Data Analysis Report

Image Classification with Convolutional Neural Networks

Interpreting Hand-Shape of ASL Interactions

Data Analytics Graduate Capstone

Brittany Kozura
Western Governors University
D214

Table of Contents

Part I: Research Question
A. Purpose of Report

Analysis Context

As machine learning research grows, there are some companies who are looking to harness Natural Language Processing for translational and interpreting work. AI development in this area of research, known as Machine Translation, has "led to very significant improvements in translation quality" in recent years (Stanford NLP Group, 2022). A new challenge in particular is presented when attempting NLP for sign languages, as they neither have a verbal nor written component. Therefore, new approaches to NLP must be considered when it comes to interpreting sign languages. This project acts as a predecessor to NLP for American Sign Language (ASL), where the first step in the process is to classify hand shape from an individual video frame or image. Our goal is to create a model that classifies hand shapes as a proof-of-concept for such a system.

Research Question

The purpose of this data analysis is to answer the research question: **"To what extent can hand shape be accurately classified from images?"** This is a pertinent research question because we need to know the maximum accuracy a model could achieve to be able to tell if the Machine Translation System could be viable.

Hypothesis

Our hypothesis is that images can be **classified with 90% accuracy using Convolutional Neural Networks**.

B. Analysis Environment

Environment Description

Environment Setup

Our environment setup is as follows:

```
# Initial Environment setup
setwd("C:/Users/bkozura/Documents/asl_images/")
library(caret)            #  Classification and Regression Training
library(jsonlite)         #  A Simple and Robust JSON Parser and Generator for R
library(keras)            #  R Interface to 'Keras'
library(tidytext)         #  Text Mining using Tidy tools
library(tidyverse)        #  R Packages for Data Science
library(imager)           #  Image data manipulation
library(magick)           #  Image data manipulation
```

```
getS3method("print", "sessionInfo")(sessionInfo()[-8][], locale = FALSE)
```

```
R version 4.2.1 (2022-06-23 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 22000)

Matrix products: default

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] magick_2.7.3      imager_0.42.13    magrittr_2.0.3    hms_1.1.2         reticulate_1.26
 [6] tensorflow_2.9.0  keras_2.9.0       forcats_0.5.2     stringr_1.4.1     dplyr_1.0.10
[11] purrr_0.3.4       readr_2.1.2       tidyr_1.2.1       tibble_3.1.8      ggplot2_3.3.6
[16] tidyverse_1.3.2
```

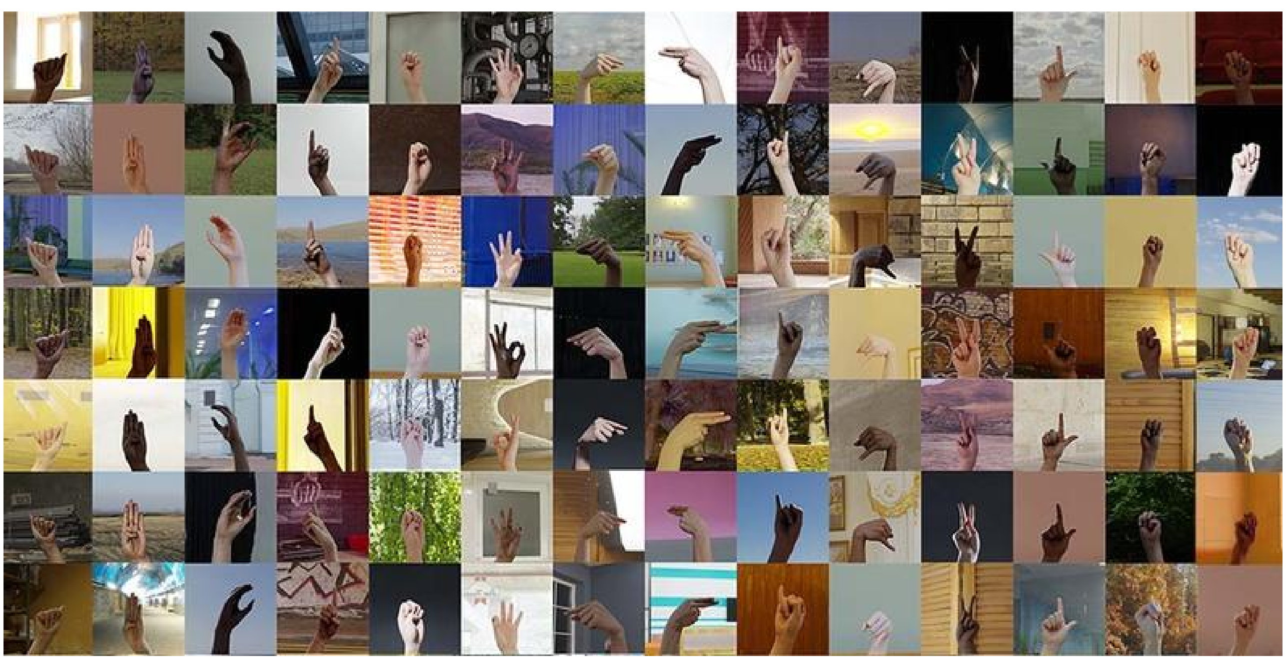
```
tf_config()
```

```
TensorFlow v2.9.2 ()
Python v3.8 (C:/Users/bkozura/AppData/Local/r-miniconda/envs/r-reticulate/python.exe)
```

