ASL Sign Language Translation

Beard, Bennion, Carlson, Napolitano

Data Science Capstone Project
Predictive Modeling Report

Date:

[8/22/2022]

Team Members:

Name: Tyler Beard

Name: Adam Bennion

Name: Zach Carlson

Name: Andrew Napolitano

1. Define the Predictive Modeling Problem

A. Input: What are the input data and define the input data clearly?

The input data is a vector of integers that represent a gray scaled pixel from 0 to 255. There are 50,176 integers per vector that represent the image pixel grayscale value in order from top left to bottom right in a row by row fashion.

Since we have a mean of 931 samples per class, the featurespace will need to be shrunk to avoid risk of overfitting. If performance is poor in all predictive models we may revisit data acquisition.

Data split is as follows:

- Training: 70%
- Validation: 20%
- Test: 10%

B. Data Representation: What is the data representation?

The data representation is digital and numerical in nature. Each integer within the vector represents a gray scaled pixel within the image. The array of integers range in values between 0 and 255. All 50,176 integers represent a 224x224 image of a hand signing the ASL alphabet that has been scaled, cropped and centered to ensure image consistency.
Samples are stored in directories according to class. Upon loading data, each directory's name will be used to automatically apply a class label to the directory's samples.

C. Output: What are you trying to predict? Define the output clearly.

We are trying to classify which letter of the ASL alphabet is shown in a given image, excluding the letters J and Z as their signage requires motion which introduces undue complexity to the project at this stage. There are 24 letters we are trying to classify and the output value will depend on the predictive model algorithm employed. Output for **CNN** and **Transfer Learning CNN** will be a percentage value (.1, .78, .02 … etc) for each of those letters that will add up to equal 1. The model classifies an image into the class with the highest percentage output since this output can be interpreted as probability. Output for **KNN** will be a single class prediction (e.g. 3, which means class D).

2. Predictive Models

A. What are the methods? Give a general introduction of the methods with references

The methods we will be using are k-Nearest Neighbor (**KNN**), convolutional neural network (**CNN**), and **Transfer Learning CNN**. To compare the performance of our model, we may use either a random guesser, or classifier that always predicts one class, to serve as a “dummy” classifier.

KNN:
KNN relies on the assumption that data points that are similar to each other will cluster with each other. The method uses proximity to decide what class a given data point belongs to. The class label itself is given based on a majority vote. The *k* value is the number of neighbors that will be assessed before deciding on a class label for a given data point. For example, if $k = 3$, the algorithm will look at the three nearest neighbors. A distance metric must be selected to calculate distances from a given data point to all others (e.g. Euclidean, Manhattan, etc.) (Reference: IBM)

CNNs:
CNNs were originally inspired by biology and how we process visual information. It was discovered that our visual cortexes contain neurons that trigger only in response to specific shapes (such as edges, or horizontal lines). CNNs operate in the same way. CNNs use kernels and convolutions to identify patterns. The kernel itself can be thought of like a window as it slides across the input image in search of patterns. The kernel attempts to identify patterns by using various types of mathematical convolutions on the image, such as edge detection, embossing, blurring, sharpening, etc. The array that comes as a result of multiplying the kernel and image together is called a feature map. It's called a feature map because it's essentially a "map" that points to where that particular pattern (or feature) is located in the original image. Feature maps become more and more complex in later layers of the neural network. Initial layers can identify things such as diagonal lines, intermediate layers can identify things such as eyes, and advanced layers identify things like faces and buildings. (Reference: DataCamp)

Transfer Learning CNNs:
Transfer Learning CNNs use the same principles of CNNs, however, rather than training an entire neural network from scratch, Transfer Learning CNNs allow you to take a pre-trained model and apply it to another dataset. It allows one to take a model that was trained on a large dataset and use it on a small dataset, potentially one with less-than-ideal number of images. This is especially enticing for those who do not have access to large datasets, don't have sufficient hardware to process or store these datasets, or don't have the funds to use a cloud-based tool to store/process these datasets and models quickly. However, there is a major assumption that whatever model you are using was trained on a dataset similar to your own. (Reference: TowardsDataScience.com)

B. Describe the methods with pseudo code using the definitions in Section 1.

KNN:

