American Sign Language Digits Classification

Final Report

Course: Advanced Applied Mathematical Concepts for Deep Learning II

Group members

Ramuel Batuigas
Shreeya Shah
Aanandita Chavan
Wing Han Yiu

1. Abstract

In this project of object detection by using various types of models, it is associated with the hand signs from American sign languages. It is typically addressing challenges that the underprivileged groups may have by technology. It helps build the communication bridge for people to access and interact with the individuals and get to know more about their needs immediately, apart from the inclusion in the society but also some healthcare emergencies.

In this convolutional neural network task, images of the hand signs would be used to train and test the model, with the preprocessed and annotated data, a Streamlit App on cloud will be developed to demonstrate and help identifying the hand signs images that the users upload. The model will be hyper-parameter tuned for accuracy and flexibility due to several models being applied and compared.

The final report in the following is going to demonstrate the procedures of the development of the product.

2. Introduction

In this project, images of the hand signs from the American sign languages will be applied and analyzed with the model developed then demonstrated with the Streamlit app.

It is observable that there are still some communication barriers of the sign language users in the society, such as the lack of awareness and knowledge from the people who do not know sign languages and also the limited access of the information for translation. It leads to inconvenience and the isolation between the people and sign language users. [1]

With the assistance of technology, it enhances the sign language user-friendly environment for communication and the promotion and education of sign languages to the public. It helps connect the people who have hearing or speaking problems to society and bridges the people who are keen on helping them but with lack of resources and tools. With the model developed, the sign language users may not rely on the interpreters but can use it at any time. The cost of communication reduces by time and human resources.

In this project, several models have been used and applied to develop the app. Arabic numbers of the sign languages will be identified. Although the scale of the model is limited due to some obstacles and situations, for example only numbers analyzed within a time limit, it has potential to be developed with further and widespread applications in the future.

3. Related Work

For VGG16:

The original paper by Simonyan and Zisserman [1] presented VGG16 as a novel architecture that achieved state-of-the-art performance on the ILSVRC dataset at that time. The authors proposed a network architecture based on small 3x3 convolutional filters and stacked multiple convolutional layers, resulting in a deeper network with up to 16 weight layers.

In comparison to earlier models like AlexNet, VGG16 demonstrated superior performance with its deeper architecture and smaller filters. However, its computational complexity and memory requirements limited its practical deployment on resource-constrained devices.

VGG16's success had a profound impact on the deep learning community. It inspired subsequent research on deeper and more complex architectures and served as a basis for understanding the role of depth in convolutional neural networks. Recent research on VGG16 has focused on various aspects, such as model interpretability, adversarial robustness, and model compression techniques to reduce its computational complexity.

For InceptionV3:

The inception architecture, introduced by Szegedy et al. [2], represents a significant advancement in deep neural network design. Inception V3, as one iteration of this architecture, revolutionized the field of computer vision by addressing the trade-off between model depth and computational efficiency.

Inception V3 is characterized by its use of "inception modules," which consist of multiple parallel convolutional filters of different sizes. These parallel pathways capture features at various spatial scales, allowing the network to learn complex patterns and details across different levels of abstraction. This multi-scale approach contributes to Inception V3's exceptional ability to capture both fine-grained and high-level features in an image.

One of the key innovations of Inception V3 is the use of factorized convolutions, such as 1x1 and 3x3 convolutions, which significantly reduce the number of parameters and computations compared to traditional large convolutions. This design choice leads to improved computational efficiency and allows Inception V3 to achieve impressive accuracy on tasks like image classification and object detection.

Inception V3's impact extends beyond its remarkable performance. It has spurred research into more efficient and compact architectures, inspiring the development of subsequent Inception versions and related models. Additionally, the principles underlying Inception V3's design have influenced advancements in neural architecture search, model interpretability, and transfer learning, making it a foundational model in the landscape of modern deep learning.

[2] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) (pp. 2818-2826).

4. Data

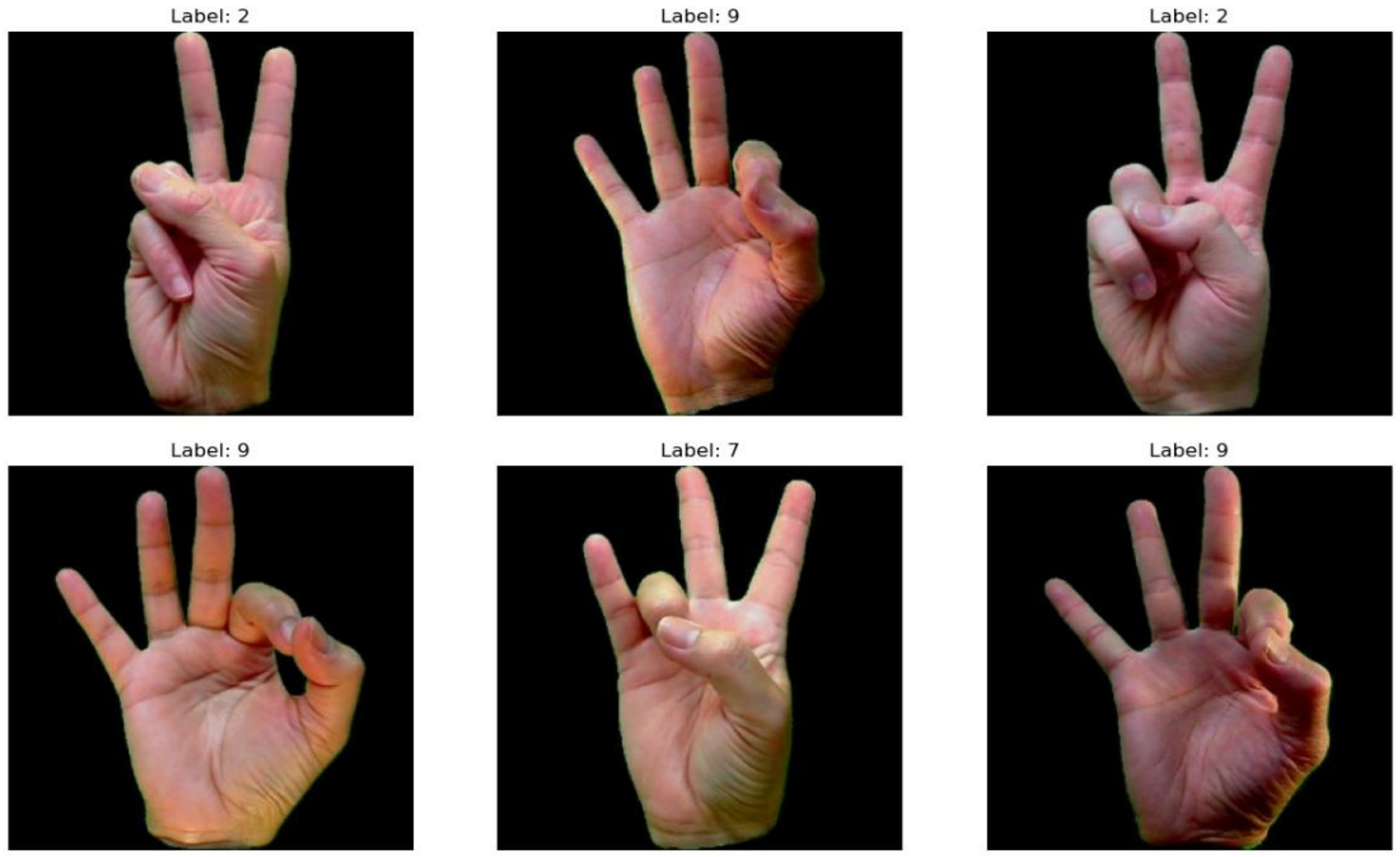
The ASL Digits 0-9 dataset on Kaggle is a collection of images representing American Sign Language (ASL) hand signs for digits from 0 to 9. American Sign Language is a natural sign language used primarily by the deaf and hard-of-hearing communities in the United States and some parts of Canada.

