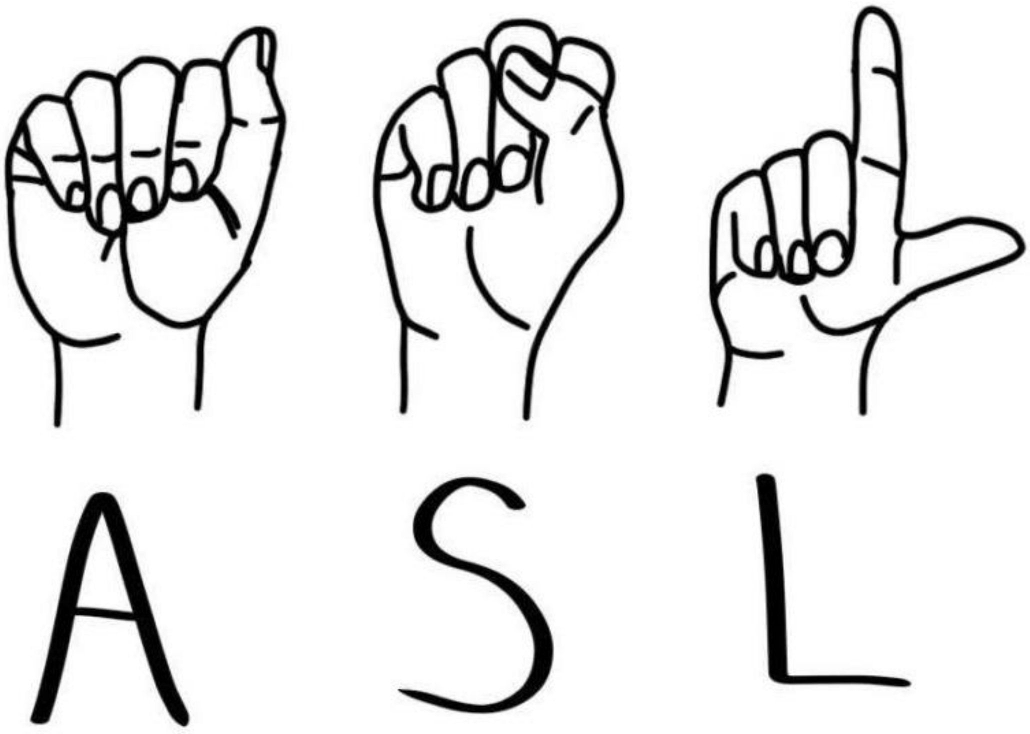
UNIVERSITY OF SALERNO

Department of Computer Science

Master of Science in IOT

Context Aware Security Analytics in Computer Vision



Project Report: American Sign Language (ASL) recognition

Supervisors:

Prof. Fabio Narducci

Dr. Carmen Bisogni

Author:

Mohamed Rammszy Careem

Table of content:

Abstract:

Sign language is a form of communication language designed to link a deaf-mute person to the world. To express an idea, it requires the use of hand gestures and body movement. However, the bulk of the general population remain uneducated to understand the sign language. Therefore, a translator must ease the communication. Thus, this project will aim to classify various American Sign Language (ASL) alphabets coupled in a Dataset using Convolutional Neural Network (CNN) to achieve precise accuracy.

Introduction:

American Sign Language (ASL) is a comprehensive, natural language that is communicated with the movement of hands and face. Moreover, American Sign Language (ASL) enables the deaf group a way to interact within the group itself as well as to the outside world. However, American Sign Language (ASL) is known by limited people, be it the signs or gestures. Nonetheless, due to the evolving field of Artificial Neural Networks and Deep Learning, it is now possible to build a system that can recognize objects or even objects of various categories (like cat’s vs dogs). Similar approach can be taken to understand American Sign Language using deep learning models trained with the help of a dataset to distinguish the signs.

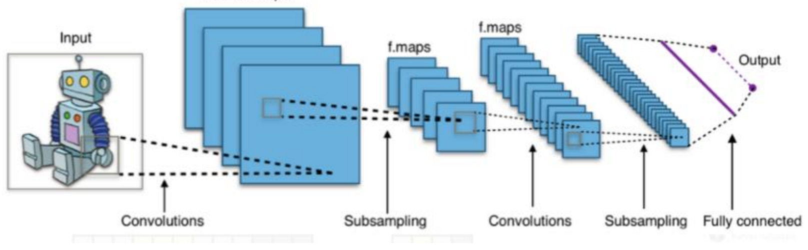
Related Work:

Having not focused on deep learning before and having no knowledge about American Sign Language (ASL) dataset classified using Convolutional Neural Network, I had to undergo further research on the current solutions out there in the wider work. During the last few months, more developers that have begun to appear with similar applications. One being “deeplens-asl,” an American Sign Language alphabet classifier. Training was done using transfer learning from squeezeNet with eighteen layers. The data for this training was collected using Amazon SageMaker. SageMaker took images one user at a time, using their input to capture and label the image. The data collected from this tool consisted of all alphabets but had to use special signs for letters ‘j’ and ‘z’ as the classifier could not train the model using their special characters. Due to this, the final model can also only detect these two letters if the user uses these special signs for these letters. Deeplens-als’s final model could classifies ASL alphabet gestures, but with only with 40% accuracy. The application uses amazon’s device stream to get input. The input videos for this application needs to account for a couple of caveats. This includes having to go slower if there are words with repeating letters like “letter” for it to detect the two “tt” apart and having to wait between words for the classifier to complete detecting the earlier letter. If the pictures have an almost white background, with only the signs visible in the images, it can classify the gestures with higher level of accuracy.

Proposed Method:

The aim of this project was to classify images of the American Sign Language alphabet from a Kaggle dataset and to build a neural network that could classify the images with a prominent level of accuracy. Thus, labeling this as a vision-based approach, due to its nature of using human input rather than machine-based input. The ASL alphabet dataset has 87,000 images spanning 29 classes containing 3,000 images each. 26 of these classes are the letters A-Z and the other 3 are the signs for nothing, space and delete. These 87,000 images were divided into 78,300 images that would be fed into the model as training data and 8,700 that would be used as validation data. In addition to splitting the data into training and validation data, a generator was also used to augment the images (rotate them, shift them sideways, etc.) so that the data became less similar, and the model would have to generalize rather than just memorizing certain images. The code is laid out in a way such that all the work is split up into functions which are called in order at the end of the code. The training, validation and test data was all run through two models, one consisting of fully connected layers and the other being a convolutional neural network, so all the functions were run twice (once for each model) and written so that depending on the model type the data is processed and run through the model accordingly. The fully connected model is a standard deep learning model with two hidden layers, the input layer having 4096 nodes in to take in the grayscale 73*73 image. The convolutional neural network was adapted from Running Kaggle Kernels with a GPU, with the addition of Batch Normalization, Dropout, and kernel regularizes to reduce overfitting.

