CAI 4104/6108: Machine Learning Engineering

Project Report: An ASL Ensemble

David Magda
(Point of Contact)
magdad@ufl.edu

Sushmitha Tirumalla
tirumalla.s@ufl.edu

Adam Horton
adam.horton@ufl.edu

Vaishnavi Kalva
vkalva@ufl.edu

Reema Solan
reema.solan@ufl.edu

April 25, 2024

1 Introduction

This project pertains to the multi-classification problem of static American Sign Language (ASL) alphabet signs. While this project has been done before, we are utilizing an ensemble approach with a CNN, transfer learning with MobileNet, KNN, and VGG. This approach should allow us to achieve a high accuracy regardless of the individual performance of the models.

The classification of ASL signs has been a common project for various machine learning uses as it has both a seemingly positive social impact and is, in reality, a simple multi-classification problem.

2 Approach: Dataset(s) & Pipeline(s)

To solve our problem of multi-classification, we decided to utilize an ensemble approach which we had not seen previously in the literature. We decided on using four models together: a Convulational Neural Network (CNN), a MobileNet (via transfer learning), a K-Nearest Neighbor model, and a Visual Geometry Group (VGG) model. These models, once trained, would be put into an ensemble that would utilize majority voting for classification.

The start of our pipeline included the loading and pre-processing of our data. We opted to only due pixel normalization for the data and flattening of the data that needed it. The neural networks used the image data rather than the flattened data.

The dataset we utilized was a dataset compiled by Kaggle user Akash. This dataset included 87,000 200x200 RGB images of the ASL alphabet as well as the signs for delete, space, and photos of nothing at all for training models [2]. This created 29 classes for images.

For the data, we split it into a 80%, 10%, 10% ratio for training, testing, and validation. We had 69,600 training samples, 8,700 validation samples, and 8,728 testing samples. Our data is equally distributed among all classes.

Once each directory was set up with the images, we created three data generators that allowed us to control the size of the image and implement any data augmentation if we wanted to do so later. Since each image was originally 200x200 pixels, we had to turn them into 32x32 due to memory constraints. We did this in the generator along with setting our class_mode to categorical as we had mutually exclusive labels.

Our pipeline also included the creation of multiple models to be done in an ensemble. We will discuss each of the models in the following subsections including their results and architecture. The models were made independently, tested, and then moved into the ensemble so that we could track their individual performance as well as the ensemble performance.

2.1 CNN

Our CNN had the architecture seen in Table 1. This allowed us to have 359,581 total parameters with 359,133 of those being trainable. We utilized the categorical crossentropy loss function with the adam optimizer.

2.2 Transfer Learning with MobileNet

MobileNets are an architecture type developed by Howard et al. as "light weight deep neural networks." This architecture consists of multiple convulational depthwise and pointwise layers with average pooling and the softmax

Layer
Conv2D
Max Pooling 2D
Batch Normalization
Conv2D
Max Pooling 2D
Batch Normalization
Conv2D
Max Pooling 2D
Batch Normalization
Flatten
Dense
Dense

Table 1: CNN Model Architecture

function at the end. The full architecture can be found in their paper [1]. We added two more layers on top of the MobileNet architecture. These were a global average pooling 2D later and a dense layer with softmax activation.

2.3 KNN

The KNN model required the data to be flattened. To do this, we looped through the batches from the generators and appended each flattened batch to a list while keeping the labels the same. The model itself had a grid search implemented for tuning the number of neighbors hyperparameter. The grid search presented the best n neighbors as 1 using the validation set.

2.4 Transfer Learning with VGG19

We utilized the Visual Geometry Group (VGG) model with 19 layers for our last model. This model is a deep convulational neural network generally used for image classification. This model was developed by Simonyan et al. (2015) [5] with an architecture of maxpools between groups of convulational layers, three hidden layers and a softmax activation. Our VGG also had some custom dense and dropout layers as well as early stopping implemented to help prevent overfitting.

3 Evaluation Methodology

Our evaluation consists of finding the F1-Score, Recall, and accuracy of each of our models separately and our ensemble as a whole. The F1-Score was chosen for its usefulness in classification problems and ability to give insight into precision and recall. Recall was chosen to find the rate at which true positives were chosen. Accuracy was chosen as a general performance measurement against other models.