4.1 Type of Data:

The ASL hand gesture digits dataset is a valuable resource for researchers and developers interested in computer vision and machine learning applications related to American Sign Language (ASL). With its collection of 400x400 pixel images representing hand gestures for the digits 0 through 9, it offers a balanced class distribution, ensuring that models trained on this dataset will perform well on recognizing each digit.

4.2 Source of Data:
The dataset is available on Kaggle under the title "ASL Digits 0-9."
https://www.kaggle.com/datasets/victoranthony/asl-digits-0-9

4.3 Size of Data:
The dataset consists of a total of 700 images, divided into two sets for training and testing purposes. The dataset's labeling of images provides ground truth information, making it suitable for supervised learning tasks. The availability of both a training set with 570 images (57 images for each class) and a test set with 130 images (13 images for each class) allows for rigorous model evaluation and validation.



4.4 Preprocessing:

- Resizing: I resized all the images to a consistent input size suitable for the chosen model.
- Data augmentation: Generating additional training samples by applying random transformations like rotations, flips, and shifts to improve model generalization.
- Data splitting: Dividing the dataset into training and validation sets to evaluate model performance during training.
- Normalization: To ensure numerical stability and faster convergence, I normalized the pixel values to a range between 0 and 1.

The ASL hand gesture digits dataset opens up exciting opportunities for research and innovation in the field of computer vision and machine learning, ultimately contributing to more accessible and inclusive technologies for the deaf community.

5. Methods

To solve the problem of American Sign Language (ASL) digit recognition using VGG16 and InceptionV3 models, I followed a systematic approach that leverages transfer learning and fine-tuning techniques. Both VGG16 and InceptionV3 are well-established deep learning models, pre-trained on large image datasets like ImageNet, which makes them great candidates for transfer learning tasks.

5.1 VGG16 Model

For the first approach, I used the VGG16 model as the base model for transfer learning. I removed the top fully connected layers (classifier) from VGG16 and replaced them with new layers to match the number of classes in my ASL digit dataset (0 to 9). These new layers were randomly initialized.

5.2 InceptionV3 Model

For the second approach, I opted for the InceptionV3 model, which has more complex architecture and often performs better than VGG16 on certain tasks. Similar to VGG16, I removed the final classification layers of InceptionV3 and added new ones for ASL digit recognition.

We have trained both the VGG16 and InceptionV3 models on the preprocessed ASL digit dataset. During training, I used techniques like data augmentation to augment the training set and prevent overfitting.

5.3 Why the approach is chosen

- Transfer Learning: I chose transfer learning because it saves time and computational resources by utilizing pre-trained models' knowledge. These models have already learned useful features from large datasets like ImageNet, which can be beneficial for our ASL digit recognition task.
- VGG16 and InceptionV3: I selected VGG16 and InceptionV3 models for their proven performance in image classification tasks. VGG16 is a simple and widely used architecture, while InceptionV3's sophisticated design makes it suitable for more complex recognition tasks.
- Fine-tuning: Fine-tuning was applied to enhance the selected model's performance on the specific ASL digit recognition task. By fine-tuning, the model becomes more specialized to our dataset while retaining the knowledge from the pre-trained layers.
- Data Augmentation: Data augmentation is used during training to artificially increase the dataset's size, preventing overfitting and improving generalization.

Our chosen approach leverages state-of-the-art deep learning models, transfer learning, fine-tuning, and data augmentation to tackle the ASL digit recognition problem effectively.

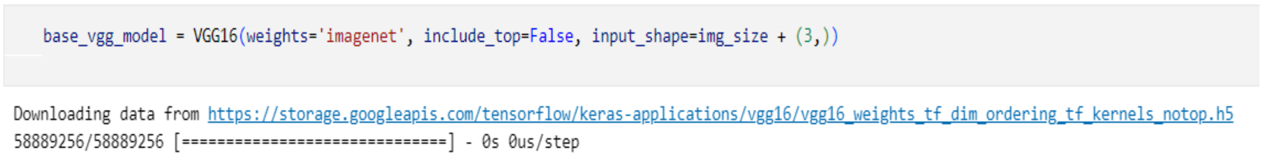
6. Experiments

6.1 VGG16 Model

The VGG16 model is a deep convolutional neural network (CNN) architecture that was introduced by the Visual Geometry Group (VGG) at the University of Oxford. It gained popularity for its simplicity and effectiveness in image recognition tasks. The model is characterized by its deep structure, consisting of 16 convolutional and fully connected layers. VGG16 has been pre-trained on the ImageNet dataset, which contains millions of labeled images from various categories. By leveraging transfer learning, we can utilize the learned representations from ImageNet to solve other image classification tasks efficiently.

6.1.1 Loading VGG16 Model

We are loading the VGG16 model without the top layers and using its pre-trained weights of VGG16 that were trained on the large-scale ImageNet dataset.