1. **Convolution Neural Network (CNN)**
   Convolution Neural Network (CNN) are one of the ways in which a computer can classify an image. They can train on thousands of photos and learn to classify each one in its correct category, or translation in this case. I chose to do a simple ASL translation, where I translated the ASL *alphabet*. It consists of 26 hand signs, and one delete and space sign. I translated the signs into English letters. CNNs are cool in that they can classify images. It's quite easy for a human to identify the difference between two items, such as a dog and a cat, but it's a lot harder for a computer to do this. Computers don't truly "see" images. They read them as a series of numbers arranged in an array. A CNN consists of multiple layers. The layers are usually convolutional, pooling and fully connected layers.



2. **Data Preparation**
   The dataset used was taken from ASL Alphabet (Kaggle) . The data is organized into 29 folders with 3000 pictures in each folder for each letter of the alphabet.

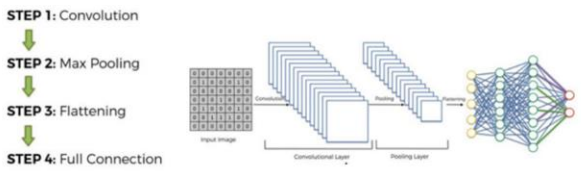
3. **Convolution**
   The convolutional layer consists of a kernel/filter with a designated size that slides, or convolves, over the pixels multiplying and summing values, finally outputting it into a new smaller simplified matrix. The filter travels over every pixel in the photo creating a new matrix called a feature matrix. This new smaller matrix is important because it highlights the most key features (hence the name) in the picture. It is also easier to train on because smaller = less weights = less training needed to find those weights.

4. **ReLu**
   An added operation called ReLU has been used after every Convolution operation. ReLU stands for the Rectified Linear Unit and is a non-linear operation. ReLU is an element-wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our Convolution Neural Network as most real-world data we would want our Convolution Neural Network to learn would be non-linear (Convolution is a linear operation — element-wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

5. **Pooling**
   The next layer is a pooling layer. The pooling layer further reduces the matrix size. It passes a pooling kernel over the feature matrix and takes either the highest pixel value(max-pooling) or the average(average-pooling).

6. **Flattening**
   Flattening is the process of converting al the resultant 2-dimensional arrays into a single long continuous linear vector

7. **Fully Connected Layer**
   Here’s where the classification happens. The matrix is first flattened into a vector and then passed through a neural network. The neural network it passes through is like an Artificial Neural Network in that it passes the vector through, applying weights and biases finally ending up with a classification. The CNN classifies the image by using a SoftMax activation function which gives the probability the input is from a certain class.



- **Convolutional and Pooling Layers**

  - In the first line, the number of filters and filter size is defined.
  - After the matrix undergoes convolution, forming a feature matrix, it passes through batch normalization. This reduces the shift of hidden layer values. This makes training easier because it stabilizes the weights, improving accuracy.
  - We next run it through a Re-LU function. This brings limited non-linearity to the layer, allowing the CNN to understand the complicated pictures inputted.
  - The next line is where pooling occurs.
  - Finally, the matrix is passed through a dropout layer.
  - The output of the first convolutional layer now becomes the input of the next layer.

- **Fully Connected Layers**

  - Now that the data can pass through the neural network, a dense layer is used. A
  - Then, it goes through batch normalization.
  - Then a ReLU function is used for activation.
  - The nodes are once again dropped out using dropout.
  - There are two fully connected layers.
  - Once past the second fully connected layer, the output is put through a SoftMax function which
  - is used to give the probability the image belongs to one of the 29 classes.
  - The last few lines set the learning rate and evaluate the accuracy of the model.

- **Tech Stack used in this project are.**

  1. Jupyter Notebook is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages".
  2. Python is a high-level, general-purpose programming language.
  3. NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays
  4. Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.
  5. TensorFlow is one of the most popular Machine Learning (ML) APIs, which allows to automate multiple real-world tasks such as image detection. Initially the computational graph nodes are defined, resulting in the final computation taking place inside the session.
  6. Keras is the high-level library created on TensorFlow. Supplies a type of sci-kit learning API written in Python for building neural networks. The key idea behind the development of Keras is to simplify investigations through rapid prototyping.

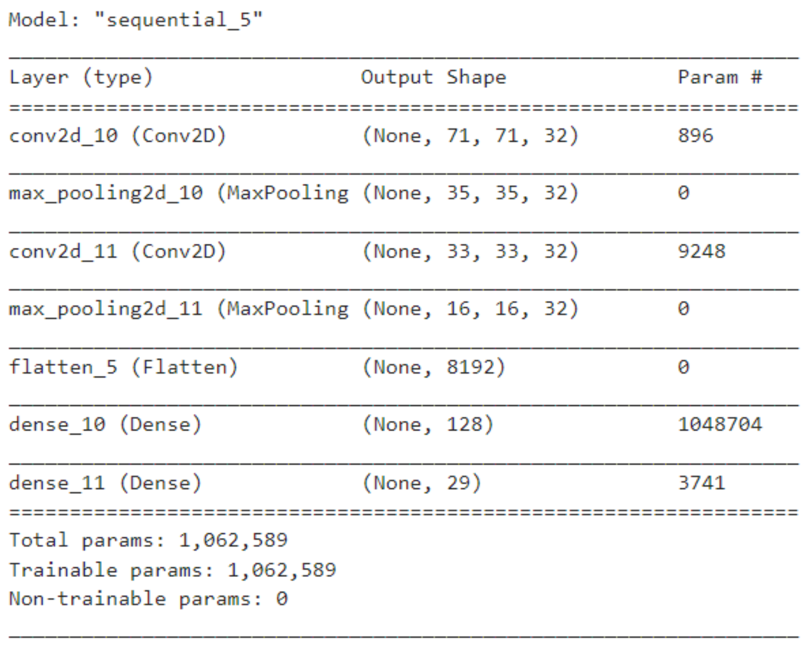
Experiment:

1. **Dataset Preparation**
   The data is organized into 29 folders with 3000 pictures in each folder for each letter of the alphabet. The 3 extra folders are space, delete and nothing. From the total 87,000 images across all the folders, 90% of the photos are in training, and 10% are in testing. To have more generalized image data, a generator was used to amplify the images such as rotating them etc.

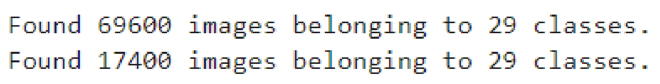
2. **Models**
   For this experiment, 3 different types of models with different layers and epochs were used, to have a better variation of the results. Using the Convolution Neural Network (CNN) the transfer learning concept is used, where the initial model (ASLModel01) is trained first with reference to the dataset, enabling the knowledge gained in the manner of "weights" by this model to be transferred to the other two models/neural networks, hence with addition of fully connected layers on top of it, the pre-trained model was used as a feature extractor.

