project using the below link

Kaggle Dataset: https://www.kaggle.com/datasets/grassknoted/asl-alphabet

Activity 2: Importing files and dataset

```
from google.colab import files
```

```
# From Desktop, upload: utils.py
uploaded = files.upload()
for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))
```

```
Saving utils.py to utils.py
User uploaded file "utils.py" with length 2751 bytes
```

```
# GDrive: upload asl dataset downloaded from Kaggle (see link above)
# Mount GDrive to Colab folder
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Copy things to a local folder and unzip
!mkdir data
!cp -r drive/MyDrive/Data/asl/* data/
!unzip -q data/asl_alphabet_train.zip -d data/
!unzip -q data/asl_alphabet_test.zip -d data/
```

```
# Flush and unmount; changes persist in Google Drive
drive.flush_and_unmount()
```

MILESTONE 2: DATA PREPARATION

Activity1: Installing necessary Libraries

The first step in preparing data for ASL Alphabet Recognition involves setting up the development environment by installing the necessary libraries and dependencies.

```
import pickle
import numpy as np
np.random.seed(5)
import pandas as pd
import tensorflow as tf
import utils
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

```
# Load pre-shuffled training and test datasets
# Both the train & test splits are taken from the same dataset
(x_train, y_train), (x_test, y_test) = utils.load_data(
    container_path = 'data/asl_alphabet_train/asl_alphabet_train')
```

```
x_train.shape
```

```
(60000, 50, 50, 3)
```

```
y_train.shape
```

```
(60000,)
```

```
y_train
```

```
array([16, 19, 11, ...,  4, 21, 21])
```

```
x_test.shape
```

```
(15000, 50, 50, 3)
```

```
x_train.min()
```

```
0.0
```

```
x_train.max()
```

```
1.0
```

Activity 2 : Visualize and Examine Dataset

It involves exploring the distribution of sign images, assessing image quality, and understanding label consistency. Utilizing tools like matplotlib or seaborn can aid in creating histograms or heatmaps to analyze class imbalance, while previewing sample images helps identify potential challenges in the dataset. Additionally, inspecting metadata such as image dimensions and pixel intensity contributes to a comprehensive understanding of the ASL dataset before model development.

```
# Store labels of dataset
#labels = ['A', 'B', 'C']
labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
          'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
          'S', 'T', 'U', 'W', 'X', 'Y', 'Z']

# Print the first several training images, along with the labels
fig = plt.figure(figsize=(20,5))
for i in range(36):
    ax = fig.add_subplot(3, 12, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_train[i]))
    ax.set_title("{}".format(labels[y_train[i]]))
plt.show()
```

```
# The dataset is quite balanced!
print("Train/test samples per class:")
for i, label in enumerate(labels):
    num_train = sum(y_train==i)
    num_test = sum(y_test==i)
    print(f"{label}: {num_train} / {num_test} = {round(num_train/num_test, 2)}")
```

```
Train/test samples per class:
A: 2384 / 616 = 3.87
B: 2392 / 608 = 3.93
C: 2454 / 546 = 4.49
D: 2395 / 605 = 3.96
E: 2379 / 621 = 3.83
F: 2386 / 614 = 3.89
G: 2408 / 592 = 4.07
H: 2416 / 584 = 4.14
I: 2354 / 646 = 3.64
J: 2387 / 613 = 3.89
K: 2412 / 588 = 4.1
L: 2382 / 618 = 3.85
M: 2399 / 601 = 3.99
N: 2360 / 640 = 3.69
O: 2397 / 603 = 3.98
P: 2391 / 609 = 3.93
Q: 2409 / 591 = 4.08
R: 2398 / 602 = 3.98
S: 2396 / 604 = 3.97
T: 2418 / 582 = 4.15
U: 2436 / 564 = 4.32
W: 2429 / 571 = 4.25
X: 2410 / 590 = 4.08
Y: 2403 / 597 = 4.03
Z: 2405 / 595 = 4.04
```

MILESTONE 3 : DATA PREPROCESSING

- Resize and normalize ASL alphabet images for consistent input dimensions and pixel value ranges.
- Apply data augmentation to diversify the training set, while encoding labels and addressing class imbalances.
- Split the dataset, handle missing data, and set up a memory-efficient pipeline for efficient model training.

```
from keras.utils import np_utils

# One-hot encode the training labels
y_train_OH = np_utils.to_categorical(y_train, len(labels))

# One-hot encode the test labels
y_test_OH = np_utils.to_categorical(y_test, len(labels))
```

MILESTONE 4 : MODEL BUILDING

Training the Model:

Define Model 1: Simple CNN from Scratch

Need to consider various components such as the architecture, layers, and parameters.

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Flatten, Dense, Dropout, Reshape, Conv2DTranspose
from tensorflow.keras.models import Sequential
```

```
model = Sequential()
# First convolutional layer accepts image input
model.add(Conv2D(filters=16,
                 kernel_size=3,
                 padding='same',
                 activation='relu',
                 input_shape=(50, 50, 3)))
# Add a max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2))) # 50/2 = 25
# Add a convolutional layer
model.add(Conv2D(filters=32,
                 kernel_size=3,
                 padding='same',
                 activation='relu'))
# Add another max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2))) # 25/2 = 12
# Add a convolutional layer
model.add(Conv2D(filters=64,
                 kernel_size=3,
                 padding='same',
                 activation='relu'))
# Add another max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2))) # 12/2 = 6
```

```
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(labels), activation='softmax'))
# Summarize the model
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 50, 50, 16)        448
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 25, 25, 16)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 25, 25, 32)        4640
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 6, 6, 128)         73856
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 3, 3, 128)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               590336
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 25)                12825
```

Train Model 1

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Compile the model
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
# Early stopping when validation loss stops decreasing
# Arguments:
# - monitor: value to be monitored -> val_loss: loss of validaton data
# - mode: min -> training stops when monitored value stops decreasing
# - patience: number of epochs with no improvement after which training will be stopped
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=3)
```

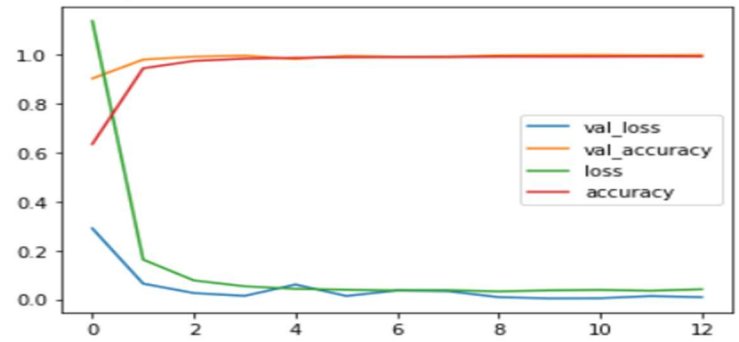
```
# Train the model
hist = model.fit(x_train,
                 y_train_OH,
                 epochs=20,
                 validation_split=0.2,
                 batch_size=32,
                 shuffle=True,
                 callbacks=[early_stop])
```

MILESTONE 5: MODEL EVALUATION

Evaluate Model 1

```
# Plot learning curves
losses = pd.DataFrame(model.history.history)
losses.plot()
```