As a baseline, we defer to Pathan et al. (2023) in their gathering of various models and their accuracies for this type of task. The first baseline we chose were various models from Kaggle users on the same dataset which ranged from 95.81% - 96.96%. The second baseline we chose was Pathan et al.'s multi-headed CNN as it performed better than the other models that they had referenced in their paper at 98.98% [3].

The splitting of the data was 80%, 10%, 10%. The overall distribution of this data ended up being 69,600 images for training, 8700 images for validation, and 8728 images for testing. There was an additional testing dataset that we created ourselves. For this test set, one image of each of the signs (and 'nothing') was taken via a USB web camera and placed in its own test folder. Overall, this custom test dataset had 29 images in it (one per class).

4 Results

4.1 Individual Models

Of the individual models, the transfer learning with VGG19 and the CNN performed the best at both above 99.1% accuracy while the MobileNet transfer learning performed the worst at 85.71%, which was well below our baselines. All test accuracies and test losses can be found in table 2.

Model	Test Accuracy	Test Loss
CNN	99.10%	0.0286
Transfer Learning MobileNet	85.71%	0.5350
KNN	96.14%	N/A
Transfer Learning VGG19	99.15%	0.19
Ensemble	99.34%	N/A

Table 2: The results and test loss for each model and the ensemble

Metric	Score
F1-Score	0.9934
Recall	0.9934
Accuracy	99.34%
Baseline 1	96.96%
Baseline 2	98.98%

Table 3: Ensemble metrics

4.2 The Ensemble

Overall, our results were very promising in terms of f1-score, recall, and accuracy. We scored roughly the same on all 3 (see table 3 for results) which seems to indicate that our data was well balanced and well-performing. Interestingly, when introducing a test set of one of us making the symbols, the ensemble and individual models performed poorly. With this test set, our ensemble achieved an accuracy of 10.34%.

Our baselines of 95.81% - 96.96% and 98.98% were both improved upon with our ensemble method. The first baseline came from the same dataset with a KNN model, ORB model [4], and an MLP model from Kaggle users [3]. The second baseline was the accuracy from a multi-headed CNN network which was trained with a different dataset, but for a similar task [3].

5 Conclusions

Our first key takeaway is that our ensemble was too complex for very little gain. By this, we mean that we had already achieved 99.15% accuracy with the Transfer Learning VGG19 model and 99.10% with the CNN, but our ensemble was only slightly above that at 99.34% with more work and computational resources behind it.

Second, we saw a problem with generalization in a different test dataset that we created ourselves. This could be a problem with the data that we created (such as performing the sign incorrectly) or an issue with the model itself. More work would need to be done to figure out the actual cause for the lack of generalization.

Finally, it was also difficult to train the various models on images that were too large (we had to scale them down to 32x32) due to various memory issues. We utilized a machine with a dedicated GPU and still had to scale down the data and utilize batches in order to be able to train the models, specifically the KNN which had to utilize flattened data for its training. It was simpler to train the neural networks as the data could come directly from the directory instead of loading the entire dataset into memory.

References

[1] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[2] Akash Nagaraj. Asl alphabet, 2018. Available at https://www.kaggle.com/datasets/grassknoted/asl-alphabet.

[3] Refat Khan Pathan, Munmun Biswas, Suraiya Yasmin, Mayeen Uddin Khandaker, Mohammad Salman, and Ahmed A. F. Youssef. Sign language recognition using the fusion of image and hand landmarks through multi-headed convolutional neural network. *Scientific Reports*, 2023.

[4] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

A Jupyter Notebook

This code is also available as a .ipynb included in the zip folder for this project.

In [ ]:
```
import numpy as np
import pandas as pd

import os
import cv2

import splitfolders

from tensorflow.keras import models, layers, Input
from tensorflow.keras.models import load_model, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization, Flatten, Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.regularizers import l2

from tensorflow.keras.applications.mobilenet import preprocess_input

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score, recall_score, accuracy_score

import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
```

In [ ]:
```
os.makedirs('datasets', exist_ok=True)

# List files in a directory as a sanity check
"""for dirname, _, filenames in os.walk('Data/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))"""

# List files in a specific directory
# This will list the labels
print(os.listdir('Data/kaggle/input/asl-alphabet/asl_alphabet_train'))
```

In [ ]:
```
"""train_src = "Data/kaggle/input/asl-alphabet/asl_alphabet_train/"

splitfolders.ratio(train_src, output="datasets/asl_alphabet",
    seed=1337, ratio=(.8, .1, .1), group_prefix=None, move=False) # default values"""
```