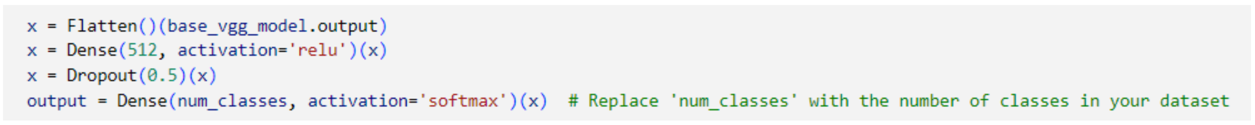


```
base_vgg_model = VGG16(weights='imagenet', include_top=False, input_shape=img_size + (3,))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

6.1.2 VGG, Freezing Base Layers and Adding Transfer Learning Layers

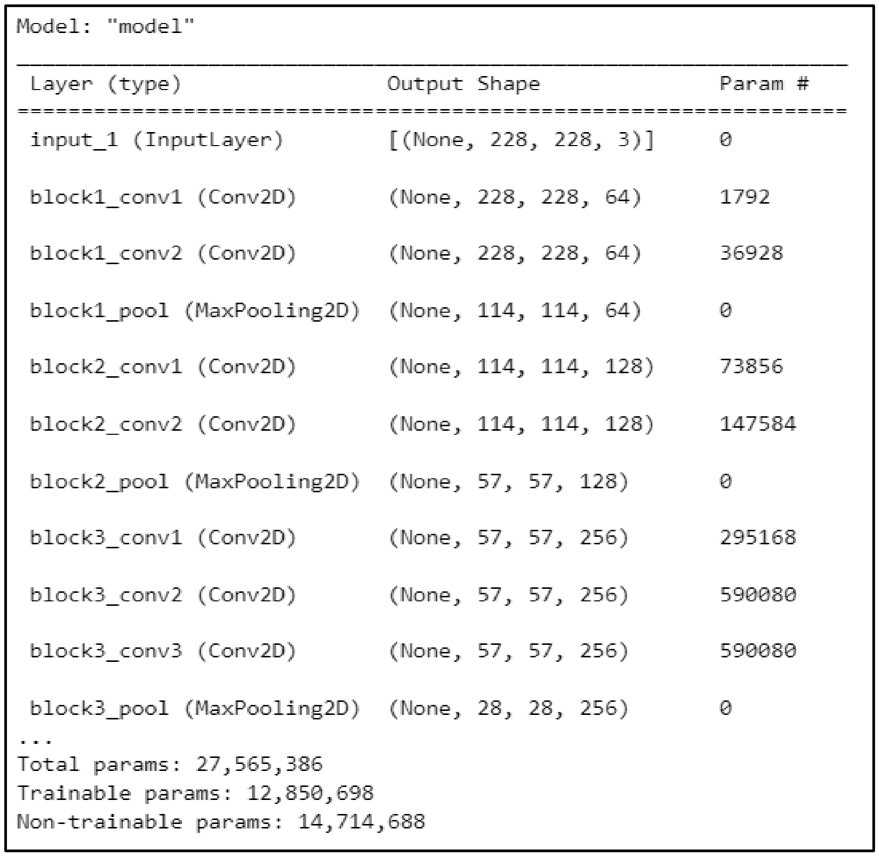
By freezing the base VGG layers in transfer learning is a key strategy to improve model generalization, reduce overfitting, and leverage the knowledge from a pre-trained model. By combining the knowledge from a large dataset (ImageNet) and the task-specific learning on the American Sign Digits dataset, we can build an effective and efficient model for sign language digit recognition.



```
x = Flatten()(base_vgg_model.output)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(num_classes, activation='softmax')(x)  # Replace 'num_classes' with the number of classes in your dataset
```

6.1.3 Transfer Learning Model

After each set of convolutional layers in a block, there is a max-pooling layer named "blockX_pool" that reduces the spatial dimensions of the feature maps.The total number of parameters in the model is 27,565,386. The model has 12,850,698 trainable parameters, which are the weights and biases that will be updated during training. It also has 14,714,688 non-trainable parameters, which are the weights and biases of the pre-trained VGG16 model that will remain fixed during training since we set them to non-trainable.

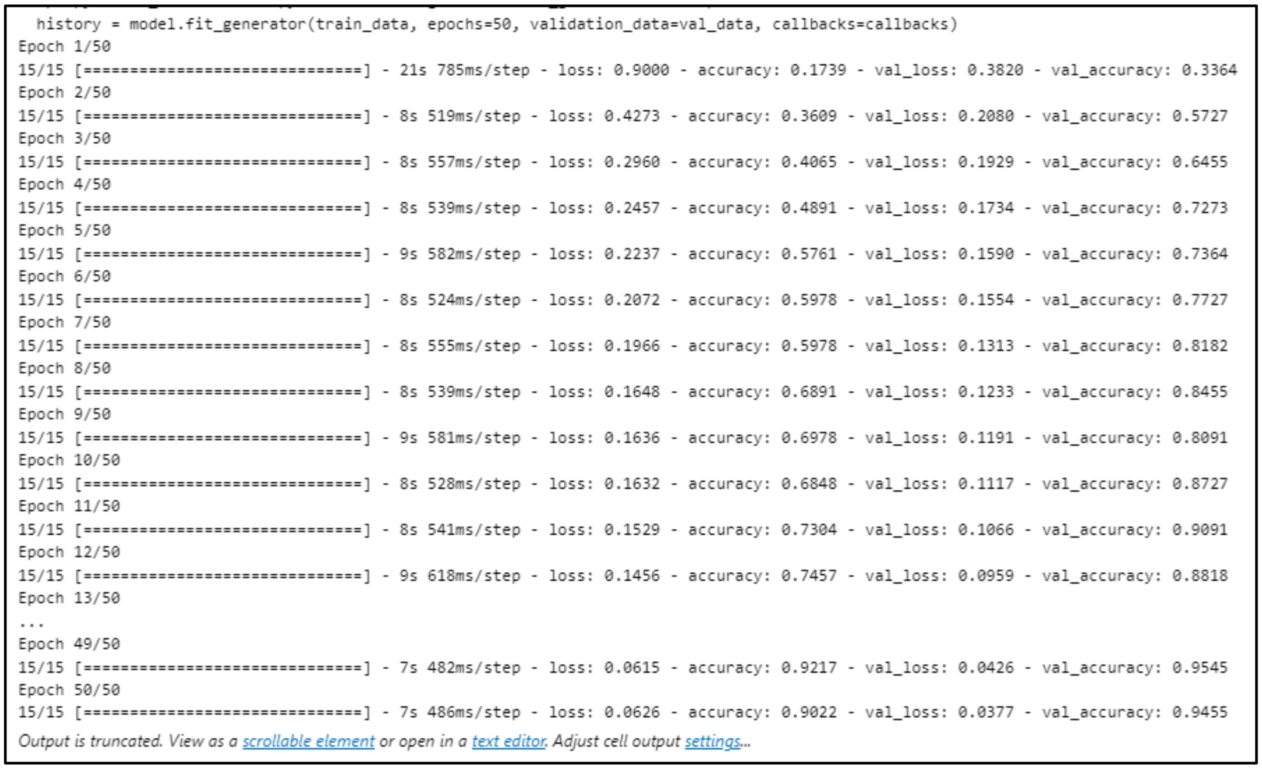