```
<AxesSubplot:>
```




```
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

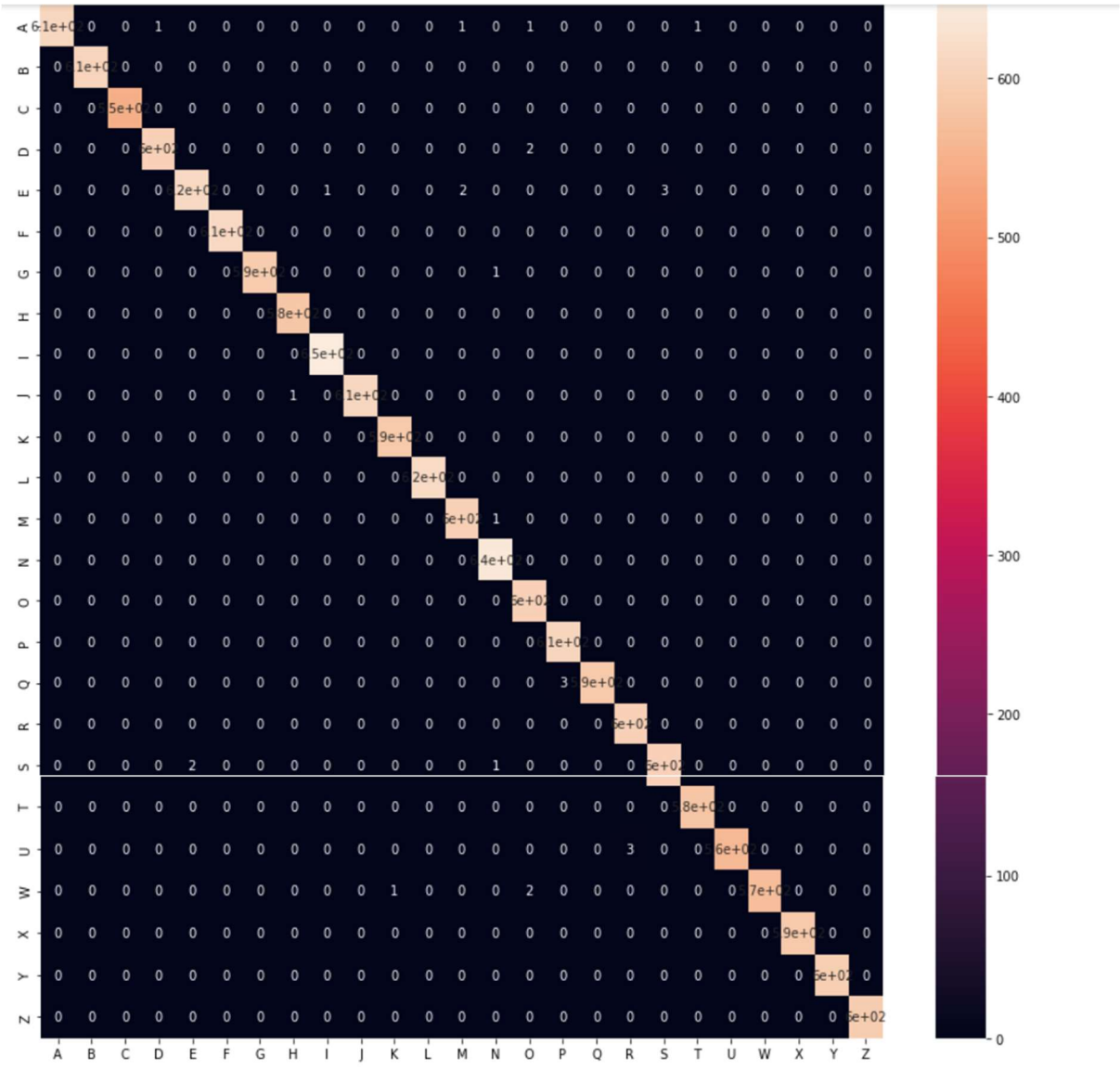
```
# Obtain accuracy on test set
score = model.evaluate(x=x_test,
                       y=y_test_OH,
                       verbose=0)
print('Test accuracy:', score[1])
```

```
Test accuracy: 0.998199999332428
```

```
# Get predicted probabilities for test dataset
y_probs = model.predict(x_test)

# Get predicted labels for test dataset
#y_preds = model.predict_classes(x_test)
y_preds = np.argmax(y_probs, axis=1)
```

```
outcome = pd.DataFrame(np.array([y_test,y_preds]).T, columns=['true', 'pred'])
id2label = {i:value for i,value in enumerate(labels)}
outcome = outcome[['true','pred']].replace(id2label)
```

```
plt.figure(figsize=(15,15))
sns.heatmap(confusion_matrix(outcome['true'],outcome['pred']),
            xticklabels=labels,
            yticklabels=labels,
            annot=True);
```

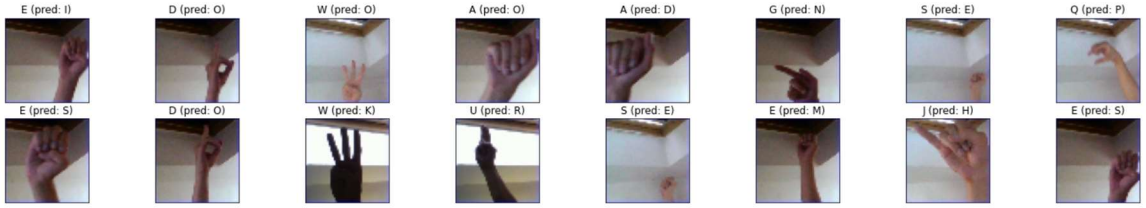


```
# Indices corresponding to test images which were mislabeled
bad_test_idxs = np.where(y_preds!=y_test)[0]
```

```
len(bad_test_idxs)
```

```
27
```

```
# Print a random subset of mislabeled examples
fig = plt.figure(figsize=(25,4))
subset_size = 16
try:
    assert subset_size < len(bad_test_idxs)
except AssertionError:
    print("Use a smaller subset size!")
# Pick a random subset
z = int(np.random.rand()*(len(bad_test_idxs)-subset_size))
subset = bad_test_idxs[z:(z+subset_size)]
for i, idx in enumerate(subset):
    ax = fig.add_subplot(2, int(np.ceil(len(subset)/2)), i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    ax.set_title("{} (pred: {})".format(labels[y_test[idx]], labels[y_preds[idx]]))
```



Define, Train & Evaluate Model 2: ResNet50/VGG16 with Transfer Learning

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Flatten, Dense, Dropout, Reshape, Conv2DTranspose
from tensorflow.keras.models import Sequential

#from tensorflow.keras.applications import ResNet50
#from tensorflow.keras.applications.resnet50 import preprocess_input

from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
```

```
# Apply ResNet50-specific preprocessing
def preprocess(images, labels):
    return preprocess_input(images), labels

x_train_trans, y_train_OH = preprocess(x_train, y_train_OH)
x_test_trans, y_test_OH = preprocess(x_test, y_test_OH)
```

```
x_train_trans.min()
```

```
-123.68
```

```
transfer_model.add(Dropout(0.5))
transfer_model.add(Dense(len(labels), activation='softmax'))

