Machine Learning
Final Project

American Sign Language Recognition using Convolutional Neural Network

By

Aditya Gidda

Jaswanth Das Gunturu

Siddhant Sharma

Motivation

Our motivation for doing this project for out final in the Machine Learning was that, when we were trying to find a data set which is really new. So, we thought of using a new topic for the project. We saw a video online about the American sign Language (ASL) where we found the dumb and deaf people were using different types of signs to communicate between each other. It was very motivational for us and we started to do some research on the ASL and it was very tough to remember even some signs, which made us think why don't we do a project on the signs. It was the time when we thought we can do a project regarding the American Sign Language(ASL). So, we started to search for the data sets which are compatible to the methods we use in machine learning. We found a 2.2GB data from NICO's Homepage which had the ASL Finger Spelling Dataset. We found that the Convolutional Neural Networking(CNN) the best method we could use for this type of data set.

Introduction

A real-time sign language translator is an important milestone in facilitating communication between the deaf community and the public. We hereby present the development and implementation of an **American Sign Language (ASL)** fingerspelling translator based on a convolutional neural network. American Sign Language (ASL) is a natural language that serves as the predominant sign language of Deaf communities in the United States and most of Anglophone Canada. ASL originated in the early 19th century in the American School for the Deaf (ASD) in Hartford, Connecticut, from a situation of language contact. Since then, ASL use has propagated widely via schools for the deaf and Deaf community organizations. Despite its wide use, no accurate count of ASL users has been taken, though reliable estimates for American ASL users range from 250,000 to 500,000 persons, including a number of children of deaf adults. ASL users face stigma due to beliefs in the superiority of oral language to sign language, compounded by the fact that ASL is often glossed in English due to the lack of a standard writing system.

ASL signs have several phonemic components, including movement of the face and torso as well as the hands. ASL is not a form of pantomime, but iconicity does play a larger role in ASL than in spoken languages. English loan words are often borrowed through fingerspelling, although ASL grammar is unrelated to that of English. ASL has verbal agreement and aspectual marking and has a productive system of forming agglutinative classifiers. Many linguists believe ASL to be a subject–verb–object (SVO) language, but there are several alternative proposals to account for ASL word order.

We produced a robust model that consistently classifies letters a-I for 5 different users. Given the limitations of the datasets and the encouraging results achieved, we are confident that with further research and more data, we can produce a fully generalizable translator for all ASL letters.

Approach and Methods:

Classifier Development:

Transfer Learning Our ASL letter classification is done using a convolutional neural network (CNN or ConvNet). CNNs are machine learning algorithms that have seen incredible success in handling a variety of tasks related to processing videos and images. Since 2012, the field has experienced an explosion of growth and applications in image classification, object localization, and object detection. A primary advantage of utilizing such techniques stems from CNNs abilities to learn features as well as the weights corresponding to each feature. Like other machine learning algorithms, CNNs seek to optimize some objective function, specifically the loss function. Using a SoftMax-based classification head allows us to output values akin to probabilities for each ASL letter. This differs from another popular choice: the SVM loss. 227 Using an SVM classification head would result in scores for each ASL letter that would not directly map to probabilities. These probabilities afforded to us by the SoftMax loss allow us to more intuitively interpret our results and prove useful when running our classifications through a language model.

Using a SoftMax-based classification head allows us to output values akin to probabilities for each ASL letter. This differs from another popular choice: the SVM loss. Using an SVM classification head would result in scores for each ASL letter that would not directly map to probabilities. These probabilities afforded to us by the SoftMax loss allow us to more intuitively interpret our results and prove useful when running our classifications through a language model.

Transfer Learning is a machine learning technique where models are trained on (usually) larger data sets and refactored to fit more specific or niche data. This is done by recycling a portion of the weights from the pre-trained model and reinitializing or otherwise altering weights at shallower layers. The most basic example of this would be a fully trained network whose final classification layer weights have been reinitialized to be able to classify some new set of data. The primary benefits of such a technique are its less demanding time and data requirements. However, the challenge in transfer learning stems from the differences between the original data used to train and the new data being classified. Larger differences in these data sets often require re-initializing or increasing learning rates for deeper layers in the net.

Methodology:

We first obtained different images of five different persons showing the symbols of a, b, c, d, e, f, g, h and i. And after that we resize all the images to a resolution of about 32*32. Here the five persons are considered as different classes.