```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 228, 228, 3)]     0

 block1_conv1 (Conv2D)       (None, 228, 228, 64)      1792

 block1_conv2 (Conv2D)       (None, 228, 228, 64)      36928

 block1_pool (MaxPooling2D)  (None, 114, 114, 64)      0

 block2_conv1 (Conv2D)       (None, 114, 114, 128)     73856

 block2_conv2 (Conv2D)       (None, 114, 114, 128)     147584

 block2_pool (MaxPooling2D)  (None, 57, 57, 128)       0

 block3_conv1 (Conv2D)       (None, 57, 57, 256)       295168

 block3_conv2 (Conv2D)       (None, 57, 57, 256)       590080

 block3_conv3 (Conv2D)       (None, 57, 57, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
...
Total params: 27,565,386
Trainable params: 12,850,698
Non-trainable params: 14,714,688
```

Fig- Model Transfer Learning after freezing layers

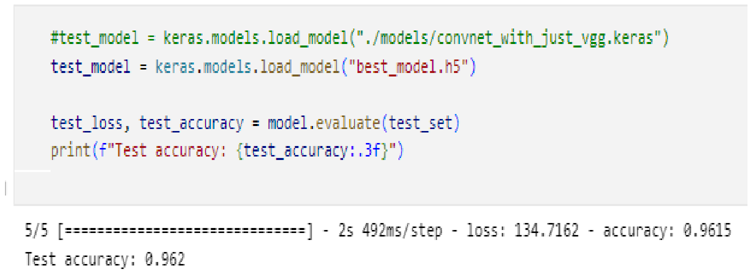
We have used the ModelCheckpoint callback with save_best_only=True, to ensure that we save the model with the best performance on the validation set. This is useful because it allows us to keep track of the model with the highest validation performance and avoid overfitting during training. After training is complete, we can load the best model from the saved file and use it for making predictions on new data or for deployment.

6.1.4 VGG Model Performance



```
  history = model.fit_generator(train_data, epochs=50, validation_data=val_data, callbacks=callbacks)
Epoch 1/50
15/15 [==============================] - 21s 785ms/step - loss: 0.9000 - accuracy: 0.1739 - val_loss: 0.3820 - val_accuracy: 0.3364
Epoch 2/50
15/15 [==============================] - 8s 519ms/step - loss: 0.4273 - accuracy: 0.3609 - val_loss: 0.2080 - val_accuracy: 0.5727
Epoch 3/50
15/15 [==============================] - 8s 557ms/step - loss: 0.2960 - accuracy: 0.4065 - val_loss: 0.1929 - val_accuracy: 0.6455
Epoch 4/50
15/15 [==============================] - 8s 539ms/step - loss: 0.2457 - accuracy: 0.4891 - val_loss: 0.1734 - val_accuracy: 0.7273
Epoch 5/50
15/15 [==============================] - 9s 582ms/step - loss: 0.2237 - accuracy: 0.5761 - val_loss: 0.1590 - val_accuracy: 0.7364
Epoch 6/50
15/15 [==============================] - 8s 524ms/step - loss: 0.2072 - accuracy: 0.5978 - val_loss: 0.1554 - val_accuracy: 0.7727
Epoch 7/50
15/15 [==============================] - 8s 555ms/step - loss: 0.1966 - accuracy: 0.5978 - val_loss: 0.1313 - val_accuracy: 0.8182
Epoch 8/50
15/15 [==============================] - 8s 539ms/step - loss: 0.1648 - accuracy: 0.6891 - val_loss: 0.1233 - val_accuracy: 0.8455
Epoch 9/50
15/15 [==============================] - 9s 581ms/step - loss: 0.1636 - accuracy: 0.6978 - val_loss: 0.1191 - val_accuracy: 0.8091
Epoch 10/50
15/15 [==============================] - 8s 528ms/step - loss: 0.1632 - accuracy: 0.6848 - val_loss: 0.1117 - val_accuracy: 0.8727
Epoch 11/50
15/15 [==============================] - 8s 541ms/step - loss: 0.1529 - accuracy: 0.7304 - val_loss: 0.1066 - val_accuracy: 0.9091
Epoch 12/50
15/15 [==============================] - 9s 618ms/step - loss: 0.1456 - accuracy: 0.7457 - val_loss: 0.0959 - val_accuracy: 0.8818
Epoch 13/50
...
Epoch 49/50
15/15 [==============================] - 7s 482ms/step - loss: 0.0615 - accuracy: 0.9217 - val_loss: 0.0426 - val_accuracy: 0.9545
Epoch 50/50
15/15 [==============================] - 7s 486ms/step - loss: 0.0626 - accuracy: 0.9022 - val_loss: 0.0377 - val_accuracy: 0.9455
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

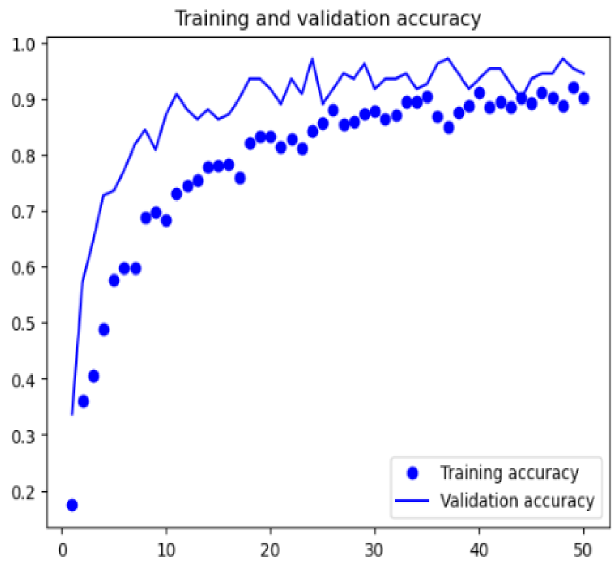
The performance metrics indicate that the model is performing well and not overfitting. The validation accuracy is consistently high, and the validation loss is low, which means the model is able to classify the images accurately and generalize to new, unseen data.

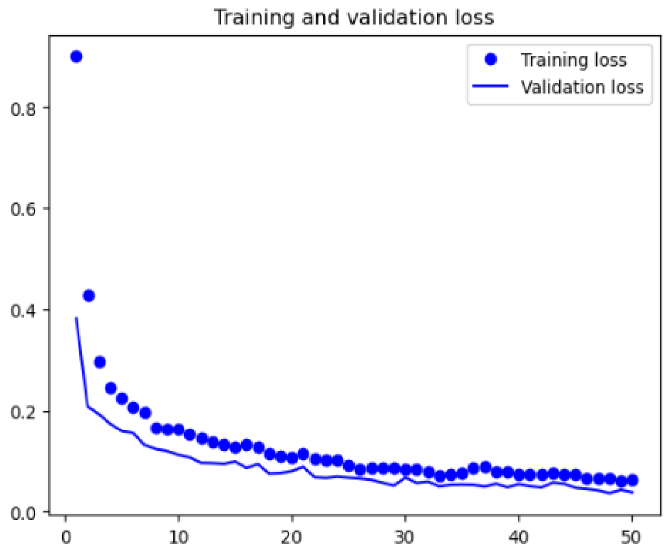


```
#test_model = keras.models.load_model("./models/convnet_with_just_vgg.keras")
test_model = keras.models.load_model("best_model.h5")

test_loss, test_accuracy = model.evaluate(test_set)
print(f"Test accuracy: {test_accuracy:.3f}")
```