In [ ]:
```
train_dir = 'datasets/asl_alphabet/train'
val_dir = 'datasets/asl_alphabet/val'
```

```
test_dir  = 'datasets/asl_alphabet/test'
```

In [ ]:
```
batch_size = 32
target_size = (32,32) # dataset pic = 200x200

train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True)
val_datagen   = ImageDataGenerator(rescale=1./255, horizontal_flip=True)
test_datagen  = ImageDataGenerator(rescale=1./255, horizontal_flip=True)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=target_size,
        batch_size=batch_size,
        color_mode="rgb",
        class_mode='categorical',
        shuffle=True)

val_generator = val_datagen.flow_from_directory(
        val_dir,
        target_size=target_size,
        batch_size=batch_size,
        color_mode="rgb",
        class_mode='categorical',
        shuffle=False)

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=target_size,
        batch_size=batch_size,
        color_mode="rgb",
        class_mode='categorical',
        shuffle=False)
```

In [ ]:
```
labels = list(train_generator.class_indices.keys())
print(labels)
```

In [ ]:
```
TRAIN_PATH = train_dir
def sample_images(labels):
    # Create Subplots
    y_size = 12
    if(len(labels)<10):
        y_size = y_size * len(labels) / 10
    fig, axs = plt.subplots(len(labels), 9, figsize=(y_size, 13))

    for i, label in enumerate(labels):
        axs[i, 0].text(0.5, 0.5, label, ha='center', va='center', fontsize=8)
        axs[i, 0].axis('off')
```

```
        label_path = os.path.join(TRAIN_PATH, label)
        list_files = os.listdir(label_path)

        for j in range(8):
            img_label = cv2.imread(os.path.join(label_path, list_files[j]))
            img_label = cv2.cvtColor(img_label, cv2.COLOR_BGR2RGB)
            axs[i, j+1].imshow(img_label)
            axs[i, j+1].axis("off")

    # Title
    plt.suptitle("Sample Images in ASL Alphabet Dataset", x=0.55, y=0.92)

    # Show
    plt.show()
```

In [ ]:
```
sample_images(labels[:10])
```

CNN Model

In [ ]:
```
num_classes = len(labels)
input_shape = (32,32,3)

input_layer = layers.Input(shape=input_shape)

# Build Model
model = models.Sequential()

model.add(input_layer)

# 1st convolution layer
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
# 2nd convolution layer
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
# 3rd convolution layer
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
# fully-connected layers
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))
```

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

model.summary()
```

In [ ]:
```
# Define checkpoint path
checkpoint_path = "best_model.keras"

# Create ModelCheckpoint callback
checkpoint = ModelCheckpoint(checkpoint_path,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
```

In [ ]:
```
# Compile Model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [ ]:
```
history = model.fit(train_generator, validation_data=val_generator, epochs=10, callbacks=[checkpoint])
```

In [ ]:
```
scores = model.evaluate(test_generator)
print('Test loss: ', scores[0])
print('Test accuracy: ', scores[1])
```

In [ ]:
```
model.save("asl_alphabet_cnn.h5")
```

MobileNet Transfer Learning

In [ ]:
```
image_size = 32
batch_size = 32
num_classes = 29

base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
x = GlobalAveragePooling2D()(base_model.output)
output = Dense(num_classes, activation='softmax')(x)  # Output layer with softmax activation
mobile_model = Model(inputs=base_model.input, outputs=output)

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

# Train the model
model.fit(train_generator, epochs=5, batch_size=32, validation_data=val_generator)
```

In [ ]:
```
# Define checkpoint path
checkpoint_path = "best_model.keras"
```

```
# Create ModelCheckpoint callback
checkpoint = ModelCheckpoint(checkpoint_path,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')

model.save("asl_alphabet_mobilenet.h5")
```

KNN

In [ ]:
```
def flatten_data(generator):
    flattened_data_batches = []
    label_batches = []

    for i in range(len(generator)):
        batch_x, batch_y = generator[i]
        batch_x_flat = batch_x.reshape(batch_x.shape[0], -1)

        flattened_data_batches.append(batch_x_flat)
        label_batches.append(batch_y)

        print(f"Flattening progress: {i+1} of {len(generator)} batches", end="\r", flush=True)

    return flattened_data_batches, label_batches
```

In [ ]:
```
train_x_flat_batches, train_y_batches = flatten_data(train_generator)
```