# The complete model (fine-tuning) has 24/0.7 = 34x more parameters than the previous one!
# The transfer learning model has 1/0.7 = 1.4x more parameters than the previous one
transfer_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 2s 0us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 512)               14714688

 flatten_1 (Flatten)         (None, 512)               0

 dense_2 (Dense)             (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 25)                12825

=================================================================
Total params: 14,990,169
Trainable params: 275,481
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
x_train_trans.max()
```

```
-102.939
```

```
x_train_trans.shape
```

```
(60000, 50, 50, 3)
```

```
# Empty sequential model
transfer_model = Sequential()

# From Keras Applications, we can download many pre-trained models
# If we specify include_top=False, the original input/output layers
# are not imported.
# Note that we can specify our desired the input and output layer sizes!
pretrained_model= VGG16(include_top=False,
                   input_shape=(x_train.shape[1:]),
                   pooling='avg',
                   classes=len(labels),
                   weights='imagenet')
# Freeze layers; if not active, fine-tuning, else transfer learning
for layer in pretrained_model.layers:
    layer.trainable = False

# Add ResNet to empty sequential model
transfer_model.add(pretrained_model)

# Now, add the last layers of our model which map the extracted features
# too the classes - that's the classifier, what's really trained
transfer_model.add(Flatten())
transfer_model.add(Dense(512, activation='relu'))
```

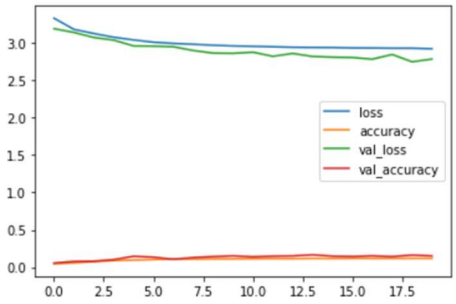
```python
from tensorflow.keras.callbacks import EarlyStopping

# Early stopping when validation loss stops decreasing
# Arguments:
# - monitor: value to be monitored -> val_loss: loss of validaton data
# - mode: min -> training stops when monitored value stops decreasing
# - patience: number of epochs with no improvement after which training will be stopped
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=3)
```

```python
from tensorflow.keras.optimizers import Adam

# Compile
#resnet_model.compile(optimizer=Adam(learning_rate=0.0001),loss='categorical_crossentropy',metrics=['accuracy'])
transfer_model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy'])

# Train/Fit
# I use less epochs, since in the previous model with 10 we were already good enough
history = transfer_model.fit(
                x_train_trans,
                #x_train,
                y_train_OH,
                epochs=20,
                validation_split=0.2,
                batch_size=32,
                shuffle=True,
                callbacks=[early_stop])
```

```
Epoch 1/20
1500/1500 [==============================] - 36s 19ms/step - loss: 3.3338 - accuracy: 0.0443 - val_loss: 3.1931 - val_accuracy: 0.0547
Epoch 2/20
1500/1500 [==============================] - 29s 19ms/step - loss: 3.1838 - accuracy: 0.0572 - val_loss: 3.1429 - val_accuracy: 0.0770
Epoch 3/20
1500/1500 [==============================] - 29s 19ms/step - loss: 3.1283 - accuracy: 0.0749 - val_loss: 3.0754 - val_accuracy: 0.0798
Epoch 4/20
1500/1500 [==============================] - 28s 19ms/step - loss: 3.0782 - accuracy: 0.0877 - val_loss: 3.0402 - val_accuracy: 0.1013
Epoch 5/20
1500/1500 [==============================] - 28s 19ms/step - loss: 3.0426 - accuracy: 0.0958 - val_loss: 2.9623 - val_accuracy: 0.1460
Epoch 6/20
1500/1500 [==============================] - 28s 19ms/step - loss: 3.0115 - accuracy: 0.1020 - val_loss: 2.9592 - val_accuracy: 0.1334
Epoch 7/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9961 - accuracy: 0.1084 - val_loss: 2.9512 - val_accuracy: 0.1067
Epoch 8/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9841 - accuracy: 0.1081 - val_loss: 2.9007 - val_accuracy: 0.1286
Epoch 9/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9715 - accuracy: 0.1097 - val_loss: 2.8660 - val_accuracy: 0.1418
Epoch 10/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9623 - accuracy: 0.1107 - val_loss: 2.8630 - val_accuracy: 0.1513
Epoch 11/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9570 - accuracy: 0.1136 - val_loss: 2.8782 - val_accuracy: 0.1394
Epoch 12/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9510 - accuracy: 0.1154 - val_loss: 2.8232 - val_accuracy: 0.1478
Epoch 13/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9440 - accuracy: 0.1143 - val_loss: 2.8613 - val_accuracy: 0.1506
Epoch 14/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9414 - accuracy: 0.1173 - val_loss: 2.8213 - val_accuracy: 0.1658
Epoch 15/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9398 - accuracy: 0.1157 - val_loss: 2.8118 - val_accuracy: 0.1475
Epoch 16/20
1500/1500 [==============================] - 33s 22ms/step - loss: 2.9349 - accuracy: 0.1180 - val_loss: 2.8070 - val_accuracy: 0.1437
Epoch 17/20
1500/1500 [==============================] - 28s 19ms/step - loss: 2.9340 - accuracy: 0.1173 - val_loss: 2.7845 - val_accuracy: 0.1517
```

```
# Plot learning curves
losses = pd.DataFrame(transfer_model.history.history)
losses.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f12a6147a60>
```




```
# Obtain accuracy on test set
score = transfer_model.evaluate(x=x_test_trans,
                      y=y_test_OH,
                      verbose=0)
print('Test accuracy:', score[1])
```

