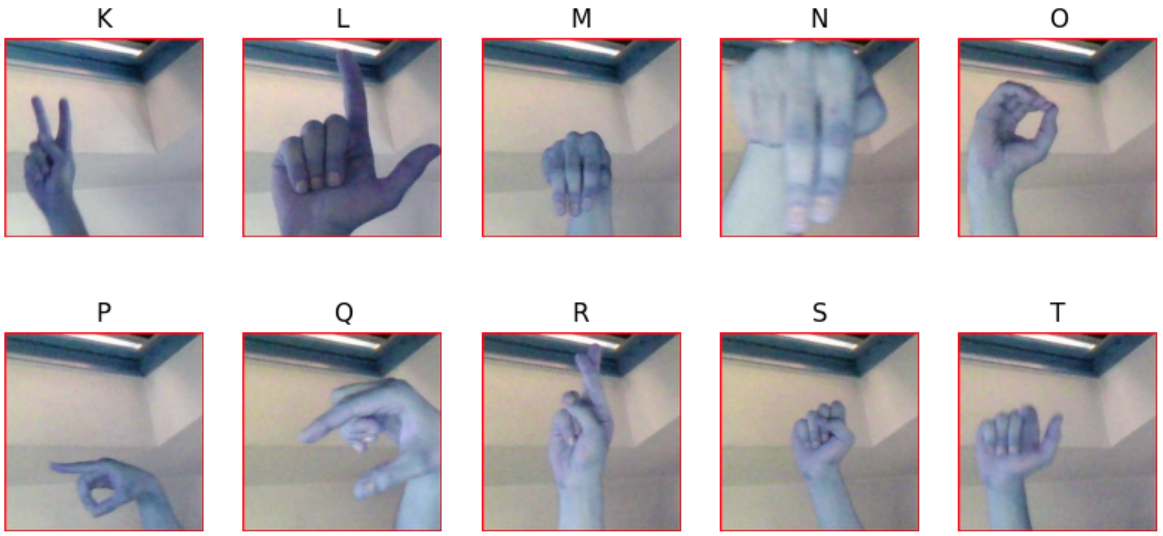
Classification of American Sign Language: Assignment 3

Yehudit Brickner 328601018
Yaakov Khodorkovski 207045063
Daniel Zaken 207296989



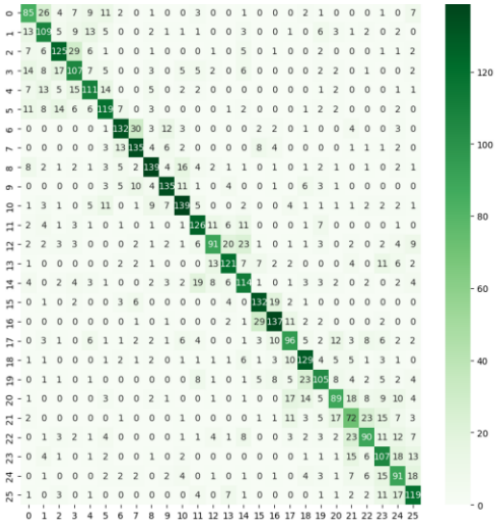
Our project is classifying asl (American sign language) letters. The data set can be found here: https://www.kaggle.com/datasets/grassknoted/asl-alphabet .
In this dataset, there are 78,000 images in 26 classes: A-Z. (3000 images for each class)
The images are 200p x 200p = 40,000p.
Because that is a lot of features we decided to resize the images to 64 x 64.
We split the data into 70% training and 30% testing.
We took 80% of the testing data for validation.
For each class, we will have approximately 2100 images in the train set, 720 images in the validation set, and 180 images in the test set.

For the first part of the project, we created a simple multi-layer perceptron (MLP) with 0,2,5 hidden layers with softmax for the classification.
For each of the above neural networks, we tried changing the respective parameters of the model: learning rate, epochs, with/without mini-batches, and the size of the layers, until we found a network that gave us good validation results.