#Load the data, feature reduce, split the data into 70% training, 20% validation, 10% test.
```
dataset  = pandas.read_csv(imgscsv)
dataset = PCA(dataset,num_components = 931 ) # will see how slow this is
or
dataset = ImageDeresolution(dataset,30x30) #if PCA is too slow
```

train_ratio = 0.70 validation_ratio = 0.20 test_ratio = 0.10
x_train, x_test, y_train, y_test = train_test_split(dataX, dataY, test_size=1 - train_ratio)
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=test_ratio/(test_ratio + validation_ratio))

#Create the model and set the n=10
model = KNeighborsClassifier(n_neighbors = 10)
model.fit(X_train, y_train)

#Get the models predictions
y_pred = model.predict(X_test)

#Evaluate how the model did and repeat with both validation and test data
confusion_matrix(y_test, y_pred)
classification_report(y_test, y_pred)

CNN:

#**Load data -- 224x224 samples**
X_train, y_train,X_validate, y_validate, X_test, y_test = load_data() #already split

#**Split data into trainX,trainY,validateX,validateY,testX,testY**

X_train, y_train,X_validate, y_validate, X_test, y_test = split(data)

#**Define CNN**

cnn = Sequential()
cnn.add(Conv2D(64, (3, 3), input_shape = (224, 224, 1), relu_activation))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Dropout)
cnn.add(Conv2D(32, kernel_size, relu_activation))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Dropout)
cnn.add(Flatten)
cnn.add(Dense(24, softmax_activation))
cnn.compile(adam_optimizer, categorical_crossentropy_loss, accuracy_metric)

#**Train CNN with trainX,trainY**
cnn_model.fit(X_train, y_train, X_validate, y_validate, epoch_count)
#Add Hyperparameter Loop -- num epochs, kernel_size, dropout
#Generate Confusion Matrix

Transfer Learning CNN:

#Load data, 244x244 samples
X_train, y_train,X_validate, y_validate, X_test, y_test = load_data() #already split

#Create VGG16 model without classifier layer, no VGG16 training
vgg = VGG16(input_shape=IMAGE_SIZE, weights='imagenet', include_top=False)
vgglayers.trainable = False **#at least at first to reduce processing costs**

#Flatten VGG16 output, use our class count
x = Flatten()(vgg.output)
prediction = Dense(numClasses, activation='softmax')(x)

#Creating model object
model = Model(inputs=vgg.input, outputs=prediction)

#Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(training_set, validation_data=validation_set, epochs=20, batch_size=32)

C. Justify the choice of the method.

KNN:
The KNN algorithm is an appropriate algorithm to begin an image classification problem with as it tends to be one of the most simple algorithms to understand and implement - especially when considering algorithms 2 and 3 are variations of neural networks that have the potential to grow rather complex and opaque. The model *simply* calculates a distance metric between the test image and each image in the training set, takes the nearest *k* images (the *k* images with the lowest distance calculation), and rather diplomatically takes a majority vote among the neighborhood to determine a classification and can usually be displayed and understood visually in a simple manner. Additionally, the most basic forms of the algorithm can use just 2 hyperparameters: choosing an appropriate distance function and determining the appropriate k value (aws.amazon.com). KNNs also do not require training time meaning we can launch right into prediction and quickly begin to gauge success of the hyperparameters. Lastly, other algorithms may struggle with a dataset such as ours that boasts 24 different classes, whereas the KNN algorithm has the capacity to handle this requirement quite well. For these reasons, in addition to being a rather fundamental classification method, the KNN algorithm will allow our team to explore our problem set with a simplified initial approach and set the table for additional complexities and efficiencies to be implemented in later, more complex methods if necessary (machinelearninginterviews.com).

CNN:
Our dataset has one main limitation - its size. Thankfully, CNNs leverage a few key ideas to deliver best-in-class accuracy while driving computational efficiency where possible. The first is

the idea of sparse interaction, or the ability of the CNN to extract meaningful information from a large input feature pool and store a much smaller collection of parameters. In our case, the model will read in about 930 samples of 50,176-integer vectors for each of our 24 classes and simplify into a much smaller set of key parameters during the convolutional layer. Reducing parameter complexity directly reduces the memory requirement and increases the statistical efficiency of the model. This leads to the second key idea which is that of shared parameters. Instead of assigning and then discarding each element in the weight matrix, a CNN uses the same weights across several inputs. Again, this reduces the memory load during operation. Lastly, parameter sharing lends to the property of equivariance to translation, meaning that model output changes in a consistent manner to changes in the inputs to the model (towardsdatascience.com). These ideas combine to enable an algorithm known for delivering accuracy to the field of image classification. This method has been used previously to achieve 98% accuracy on classifying an ASL alphabet dataset using a 7 layer CNN.