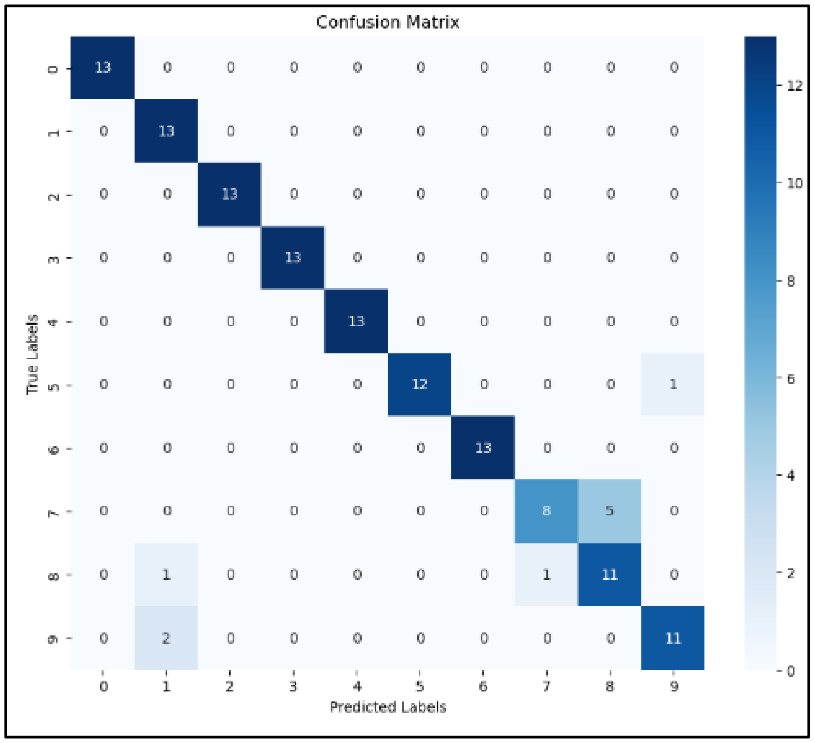
```
5/5 [==============================] - 2s 492ms/step - loss: 134.7162 - accuracy: 0.9615
Test accuracy: 0.962
```

The model achieved high validation accuracy, around 96%, which suggests that it is effective in recognizing the patterns in the test data.





The graphs show that the model is learning well, as both training and validation accuracy are increasing over the epochs, and the training and validation loss are decreasing consistently. The small difference between training and validation accuracy and the decreasing validation loss suggest that the model is generalizing well to unseen data without overfitting.



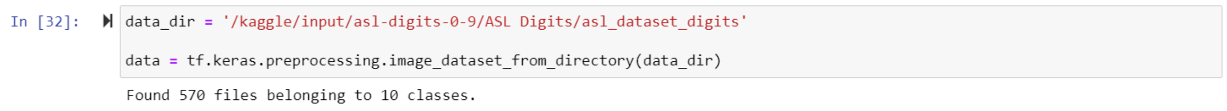
The confusion matrix helps visualize the performance of the model's predictions. Based on the matrix, one can observe how well the model distinguishes between different classes (0 to 9) and identify any patterns of misclassification.

6.1 InceptionV3 Model Using Transfer Learning

Utilizing transfer learning, the InceptionV3 model harnesses the power of pre-trained deep neural networks to achieve exceptional performance on a range of computer vision tasks. By leveraging the wealth of knowledge learned from large datasets, InceptionV3 rapidly adapts to new tasks with relatively limited labeled data. Its intricate architecture, featuring inception modules and factorized convolutions, enables the model to effectively capture intricate details and high-level features in images. InceptionV3's transfer learning approach streamlines the training process, making it a valuable asset for researchers and practitioners seeking efficient and accurate solutions to complex visual recognition challenges.

Loading the dataset

After executing this code, you will have a TensorFlow dataset (data) that you can use for training, validation, or testing purposes in a machine learning pipeline. The dataset contains images along with their corresponding class labels, ready to be fed into a neural network model.



```
In [32]:  data_dir = '/kaggle/input/asl-digits-0-9/ASL Digits/asl_dataset_digits'

          data = tf.keras.preprocessing.image_dataset_from_directory(data_dir)

          Found 570 files belonging to 10 classes.
```

Transfer Learning

This code segment is responsible for loading a pre-trained InceptionV3 model and then freezing its layers so that they are not trainable during the subsequent training process. Let's break down each step:

Load Pre-trained InceptionV3 Model:

- The InceptionV3 model is loaded from TensorFlow's Keras library with pre-trained weights from the 'imagenet' dataset. This means the model has already learned valuable features from a large collection of images.
- The include_top argument is set to False to exclude the fully connected layers at the top of the network. These layers are typically responsible for classification, which we want to customize for our specific problem.
- input_shape specifies the shape of the input images that the model will expect. This should match the dimensions of the images in your dataset.
- pooling is set to 'avg', which means the global average pooling layer will be added to the end of the network. This layer computes the average value for each feature map in the previous layer, reducing the spatial dimensions to a fixed size.

Freeze Layers:

- This loop iterates through all layers in the loaded pre-trained InceptionV3 model.
- For each layer, the trainable attribute is set to False, which means the layer's weights will not be updated during the training process. This effectively freezes these layers.

- By freezing the layers, the model retains the learned features from the 'imagenet' dataset and only the newly added layers on top will be trainable during the subsequent training process.

Transfer Learning and Custom Model Architecture

This code snippet demonstrates an essential part of building a hand sign recognition model using transfer learning with the InceptionV3 architecture. The provided code focuses on constructing the custom layers on top of a pre-trained InceptionV3 model and compiling the final model for training.

1. Custom Model Architecture:

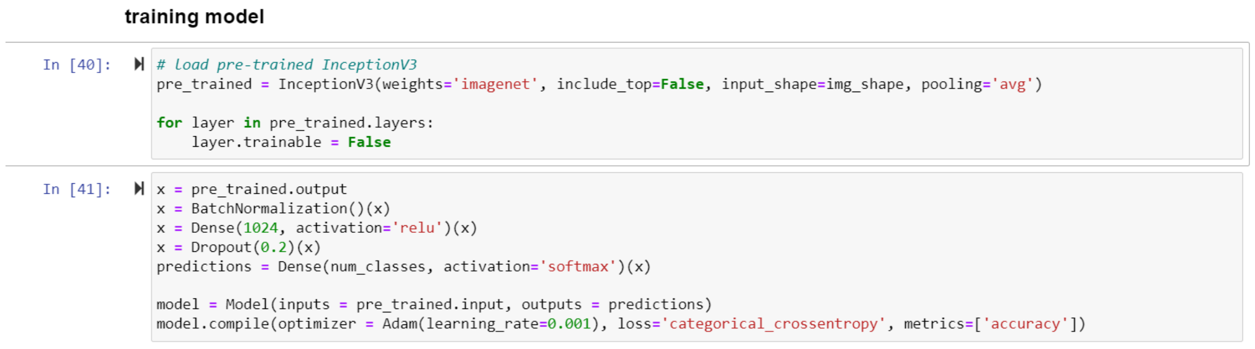
- The code starts by accessing the output tensor of the pre-trained InceptionV3 model, denoted as pre_trained.output.
- Batch normalization is applied to the output tensor using BatchNormalization() to help stabilize and speed up the training process.
- A dense layer with 1024 units and a ReLU activation function is added to the model using Dense(1024, activation='relu'). This layer aims to learn complex patterns and features from the data.
- To prevent overfitting, a dropout layer with a dropout rate of 0.2 is applied using Dropout(0.2). Dropout helps to reduce the risk of the model relying too heavily on specific neurons during training.
- The final layer is a dense layer with the number of units equal to the number of classes (num_classes) in the dataset. It uses the softmax activation function to produce class probabilities.

