Report

**HAND GESTURE RECOGNITION**

**ABI PRIYANKA PL**

**MSc Data Science**

**Table of Contents:**

[**ABSTRACT**: 1](#_Toc133745799)

[**INTRODUCTION:** 1](#_Toc133745800)

[**LITERATURE REVIEW:** 2](#_Toc133745801)

[**PROCEDURE:** 3](#_Toc133745802)

[**OBSERVATION:** 8](#_Toc133745803)

[**CONCLUSION:** 9](#_Toc133745804)

[**FUTURE SCOPE:** 10](#_Toc133745805)

[**REFERENCES:** 11](#_Toc133745806)

Data

augmentation like re-scaling, zooming, shearing, rotation, width

and height shifting was applied to the dataset. The model was

trained on 8000 images and tested on 1600 images which were

divided into 10 classes.

# 

# **ABSTRACT**:

Static hand gesture recognition is the process of identifying and interpreting the meaning of a gesture made with the hand that remains still. It is an important application of computer vision, which is a field of study focused on enabling machines to interpret and understand visual data from the world around them. Convolutional neural networks (CNNs) have emerged as a powerful tool for solving a wide range of computer vision tasks, including image classification, object detection, and segmentation. In this report, we will explore the implementation of static hand gesture recognition using PyTorch and CNNs, which have shown excellent performance on image recognition tasks. Data augmentation like re-scaling, zooming, cropping was applied to the dataset. The model was trained on 600 images and tested on 300 images which were divided into 6 classes. The model achieved a training accuracy of 96.71% and a testing accuracy of 97.07%.

# **INTRODUCTION:**

Gesture is the term used to describe the motion of the body used to communicate with other agents. For a gesture to be successful, the sender and the recipient need to have the same kind of information. There are two types of gestures: Static and Dynamic. A static gesture tends to stay largely identical over time, whereas a dynamic gesture intended to change over time. The recognition of static gestures is particularly highlighted in this project. Automated hand gesture detection has potential uses in a variety of fields, including design, robotics, virtual reality, and most significantly, sign language.

Making a computer capable of comprehending hand gestures is the main challenge. Different hand gestures have different hand shapes and finger orientations. Therefore, one of the aspects of hand gestures that needs to be dealt with is non-linearity. The images' metadata and content data can be used to accomplish this. Images of hand movements can be recognised using their meta information. The method combines both tasks of feature extraction and classification. The features of a picture must be retrieved before any gesture can be recognised.

Convolutional Neural Networks (CNNs) are a type of deep learning neural network that can extract useful features from input data, such as images, on the fly. This is achieved through the use of convolutional layers, which apply a series of filters to the input data to extract relevant features. In addition to convolutional layers, CNNs also typically include fully connected layers, which can be used for classification. One of the key benefits of using a CNN is that it can combine the feature extraction and classification steps, reducing both memory requirements and computational complexity while improving performance. Furthermore, CNNs are particularly well-suited for understanding the complex and non-linear relationships that exist between different features in an image.

In the context of gesture recognition using PyTorch, a CNN-based approach is ideal for accurately identifying and classifying different types of gestures based on their visual features. By using a CNN to extract these features and classify the input data, we can achieve high accuracy and robustness in our gesture recognition system.

In this project, a neural network is trained to classify hand gestures that correspond to music player commands such as "next", "pause", "play", "previous", "volume\_down", and "volume\_up".