Transfer Learning CNN:
Transfer learning CNNs carry many of the same benefits of a CNN trained from scratch but enable the researcher to leverage weights that have been pre-trained on similar datasets. In our case, we have identified CNN weights that have been trained based on a 16-layer deep pre-trained CNN that achieved 97% accuracy for an ASL alphabet dataset developed by a group of Oxford researchers. Using athe weights associated with a CNN of such strong performance can increase the efficiency of our research if implemented correctly.

3. Evaluations

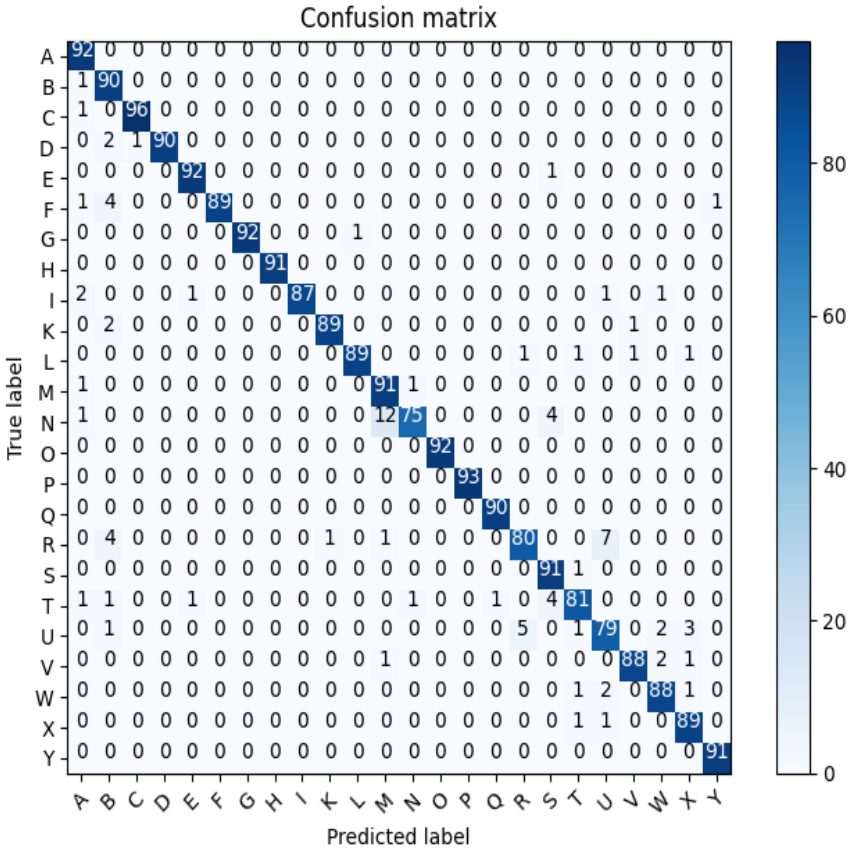
A. What metrics do you use for evaluation?

Our team chose to use accuracy as our primary metric to evaluate the performance of the model overall due to the high number of classes we are predicting (24). Although it is possible to determine measurements that are adjacent to precision/recall for multiclass applications by using macro/micro averaging, we decided that we will heed the side of simplicity at the model level and dive into more granular measures at the class level. We intend to analyze the accuracy, precision, and recall values at the class level to identify classes of higher or lower relative success and may inform revised preprocessing strategies or inspire model parameter adjustments.

However, most of the focus will be on accuracy for our evaluation metric. Our project's purpose is to simply identify hand signs correctly. In a multiclass classification model, false positives and false negatives rely on looking at performance from the perspective of a singular class. In our case, whether the model labels a hand sign as "C" when it's "D" (false positive) or labels a hand sign as "D" when it's "C" (false negative) are equally undesirable.