```
Test accuracy: 0.14873333275318146
```

```
# Get predicted probabilities for test dataset
y_probs = transfer_model.predict(x_test_trans)
```

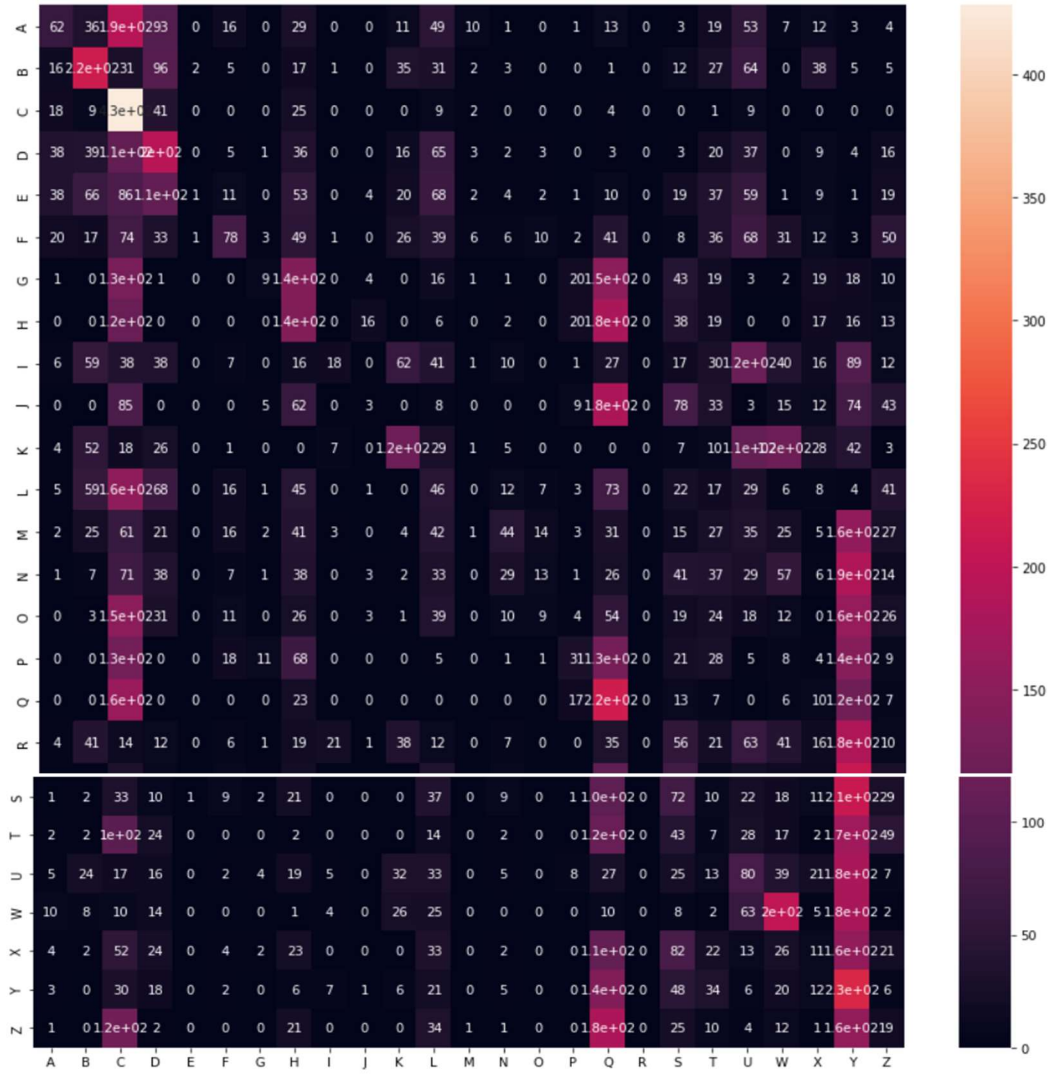
```
# Get predicted probabilities for test dataset
y_probs = transfer_model.predict(x_test_trans)

# Get predicted labels for test dataset
#y_preds = transfer_model.predict_classes(x_test_trans)
y_preds = np.argmax(y_probs, axis=1)
```

```
469/469 [==============================] - 6s 13ms/step
```

```
outcome = pd.DataFrame(np.array([y_test,y_preds]).T, columns=['true', 'pred'])
id2label = {i:value for i,value in enumerate(labels)}
outcome = outcome[['true','pred']].replace(id2label)
```

```
#from sklearn.metrics import confusion_matrix
#import seaborn as sns
```

```
plt.figure(figsize=(15,15))
sns.heatmap(confusion_matrix(outcome['true'],outcome['pred']),
            xticklabels=labels,
            yticklabels=labels,
            annot=True);
```




```
# Indices corresponding to test images which were mislabeled
bad_test_idxs = np.where(y_preds!=y_test)[0]
```

```
len(bad_test_idxs)
```

```
12769
```

Define, Train & Evaluate Model 3: Autoencoder + Random Forest

This model is probably a little bit peculiar. Since the neural networks seem to be working that well, I will try to use them to compress the images to a latent representation. Then, with that latent representation:

- I attach a classifier: a Random Forest.
- I plot the samples in a 2D space with manifold learning.

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Flatten, Dense, Dropout, Reshape, Conv2DTranspose
from tensorflow.keras.models import Sequential
```

```
# Encoder: we compress the image
encoder = Sequential()
# First convolutional layer accepts image input
encoder.add(Conv2D(filters=32,
                   kernel_size=3,
                   padding='same',
                   activation='relu',
                   input_shape=(50, 50, 3)))
# Add a max pooling layer
encoder.add(MaxPooling2D(pool_size=(2, 2))) # 50/2 = 25
# Add a convolutional layer
encoder.add(Conv2D(filters=64,
                   kernel_size=3,
                   padding='same',
                   activation='relu'))
# Add another max pooling layer
encoder.add(MaxPooling2D(pool_size=(2, 2))) # 25/2 = 12
# Add a convolutional layer
encoder.add(Conv2D(filters=128,
                   kernel_size=3,
                   padding='same',
                   activation='relu'))
# Add another max pooling layer
encoder.add(MaxPooling2D(pool_size=(2, 2))) # 12/2 = 6
# Add a convolutional layer
encoder.add(Conv2D(filters=256,
                     kernel_size=3,
                     padding='same',
                     activation='relu'))
# Add another max pooling layer
encoder.add(MaxPooling2D(pool_size=(2, 2))) # 6/2 = 3
# Compressed size: (batch, 3, 3, 256): 2304
# This flattening is not really necessary...
encoder.add(Flatten()) # (batch, 2304)
encoder.add(Dense(512)) # 2304 -> 512
encoder.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 50, 50, 32)        896

 max_pooling2d (MaxPooling2D  (None, 25, 25, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 25, 25, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 conv2d_3 (Conv2D)           (None, 6, 6, 256)         295168

 max_pooling2d_3 (MaxPooling  (None, 3, 3, 256)        0
 2D)
```

```
 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 512)               1180160

=================================================================
Total params: 1,568,576
Trainable params: 1,568,576
Non-trainable params: 0
_________________________________________________________________
```

```
# Decoder: we expand the compressed representation
decoder = Sequential()
decoder.add(Dense(2304, input_shape=[512])) # 512 -> 2304
# Reshape: un-flatten the compressed representation
#decoder.add(Reshape([3, 3, 128], input_shape=[1152]))
decoder.add(Reshape([3, 3, 256]))
# (3, 3, 128) -> (6, 6, 64)
# W_out = ((W_in - 1) * strides[0] + kernel_size[0] - 2 * padding[0] + output_padding[0])
# W_out = (12-1)*2 + 3 - 2*P + OP
decoder.add(Conv2DTranspose(filters=128,
                            kernel_size=2,
                            strides=2,
                            padding="same",
                            activation="relu"))
# (6, 6, 64) -> (12, 12, 32)
decoder.add(Conv2DTranspose(filters=64,
                            kernel_size=2,
                            strides=2,
                            padding="same",
                            activation="relu"))
# (12, 12, 32) -> (25, 25, 16)
decoder.add(Conv2DTranspose(filters=32,
```

```
                            activation="relu"))
# (6, 6, 64) -> (12, 12, 32)
decoder.add(Conv2DTranspose(filters=64,
                            kernel_size=2,
                            strides=2,
                            padding="same",
                            activation="relu"))
# (12, 12, 32) -> (25, 25, 16)
decoder.add(Conv2DTranspose(filters=32,
                            kernel_size=3,
                            strides=2,
                            padding="valid",
                            activation="relu"))
# (25, 25, 16) -> (50, 50, 3)
decoder.add(Conv2DTranspose(filters=3,
                            kernel_size=2,
                            strides=2,
                            padding="same",
                            activation="relu"))

decoder.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 2304)              1181952

 reshape (Reshape)           (None, 3, 3, 256)         0

 conv2d_transpose (Conv2DTra  (None, 6, 6, 128)        131200
 nspose)

 conv2d_transpose_1 (Conv2DT  (None, 12, 12, 64)       32832
```

```
 conv2d_transpose_1 (Conv2DT  (None, 12, 12, 64)       32832
 ranspose)

 conv2d_transpose_2 (Conv2DT  (None, 25, 25, 32)       18464
 ranspose)

 conv2d_transpose_3 (Conv2DT  (None, 50, 50, 3)        387
 ranspose)

=================================================================
Total params: 1,364,835
Trainable params: 1,364,835
Non-trainable params: 0
_________________________________________________________________
```

```
# Autoencoder
autoencoder = Sequential([encoder,decoder])

autoencoder.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 512)               1568576

 sequential_1 (Sequential)   (None, 50, 50, 3)         1364835

=================================================================
Total params: 2,933,411
Trainable params: 2,933,411
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.callbacks import EarlyStopping

# Early stopping when validation loss stops decreasing
# Arguments:
# - monitor: value to be monitored -> val_loss: loss of validaton data
# - mode: min -> training stops when monitored value stops decreasing
# - patience: number of epochs with no improvement after which training will be stopped
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
```