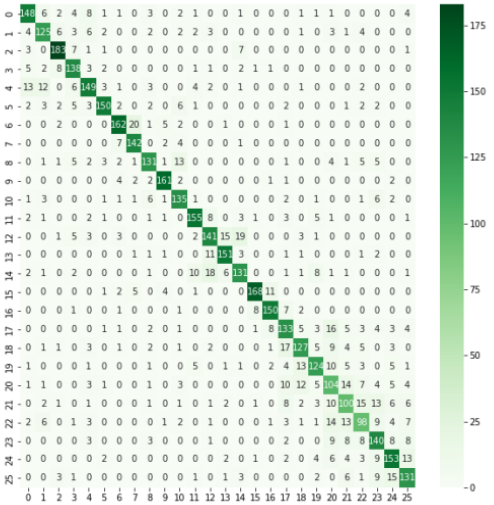
	Model type	Epochs	Learning rate	Layers	Accuracy
Model I	Logistic Regression SoftMax	75000 using minibatches, each epoch had 1/5 of the train set	0.01		63.14%
Model II	MLP Soft SoftMax max	30000	0.001	2 layers (1040,520)	77.54%
Model III	MLP SoftMax	17000	0.001	5 layers (1040,520,260, 260,130)	77.99%

Confusion Matrix

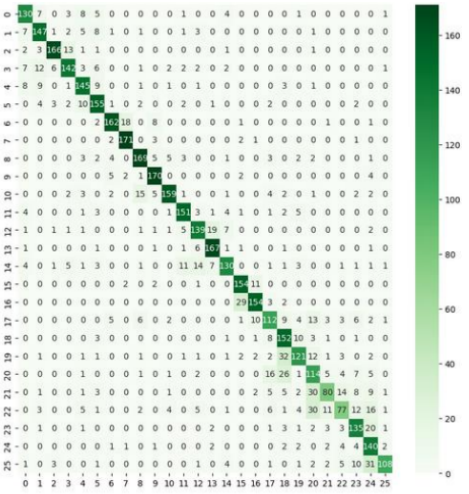
Logistic Regression SoftMax



MLP 2 Layers SoftMax



MLP 5 Layers SoftMax



For the second part, we used a Convolution Neural Network to classify the data. Again we tried playing with the parameters, we tried adding different convolutional layers with different-sized kernels, and different padding to the kernel. We tried changing where we have max pooling layers. We played with the learning rate, number of epochs, and batch size.

This model has 3 convolutional layers with a kernel of size 5x5. The activation is ReLu.

The first convolutional layer has 8 kernels, and everyone after has double the amount of the one before. After each convolutional layer we have a max pooling layer of 2x2. After the last max pooling layer, we used a flatten layer to get a vector. We then did 3 fully connected layers.

We ran this model for 30 epochs with a batch size of 10 and a learning rate of 0.001, and then another 15 epochs with batch size of 10 and learning rate of 0.0001.