# **LITERATURE REVIEW:**

* "Real-time hand gesture recognition using a depth camera and a convolutional neural network" by Wang et al. (2018) - In this work, the researchers used a depth camera to capture hand gestures and used a convolutional neural network (CNN) to classify the gestures in real-time. They achieved an accuracy of 94.4% on a dataset of 7 hand gestures.
* "Hand gesture recognition using depth and skeleton information with convolutional neural networks" by Choi et al. (2018) - In this work, the researchers used both depth and skeleton information to recognize hand gestures. They used a CNN with 2D and 3D convolutional layers to classify the gestures. They achieved an accuracy of 95.7% on a dataset of 12 hand gestures.
* "A vision-based approach for hand gesture recognition using deep learning" by Kumar et al. (2017) - In this work, the researchers used a dataset of 25 hand gestures and used a CNN with 5 convolutional layers and 2 fully connected layers to classify the gestures. They achieved an accuracy of 95.6%.
* "Dynamic hand gesture recognition using a depth camera and a convolutional neural network" by Kim et al. (2017) - In this work, the researchers used a depth camera to capture dynamic hand gestures and used a CNN with 2D and 3D convolutional layers to classify the gestures. They achieved an accuracy of 97.9% on a dataset of 11 dynamic hand gestures.
* "A comprehensive study of hand gesture recognition techniques and databases" by Rautaray et al. (2015) - In this paper, the authors reviewed various techniques for hand gesture recognition, including traditional methods such as template matching and model-based approaches, as well as more recent deep learning-based approaches such as CNNs. They also reviewed various datasets for hand gesture recognition.

# **PROCEDURE:**

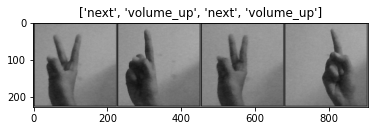
**Static Hand Gesture Recognition using PyTorch:**

The static hand gesture recognition model can be built using a convolutional neural network (CNN) architecture. The input to the model is an image of the hand gesture, and the output is a prediction of the corresponding gesture class. The following steps are involved in building the model:

1. Data Collection: The first step is to collect a dataset of hand gesture images. The dataset includes images of six hand gestures with different hand shapes and lighting conditions separated into train and valid with 100 images per gesture for train and 50 images per gesture for valid totalling 600 images under train and 300 under valid.
2. Data transformation: Data transformation is a critical step in deep learning. It involves applying different image processing operations on the dataset to augment it and improve model accuracy. In this segment, we apply image resizing, grayscale conversion, and normalization transformations to the dataset.

When an image is grayscale, it is changed from a full-colour to a grayscale version in which each pixel is represented by a single value between 0 and 255. Because it simplifies the image data and lowers the dimensionality and complexity of the image, gray scaling is a frequently used pre-processing step in computer vision tasks. This simplification minimises the amount of processing needed to process the image and makes it simpler for the model to identify patterns and features. The edges and contours of the hand, which are crucial details for identifying hand motions, can be highlighted by gray scaling in the context of hand gesture recognition to help remove the colour information from the image.

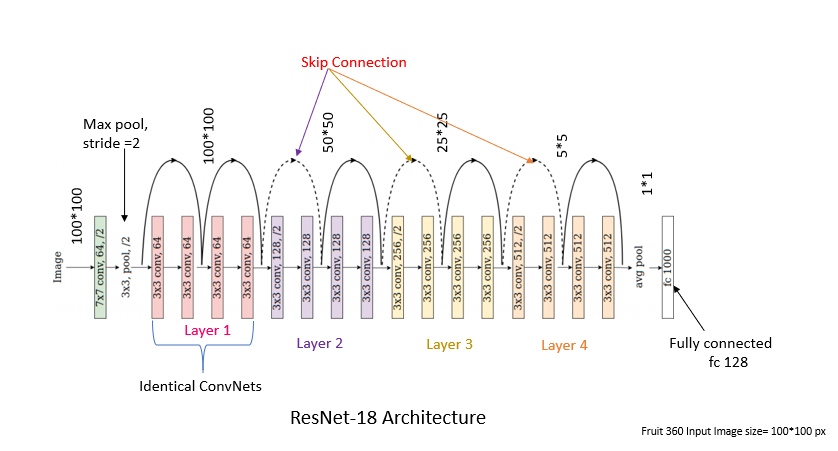
Normalizing the images can also help to improve the accuracy of the model by reducing the effect of illumination variations, which can affect the intensity of the pixels in the image. By scaling the pixel values, the model becomes less sensitive to changes in illumination, which helps to improve its robustness and generalization ability.



1. Model Architecture: The model architecture can be a CNN that consists of convolutional layers, pooling layers, and fully connected layers. The convolutional layers extract features from the input image, and the fully connected layers classify the extracted features into different gesture classes. The number of convolutional and fully connected layers and the number of neurons in each layer can be tuned based on the dataset size and complexity.