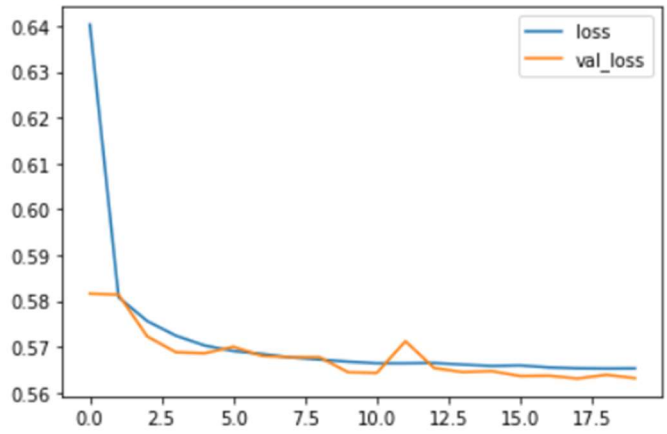
```
# We use the BINARY cross-entropy loss
# because we want to know whether the input and output images are equivalent
autoencoder.compile(loss='binary_crossentropy', # mse
                    optimizer='rmsprop')
```

```
# Train
autoencoder.fit(x_train,
                x_train,
                epochs=20,
                validation_split=0.2,
                batch_size=32,
                shuffle=True,
                callbacks=[early_stop])
```

```
Epoch 1/20
1500/1500 [==============================] - 28s 13ms/step - loss: 0.6403 - val_loss: 0.5816
Epoch 2/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5808 - val_loss: 0.5814
Epoch 3/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5756 - val_loss: 0.5723
Epoch 4/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5724 - val_loss: 0.5688
Epoch 5/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5703 - val_loss: 0.5686
Epoch 6/20
1500/1500 [==============================] - 17s 11ms/step - loss: 0.5691 - val_loss: 0.5700
Epoch 7/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5684 - val_loss: 0.5680
Epoch 8/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5677 - val_loss: 0.5677
Epoch 9/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5672 - val_loss: 0.5677
Epoch 10/20
Epoch 9/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5672 - val_loss: 0.5677
Epoch 10/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5667 - val_loss: 0.5644
Epoch 11/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5664 - val_loss: 0.5643
Epoch 12/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5664 - val_loss: 0.5712
Epoch 13/20
1500/1500 [==============================] - 17s 11ms/step - loss: 0.5665 - val_loss: 0.5654
Epoch 14/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5661 - val_loss: 0.5645
Epoch 15/20
1500/1500 [==============================] - 17s 11ms/step - loss: 0.5658 - val_loss: 0.5647
Epoch 16/20
1500/1500 [==============================] - 17s 11ms/step - loss: 0.5659 - val_loss: 0.5636
Epoch 17/20
1500/1500 [==============================] - 17s 11ms/step - loss: 0.5655 - val_loss: 0.5637
Epoch 18/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5653 - val_loss: 0.5631
Epoch 19/20
1500/1500 [==============================] - 16s 11ms/step - loss: 0.5652 - val_loss: 0.5639
Epoch 20/20
1500/1500 [==============================] - 17s 11ms/step - loss: 0.5653 - val_loss: 0.5631
<keras.callbacks.History at 0x7fb5939a0ac0>
```

```
losses = pd.DataFrame(autoencoder.history.history)
losses.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb592a77e50>
```



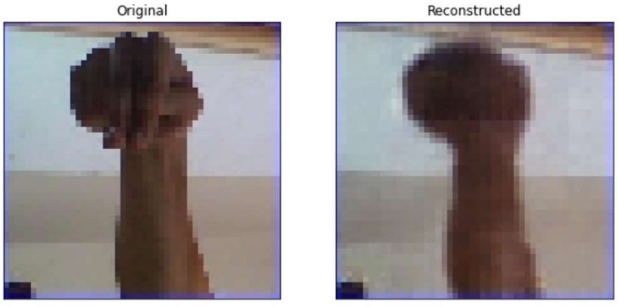

```
x_test_enc_0 = encoder.predict(np.expand_dims(x_test[0], axis=0))
print(x_test_enc_0)
```

```
1/1 [==============================] - 0s 211ms/step
[[ 6.40319958e-02  4.25598323e-02 -2.92496029e-02 -1.83627084e-02
   6.63838023e-03  1.48624806e-02  3.08103021e-03 -8.04711413e-03
  -1.71038546e-02  1.01984339e-02  2.05837432e-02 -1.04462905e-02
   1.06213326e-02 -1.21105164e-02 -3.15149128e-02  1.79034937e-02
  -2.44427118e-02 -2.49820482e-03  8.46669264e-03 -8.72136932e-03
  -1.09536352e-03  9.62975249e-03  1.64383054e-02  2.18578316e-02
  -3.63428146e-04 -5.15409280e-04  5.83062768e-02  1.76272988e-02
   5.99097228e-03  2.05886140e-02  1.83729138e-02  2.04667188e-02
  -4.54637595e-03 -3.18964347e-02  9.30061750e-03 -1.13678277e-02
  -9.97534394e-03 -2.19083130e-02  1.25905387e-02  3.67516242e-02
   7.41622457e-03 -7.05099106e-03 -7.24637788e-03  3.35478187e-02
  -1.11051230e-02 -6.48978027e-03  5.55772334e-04  1.39158759e-02
  -1.87839586e-02 -2.00955328e-02  2.34440286e-02  6.70470437e-03
  -8.38374253e-04  1.91740822e-02 -3.78771797e-02  9.11231712e-03
  -1.41417691e-02  1.73400287e-02 -6.00988138e-03  2.46732645e-02
   9.70256794e-03 -6.28837012e-03  1.06869815e-02  1.36228232e-02
   1.19295120e-02 -2.11731307e-02  1.61577128e-02 -6.53648283e-04
   1.44341514e-02  1.67108122e-02 -6.81834808e-03 -1.12451566e-02
   1.23809222e-02 -4.39880509e-03  3.45890969e-02 -3.10720000e-02
  -2.31309608e-02  1.67202111e-02 -1.16612203e-02  1.82861984e-02
  -1.30423065e-02  3.28779081e-03 -5.20270392e-02  6.23836275e-03
   3.49616050e-03 -8.10684822e-03  1.52900033e-02  1.26747768e-02
  -2.50589754e-03 -7.34254904e-03 -2.73997784e-02  8.34686495e-03
   4.95480746e-02 -8.81912746e-03  2.39179973e-02 -1.55034205e-02
   2.59685852e-02 -4.60348325e-03 -3.07247806e-02  3.97056229e-02
   1.35227600e-02 -5.29309083e-03  2.21932810e-02 -4.49494924e-04
   1.64328106e-02  3.31521221e-03  9.74468328e-03  6.79124240e-03
   7.14538852e-03  4.13148897e-04 -1.05538848e-03  1.93623211e-02
   1.37470178e-02  2.59515792e-02 -2.13645026e-02  8.84152949e-04
   1.49654690e-02 -4.90718149e-03 -1.16483038e-02 -3.43340710e-02
```