Now we divided the dataset into two parts X and y, where X denotes the images and y denotes the labels.

We here assigned two paths for the code unlike we do regularly, as the images are in different files in a folder, we created root path for the initial folder and final path for the inner images. We use cv2 library to read the images here.

We Split the data into two parts as training data and test data in the ratio 80: 20 using Sklearn.

Mini-Batcher is being used which is a function that takes an input in batches so that we can iterate through all the divisions.

We used the library Tensor flow so that we can use the Convolutional Neural Networks(CNN) for our project.

Convolutional Neural Network:

We use four layers to identify the images

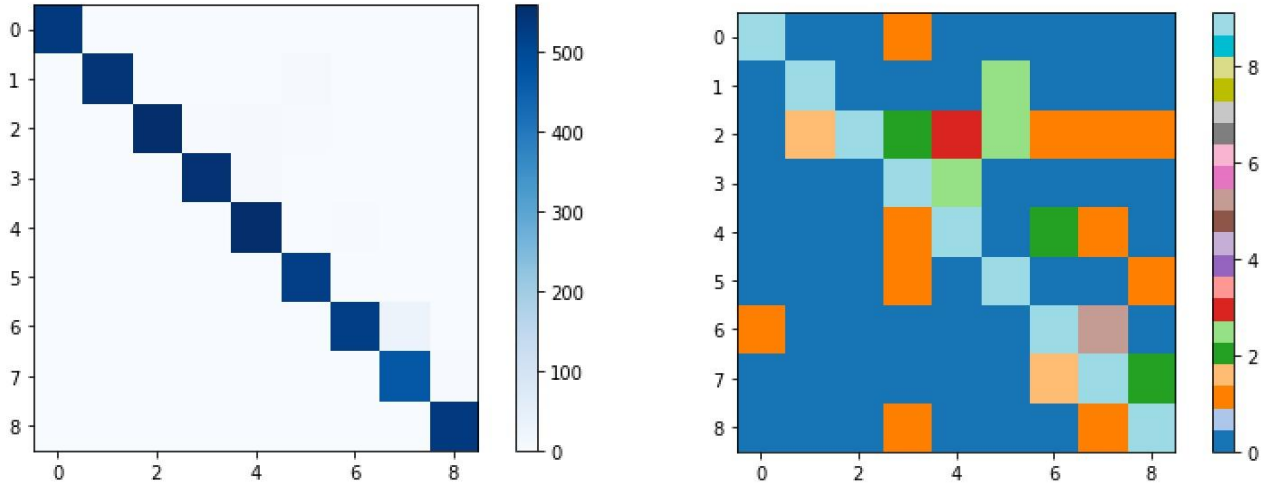
Convolutional Layer: Convolution layer has no. of filters that does the convolutional operations. We use the convolutional layer two times where we find low activation fields for the first convolution and high activation fields for the second convolution.

Maxpool: Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.

Activation Layer: Activation layer is an activation function that decides the final values of a neuron. It makes the value of a cell ideal, i.e., by rounding of the value to 1 if it is 0.7 or more else rounding it to 0.

Fully-Connected Layer: It is a layer which takes in the output from the second convolutional layer which are high activation fields and compares with all the 9 classes we've chosen and gives a probability of them matching with the classes.

SoftMax: The softmax function is often used in the final layer of a neural network-based classifier. Such networks are commonly trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression. Which compares the non-linear probabilities and gives the best class.



Accuracy using Color Bars

Classification Report:

	Precision	Recall	F1-score	Support
0	1	1	1	533
1	1	0.99	0.99	547
2	1	0.97	0.98	577
3	0.99	0.99	0.99	555
4	0.98	0.99	0.99	562
5	0.98	1	0.99	523
6	0.99	0.94	0.96	557
7	0.93	0.99	0.96	473
8	0.99	1	0.99	534
avg/Total	0.98	0.98	0.98	4861

Confusion Matrix:

[[532 0 0 1 0 0 0 0 0]
[0 542 0 0 0 5 0 0 0]
[0 2 559 3 6 4 1 1 1]
[0 0 0 550 5 0 0 0 0]
[0 0 0 1 557 0 3 1 0]
[0 0 0 1 0 521 0 0 1]
[1 0 0 0 0 0 524 32 0]
[0 0 0 0 0 0 2 468 3]
[0 0 0 1 0 0 0 1 532]]