ResNet-18 is a convolutional neural network architecture that was introduced by Microsoft researchers in 2015. It is a deep neural network that consists of 18 layers, including convolutional layers, pooling layers, fully connected layers, and a softmax layer for classification. The architecture uses residual connections to help solve the problem of vanishing gradients in deep neural networks, which can make it difficult for the network to learn from very deep layers.

In the above code, the ResNet-18 architecture is used as a pre-trained model that is fine-tuned on the hand gesture recognition task. The pre-trained ResNet-18 model is loaded using the torchvision library, and the last fully connected layer is replaced with a new layer that outputs the number of classes in the hand gesture recognition task. The new layer is trained using the hand gesture dataset to learn the specific features of the hand gesture images. The pre-trained weights of the ResNet-18 model are used as a starting point for training the new layer, which can help to speed up the training process and improve the performance of the model.



**CNN CONFIGURATION:**

**TRANSFER LEARNING:**

The reuse of a pre-trained model on a new problem is known as transfer learning in machine learning. A machine uses the knowledge learned from a prior assignment to increase prediction about a new task in transfer learning. There are a number of popular pre-trained machine learning models available.

In this project, **ResNet18** is used as the pre-trained convolutional neural network (CNN) architecture to extract features from the input images. The ResNet-18 architecture is used as a pre-trained model that is fine-tuned on the hand gesture recognition task. The pre-trained ResNet-18 model is loaded using the torchvision library, and the last fully connected layer is replaced with a new layer that outputs the number of classes in the hand gesture recognition task. The new layer is trained using the hand gesture dataset to learn the specific features of the hand gesture images. The pre-trained weights of the ResNet-18 model are used as a starting point for training the new layer, which can help to speed up the training process and improve the performance of the model.

1. Training: The next step is to train the model using the training set. The model is trained using backpropagation and stochastic gradient descent (SGD) optimization. The hyperparameters such as learning rate, batch size, and number of epochs can be tuned to achieve better performance.

The SGD optimizer is used to update the weights of the ResNet18 model during training. It iteratively performs the following steps for each mini-batch of training examples:

* + Compute the gradients of the cost function with respect to the model parameters.
  + Update the model parameters by subtracting a scaled version of the gradients. The scaling factor is called the learning rate and determines the step size taken in the parameter space.
  + Repeat the process for the next mini-batch until all training examples have been processed.

**Fine-tuning** is a common technique used in deep learning, where a pre-trained model is used as a starting point, and the weights of one or more layers are updated by training the model on a new dataset or task. Here, the ResNet18 model is fine-tuned by updating the weights of the last fully connected layer and the layer before it. The rest of the layers are frozen, meaning their weights are not updated during training. This approach allows the model to adapt to the specific task of hand gesture recognition, while still leveraging the pre-trained weights from the ImageNet dataset.

The optimizer has a learning rate (lr) of 0.001, which determines the step size at each iteration of the optimization algorithm. The momentum (momentum) parameter is set to 0.9, which helps the optimizer to continue moving in the same direction if the gradients are pointing in the same direction. This can help to accelerate convergence and improve the optimization results.

The next layer is first fully connected layer and ReLU is used as the activation function. The layer is followed by a dropout layer which excludes 50% of the neurons to prevent overfitting.

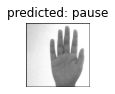
Dropout is a technique used to prevent overfitting in neural networks. It randomly drops out (i.e., sets to zero) a fraction of the output values of the previous layer during training. This forces the network to learn redundant representations of the data, making it more robust and less likely to overfit. The p parameter in the nn.Dropout(p) constructor specifies the probability that an element is set to zero, and its default value is 0.5. This means that during training, each neuron in the previous layer of the dropout layer will be dropped out with a probability of 0.5. Therefore, roughly half of the neurons will be dropped out during training.

The Cross Entropy Loss function is used as the loss criterion to optimize the model. The Cross Entropy Loss function combines both the LogSoftmax and NLLLoss (Negative Log Likelihood Loss) functions in a single class.