Classification Report:

	precision	recall	f1-score	support
A	0.92	1.00	0.96	92
B	0.87	0.99	0.92	91
C	0.99	0.99	0.99	97
D	1.00	0.97	0.98	93
E	0.98	0.99	0.98	93
F	1.00	0.94	0.97	95
G	1.00	0.99	0.99	93
H	1.00	1.00	1.00	91
I	1.00	0.95	0.97	92
K	0.99	0.97	0.98	92
L	0.99	0.96	0.97	93
M	0.87	0.98	0.92	93
N	0.97	0.82	0.89	92
O	1.00	1.00	1.00	92
P	1.00	1.00	1.00	93
Q	0.99	1.00	0.99	90
R	0.93	0.86	0.89	93
S	0.91	0.99	0.95	92
T	0.94	0.90	0.92	90
U	0.88	0.87	0.87	91
V	0.98	0.96	0.97	92
W	0.95	0.96	0.95	92
X	0.94	0.98	0.96	91
Y	0.99	1.00	0.99	91
accuracy			0.96	2214
macro avg	0.96	0.96	0.96	2214
weighted avg	0.96	0.96	0.96	2214



B. What is your ground truth?

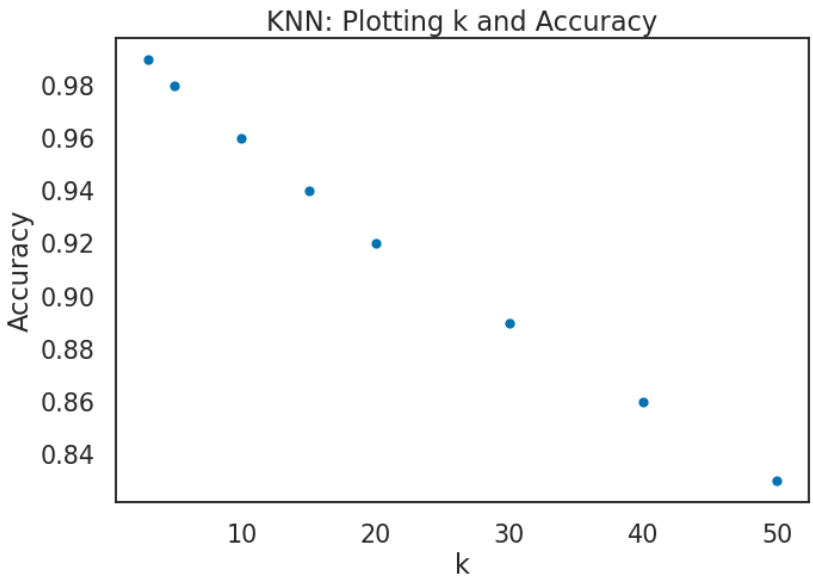
Each image in our training set is stored in a labeled directory identifying which alphabetical character the images are signing. Ground truth is assigned by labeling each image vector according to the class directory in which it was initially stored..

C. Discuss the performance and the limitation of the method.

KNN

By tweaking the hyperparameter "number of neighbors", we were able to find a pattern of decreasing accuracy the more neighbors considered when classifying.

3 neighbors gave a test accuracy of 99%, while 50 neighbors proved the worst with a test accuracy of 85%.



KNN has proven highly accurate with the given samples. Although there may be significant limitations to KNN for ASL sign classification, it will remain unknown until a larger and more diverse dataset is acquired. Experimentation with distance calculation and dimension reduction component count may help further improve the performance. Some limitations with KNN are the overlap of different class features, poor scalability with larger sample sets, and poor reaction to outliers.

CNN

Two CNN architectures were attempted.

The simple architecture:

Conv2D -> MaxPooling2D -> Dropout -> Conv2D -> MaxPooling2D -> Dropout -> Flatten() -> Dense softmax

The more complex architecture:

Conv2D -> MaxPooling2D -> Dropout -> Conv2D -> Dropout -> Conv2D -> MaxPooling2D -> Dropout -> Flatten() -> Dense softmax

Hyperparameters included in the simpler architecture:

- Dropout [0.2, 0.2],[0.25, 0.25],[0.3, 0.3]
- Kernel size [5, 3],[3, 3],[10, 6]

  Where each brack details the hyperparameter for each layer utilizing said hyperparameter.