MILESTONE 6: LOAD AND TEST THE MODEL

```
x_org = x_test[2]
x_test_rec = autoencoder.predict(np.expand_dims(x_org, axis=0))

fig = plt.figure(figsize=(10,5))
ax1 = fig.add_subplot(1, 2, 1, xticks=[], yticks=[])
ax1.imshow(np.squeeze(x_org))
ax1.set_title("Original")
ax1 = fig.add_subplot(1, 2, 2, xticks=[], yticks=[])
ax1.imshow(np.squeeze(x_test_rec[0]))
ax1.set_title("Reconstructed")
```

```
1/1 [==============================] - 0s 235ms/step
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Text(0.5, 1.0, 'Reconstructed')
```




```
x_train_enc = encoder.predict(x_train)
x_test_enc = encoder.predict(x_test)
print(x_train_enc.shape)
print(x_test_enc.shape)
```

```
1875/1875 [==============================] - 5s 3ms/step
469/469 [==============================] - 1s 3ms/step
(60000, 512)
(15000, 512)
```

```
print(x_train_enc.shape)
print(x_test_enc.shape)
data_dict = {
    "x_train_enc": x_train_enc,
    "x_test_enc": x_test_enc,
    "y_train_OH": y_train_OH,
    "y_train": y_train,
    "y_test_OH": y_test_OH,
    "y_test": y_test,
    "labels": labels
}
```

```
(60000, 512)
(15000, 512)
```

```
#import pickle
pickle.dump(data_dict, open('data_dict.pkl','wb')) # wb: write bytes
```

```
data_dict = pickle.load(open('data_dict.pkl','rb')) # rb: read bytes
x_train_enc = data_dict["x_train_enc"]
```

```
data_dict = pickle.load(open('data_dict.pkl','rb')) # rb: read bytes
x_train_enc = data_dict["x_train_enc"]
x_test_enc = data_dict["x_test_enc"]
y_train_OH = data_dict["y_train_OH"]
y_train = data_dict["y_train"]
y_test_OH = data_dict["y_test_OH"]
y_test = data_dict["y_test"]
labels = data_dict["labels"]
```

```
# Scale (for the logistic regression)
from sklearn.preprocessing import StandardScaler

scaler_enc = StandardScaler()
x_train_enc_scaled = scaler_enc.fit_transform(x_train_enc)
x_test_enc_scaled = scaler_enc.transform(x_test_enc)
```

```
search.fit(x_train_enc_scaled, y_train)
search_best = search.best_estimator_

print(search.best_score_)
print(search.best_params_)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
[CV 1/3; 1/4] START max_depth=10, n_estimators=100.............................
[CV 1/3; 1/4] END max_depth=10, n_estimators=100;, score=0.856 total time=  14.4s
[CV 2/3; 1/4] START max_depth=10, n_estimators=100.............................
[CV 2/3; 1/4] END max_depth=10, n_estimators=100;, score=0.859 total time=  13.6s
[CV 3/3; 1/4] START max_depth=10, n_estimators=100.............................
[CV 3/3; 1/4] END max_depth=10, n_estimators=100;, score=0.866 total time=  13.2s
[CV 1/3; 2/4] START max_depth=10, n_estimators=150.............................
[CV 1/3; 2/4] END max_depth=10, n_estimators=150;, score=0.872 total time=  19.9s
[CV 2/3; 2/4] START max_depth=10, n_estimators=150.............................
[CV 2/3; 2/4] END max_depth=10, n_estimators=150;, score=0.873 total time=  20.8s
[CV 3/3; 2/4] START max_depth=10, n_estimators=150.............................
[CV 3/3; 2/4] END max_depth=10, n_estimators=150;, score=0.879 total time=  19.9s
[CV 1/3; 3/4] START max_depth=25, n_estimators=100.............................
[CV 1/3; 3/4] END max_depth=25, n_estimators=100;, score=0.986 total time=  19.9s
[CV 2/3; 3/4] START max_depth=25, n_estimators=100.............................
[CV 2/3; 3/4] END max_depth=25, n_estimators=100;, score=0.988 total time=  19.7s
[CV 3/3; 3/4] START max_depth=25, n_estimators=100.............................
[CV 3/3; 3/4] END max_depth=25, n_estimators=100;, score=0.987 total time=  19.7s
[CV 1/3; 4/4] START max_depth=25, n_estimators=150.............................
[CV 1/3; 4/4] END max_depth=25, n_estimators=150;, score=0.988 total time=  29.6s
[CV 2/3; 4/4] START max_depth=25, n_estimators=150.............................
[CV 2/3; 4/4] END max_depth=25, n_estimators=150;, score=0.989 total time=  29.3s
[CV 3/3; 4/4] START max_depth=25, n_estimators=150.............................
[CV 3/3; 4/4] END max_depth=25, n_estimators=150;, score=0.988 total time=  29.3s
0.9882666666666666
{'max_depth': 25, 'n_estimators': 150}
```

```
# Get predicted probabilities for test dataset
y_probs = search_best.predict_proba(x_test_enc_scaled)

# Get predicted labels for test dataset
y_preds = search_best.predict(x_test_enc_scaled)
#y_preds = np.argmax(yp, axis=1)
```

```
y_preds.shape
```