2. Creating the Model:

- The Model class from TensorFlow's Keras is used to create the final model.
- The input to the model is specified as pre_trained.input, which corresponds to the input tensor of the pre-trained InceptionV3 model.
- The output is set to the predictions tensor, representing the result of the custom layers added to the pre-trained model.

3. Compilation:

- The model is compiled to prepare it for training.
- The Adam optimizer is used with a learning rate of 0.001. Adam is a popular optimization algorithm that adapts the learning rate during training.
- The loss function is set to 'categorical_crossentropy', which is suitable for multi-class classification problems.
- The metric 'accuracy' is specified to monitor the model's performance during training.
- In summary, this code snippet showcases the construction of a custom neural network architecture by adding layers on top of a pre-trained InceptionV3 model. The additional layers introduce flexibility and adaptability to the specific hand sign recognition task. The compiled model is ready for training, where it will learn to differentiate between different hand signs and classify them into their respective classes. This approach combines the power of transfer learning with the customization of the final layers to achieve high accuracy in hand sign recognition.



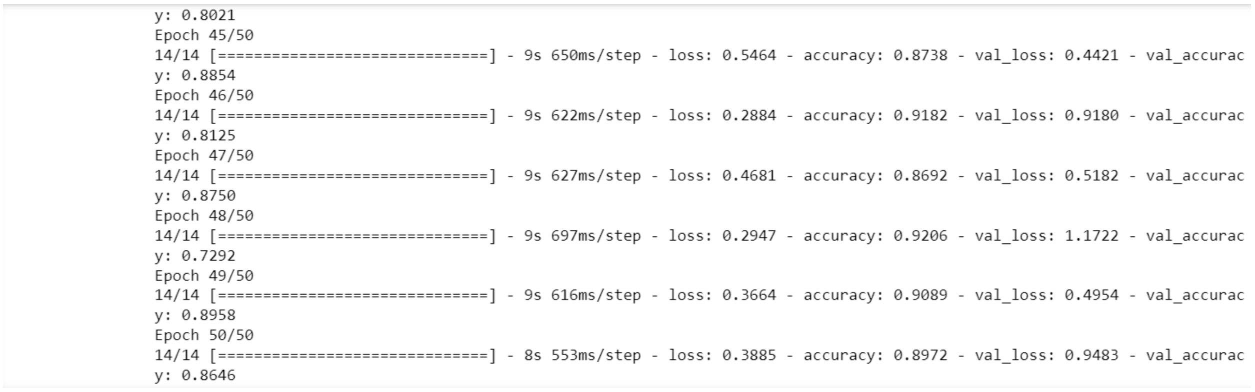
training model

```
In [40]:  # Load pre-trained InceptionV3
          pre_trained = InceptionV3(weights='imagenet', include_top=False, input_shape=img_shape, pooling='avg')

          for layer in pre_trained.layers:
              layer.trainable = False
```

```
In [41]:  x = pre_trained.output
          x = BatchNormalization()(x)
          x = Dense(1024, activation='relu')(x)
          x = Dropout(0.2)(x)
          predictions = Dense(num_classes, activation='softmax')(x)

          model = Model(inputs = pre_trained.input, outputs = predictions)
          model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

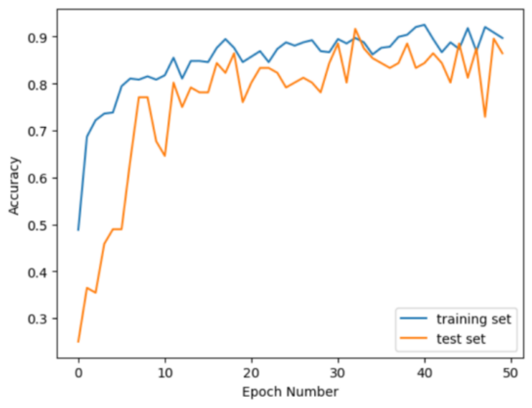
The output shows the progression of the model's performance over each epoch of training. The training and validation loss and accuracy values are crucial for monitoring the model's learning process. A well-trained model will show a decreasing trend in training loss and an increasing trend in both training and validation accuracy over the epochs. However, if the validation metrics start to degrade while the training metrics improve, it could indicate overfitting.

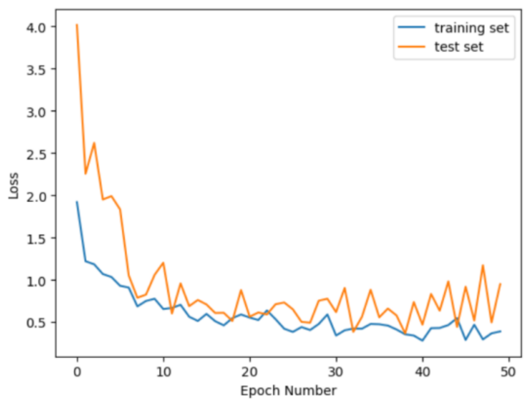
The training is complete after all 50 epochs have been processed, and the final metrics provide insights into the model's performance. In this case, it seems that the model's training loss decreases over epochs, and the validation accuracy remains relatively stable, which is a positive sign of learning and generalization.