Accuracy: 0.984365356922444 i.e., 98.43%

Appendix:

Code:

```
import matplotlib.pyplot as plt
import glob
from skimage.color import rgb2lab
from skimage.transform import resize
from collections import namedtuple
import numpy as np
from string import ascii_lowercase
import os
import cv2
np.random.seed(101)
N_CLASSES = 9
RESIZED_IMAGE = (16,16)
Dataset = namedtuple('Dataset', ['X', 'y'])
```

```
def read_dataset_ppm(rootpath, n_labels, resize_to):
  images = []
  labels = []
  small = ['a','b','c','d','e','f','g','h','i']
  big = ['A','B','C','D','E']
  for each in big:
    for c,s in enumerate(small):
     full_path = rootpath + '/' + each + '/' + s + '/'
     for img_name in glob.glob(full_path + "color*.png"):
      img = cv2.imread(img_name).astype(np.float16)
       reimg = cv2.resize(img, resize_to)
      label = np.zeros((n_labels, ), dtype=np.float16)
      label[c] = 1.0

      images.append(reimg.astype(np.float16))
      labels.append(label)
  return Dataset(X = np.array(images).astype(np.float16),
                 y = np.matrix(labels).astype(np.float16))
dataset = read_dataset_ppm('dataset5', N_CLASSES, RESIZED_IMAGE)
print(dataset.X.shape)
print(dataset.y.shape)
from sklearn.model_selection import train_test_split
idx_train, idx_test = train_test_split(range(dataset.X.shape[0]), test_size=0.20,
random_state=101)
X_train = dataset.X[idx_train, :, :, :]
X_test = dataset.X[idx_test, :, :, :]
y_train = dataset.y[idx_train, :]
y_test = dataset.y[idx_test, :]
print(X_train.shape)
print(y_train.shape)
```

```
print(X_test.shape)
print(y_test.shape)
def minibatcher(X, y, batch_size, shuffle):
  assert X.shape[0] == y.shape[0]
  n_samples = X.shape[0]
  if shuffle:
    idx = np.random.permutation(n_samples)
  else:
    idx = list(range(n_samples))
  for k in range(int(np.ceil(n_samples/batch_size))):
    from_idx = k*batch_size
    to_idx = (k+1)*batch_size
    yield X[idx[from_idx:to_idx], :, :, :], y[idx[from_idx:to_idx], :]
for mb in minibatcher(X_train, y_train, 20000, True):
    print(mb[0].shape, mb[1].shape)
import tensorflow as tf
def fc_no_activation_layer(in_tensors, n_units):
  w = tf.get_variable('fc_W',
    [in_tensors.get_shape()[1], n_units],
    tf.float32,
    tf.contrib.layers.xavier_initializer())
  b = tf.get_variable('fc_B',
    [n_units, ],
    tf.float32,
    tf.constant_initializer(0.0))
  return tf.matmul(in_tensors, w) + b
def fc_layer(in_tensors, n_units):
  return tf.nn.leaky_relu(fc_no_activation_layer(in_tensors, n_units))
def maxpool_layer(in_tensors, sampling):
```

```
  return tf.nn.max_pool(in_tensors, [1, sampling, sampling, 1], [1, sampling, sampling, 1], 'SAME')
def conv_layer(in_tensors, kernel_size, n_units):
  w = tf.get_variable('conv_W',
    [kernel_size, kernel_size, in_tensors.get_shape()[3], n_units],
    tf.float32,
    tf.contrib.layers.xavier_initializer())
  b = tf.get_variable('conv_B',
    [n_units, ],
    tf.float32,
    tf.constant_initializer(0.0))
  return tf.nn.leaky_relu(tf.nn.conv2d(in_tensors, w, [1, 1, 1, 1], 'SAME') + b)

def dropout(in_tensors, keep_proba, is_training):
  return tf.cond(is_training, lambda: tf.nn.dropout(in_tensors, keep_proba), lambda: in_tensors)
def model(in_tensors, is_training):
  # First layer: 5x5 2d-conv, 32 filters, 2x maxpool, 20% drouput
  with tf.variable_scope('l1'):
    l1 = maxpool_layer(conv_layer(in_tensors, 5, 32), 2)
    l1_out = dropout(l1, 0.8, is_training)
  # Second layer: 5x5 2d-conv, 64 filters, 2x maxpool, 20% drouput
  with tf.variable_scope('l2'):
    l2 = maxpool_layer(conv_layer(l1_out, 5, 64), 2)
    l2_out = dropout(l2, 0.8, is_training)
  with tf.variable_scope('flatten'):
    l2_out_flat = tf.layers.flatten(l2_out)