Hyperparameters included in the more complex architecture:

- Dropout [0.2, 0.2, 0.2],[0.25, 0.25, 0.25]
- Kernel size [5, 3, 2], [4, 3, 2], [5, 4, 3]

The best result was the simpler architecture with epochs 10, kernel size[5, 3], and dropout 0.2 for which the test accuracy was **98.7%**.

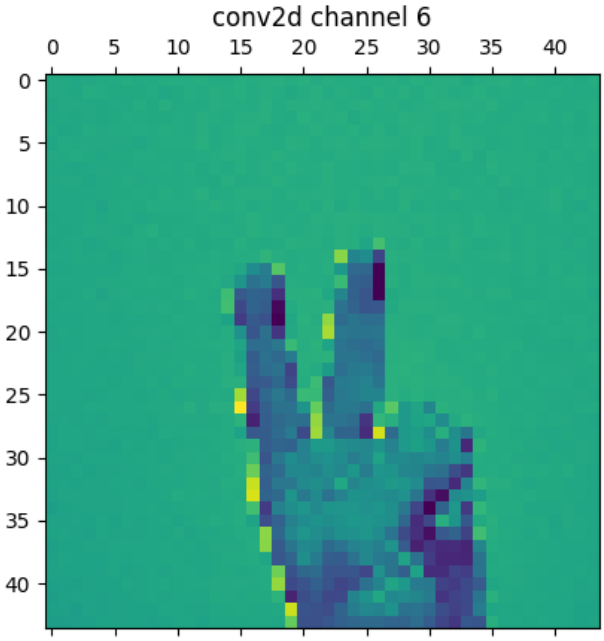
The more complex architecture's best performance was 93.75% with 29 epochs, kernel size[5, 4, 3], and dropout 0.2.

Time constraints limit the number of hyperparameters attempted as well as number of architectures considered.

Appendix

[Additional materials that are not included in the above sections.]

Feature Map Extraction


conv2d channel 6
0
5
10
15
20
25
30
35
40

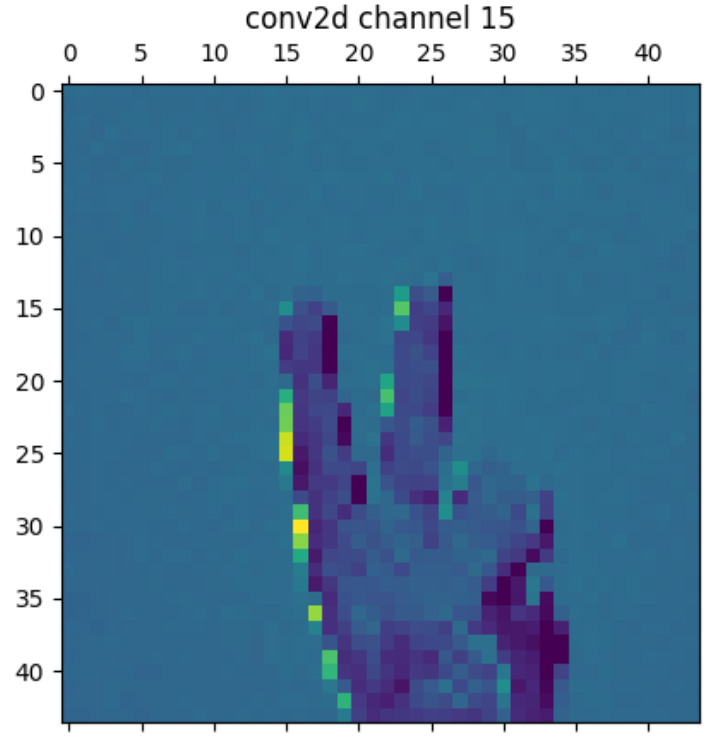

conv2d channel 15
0
5
10
15
20
25
30
35
40

Table of Contributions

The table below identifies contributors to various sections of this document.

	Section	Writing	Editing
1	**Predictive Modeling Problem Definition**	TB, AB, AN, ZC	TB, AB, AN, ZC
2	**Predictive Models**	AB, ZC, AN, TB	AB, ZC, AN, TB
3	**Evaluations**	ZC, AN, AB, TB	ZC, AN, AB, TB
4	**Appendix**		

Grading

The grade is given on the basis of quality, clarity, presentation, completeness, and writing of each section in the report. This is the grade of the group. Individual grades will be assigned at the end of the term when peer reviews are collected.