Part II: Data Preparation

C. Data Collection

The dataset used for this analysis "Synthetic ASL Alphabet" was created by a software development company Lexset (Lexset, 2022) and was listed as Creative Commons Attribution-NonCommercial as a way to promote Lexset's synthetic data generator platform “Seahaven” which was used to create the data set. The data was obtained from open data repository Kaggle and was split into 'train' and 'test' data sets–each with images divided into folders by classification. It consists of 27,000 images across 27 categories (26 alphabet + "Blank"). One advantage of Synthetic data is that you can create a lot of it at a time, but a disadvantage is that it can result in data that is too ideal/ not diverse. One challenge to overcome in this dataset was to figure out the best method of retrieving the image data, since 27000 x 277x277x3 results in a huge amount of data. Our solution was to rent a VM to handle the large memory needs.



Example images from the dataset

C. Data Extraction & Preparation

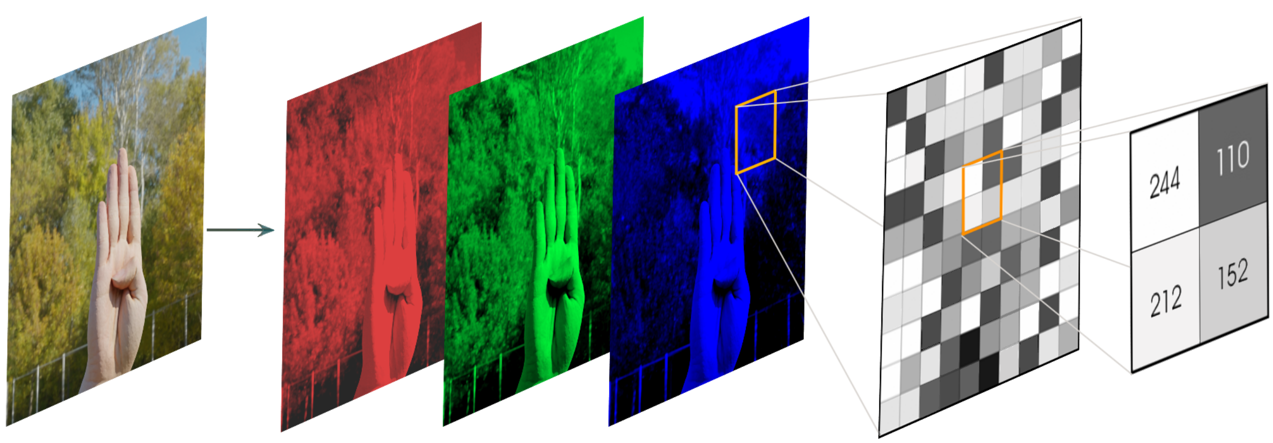
Data Preparation Steps

The Data Preparation phase of this project can be broken down into multiple steps:

(a) Data Extraction
(b) Image Preprocessing
(c) Data splitting

(a) Data Extraction & Organization

RGB data is extracted from the images using Kera's internal `flow_images_from_directory` function, which allows us to include our preprocessing steps in the data extraction phase. One advantage of using Keras is that it has a lot of built-in machine learning functionality, but a disadvantage is that it requires the use of both Python and R. During this process all data augmentation, resizing, and manipulations occur. Additionally, we split the dataset into a 72-19-10 train-validation-test split.