❖ **ASLModel01**

- **Layers:** 02
- **Epochs:** 13

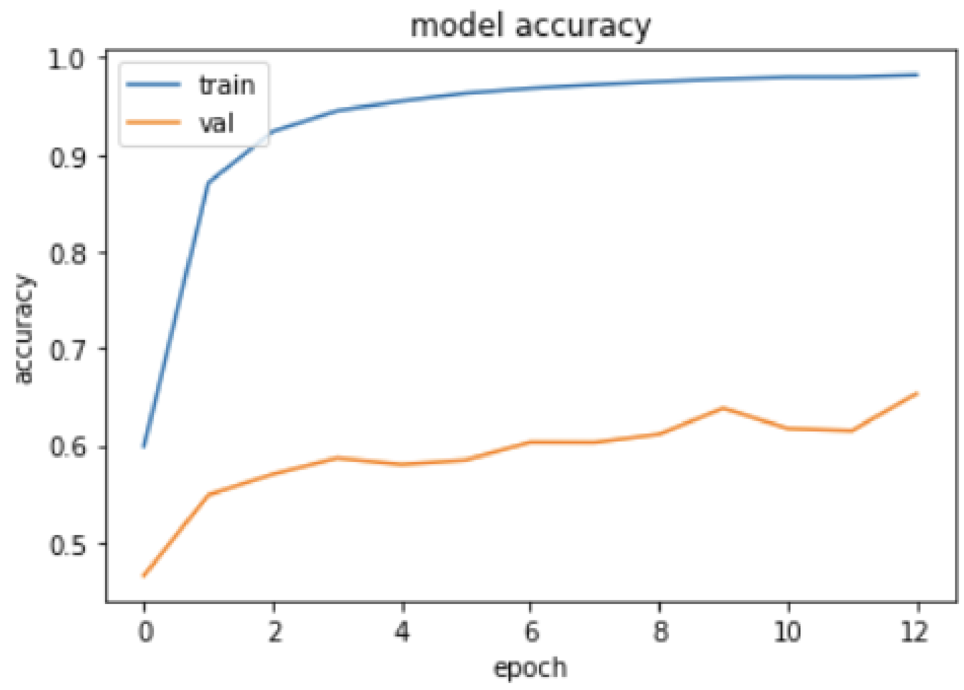


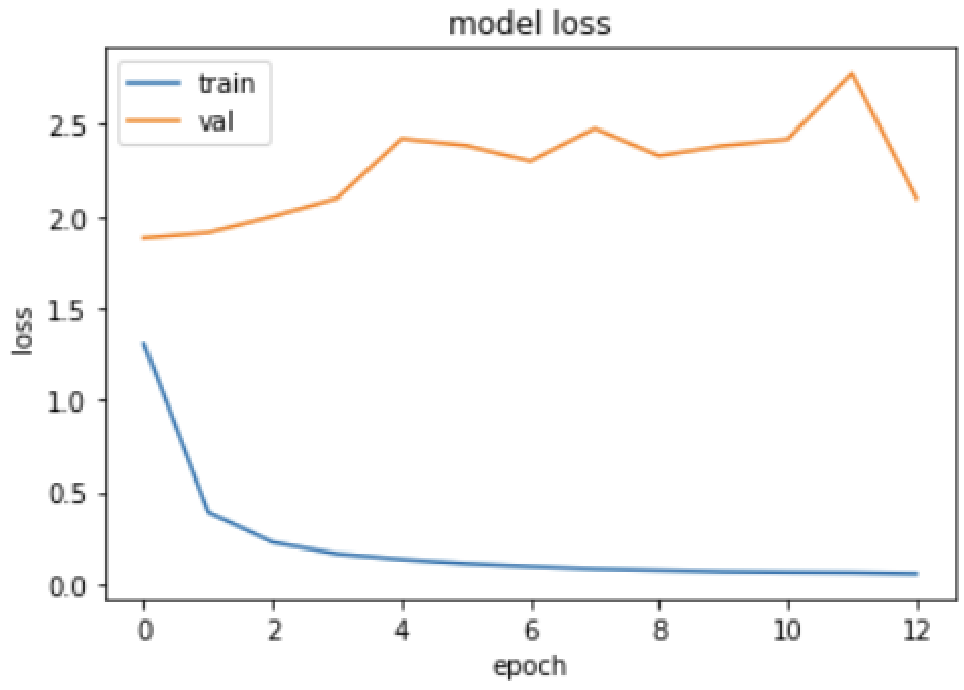
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 71, 71, 32)        896
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 35, 35, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 33, 33, 32)        9248
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               1048704
_________________________________________________________________
dense_11 (Dense)             (None, 29)                3741
=================================================================
Total params: 1,062,589
Trainable params: 1,062,589
Non-trainable params: 0
_________________________________________________________________
```



```
Found 69600 images belonging to 29 classes.
Found 17400 images belonging to 29 classes.
```

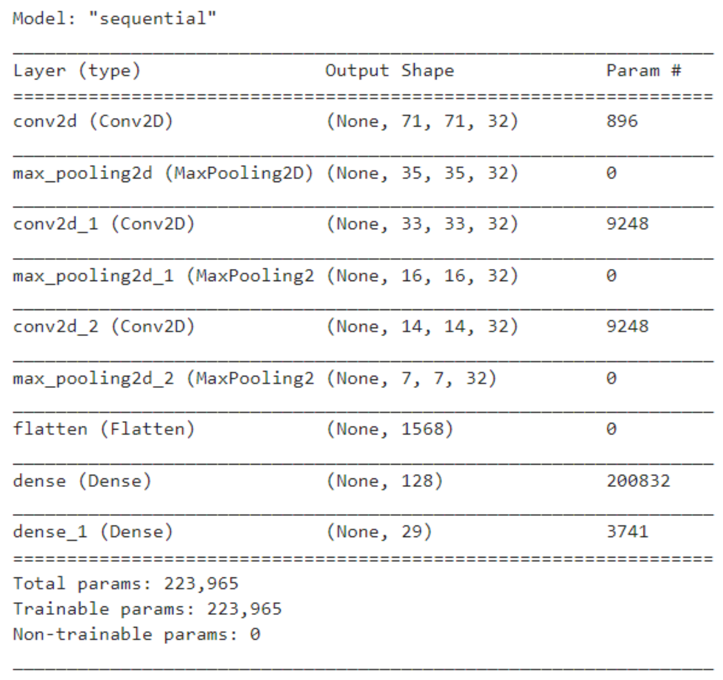
```
Epoch 1/13
2175/2175 [==============================] - 930s 427ms/step - loss: 1.3073 - accuracy: 0.5991 - val
_loss: 1.8826 - val_accuracy: 0.4657
Epoch 2/13
2175/2175 [==============================] - 844s 388ms/step - loss: 0.3889 - accuracy: 0.8709 - val
_loss: 1.9129 - val_accuracy: 0.5487
Epoch 3/13
2175/2175 [==============================] - 822s 378ms/step - loss: 0.2265 - accuracy: 0.9237 - val
_loss: 2.0009 - val_accuracy: 0.5702
Epoch 4/13
2175/2175 [==============================] - 825s 379ms/step - loss: 0.1627 - accuracy: 0.9453 - val
_loss: 2.0987 - val_accuracy: 0.5870
Epoch 5/13
2175/2175 [==============================] - 771s 354ms/step - loss: 0.1331 - accuracy: 0.9551 - val
_loss: 2.4232 - val_accuracy: 0.5803
Epoch 6/13
2175/2175 [==============================] - 479s 220ms/step - loss: 0.1102 - accuracy: 0.9633 - val
_loss: 2.3859 - val_accuracy: 0.5849
Epoch 7/13
2175/2175 [==============================] - 296s 136ms/step - loss: 0.0946 - accuracy: 0.9683 - val
_loss: 2.3012 - val_accuracy: 0.6031
Epoch 8/13
2175/2175 [==============================] - 292s 134ms/step - loss: 0.0817 - accuracy: 0.9722 - val
_loss: 2.4778 - val_accuracy: 0.6029
Epoch 9/13
2175/2175 [==============================] - 294s 135ms/step - loss: 0.0738 - accuracy: 0.9754 - val
_loss: 2.3308 - val_accuracy: 0.6114
Epoch 10/13
2175/2175 [==============================] - 292s 134ms/step - loss: 0.0665 - accuracy: 0.9781 - val
_loss: 2.3847 - val_accuracy: 0.6382
Epoch 11/13
2175/2175 [==============================] - 285s 131ms/step - loss: 0.0629 - accuracy: 0.9800 - val
_loss: 2.4200 - val_accuracy: 0.6174
Epoch 12/13
2175/2175 [==============================] - 360s 165ms/step - loss: 0.0607 - accuracy: 0.9802 - val
_loss: 2.7792 - val_accuracy: 0.6151
Epoch 13/13
2175/2175 [==============================] - 406s 187ms/step - loss: 0.0546 - accuracy: 0.9823 - val
_loss: 2.0981 - val_accuracy: 0.6533
```



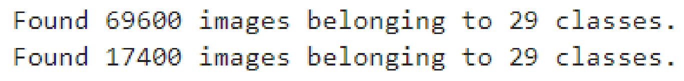