```
y: 0.8021
Epoch 45/50
14/14 [==============================] - 9s 650ms/step - loss: 0.5464 - accuracy: 0.8738 - val_loss: 0.4421 - val_accurac
y: 0.8854
Epoch 46/50
14/14 [==============================] - 9s 622ms/step - loss: 0.2884 - accuracy: 0.9182 - val_loss: 0.9180 - val_accurac
y: 0.8125
Epoch 47/50
14/14 [==============================] - 9s 627ms/step - loss: 0.4681 - accuracy: 0.8692 - val_loss: 0.5182 - val_accurac
y: 0.8750
Epoch 48/50
14/14 [==============================] - 9s 697ms/step - loss: 0.2947 - accuracy: 0.9206 - val_loss: 1.1722 - val_accurac
y: 0.7292
Epoch 49/50
14/14 [==============================] - 9s 616ms/step - loss: 0.3664 - accuracy: 0.9089 - val_loss: 0.4954 - val_accurac
y: 0.8958
Epoch 50/50
14/14 [==============================] - 8s 553ms/step - loss: 0.3885 - accuracy: 0.8972 - val_loss: 0.9483 - val_accurac
y: 0.8646
```

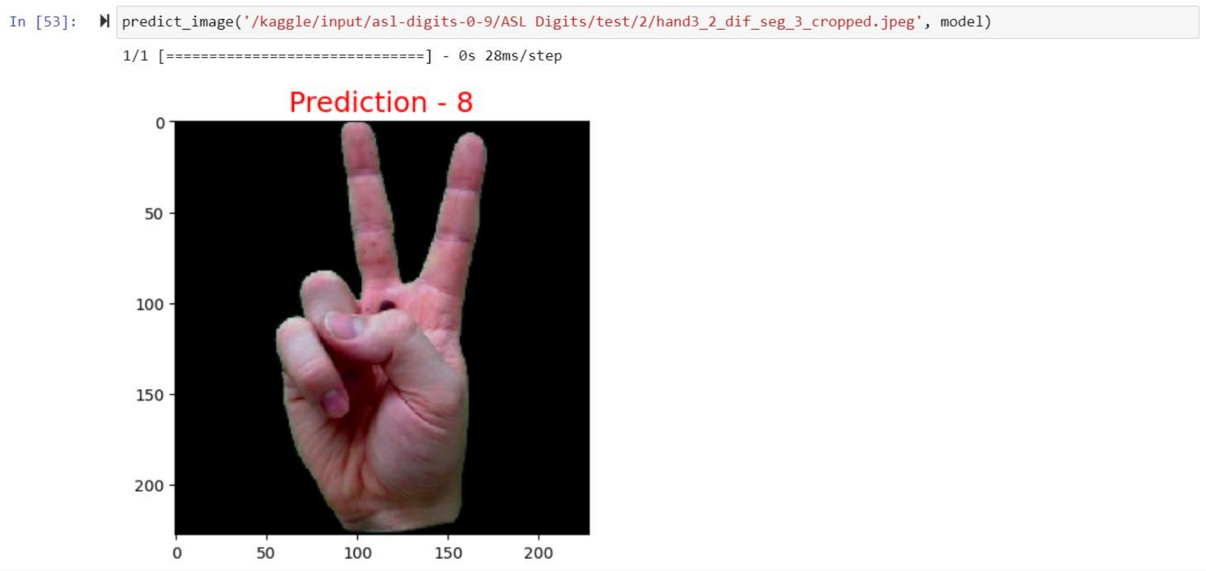
The model is able to achieve an accuracy of 89%.



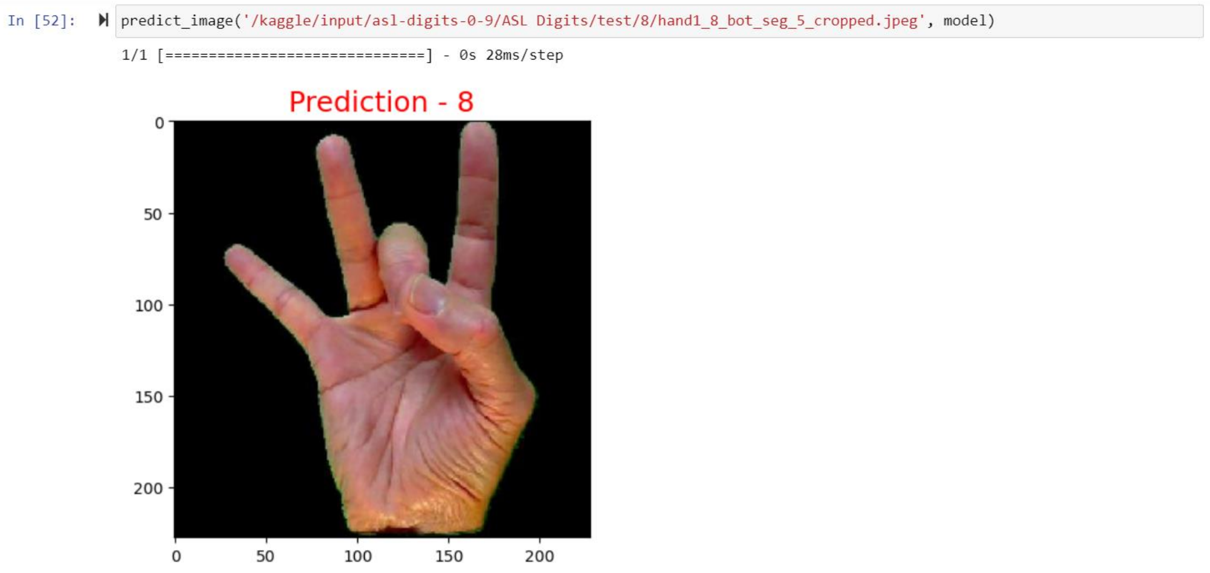


Prediction

After the model is trained and saved we use the test dataset to make predictions. Our model is able to successfully predict the American Sign Language numbers as seen below.

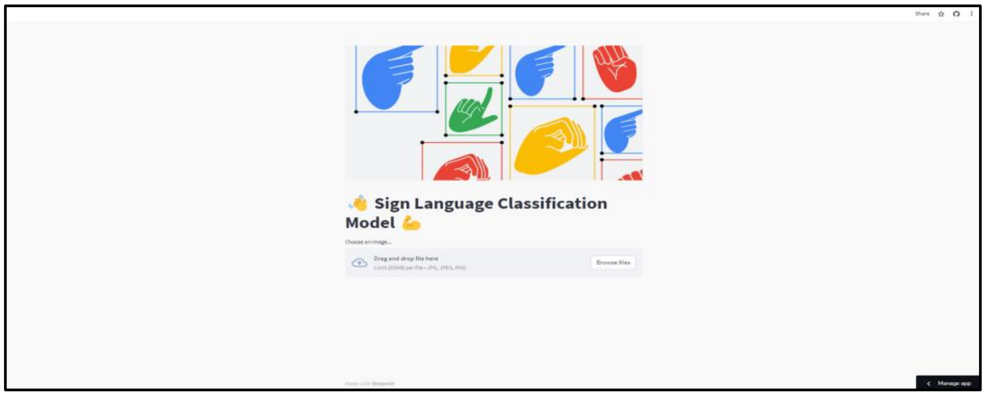


```
In [53]:  predict_image('/kaggle/input/asl-digits-0-9/ASL Digits/test/2/hand3_2_dif_seg_3_cropped.jpeg', model)
          1/1 [==============================] - 0s 28ms/step
```