RGB data from image

```
# Load Data
  dir(wd)
 [1] "example.png"   "test"    "train"    "tuning_results.csv"

path_train <- paste(wd,"/train", sep="")
path_test <- paste(wd,"/test", sep="")
label_list <- dir(path_train)
n_classes <- length(label_list)  # 27 Classes
save(label_list, file="label_list.R")
label_list
 [1] "A"      "B"      "Blank" "C"      "D"      "E"      "F"      "G"
 [9] "H"      "I"      "J"      "K"      "L"      "M"      "N"      "O"
[17] "P"      "Q"      "R"      "S"      "T"      "U"      "V"      "W"
[25] "X"      "Y"      "Z"
```

(b) Image Data Preprocessing

For our image data preprocessing, we will:

- Resize the images from 513x512 pixels to 227x227 pixels
- Rescale the image data to values between 0 and 1 by dividing rgb values by 255

```
# Image Preprocessing variables
img_size <- 227 # height and width
target_size <- c(img_size, img_size)

train_data_gen <- image_data_generator(rescale = 1/255, validation_split = .2)
test_data_gen <- image_data_generator(rescale = 1/255)
```

(c) Data Set Splitting

We used a **90-10 split for the training and testing data**, with a further **training-validation split of 80-20**, resulting in the following distribution:

Data set	Percent Distribution	Number of Images
train	72%	19440
validation	18%	4860
test	10%	2700
total		**27000**

```
train_images <- flow_images_from_directory(path_train,
                                           train_data_gen,
                                           subset = 'training',
                                           target_size = target_size,
                                           class_mode = 'categorical',
                                           shuffle=TRUE,
                                           classes = label_list,
                                           seed = 2021)

> Found 19440 images belonging to 27 classes.
```

```
validation_images <- flow_images_from_directory(path_train,
                                           train_data_gen,
                                           subset = validation,
                                           target_size = target_size,
                                           class_mode = 'categorical',
                                           classes = label_list,
                                           seed = 2021)

> Found 4860 images belonging to 27 classes.
```

```
test_images <- flow_images_from_directory(path_test,
                                          test_data_gen,
                                          target_size = target_size,
                                          class_mode = "categorical",
                                          classes = label_list,
                                          shuffle = FALSE,
                                          seed = 2022)

> Found 2700 images belonging to 27 classes.
```

We can also see that the dataset is balanced because the images are evenly distributed across all 27 classes.

```
table(train_images$classes)
```