```
Found 28 images belonging to 29 classes.
1/1 [==============================] - 0s 181ms/step - loss: 0.0017 - accuracy: 1.0000
1.0
0.0017350753769278526
```

- **ASLModel02**
  - **Layers:** 03
  - **Epochs:** 26



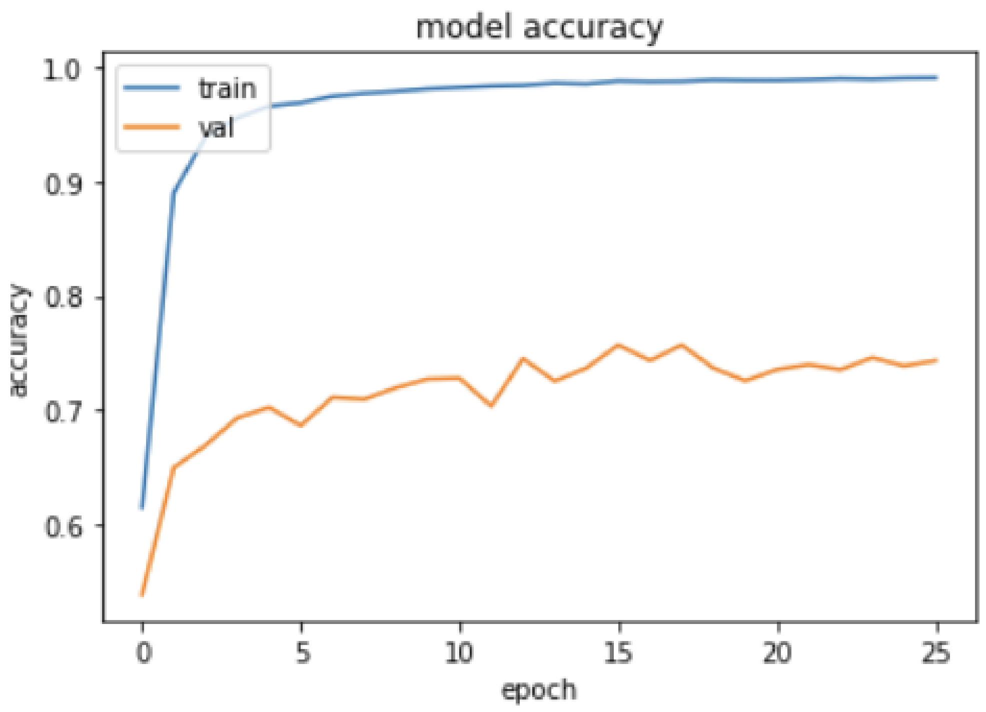
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 71, 71, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 35, 35, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 33, 33, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 1568)              0
_________________________________________________________________
dense (Dense)                (None, 128)               200832
_________________________________________________________________
dense_1 (Dense)              (None, 29)                3741
=================================================================
Total params: 223,965
Trainable params: 223,965
Non-trainable params: 0
_________________________________________________________________
```