  # Fully collected layer, 1024 neurons, 40% dropout
  with tf.variable_scope('l3'):
    l3 = fc_layer(l2_out_flat, 1024)
```

```
    l3_out = dropout(l3, 0.6, is_training)
  # Output
  with tf.variable_scope('out'):
    out_tensors = fc_no_activation_layer(l3_out, N_CLASSES)
  return out_tensors
#To get the accuracy, precision,recall and F-1 measure
from sklearn.metrics import classification_report, confusion_matrix
def train_model(X_train, y_train, X_test, y_test, learning_rate, max_epochs, batch_size):
  in_X_tensors_batch = tf.placeholder(tf.float32, shape = (None, 32,32, 3))
  in_y_tensors_batch = tf.placeholder(tf.float32, shape = (None, N_CLASSES))
  is_training = tf.placeholder(tf.bool)
  logits = model(in_X_tensors_batch, is_training)
  out_y_pred = tf.nn.softmax(logits)
  loss_score = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
labels=in_y_tensors_batch)
  loss = tf.reduce_mean(loss_score)
  optimizer = tf.train.AdamOptimizer(learning_rate).minimize(loss)
  with tf.Session() as session:
    session.run(tf.global_variables_initializer())
    for epoch in range(max_epochs):
      print("Epoch=", epoch)
      tf_score = []
      for mb in minibatcher(X_train, y_train, batch_size, shuffle = True):
        tf_output = session.run([optimizer, loss],
                                feed_dict = {in_X_tensors_batch : mb[0],
                                             in_y_tensors_batch : mb[1],
                                             is_training : True})
        tf_score.append(tf_output[1])
      print(" train_loss_score=", np.mean(tf_score))
      tf.summary.scalar("loss_plot",np.mean(tf_score))
```

```
    # after the training is done, time to test it on the test set
    print("TEST SET PERFORMANCE")
    y_test_pred, test_loss  = session.run([out_y_pred, loss],
                                feed_dict = {in_X_tensors_batch : X_test,
                                        in_y_tensors_batch : y_test,
                                        is_training : False})
    print(" test_loss_score=", test_loss)
    y_test_pred_classified = np.argmax(y_test_pred,axis=1).astype(np.int32)
    y_test_true_classified = np.argmax(y_test,axis=1).astype(np.int32)
    print(classification_report(y_test_true_classified, y_test_pred_classified))
    from sklearn.metrics import accuracy_score
    print(accuracy_score(y_test_true_classified, y_test_pred_classified))

    cm = confusion_matrix(y_test_true_classified, y_test_pred_classified)
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.colorbar()
    plt.tight_layout()
    plt.show()
    # And the log2 version, to emphasize the misclassifications
    plt.imshow(np.log2(cm + 1), interpolation='nearest', cmap=plt.get_cmap("tab20"))
    plt.colorbar()
    plt.tight_layout()
    plt.show()
tf.reset_default_graph()
train_model(X_train, y_train, X_test, y_test, 0.001, 10, 256)
```

References:

Data Link: http://empslocal.ex.ac.uk/people/staff/np331/index.php?section=FingerSpellingDataset

[1] Mitchell, Ross; Young, Travas; Bachleda, Bellamie; Karchmer, Michael (2006). "How Many People Use ASL in the United States?: Why Estimates Need Updating" (PDF). Sign Language Studies (Gallaudet University Press.) 6 (3). ISSN 0302-1475. Retrieved November 27, 2012.

[2] Singha, J. and Das, K. “Hand Gesture Recognition Based on Karhunen-Loeve Transform”, Mobile and Embedded 232 Technology International Conference (MECON), January 17-18, 2013, India. 365-371.

[3] D. Aryanie, Y. Heryadi. American Sign Language-Based Finger-spelling Recognition using k-Nearest Neighbors Classifier. 3rd International Conference on Information and Communication Technology (2015) 533-536.