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21 
720 720 720 720 720 720 720 720 720 720 720 720 720 720 720 720 720 720 720 720 720 720 
 22  23  24  25  26 
720 720 720 720 720 
```

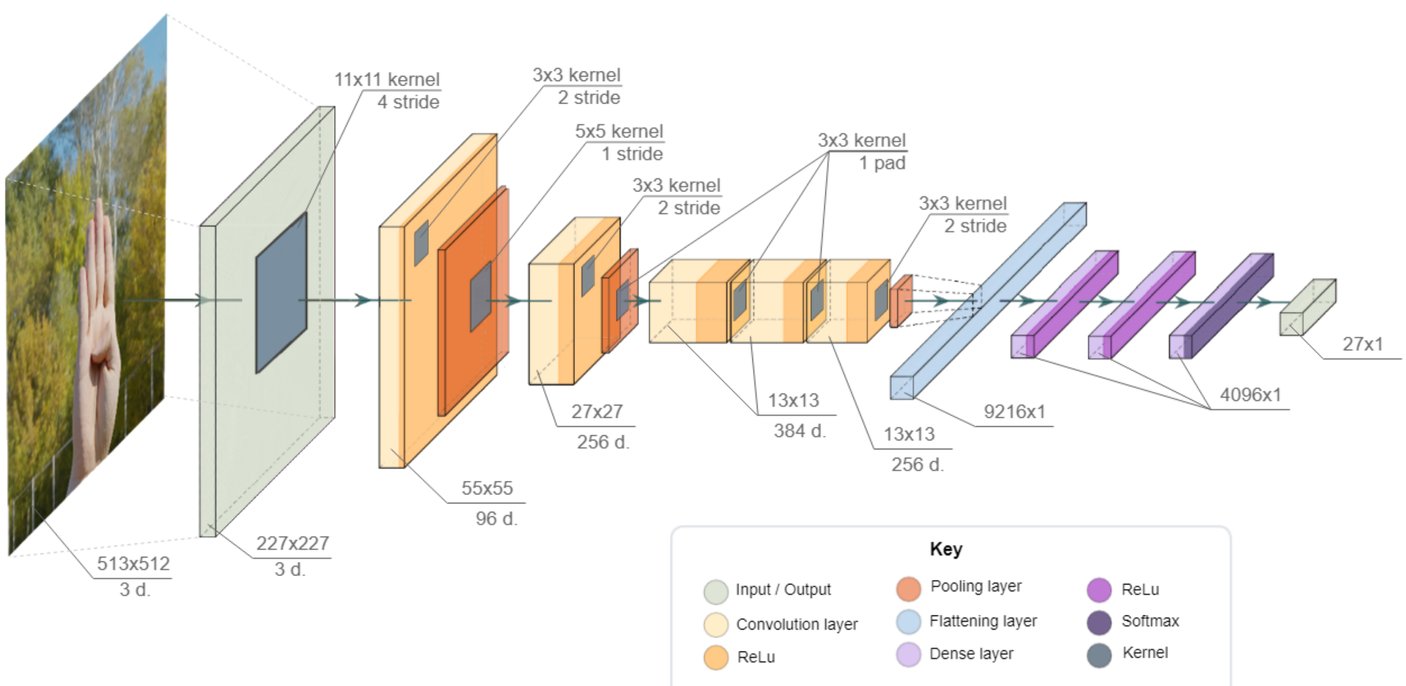
Part III: Network Architecture

D. Convolution Neural Network

The data analysis technique that will be used to classify the images is a Convolutional Neural Network using Keras/Tensorflow. CNNs are industry standards for image classification analyses since they perform "phenomenally well on computer vision tasks" (Rizvi, 2022). Rather than the analyst identifying the key features of a class of images, CNNs analyze images into three-dimensional matrices representing color components per pixel and "'learns' how to extract these features, and ultimately infer what object they constitute" (Google Developers, 2022). One advantage of a CNN is that because the network learns from the raw data rather than be interpreted by the analyst, it is able to identify features unnoticed by humans. Alternatively, a disadvantage of using a CNN is that a large amount of data is required for training to yield good results.

CNN Model Structure

Our model consists of 13 layers following a familiar CNN structure: a feature-mapping section (Convolution layers w/ ReLu and Pooling layers) and a classification section (Dense Layers w/ ReLu and final Dense layer with SoftMax). The complete structure of the CNN can be observed in the following diagram and the tables below.



CNN Model Layers

Convolutional Neural Network Model						
Layer	Type	k	Stride	Padding	Shape-out	activation
0	Input	-	-	-	227, 227, 3	-
1	Convolutional	11	4	'VALID'	55, 55, 96	'relu'
2	MaxPooling	3	2	'VALID'	27, 27, 96	-
3	Convolutional	5	1	'SAME'	27, 27, 256	'relu'
4	MaxPooling	3	2	'VALID'	13, 13, 256	-
5	Convolutional	3	1	'SAME'	13, 13, 384	'relu'
6	Convolutional	3	1	'SAME'	13, 13, 384	'relu'
7	Convolutional	3	1	'SAME'	13, 13, 256	'relu'
8	MaxPooling	3	2	'VALID'	6, 6, 256	-
9	Flatten	-	-	-	9216	-
10	Dropout	-	-	-	9216	-
11	Dense	-	-	-	4096	'relu'
12	Dense	-	-	-	4096	'relu'
13	Output (Dense)	-	-	-	27	'softmax'

Most of our convolutional and dense layers use a rectified linear activation function ("ReLU") while our final classification layer uses the softmax activation function, which is essentially " a smooth version of the winner-takes-all activation model" (Bishop 2013).

Hyperparameters Summary

Parameter	**Chosen Value**
activation functions	'relu', 'softmax'
loss function	'categorical_crossentropy'
optimizer	adam, learning_rate = 0.00001
dropout_rate	0.00000001
initializers	kernel & bias = 'random_uniform'
epochs	max = 25
stopping criteria	monitor = val_accuracy, patience = 2
evaluation metric	accuracy, kappa

Our CNN Model code is as follows:

```
# Custom Layer functions

con_layer <- function(x,filters, ksize, bias, strides=1, padding='SAME'){
  layer_conv_2d(x, filters = filters,
                kernel_size = c(ksize, ksize),
                padding = padding,
                activation = "relu",
                strides=c(strides, strides),
                use_bias = TRUE,
                kernel_initializer='random_uniform',
                bias_initializer='random_uniform')}

pool_layer <- function(x){
  layer_max_pooling_2d(x, pool_size = c(3, 3), strides=2, padding= 'VALID') }

fully_connected_layer <- function(units, activation = "relu", name){
  layer_dense(units = units,
              activation = activation,
              kernel_initializer='random_uniform',
              bias_initializer='random_uniform') }
```

```
# Create Model Function

model_a <- function(learning_rate=0.0001, dropout_rate=0.00000001){
  k_clear_session()

  model <- keras_model_sequential(input_shape = c(img_size,img_size,3)) %>%
    con_layer(filters=96, ksize=11, strides = 4, padding = 'VALID') %>%
    pool_layer() %>%
    con_layer(filters=256, ksize=5) %>%
    pool_layer() %>%
    con_layer(filters=384, ksize=3) %>%
    con_layer(filters=384, ksize=3) %>%
    con_layer(filters=256, ksize=3) %>%
    pool_layer() %>%
    layer_flatten() %>%
    layer_dropout(dropout_rate)%>%
    layer_dense(units = 4096, activation= "relu", kernel_initializer='random_uniform',
                bias_initializer='random_uniform')  %>%
    layer_dense(units = 4096, activation= "relu", kernel_initializer='random_uniform',
                bias_initializer='random_uniform')  %>%
    layer_dense(name="Output", units = n_classes, activation = "softmax")

  model %>% compile(loss = "categorical_crossentropy",
    optimizer = optimizer_adam(learning_rate= learning_rate),
    metrics = "accuracy")

  return(model)
}
my_model <- model_a(learning_rate=0.00001)
```

We can examine the **summary of our CNN model:**

```
summary(my_model)
```

```
Model: "sequential"
________________________________________________________________________________________
 Layer (type)                         Output Shape                       Param #
========================================================================================
 conv2d_4 (Conv2D)                    (None, 55, 55, 96)                 34944
 max_pooling2d_2 (MaxPooling2D)       (None, 27, 27, 96)                 0
 conv2d_3 (Conv2D)                    (None, 27, 27, 256)                614656
 max_pooling2d_1 (MaxPooling2D)       (None, 13, 13, 256)                0
 conv2d_2 (Conv2D)                    (None, 13, 13, 384)                885120
 conv2d_1 (Conv2D)                    (None, 13, 13, 384)                1327488
 conv2d (Conv2D)                      (None, 13, 13, 256)                884992
 max_pooling2d (MaxPooling2D)         (None, 6, 6, 256)                  0
 flatten (Flatten)                    (None, 9216)                       0
 dropout (Dropout)                    (None, 9216)                       0
 dense_1 (Dense)                      (None, 4096)                       37752832
 dense (Dense)                        (None, 4096)                       16781312
 Output (Dense)                       (None, 27)                         110619
========================================================================================
Total params: 58,391,963
Trainable params: 58,391,963
Non-trainable params: 0
________________________________________________________________________________________
```

Model Fitness & Addressing Overfitting

Our model at first showed evidence of overfitting, so we added a dropout layer to limit the effect of overfitting.

Saving the Pre-training RNN model

We save the untrained neural network, so we can reuse the structure again later if desired.