In [ ]:
```
val_x_flat_batches, val_y_batches = flatten_data(val_generator)
```

In [ ]:
```
# prep test data
test_x_flat_batches, test_y = flatten_data(test_generator)
test_x_flat = np.concatenate(test_x_flat_batches)
test_y = np.concatenate(test_y)
```

In [ ]:
```
test_david = 'datasets/asl_alphabet/test_david'

test_david_datagenerator = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 15,
    fill_mode = 'nearest',
    horizontal_flip = True
    #width_shift_range = 0.2,
    #height_shift_range = 0.2
    )
```

```
test_david_generator = test_david_datagenerator.flow_from_directory(
    test_david,
    target_size = (32, 32),
    class_mode = 'categorical',
    shuffle = False
)
```

In [ ]:
```
# david test flatten
test_d_flat_batches, test_d_y = flatten_data(test_david_generator)
test_d_flat = np.concatenate(test_d_flat_batches)
test_d_y = np.concatenate(test_d_y)
```

In [ ]:
```
# Concatenate flattened data and label batches
train_x_flat = np.concatenate(train_x_flat_batches)
train_y = np.concatenate(train_y_batches)
val_x_flat = np.concatenate(val_x_flat_batches)
val_y = np.concatenate(val_y_batches)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(train_x_flat, train_y)
train_accuracy = knn.score(train_x_flat, train_y)
print("Training Accuracy: ", train_accuracy)

# Evaluate the model
val_accuracy = knn.score(val_x_flat, val_y)
print("Validation Accuracy: ", val_accuracy)
```

VGG

In [ ]:
```
# Import and freeze VGG model
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(32, 32, 3), pooling='max')
base_model.trainable = False

# Add custom top layers
inputs = Input(shape=(32,32,3))
feature_maps = base_model(inputs, training=False)

dense_layer_1 = Dense(512, activation='leaky_relu', kernel_regularizer=l2(0.0001), kernel_initializer='he_normal')(feature_maps)
dropout_1 = Dropout(0.2)(dense_layer_1)
dense_layer_2 = Dense(256, activation='leaky_relu', kernel_regularizer=l2(0.0001), kernel_initializer='he_normal')(dropout_1)
dropout_2 = Dropout(0.2)(dense_layer_2)
dense_layer_3 = Dense(128, activation='leaky_relu', kernel_regularizer=l2(0.0001), kernel_initializer='he_normal')(dropout_2)
dropout_3 = Dropout(0.2)(dense_layer_3)
predictions = Dense(29, activation='softmax',kernel_regularizer=l2(0.0001))(dropout_3)
```

```
model = Model(inputs=inputs, outputs=predictions)
```

In [ ]:
```
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

model.compile(optimizer=Nadam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(
    train_generator,
    batch_size = 32,
    epochs= 10,
    validation_data=(val_generator),
    shuffle=True,
    callbacks=[early_stopping]
)
```

Ensemble

In [ ]:
```
#predictions_knn = knn.predict(test_x_flat)
predictions_knn = knn.predict(test_d_flat)
```

In [ ]:
```
# Load saved models
model_cnn = load_model("asl_alphabet_cnn.h5")
model_mobilenet = load_model("asl_alphabet_mobilenet.h5")

true_labels = test_david_generator.classes

num_train_samples = len(train_generator)
num_val_samples = len(val_generator)

predictions_cnn = model_cnn.predict(test_david_generator)
predictions_mobilenet = model_mobilenet.predict(test_david_generator)
predictions_vgg = model.predict(test_david_generator)

combined_predictions = (predictions_cnn + predictions_mobilenet + predictions_knn + predictions_vgg) / 4

ensemble_labels = np.argmax(combined_predictions, axis=1)

ensemble_accuracy_combined = accuracy_score(true_labels, ensemble_labels)

#print("Ensemble Loss (CNN + MobileNet + KNN):", ensemble_loss_combined)
print("Ensemble Accuracy (CNN + MobileNet + KNN):", ensemble_accuracy_combined)
```

In [ ]:
```
ensemble_predicted_labels = np.argmax(combined_predictions, axis=1)

true_labels = test_david_generator.classes

f1 = f1_score(true_labels, ensemble_predicted_labels, average='weighted')
```

```
recall = recall_score(true_labels, ensemble_predicted_labels, average='weighted')
accuracy = accuracy_score(true_labels, ensemble_predicted_labels)

print("F1-score:", f1)
print("Recall:", recall)
print("Accuracy:", accuracy)
```