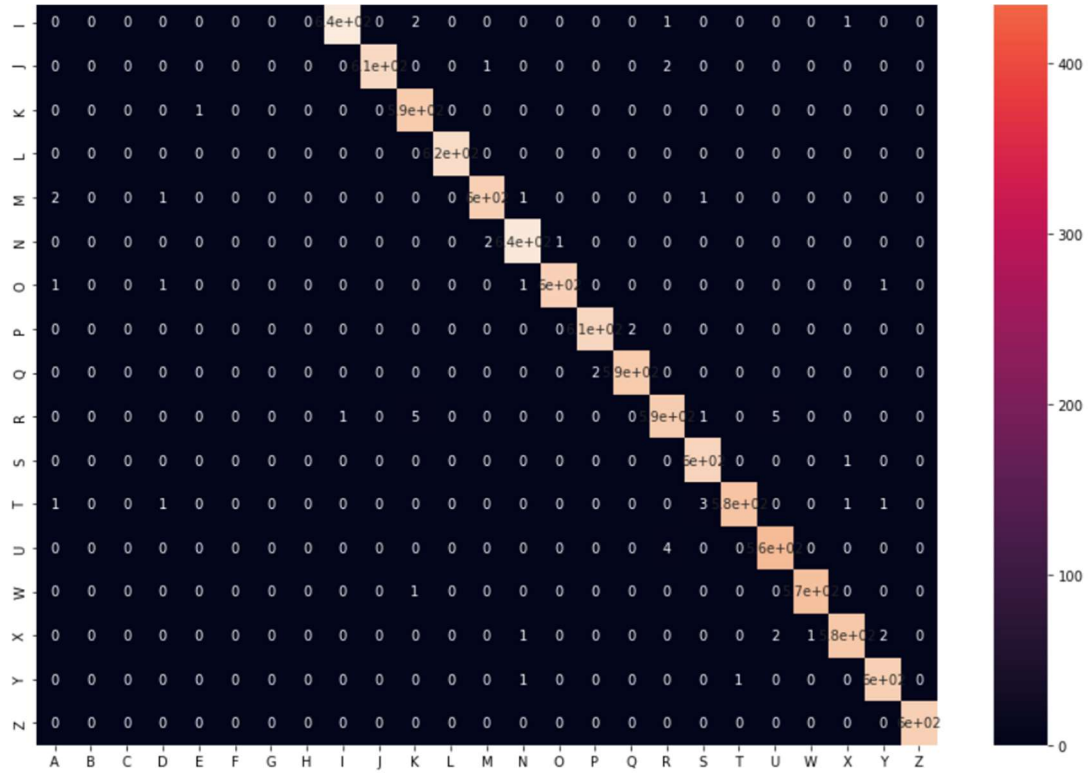
```
(15000,)
```

```
y_preds
```

```
array([ 2,  7, 13, ...,  1, 18, 16])
```

```
outcome = pd.DataFrame(np.array([y_test,y_preds]).T, columns=['true', 'pred'])
id2label = {i:value for i,value in enumerate(labels)}
outcome = outcome[['true','pred']].replace(id2label)
```

```
plt.figure(figsize=(15,15))
sns.heatmap(confusion_matrix(outcome['true'],outcome['pred']),
            xticklabels=labels,
            yticklabels=labels,
            annot=True);
```

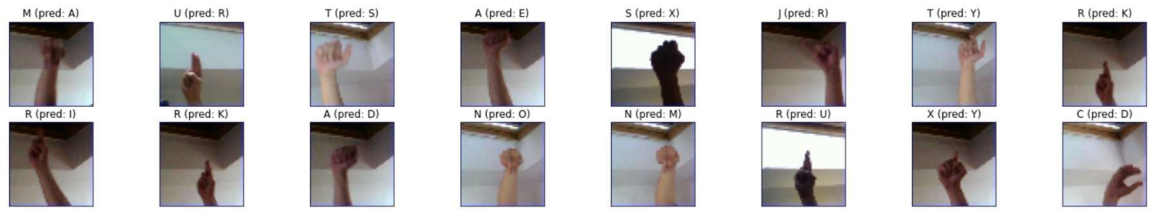


```
# Indices corresponding to test images which were mislabeled
bad_test_idxs = np.where(y_preds!=y_test)[0]
```

```
len(bad_test_idxs)
```

```
77
```

```
# Print a random subset of mislabeled examples
fig = plt.figure(figsize=(25,4))
subset_size = 16
try:
    assert subset_size < len(bad_test_idxs)
except AssertionError:
    print("Use a smaller subset size!")
# Pick a random subset
z = int(np.random.rand()*(len(bad_test_idxs)-subset_size))
subset = bad_test_idxs[z:(z+subset_size)]
for i, idx in enumerate(subset):
    ax = fig.add_subplot(2, int(np.ceil(len(subset)/2)), i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    ax.set_title("{} (pred: {})".format(labels[y_test[idx]], labels[y_preds[idx]]))
```



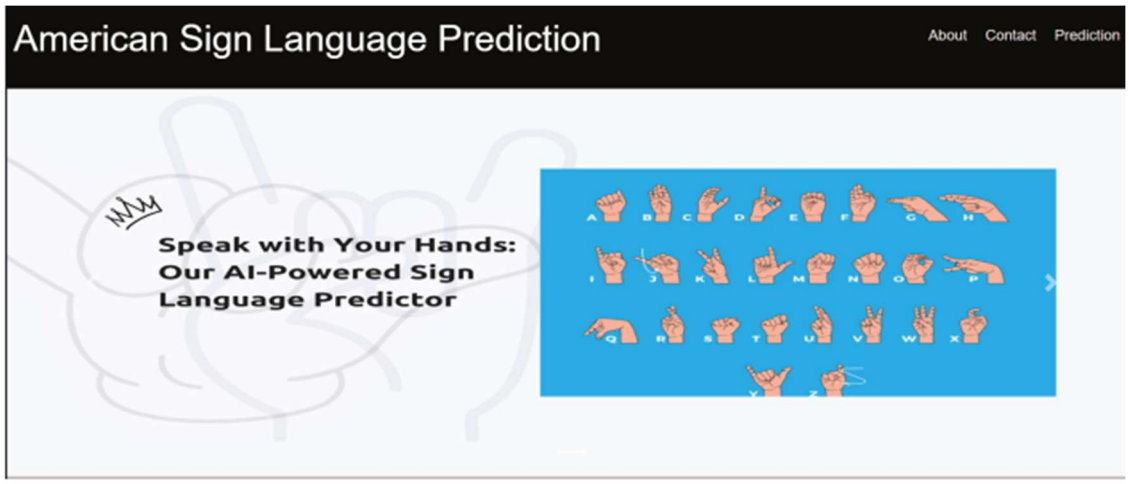
Milestone 7: APPLICATION BUILDING

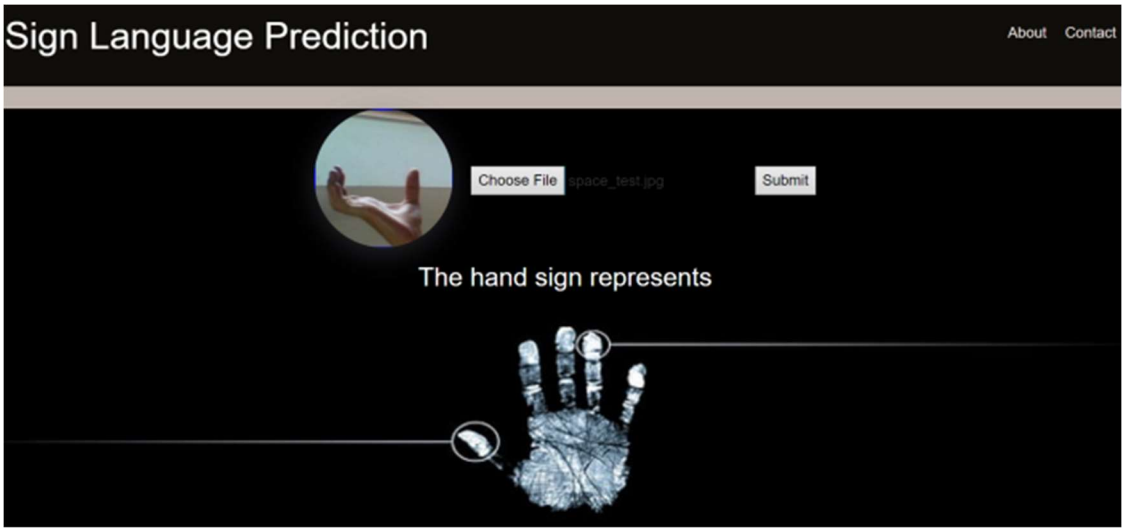
Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.