```
# Save Model
my_model %>% save_model_tf("/asl_alexnet_model_pretraining")
```

Part IV: Model Evaluation

F. Model Fitness

Number of Epochs & Stopping Criteria

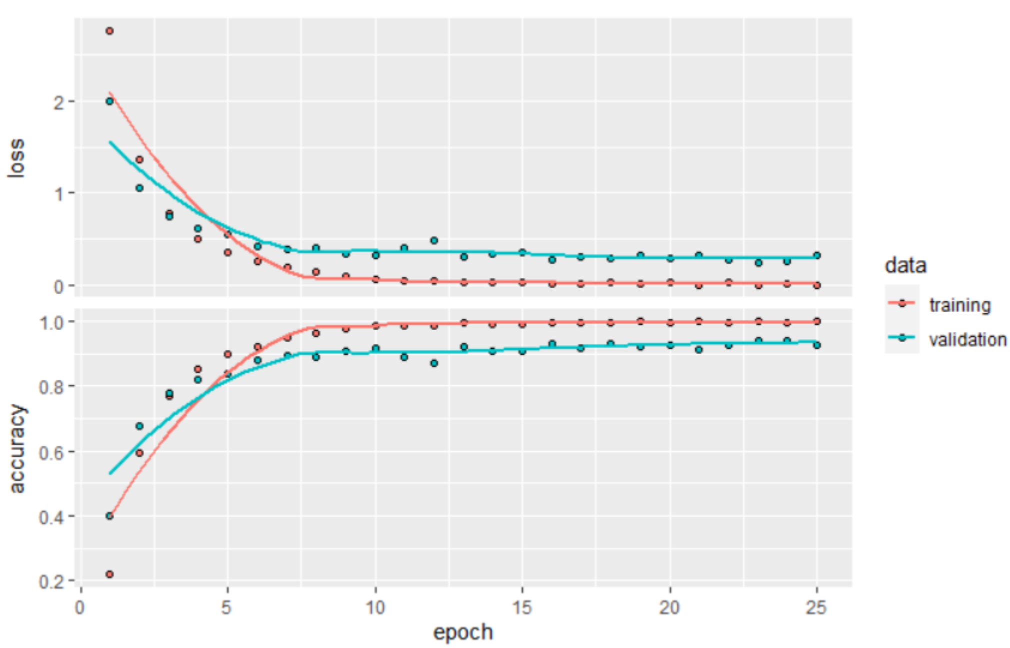
We set our 'max' number of epochs to 25 but also add a callback for early stopping with a patience of 2, meaning the training will end early if further improvement is not observed.

Model Training

We run the training data and save our epochs and save the results under **`hist`**.

```
# Train Model
batch_size <- 128
epochs <- 25
hist <- my_model %>% fit(
  train_images,
  # train_labels,
  #steps_per_epoch = train_images$n %/% batch_size,
  epochs = epochs,
  validation_data = validation_images,
  #validation_steps = validation_images$n %/% batch_size,
  verbose = 2,
 callbacks = list(callback_early_stopping(monitor = "val_accuracy", min_delta=0.001,
         patience = 2, restore_best_weights = TRUE))
)
```

Our model training resulted in an accuracy of over 99% for training data and over 94% for validation data.