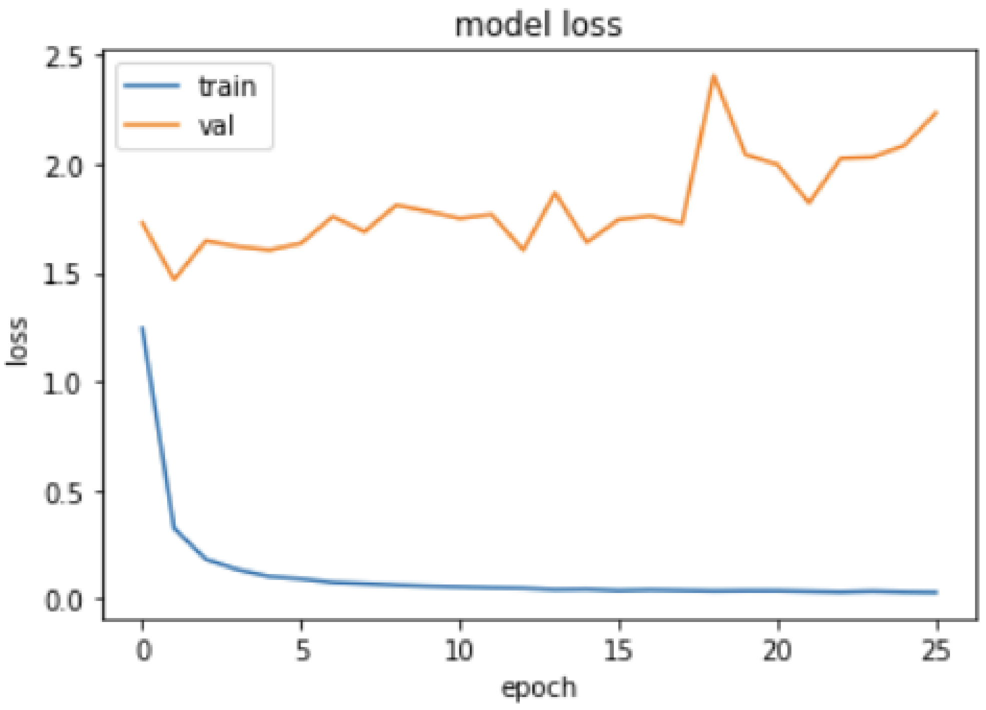


```
Found 69600 images belonging to 29 classes.
Found 17400 images belonging to 29 classes.
```

```
Epoch 1/26
2175/2175 [==============================] - 987s 453ms/step - loss: 1.2427 - accuracy: 0.6142 - val
_loss: 1.7286 - val_accuracy: 0.5378
Epoch 2/26
2175/2175 [==============================] - 970s 446ms/step - loss: 0.3231 - accuracy: 0.8905 - val
_loss: 1.4685 - val_accuracy: 0.6492
Epoch 3/26
2175/2175 [==============================] - 935s 430ms/step - loss: 0.1816 - accuracy: 0.9392 - val
_loss: 1.6460 - val_accuracy: 0.6686
Epoch 4/26
2175/2175 [==============================] - 933s 429ms/step - loss: 0.1334 - accuracy: 0.9559 - val
_loss: 1.6201 - val_accuracy: 0.6928
Epoch 5/26
2175/2175 [==============================] - 895s 411ms/step - loss: 0.1014 - accuracy: 0.9663 - val
_loss: 1.6038 - val_accuracy: 0.7019
Epoch 6/26
2175/2175 [==============================] - 909s 418ms/step - loss: 0.0907 - accuracy: 0.9693 - val
_loss: 1.6347 - val_accuracy: 0.6860
Epoch 7/26
2175/2175 [==============================] - 759s 349ms/step - loss: 0.0743 - accuracy: 0.9751 - val
_loss: 1.7575 - val_accuracy: 0.7107
Epoch 8/26
2175/2175 [==============================] - 691s 318ms/step - loss: 0.0684 - accuracy: 0.9776 - val
_loss: 1.6881 - val_accuracy: 0.7091
```

```
Epoch 9/26
2175/2175 [==============================] - 1130s 520ms/step - loss: 0.0619 - accuracy: 0.9793 - v
al_loss: 1.8106 - val_accuracy: 0.7193
Epoch 10/26
2175/2175 [==============================] - 475s 218ms/step - loss: 0.0569 - accuracy: 0.9815 - val
_loss: 1.7814 - val_accuracy: 0.7268
Epoch 11/26
2175/2175 [==============================] - 474s 218ms/step - loss: 0.0535 - accuracy: 0.9827 - val
_loss: 1.7490 - val_accuracy: 0.7276
Epoch 12/26
2175/2175 [==============================] - 471s 217ms/step - loss: 0.0499 - accuracy: 0.9840 - val
_loss: 1.7676 - val_accuracy: 0.7030
Epoch 13/26
2175/2175 [==============================] - 478s 220ms/step - loss: 0.0479 - accuracy: 0.9846 - val
_loss: 1.6043 - val_accuracy: 0.7448
Epoch 14/26
2175/2175 [==============================] - 474s 218ms/step - loss: 0.0417 - accuracy: 0.9865 - val
_loss: 1.8670 - val_accuracy: 0.7250
Epoch 15/26
2175/2175 [==============================] - 460s 211ms/step - loss: 0.0434 - accuracy: 0.9858 - val
_loss: 1.6391 - val_accuracy: 0.7366
Epoch 16/26
2175/2175 [==============================] - 464s 213ms/step - loss: 0.0374 - accuracy: 0.9883 - val
_loss: 1.7434 - val_accuracy: 0.7567
Epoch 17/26
2175/2175 [==============================] - 477s 219ms/step - loss: 0.0403 - accuracy: 0.9877 - val
_loss: 1.7606 - val_accuracy: 0.7432
Epoch 18/26
2175/2175 [==============================] - 493s 227ms/step - loss: 0.0379 - accuracy: 0.9880 - val
_loss: 1.7269 - val_accuracy: 0.7567
Epoch 19/26
2175/2175 [==============================] - 500s 230ms/step - loss: 0.0356 - accuracy: 0.9892 - val
_loss: 2.4053 - val_accuracy: 0.7364
Epoch 20/26
2175/2175 [==============================] - 485s 223ms/step - loss: 0.0368 - accuracy: 0.9889 - val
_loss: 2.0429 - val_accuracy: 0.7253
Epoch 21/26
2175/2175 [==============================] - 463s 213ms/step - loss: 0.0366 - accuracy: 0.9888 - val
_loss: 1.9974 - val_accuracy: 0.7352
Epoch 22/26
2175/2175 [==============================] - 465s 214ms/step - loss: 0.0336 - accuracy: 0.9895 - val
_loss: 1.8209 - val_accuracy: 0.7395
Epoch 23/26
2175/2175 [==============================] - 461s 212ms/step - loss: 0.0306 - accuracy: 0.9905 - val
_loss: 2.0256 - val_accuracy: 0.7351
Epoch 24/26
2175/2175 [==============================] - 459s 211ms/step - loss: 0.0342 - accuracy: 0.9900 - val
_loss: 2.0312 - val_accuracy: 0.7456
Epoch 25/26
2175/2175 [==============================] - 475s 218ms/step - loss: 0.0305 - accuracy: 0.9908 - val
_loss: 2.0847 - val_accuracy: 0.7384
Epoch 26/26
2175/2175 [==============================] - 487s 224ms/step - loss: 0.0288 - accuracy: 0.9911 - val
_loss: 2.2350 - val_accuracy: 0.7430
```

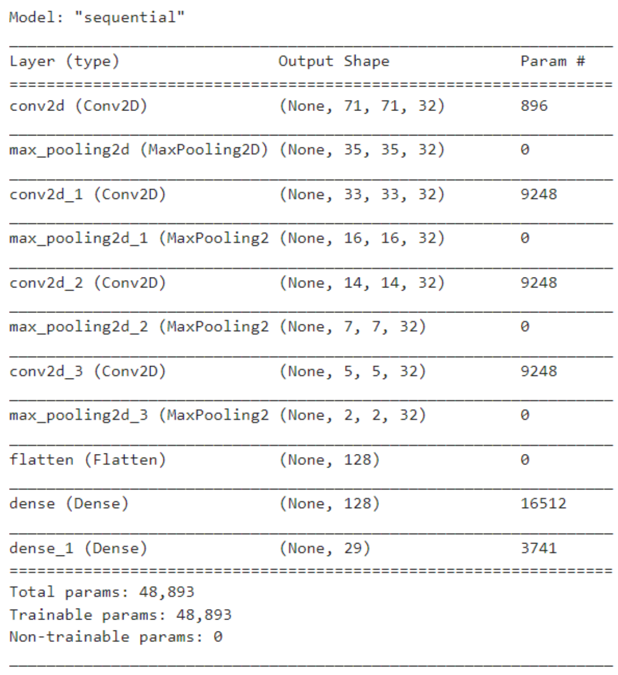




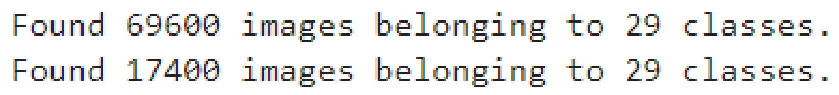
```
Found 28 images belonging to 29 classes.
1/1 [==============================] - 0s 303ms/step - loss: 0.0463 - accuracy: 0.9643
0.9642857313156128
0.04631423205137253
```