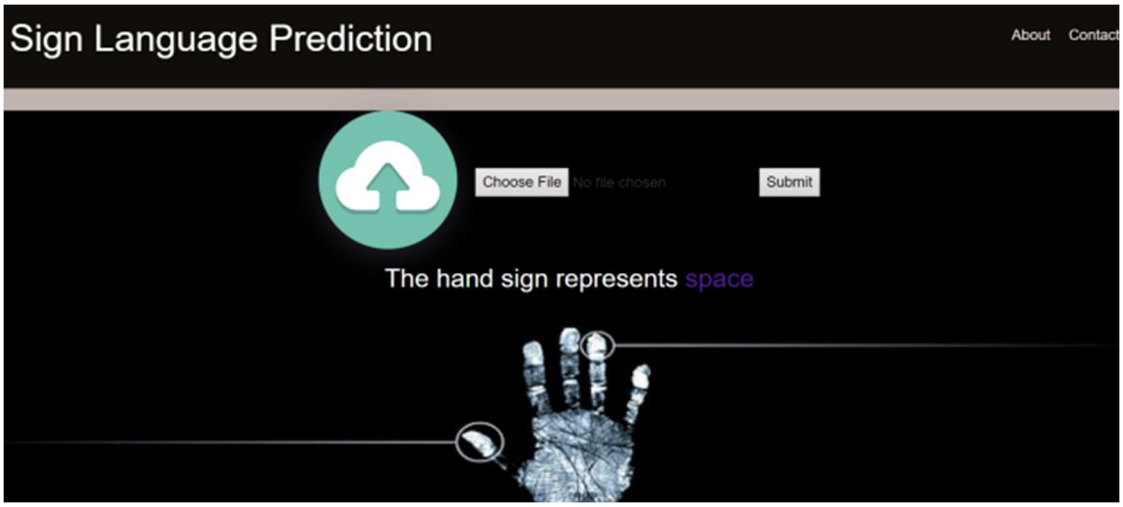
In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Image” button, the next page is opened where the user chooses the i mage and predicts the output.

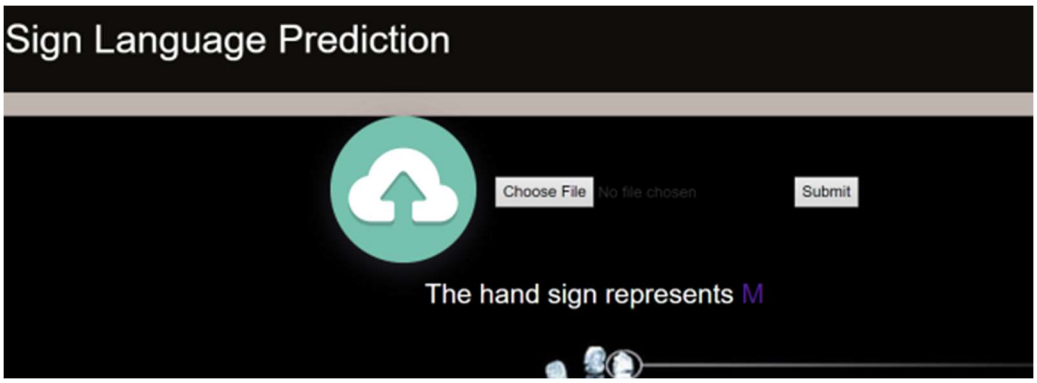
Activity 1: Create HTML Pages

- We use HTML to create the front end part of the web page.
- Here, we have created 3 HTML pages- index.html, prediction.html, and logout.html
- index.html displays the home page.
- prediction.html displays the prediction page o logout.html give the result For more information regarding HTML https://www.w3schools.com/html/
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.


American Sign Language Prediction
About
Contact
Prediction
Speak with Your Hands:
Our AI-Powered Sign
Language Predictor


Sign Language Prediction
About
Contact
Choose File
space_test.jpg
Submit
The hand sign represents


Sign Language Prediction
About
Contact
Choose File
No file chosen
Submit
The hand sign represents space



This is a Python script for a Flask web application that loads a pre-trained deep learning model for image classification and makes predictions on images uploaded by the user. The app has several routes, such as the home page ('/'), the prediction page ('/prediction.html'), and the logout page ('/logout.html'). The main prediction functionality is implemented in the '/result' route, where the uploaded image is loaded, preprocessed, and passed through the model for prediction. The predicted result is then displayed on the prediction page. The app can be run by executing the script, and it will start a local server accessible through a web browser.

```
lask > app.py > index
 2   import numpy as np
 3   import pandas as pd
 4   from PIL import Image
 5   import os
 6   import tensorflow as tf
 7   from flask import Flask, app,request,render_template
 8   from keras.models import Model
 9   from keras.preprocessing import image
10   from tensorflow.python.ops.gen_array_ops import Concat
11   from keras.models import load_model
12   #Loading the model
13   model=load_model(r"asl_vgg16_best_weights.h5",compile=False)
14   app=Flask(__name__)
15
16   #default home page or route
17   @app.route('/')
18   def index():
19       return render_template('index.html')
```

```python
def prediction():
    return render_template('prediction.html')
@app.route('/index.html')
def home():
    return render_template("index.html")
@app.route('/logout.html')
def logout():
    return render_template('logout.html')
def preprocess_image(image_path):
    img = Image.open(image_path)
    img = img.resize((64, 64))
    img = np.array(img)
    img = tf.keras.applications.mobilenet_v2.preprocess_input(img)
    return img
@app.route('/result',methods=["GET","POST"])
def res():
    if request.method=="POST":
        f=request.files['image']
        basepath=os.path.dirname(__file__) #getting the current path
```

```python
ask > app.py > prediction
@app.route('/result',methods=["GET","POST"])
def res():
    if request.method=="POST":
        f=request.files['image']
        basepath=os.path.dirname(__file__) #getting the current path i.e where app.py
        #print("current path",basepath)
        filepath=os.path.join(basepath,'uploads',f.filename) #from anywhere in the sys
        #print("upload folder is",filepath)
        f.save(filepath)
        labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N'
        img = preprocess_image(filepath)
        predictions = model.predict(np.array([img]))
        predicted_class = labels[np.argmax(predictions)]
        return render_template('prediction.html', pred=predicted_class)
""" Running our application """
if __name__ == "__main__":
    app.run(debug=True)
```

To run this Flask application, simply navigate to the project directory in the terminal and run the command "python app.py". This will start the Flask server, and you can access the web application by visiting the local host address in your web browser. Once you upload an image and submit the form, the application will use the trained model to predict the species of the plant in the image and display the result on the page.