```
Epoch 1/25
608/608 - 391s - loss: 2.7594 - accuracy: 0.2170 - val_loss: 1.9974 - val_accuracy: 0.4002 - 391s/epoch - 643ms/step
Epoch 2/25
608/608 - 394s - loss: 1.3712 - accuracy: 0.5933 - val_loss: 1.0568 - val_accuracy: 0.6772 - 394s/epoch - 648ms/step
Epoch 3/25
608/608 - 381s - loss: 0.7739 - accuracy: 0.7703 - val_loss: 0.7449 - val_accuracy: 0.7770 - 381s/epoch - 627ms/step
…
Epoch 22/25
608/608 - 379s - loss: 0.0268 - accuracy: 0.9936 - val_loss: 0.2746 - val_accuracy: 0.9270 - 379s/epoch - 624ms/step
Epoch 23/25
608/608 - 384s - loss: 0.0040 - accuracy: 0.9994 - val_loss: 0.2489 - val_accuracy: 0.9407 - 384s/epoch - 631ms/step
Epoch 24/25
608/608 - 389s - loss: 0.0232 - accuracy: 0.9939 - val_loss: 0.2587 - val_accuracy: 0.9399 - 389s/epoch - 640ms/step
Epoch 25/25
608/608 - 384s - loss: 0.0065 - accuracy: 0.9988 - val_loss: 0.3265 - val_accuracy: 0.9274 - 384s/epoch - 632ms/step
```

Example Prediction

We will look at an example image from the test set and see how the model makes a prediction:

Example image of class "D"

```
# Load example image

test_image <- image_load("example.png",
    target_size = target_size)
x <- image_to_array(test_image)
x <- array_reshape(x, c(1, dim(x)))
x <- x/255


# Classify example image

pred <- my_model %>% predict(x)
pred <- data.frame("Class" = label_list, "Probability" = t(pred))
pred <- pred[order(pred$Probability, decreasing=T),][1:5,]
pred$Probability <- paste(format(100*pred$Probability,2),"%")
pred
```

	Class <chr>	Probability <chr>
5	D	9.999998e+01 %
22	U	2.135896e-05 %
27	Z	4.700729e-06 %
25	X	2.491685e-10 %
2	B	8.922056e-12 %

5 rows

Part VI: Interpretation and Implications of Analysis Results

Prediction Accuracy

When we evaluate our model on the **training data**, we get an **accuracy of 99.9%,** and when we evaluate our model on the **testing data**, we get an **accuracy of 93.4%.**

```
my_model %>% evaluate(train_images)
>        loss    accuracy
> 0.009826553 0.999382734

my_model %>% evaluate(test_images)
>      loss  accuracy
> 0.2397213 0.9377778
```

We can now explore the predictions on the test set more in-depth

```
# Run model on test data

predictions <- my_model %>%  predict(test_images, generator = test_data_gen) %>%
               as.data.frame()
names(predictions) <- label_list
predictions$predicted_class <-  label_list[apply(predictions,1,which.max)]
predictions$true_class <- label_list[test_images$classes+1]
cmatrix <- as.matrix(table(Actual = predictions$true_class, Predicted =
           predictions$predicted_class))
cmatrix
```

Predicted

Actual	A	B	Blank	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	91	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	7	0
B	0	99	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Blank	0	0	96	2	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
C	0	0	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0
D	0	0	1	0	94	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	3	0	0
E	0	0	0	0	0	96	0	0	0	0	0	0	0	2	1	0	0	0	0	0	1	0	0	0	0	0	0
F	0	0	0	0	0	0	99	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	1	0	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
H	0	0	0	1	0	0	0	0	96	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	97	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	91	0	1	0	0	0	1	2	0	0	0	0	0	0	0	5	0
K	0	1	0	0	0	0	0	0	0	0	0	94	0	1	0	0	0	0	1	0	0	2	0	0	0	1	0
L	0	0	0	0	0	0	1	0	0	0	1	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	2	0	0	0	0	0	1	0	91	1	0	0	1	0	0	3	0	1	0	0	0	0
N	1	0	0	0	2	1	0	0	0	0	0	0	0	4	89	0	0	0	0	0	1	1	0	0	1	0	0
O	0	0	0	1	0	1	1	0	0	1	0	0	0	0	0	94	0	0	0	1	0	0	0	0	1	0	0
P	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	94	3	0	0	0	0	0	0	0	1	0
Q	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	5	91	0	0	0	0	0	0	0	1	0
R	0	1	0	0	0	0	1	0	0	1	0	0	0	2	0	1	0	0	89	0	0	1	0	0	4	0	0
S	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	13	0	0	1	79	4	0	0	0	1	0	0
T	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	2	95	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	1	0	2	0	0	94	0	0	1	0	0
V	0	1	0	1	1	0	1	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	91	2	0	0	0
W	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	3	94	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	2	0	95	0	1
Y	2	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	96	0
Z	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	2	1	0	0	93