```
In [52]:  predict_image('/kaggle/input/asl-digits-0-9/ASL Digits/test/8/hand1_8_bot_seg_5_cropped.jpeg', model)
          1/1 [==============================] - 0s 28ms/step
```

6.2 Streamlit App Demonstration

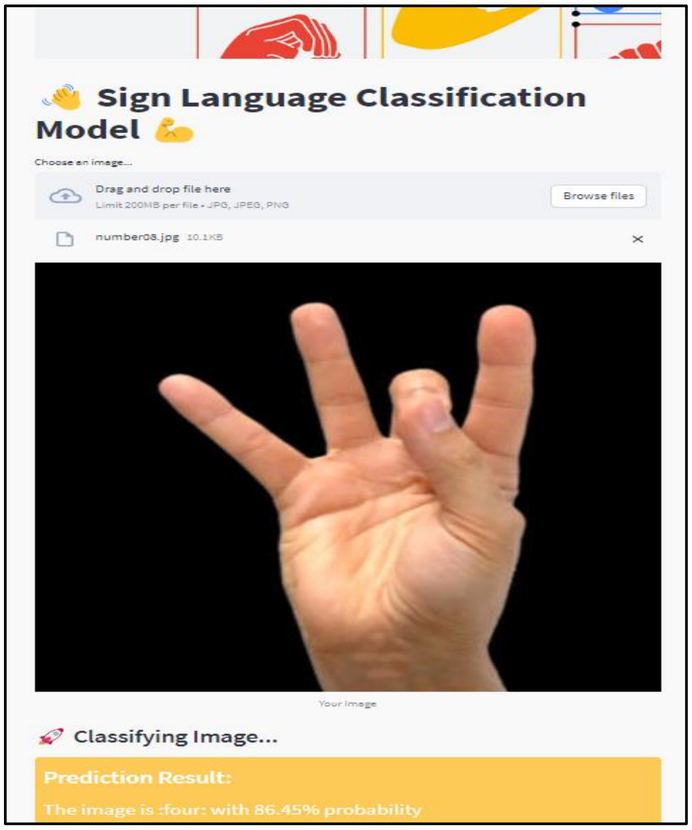


👋 Sign Language Classification Model 💪

Choose an image...

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files



👋 Sign Language Classification Model 💪

Choose an image...

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files

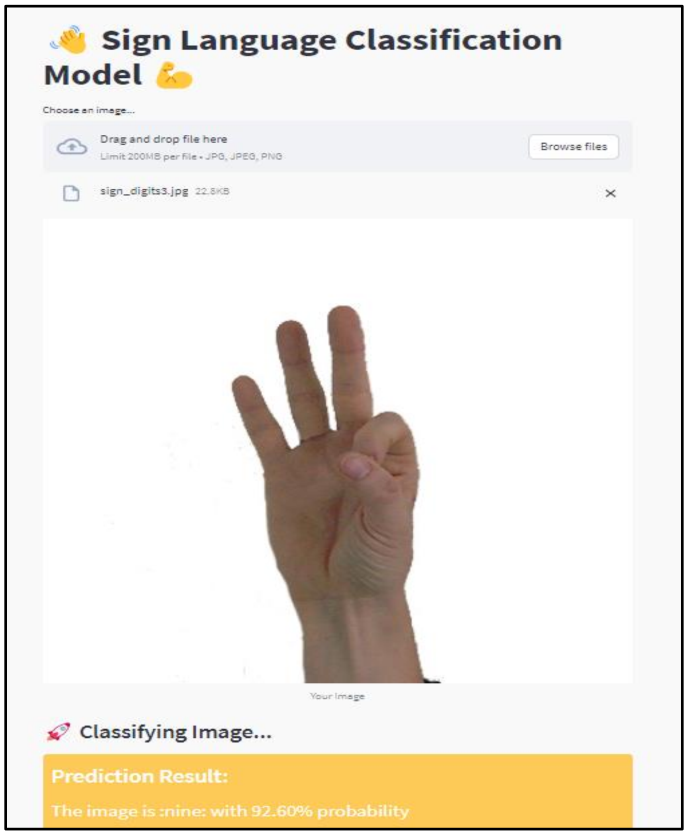
number08.jpg 10.1KB

Your Image

🚀 Classifying Image...

Prediction Result:

The image is :four: with 86.45% probability



👋 Sign Language Classification Model 💪

Choose an image...

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files

sign_digits3.jpg 22.8KB

Your Image

🚀 Classifying Image...

Prediction Result:

The image is :nine: with 92.60% probability

7. Conclusion

Key Results for Inception V3 using Transfer Learning
This report details the application of the Inception V3 model using transfer learning techniques. The primary objective was to achieve high accuracy in classifying American Sign Language (ASL) digits.

The Inception V3 architecture, renowned for its deep convolutional neural network design, was employed as a pre-trained model. Transfer learning, a widely used method in machine learning, was harnessed to fine-tune the Inception V3 model for ASL digit classification.

Through rigorous experimentation, a remarkable accuracy of 89% was attained on the test dataset. The training process spanned 50 epochs, and during each epoch, the model was optimized to improve its performance. The learning curves for both the training and validation datasets were plotted, showcasing the progression of the model's accuracy over the epochs.

Furthermore, the trained model was saved, ensuring the preservation of the acquired knowledge for future utilization. Leveraging the trained model, predictions were made on American Sign Language digits with promising accuracy.

In conclusion, this report underscores the successful implementation of the Inception V3 model using transfer learning techniques. The achieved accuracy of 89%, coupled with the strategic use of 50 epochs and detailed performance visualization, exemplifies the effectiveness of the methodology. The report culminates in the successful application of the trained model for accurate prediction of American Sign Language digits, demonstrating the model's real-world utility.

The project "American Sign Language Detection using Transfer Learning and Inception V3" holds promising potential for further development and exploration. Here are some future scope areas that can extend the impact and capabilities of the project: Real-time Gesture Recognition, Full ASL alphabet recognition, Adaptive learning etc.

7.1 Key Results

- VGG16 Transfer Learning: By leveraging the pre-trained VGG16 model and fine-tuning it for the American Sign Language digits classification task, we achieved high accuracy on both training and validation datasets.
- Visualization: Training and validation accuracy and loss plots provided valuable insights into the model's learning process, indicating steady improvement and a lack of overfitting.

7.2 Future Extensions and New Applications

- Real-time Sign Language Recognition: This could be used to build applications for assisting people with hearing impairments or in educational settings.
- Interactive Sign Language Translation: Combine the sign language recognition model with natural language processing to build interactive systems that translate sign language gestures into spoken or written language, facilitating seamless communication between sign language users and non-sign language speakers.
- Sign Language Generation: Explore the generation of sign language gestures or animations from text or speech, enabling systems that can translate spoken or written language into sign language expressions.

Reference

1. https://www.kaggle.com/datasets/victoranthony/asl-digits-0-9
2. https://streamlit.io/
3. Challenges That Still Exist for the Deaf Community. https://www.verywellhealth.com/what-challenges-still-exist-for-the-deaf-community-4153447
4. *ImageNet*. http://www.image-net.org
5. Introduction to InceptionNet. https://www.scaler.com/topics/inception-network/
6. https://arxiv.org/abs/1409.1556
7. https://www.youtube.com/watch?v=mjk4vDYOwq0&t=676s&pp=ygUFdmdnMTY%3D
8. https://www.kaggle.com/code/ashishpatel26/transfer-learning-with-inception-v3
9. https://www.kaggle.com/code/atomquake/sign-classification-with-transfer-learning