❖ **ASLModel03**

- **Layers:** 04
- **Epochs:** 39



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 71, 71, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 35, 35, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 33, 33, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 5, 32)          9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 128)               16512
_________________________________________________________________
dense_1 (Dense)              (None, 29)                3741
=================================================================
Total params: 48,893
Trainable params: 48,893
Non-trainable params: 0
_________________________________________________________________
```

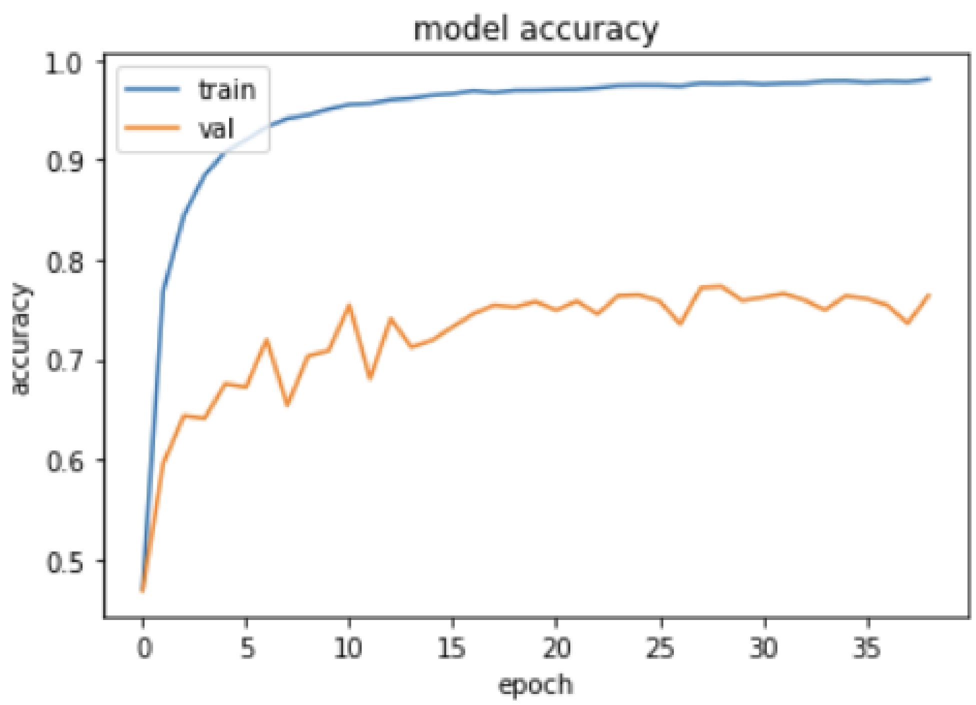


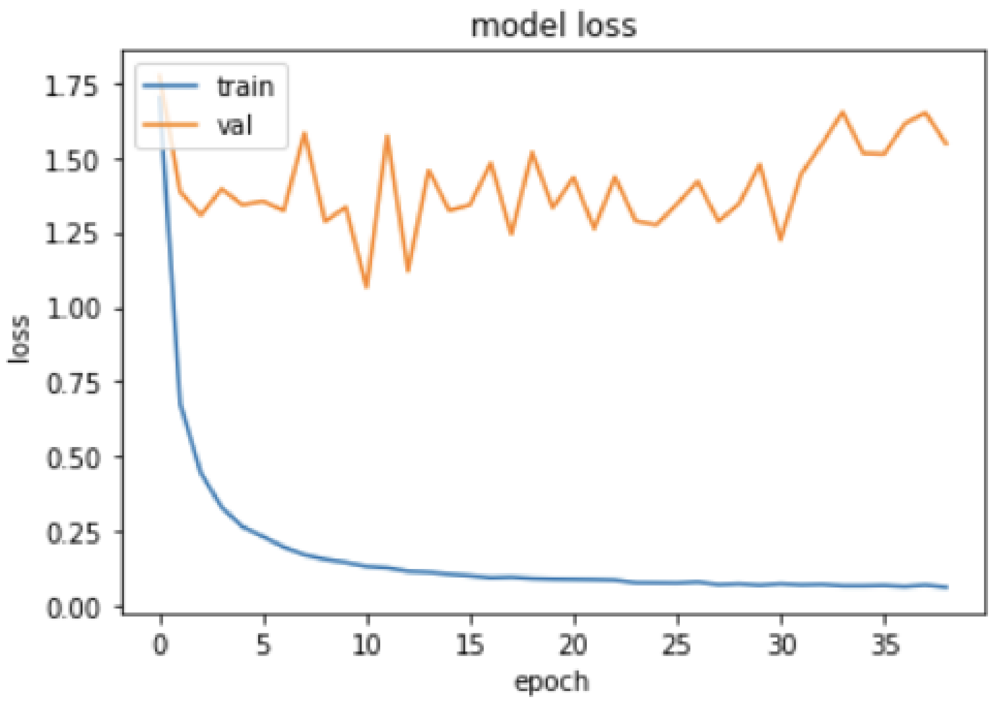
```
Found 69600 images belonging to 29 classes.
Found 17400 images belonging to 29 classes.
```

```
Epoch 1/39
2175/2175 [==============================] - 959s 440ms/step - loss: 1.7011 - accuracy: 0.4704 - val
_loss: 1.7732 - val_accuracy: 0.4683
Epoch 2/39
2175/2175 [==============================] - 958s 440ms/step - loss: 0.6755 - accuracy: 0.7689 - val
_loss: 1.3854 - val_accuracy: 0.5960
Epoch 3/39
2175/2175 [==============================] - 942s 433ms/step - loss: 0.4434 - accuracy: 0.8448 - val
_loss: 1.3080 - val_accuracy: 0.6437
Epoch 4/39
2175/2175 [==============================] - 943s 433ms/step - loss: 0.3292 - accuracy: 0.8849 - val
_loss: 1.3953 - val_accuracy: 0.6411
Epoch 5/39
2175/2175 [==============================] - 890s 409ms/step - loss: 0.2644 - accuracy: 0.9083 - val
_loss: 1.3423 - val_accuracy: 0.6757
Epoch 6/39
2175/2175 [==============================] - 920s 423ms/step - loss: 0.2311 - accuracy: 0.9203 - val
_loss: 1.3544 - val_accuracy: 0.6724
Epoch 7/39
2175/2175 [==============================] - 745s 342ms/step - loss: 0.1959 - accuracy: 0.9331 - val
_loss: 1.3224 - val_accuracy: 0.7201
Epoch 8/39
2175/2175 [==============================] - 705s 324ms/step - loss: 0.1700 - accuracy: 0.9417 - val
_loss: 1.5821 - val_accuracy: 0.6539
```