```
# Evaluation Metrics
n_images <- sum(cmatrix)
n_classes <- nrow(cmatrix)
n_correct_by_class <- diag(cmatrix)
n_per_class <- apply(cmatrix, 1, sum)
pred_per_class <- apply(cmatrix, 2, sum)
act_dist <- n_per_class / n_images
pred_dist <- pred_per_class / n_images

metrics <- data.frame("Class"= label_list)
metrics$precision <- n_correct_by_class / pred_per_class
metrics$recall <-  n_correct_by_class / n_per_class
metrics$f1 <- 2 * metrics$precision * metrics$recall /
                      (metrics$precision + metrics$recall)
summary(metrics[,c(-1)])

macros <- data.frame("Value"=colMeans(metrics[,c(-1)]))
macros["accuracy",] <- rbind(sum(n_correct_by_class) / n_images)
macros["rand_accuracy",] <- rbind(sum(act_dist*pred_dist))
macros["kappa",] <- rbind((accuracy - rand_accuracy) / (1 - rand_accuracy))
macros$Value <- round(as.numeric(macros$Value),6)
macros
```

```
  precision          recall              f1
 Min.   :0.8468   Min.   :0.7900   Min.   :0.8681
 1st Qu.:0.9249   1st Qu.:0.9100   1st Qu.:0.9160
 Median :0.9495   Median :0.9400   Median :0.9447
 Mean   :0.9396   Mean   :0.9378   Mean   :0.9378
 3rd Qu.:0.9677   3rd Qu.:0.9600   3rd Qu.:0.9606
 Max.   :1.0000   Max.   :0.9900   Max.   :0.9899
```

precision	0.939569
recall	0.937778
f1	0.937799
accuracy	0.937778
rand_accuracy	0.037037
kappa	0.935385

We evaluated the system based on accuracy and Kappa, both of which came out at over 93%, so our hypothesis was correct.

Saving the Trained RNN model

We save the trained neural network, and can load it at a later date with `load_model_tf`.

```
# Save Model
my_model %>% save_model_tf("/asl_alexnet_model")
```

G. Limitations & Recommendations

Limitations of Synthetic Data

- Synthetic Data tends to result in more 'ideal' data rather than real-world data.

  Solution: Augment data with more diverse pre-processing and real-world images.

Limitations of CNNs (Rizvi, 2020)

- CNNs require a large amount of data.
- CNNs do not encode the position and orientation of objects in images.
- CNNs do not incorporate time-series data.

  Solution: Hybridize with other models in the next phase

Recommended Course of Action

Continued Training with more Real-World Data

- Expand Training & Testing Data: Enhance with "Real-World" data and augment more synthetic data that is "less-than-ideal" to get more diversity.

Hybridize with Other Models

- Position Tracking : Add an additional network (potentially an RNN) to locate the position of the hands in the images.
- Analysis over Time: Implement a time-series component to track the location and shape of the hand over time.

Part VI: Supporting Documents

H. Analysis Report

R Notebook Link

I. Executive Summary

Document Link

J. Analysis Presentation

Panopto Link

Slides

J. Sources

Chollet François, Kalinowski, T., & Allaire, J. J. (2022). Deep learning with R. Manning.

Google Developers. (2022, July 18). ML Practicum: Image Classification. Google Developers. Retrieved August 2, 2022, from https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

Hickey, S., & Leske, H. (2021, April 22). *ASL interpreting: What you need to know about the ASL services market*. Nimdzi. Retrieved September 21, 2022, from https://www.nimdzi.com/asl-interpreting/

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional Neural Networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Lexset. (May 2022). Synthetic ASL Alphabet, 1.0. Retrieved Sept 10, 2022 from https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet.

Rizvi, M. S. Z. (2020, October 19). CNN image classification: Image Classification using CNN. Analytics Vidhya. Retrieved August 23, 2022, from https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/

The Stanford NLP Group. (2022). Machine Translation. The Stanford Natural Language Processing Group. Retrieved September 19, 2022, from https://nlp.stanford.edu/projects/mt.shtml