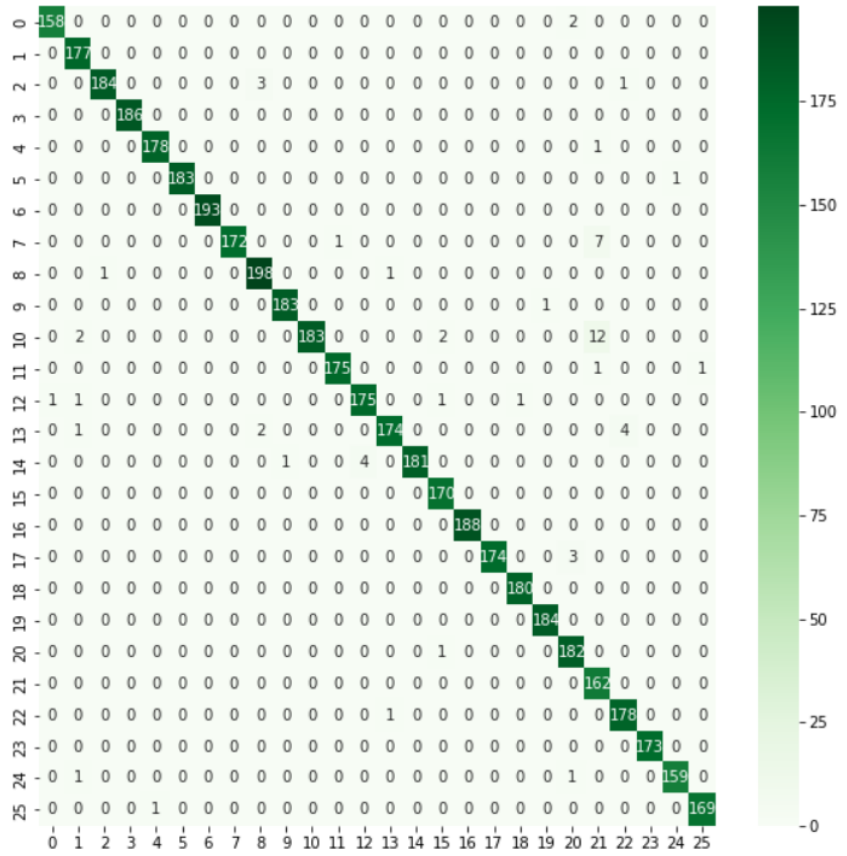
We decided to use a CNN for the second part of this assignment because it works well with images. The reason it works well with Images is that it takes into consideration each pixel and its surrounding to try and find bigger features like edges, corners, and more. Together it can find small features and slowly connect them together to bigger features.

With this model we got 98.7% accuracy

```
with tf.device('/CPU:0'):
    model = keras.models.Sequential()

    model.add(layers.Conv2D(8,(5,5), strides=(1,1), padding="valid", activation='relu', input_shape=(64,64,3)))
    model.add(layers.MaxPool2D((2,2)))
    model.add(layers.Conv2D(16, 5, activation='relu'))
    model.add(layers.MaxPool2D((2,2)))
    model.add(layers.Conv2D(32,5, activation='relu'))
    model.add(layers.MaxPool2D((2,2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(100, activation='relu'))
    model.add(layers.Dense(26))
    print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 60, 60, 8)         608

 max_pooling2d (MaxPooling2D  (None, 30, 30, 8)        0
 )

 conv2d_1 (Conv2D)           (None, 26, 26, 16)        3216

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 16)       0
 2D)

 conv2d_2 (Conv2D)           (None, 9, 9, 32)          12832

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 32)         0
 2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 100)               51300

 dense_1 (Dense)             (None, 26)                2626

=================================================================
Total params: 70,582
Trainable params: 70,582
Non-trainable params: 0
_________________________________________________________________
None
```

Confusion Matrix



Accuracy score: (TP + TN) / (TP + TN + FP + FN).A high accuracy score means that the model is making a large proportion of correct predictions.
Precision score: TP / (TP + FP).A high precision score means that the model is not making many false positive predictions.
Recall score: TP / (TP + FN). A high recall score means that the model is not missing many positive cases.
F1 score: It is calculated as 2 * (precision * recall) / (precision + recall). A high F1 score indicates a balance between precision and recall.
Because our data is multiclassed we used weighted average to calculate these scores.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
with tf.device('/CPU:0'):
  print('Accuracy score : ', accuracy_score(Y_test, mypred))
  print('Precision score : ', precision_score(Y_test, mypred, average='weighted'))
  print('Recall score : ',recall_score(Y_test,mypred, average='weighted'))
  print('F1 score : ', f1_score(Y_test,mypred, average='weighted'))
```

```
Accuracy score :  0.986965811965812
Precision score :  0.9875858083717955
Recall score :  0.986965811965812
F1 score :  0.9870160964755106
```