```
Epoch 9/39
2175/2175 [==============================] - 1083s 498ms/step - loss: 0.1546 - accuracy: 0.9453 - v
al_loss: 1.2863 - val_accuracy: 0.7036
Epoch 10/39
2175/2175 [==============================] - 514s 236ms/step - loss: 0.1444 - accuracy: 0.9511 - val
_loss: 1.3346 - val_accuracy: 0.7090
Epoch 11/39
2175/2175 [==============================] - 541s 249ms/step - loss: 0.1311 - accuracy: 0.9558 - val
_loss: 1.0656 - val_accuracy: 0.7548
Epoch 12/39
2175/2175 [==============================] - 535s 246ms/step - loss: 0.1263 - accuracy: 0.9568 - val
_loss: 1.5716 - val_accuracy: 0.6808
Epoch 13/39
2175/2175 [==============================] - 542s 249ms/step - loss: 0.1148 - accuracy: 0.9605 - val
_loss: 1.1198 - val_accuracy: 0.7410
Epoch 14/39
2175/2175 [==============================] - 526s 242ms/step - loss: 0.1116 - accuracy: 0.9621 - val
_loss: 1.4579 - val_accuracy: 0.7123
Epoch 15/39
2175/2175 [==============================] - 515s 237ms/step - loss: 0.1050 - accuracy: 0.9651 - val
_loss: 1.3227 - val_accuracy: 0.7194
Epoch 16/39
2175/2175 [==============================] - 513s 236ms/step - loss: 0.0999 - accuracy: 0.9665 - val
_loss: 1.3422 - val_accuracy: 0.7329
Epoch 17/39
2175/2175 [==============================] - 521s 240ms/step - loss: 0.0932 - accuracy: 0.9695 - val
_loss: 1.4815 - val_accuracy: 0.7459
Epoch 18/39
2175/2175 [==============================] - 543s 250ms/step - loss: 0.0951 - accuracy: 0.9677 - val
_loss: 1.2446 - val_accuracy: 0.7545
Epoch 19/39
2175/2175 [==============================] - 542s 249ms/step - loss: 0.0905 - accuracy: 0.9699 - val
_loss: 1.5177 - val_accuracy: 0.7525
Epoch 20/39
2175/2175 [==============================] - 516s 237ms/step - loss: 0.0887 - accuracy: 0.9701 - val
_loss: 1.3323 - val_accuracy: 0.7582
Epoch 21/39
2175/2175 [==============================] - 528s 243ms/step - loss: 0.0878 - accuracy: 0.9706 - val
_loss: 1.4335 - val_accuracy: 0.7495
Epoch 22/39
2175/2175 [==============================] - 521s 240ms/step - loss: 0.0869 - accuracy: 0.9710 - val
_loss: 1.2623 - val_accuracy: 0.7587
Epoch 23/39
2175/2175 [==============================] - 530s 244ms/step - loss: 0.0854 - accuracy: 0.9724 - val
_loss: 1.4350 - val_accuracy: 0.7456
Epoch 24/39
2175/2175 [==============================] - 548s 252ms/step - loss: 0.0764 - accuracy: 0.9748 - val
_loss: 1.2881 - val_accuracy: 0.7641
Epoch 25/39
2175/2175 [==============================] - 327s 150ms/step - loss: 0.0759 - accuracy: 0.9751 - val
_loss: 1.2748 - val_accuracy: 0.7649
Epoch 26/39
2175/2175 [==============================] - 292s 134ms/step - loss: 0.0753 - accuracy: 0.9751 - val
_loss: 1.3446 - val_accuracy: 0.7587
Epoch 27/39
```

```
2175/2175 [==============================] - 275s 126ms/step - loss: 0.0788 - accuracy: 0.9742 - val
_loss: 1.4202 - val_accuracy: 0.7357
Epoch 28/39
2175/2175 [==============================] - 382s 176ms/step - loss: 0.0700 - accuracy: 0.9772 - val
_loss: 1.2867 - val_accuracy: 0.7721
Epoch 29/39
2175/2175 [==============================] - 316s 145ms/step - loss: 0.0726 - accuracy: 0.9768 - val
_loss: 1.3460 - val_accuracy: 0.7733
Epoch 30/39
2175/2175 [==============================] - 283s 130ms/step - loss: 0.0681 - accuracy: 0.9774 - val
_loss: 1.4765 - val_accuracy: 0.7592
Epoch 31/39
2175/2175 [==============================] - 283s 130ms/step - loss: 0.0728 - accuracy: 0.9760 - val
_loss: 1.2243 - val_accuracy: 0.7624
Epoch 32/39
2175/2175 [==============================] - 289s 133ms/step - loss: 0.0692 - accuracy: 0.9771 - val
_loss: 1.4456 - val_accuracy: 0.7662
Epoch 33/39
2175/2175 [==============================] - 269s 124ms/step - loss: 0.0708 - accuracy: 0.9773 - val
_loss: 1.5455 - val_accuracy: 0.7598
Epoch 34/39
2175/2175 [==============================] - 271s 124ms/step - loss: 0.0661 - accuracy: 0.9792 - val
_loss: 1.6533 - val_accuracy: 0.7496
Epoch 35/39
2175/2175 [==============================] - 291s 134ms/step - loss: 0.0660 - accuracy: 0.9794 - val
_loss: 1.5145 - val_accuracy: 0.7639
Epoch 36/39
2175/2175 [==============================] - 274s 126ms/step - loss: 0.0673 - accuracy: 0.9780 - val
_loss: 1.5116 - val_accuracy: 0.7611
Epoch 37/39
2175/2175 [==============================] - 297s 137ms/step - loss: 0.0636 - accuracy: 0.9791 - val
_loss: 1.6140 - val_accuracy: 0.7547
Epoch 38/39
2175/2175 [==============================] - 270s 124ms/step - loss: 0.0686 - accuracy: 0.9785 - val
_loss: 1.6504 - val_accuracy: 0.7363
Epoch 39/39
2175/2175 [==============================] - 326s 150ms/step - loss: 0.0613 - accuracy: 0.9810 - val
_loss: 1.5467 - val_accuracy: 0.7640
```





```
Found 28 images belonging to 29 classes.
1/1 [==============================] - 0s 119ms/step - loss: 0.0105 - accuracy: 1.0000
1.0
0.010469486005604267
```

Findings:

Having 4 layers and after 39 epochs of training, the model achieved 100% validation accuracy within the dataset. Thus, making it clear that the more layers and epochs incorporated will give a more accurate output. That’s quite impressive, but there is a clear reason why the model was able to achieve this accuracy was due the dataset, looking at the pictures of one category within the dataset, they are all practically the same photo on the same background. Thus, it was easy for the model to achieve 100% accuracy because there wasn’t much variety between each photo resulting in high accuracy very quickly.

Conclusion:

This project enabled a vision-based functional real time American Sign Language translator that helps the deaf and mute community and the rest to communicate with them as needed. Maximum results were achieved with minimal loss and high accuracy with respect to the dataset used. Although this model does perform well with the given dataset, the results do not translate to more real-world scenarios and further work is needed to decide how to take such similar data and generalize it to any situation. Implementing added methods such as feature selection, F-Score selection, and Recursive Feature Elimination, which help find the most key features in a sample and end the other features, will further help increase training speeds and reduce overfitting. Dynamic Gestures, complex words to be conveyed using the motion of hand.

Bibliography:

- https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html
- https://www.kaggle.com/datasets/grassknoted/asl-alphabet
- https://jovian.ai/dianeegranger/asl-sign-language-deep-learning-project
- https://www.eecg.utoronto.ca/~jayar/mie324/asl.pdf
- https://jovian.ai/itsmohsin/asl
- https://www.astesj.com/publications/ASTESJ_050657.pdf
- https://jovian.ai/2019ucs0097/sign-detection-4b69a
- http://noiselab.ucsd.edu/ECE228-2021/projects/PresentationVideosPPT/9PPT.pdf
- https://www.kaggle.com/code/aminesnoussi/american-sign-language-recognition#overview
- https://www.kaggle.com/code/ryuodan/asl-detection-walkthrough
- https://www.kaggle.com/code/raghavrastogi75/asl-classification-using-tensorflow-99-accuracy
- https://opensource.google/projects/tensorflow
- https://keras.io/
- https://www.knime.com/deeplearning/tensorflow
- https://opensource.com/article/17/11/intro-tensorflow
- https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
- https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-keras
- https://www.tensorflow.org/api_docs/python/tf/keras

Appendix:

Code used for all 03 models

1. **ASL-Model01.ipynb**

```
#Importing tensorflow and keras libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

ASLModel01 = tf.keras.models.Sequential([
   tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(73, 73,3)),
   tf.keras.layers.MaxPooling2D(2, 2),
   tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2, 2),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(128, activation='relu'),
   tf.keras.layers.Dense(29, activation='softmax')
])
ASLModel01.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
['accuracy'])
ASLModel01.summary()

train_datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(rescale =
1./255,
                                      shear_range = 0.2,
                                      zoom_range = 0.2,
                                      horizontal_flip = True,
                                      validation_split=0.2)

training_dataset                                                              =
train_datagenerator.flow_from_directory('asl_alphabet_train/asl_alphabet_train',
                                target_size = (73, 73),  # the size of the images expected
in my cnn model
                                batch_size = 32,
                                subset = 'training',
                                class_mode = 'categorical')

validation_dataset                                                            =
train_datagenerator.flow_from_directory('asl_alphabet_train/asl_alphabet_train',
                                 target_size = (73, 73),
                                 batch_size = 32,
                                              subset = 'validation',
                                              class_mode = 'categorical')

ASLVersion1=ASLModel01.fit(training_dataset,
          epochs = 13,
          validation_data = validation_dataset)
ASLModel01.save_weights('ASLTest01.h5')

from matplotlib import pyplot as plt

plt.plot(ASLVersion1.history['accuracy'])
plt.plot(ASLVersion1.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(ASLVersion1.history['loss'])
plt.plot(ASLVersion1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Load weights of a pretrained model
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale =
1./255,shear_range = 0.2,
                                              zoom_range = 0.2,
                                              horizontal_flip = True)

test_set = test_datagen.flow_from_directory('asl_alphabet_test/asl_alphabet_test',
                              target_size = (73, 73),
                              batch_size = 32,
                              class_mode = 'categorical')


ASLModel01.load_weights("ASLTest01.h5")

test_loss, test_acc = ASLModel01.evaluate(test_set)
print(test_acc)
print(test_loss)
```

2. ASL-Model02.ipynb

```
#Importing tensorflow and keras libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

ASLModel02 = tf.keras.models.Sequential([
   tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(73, 73,3)),
   tf.keras.layers.MaxPooling2D(2, 2),
   tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2, 2),
   tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2, 2),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(128, activation='relu'),
   tf.keras.layers.Dense(29, activation='softmax')
])
ASLModel02.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
['accuracy'])
ASLModel02.summary()

train_datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(rescale =
1./255,
                                            shear_range = 0.2,
                                            zoom_range = 0.2,
                                            horizontal_flip = True,
                                            validation_split=0.2)

training_dataset                                                                  =
train_datagenerator.flow_from_directory('asl_alphabet_train/asl_alphabet_train',
                                     target_size = (73, 73),  # the size of the images expected
in my cnn model
                                     batch_size = 32,
                                     subset = 'training',
                                     class_mode = 'categorical')

validation_dataset                                                                =
train_datagenerator.flow_from_directory('asl_alphabet_train/asl_alphabet_train',
                                      target_size = (73, 73),
                                      batch_size = 32,
                                      subset = 'validation',
                                      class_mode = 'categorical')
```

```
ASLVersion2=ASLModel02.fit(training_dataset,
        epochs = 26,
        validation_data = validation_dataset)
ASLModel02.save_weights('ASLTest02.h5')

from matplotlib import pyplot as plt

plt.plot(ASLVersion2.history['accuracy'])
plt.plot(ASLVersion2.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(ASLVersion2.history['loss'])
plt.plot(ASLVersion2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Load weights of a pretrained model
test_datagen    =    tf.keras.preprocessing.image.ImageDataGenerator(rescale    =
1./255,shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_set = test_datagen.flow_from_directory('asl_alphabet_test/asl_alphabet_test',
                         target_size = (73, 73),
                         batch_size = 32,
                         class_mode = 'categorical')


ASLModel02.load_weights("ASLTest02.h5")

test_loss, test_acc = ASLModel02.evaluate(test_set)
print(test_acc)
print(test_loss)
```

3. ASL-Model03.ipynb

```
#Importing tensorflow and keras libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

ASLModel03 = tf.keras.models.Sequential([
   tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(73, 73,3)),
   tf.keras.layers.MaxPooling2D(2, 2),
   tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2, 2),
   tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2, 2),
   tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2, 2),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(128, activation='relu'),
   tf.keras.layers.Dense(29, activation='softmax')
])
ASLModel03.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics =
['accuracy'])
ASLModel03.summary()

train_datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(rescale =
1./255,
                                    shear_range = 0.2,
                                    zoom_range = 0.2,
                                    horizontal_flip = True,
                                    validation_split=0.2)

training_dataset                                                      =
train_datagenerator.flow_from_directory('asl_alphabet_train/asl_alphabet_train',
                              target_size = (73, 73),  # the size of the images expected
in my cnn model
                              batch_size = 32,
                              subset = 'training',
                              class_mode = 'categorical')

validation_dataset                                                    =
train_datagenerator.flow_from_directory('asl_alphabet_train/asl_alphabet_train',
                               target_size = (73, 73),
                               batch_size = 32,
                                              subset = 'validation',
                                              class_mode = 'categorical')
ASLVersion3=ASLModel03.fit(training_dataset,
        epochs = 39,
        validation_data = validation_dataset)
ASLModel03.save_weights('ASLTest03.h5')

from matplotlib import pyplot as plt

plt.plot(ASLVersion3.history['accuracy'])
plt.plot(ASLVersion3.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

plt.plot(ASLVersion3.history['loss'])
plt.plot(ASLVersion3.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Load weights of a pretrained model
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1./255,shear_range = 0.2,
                                              zoom_range = 0.2,
                                              horizontal_flip = True)

test_set = test_datagen.flow_from_directory('asl_alphabet_test/asl_alphabet_test',
                                 target_size = (73, 73),
                                 batch_size = 32,
                                 class_mode = 'categorical')


ASLModel03.load_weights("ASLTest03.h5")

test_loss, test_acc = ASLModel03.evaluate(test_set)
print(test_acc)
print(test_loss)
```