After the model makes a prediction for each input, the Cross Entropy Loss function takes the predicted probability distribution and the true label for each input and computes the negative log likelihood loss. This loss is then used to update the model parameters during training via backpropagation.

In summary, the Cross Entropy Loss function is used to measure the difference between the predicted probability distribution and the true label, and this difference is minimized during training to improve the accuracy of the model.

1. Validation: After training, the model is validated using the validation set. The validation set is used to tune the hyperparameters and prevent overfitting.

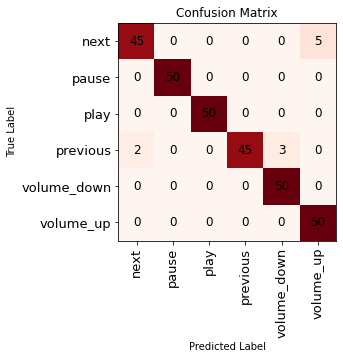
1. Testing: Finally, the model is tested. Here, a saved PyTorch model is loaded where it sets webcam, captures frames from the webcam, applies an image transformation to each frame, and passes the transformed image through the loaded model to make a prediction about a gesture being performed. The predicted label and probability are then displayed on the captured frame using OpenCV's cv2.putText function. The loop continues until the user presses 'q' to exit, and then the webcam is released and all windows are closed.

# **OBSERVATION:**

The model is set to run for 4 epochs and the following is observed for each epoch:

|  |
| --- |
| The model runs for four iterations with the learning rate of 0.001 and with the momentum 0.9.  The model uses SGD optimizer and the following results are observed. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Epoch | Training loss | Training accuracy  (in %) | Validation  loss | Validation  Accuracy  (in %) |
| 1 | **1.0161** | **65.57** | **0.2730** | **97.07** |
| 2 | **0.3000** | **93.08** | **0.3547** | **87.62** |
| 3 | **0.1871** | **95.06** | **0.1222** | **94.79** |
| 4 | **0.1157** | **96.71** | **0.1041** | **97.07** |



* This confusion matrix is showing the results of the classification model with 6 classes: 'next', 'pause', 'play', 'previous', 'volume\_down', and 'volume\_up'.
* The rows of the matrix represent the true classes, while the columns represent the predicted classes.
* Looking at the matrix, we can see that the model correctly predicted 45 instances of the 'next' class and 45 instances of the ‘previous’ class.
* 50 instances of each of the other classes ('pause', 'play', 'volume\_down' and 'volume\_up') are correctly predicted.
* However, there were 5 instances of the 'next' class that were misclassified as the 'volume\_up' class, resulting in 5 false negatives for the 'next' class.
* And there were 5 instances of the ‘previous’ class that were misclassified as ‘volume\_down’ and ‘next’ classes, resulting in 5 false negatives.

# **CONCLUSION:**

In conclusion, hand gesture recognition for music players is a potential technology that can improve user satisfaction and make music control simpler and more practical. Users can control music playing, volume, and other features with easy hand gestures thanks to computer vision algorithms that recognise and interpret hand movements. Numerous settings, including home entertainment, music festivals, and public spaces, could use this technology. For even more immersive and interactive experiences, it can be combined with other technologies like virtual reality and smart homes. However, there are still some problems that need to be solved, including enhancing the precision and speed of gesture recognition algorithms, lowering latency, and making sure that different platforms and devices are compatible. Despite these difficulties, it is likely that hand gesture recognition for music players will advance and get better in the years to come given the rising interest in this technology and the constant creation of new methods and tools.

# **FUTURE SCOPE:**

There are several ways to further develop a hand gesture recognition project and make it more scalable and useful for applications. Here are some potential avenues for improvement:

* Increase the dataset size: Collecting more hand gesture images with a wide range of backgrounds, lighting conditions, and skin tones can help to improve the model's accuracy and robustness.
* Augment the dataset: Data augmentation techniques such as rotating, flipping, and shifting images can help to create more diverse training data and improve the model's ability to generalize to new images.
* Use transfer learning: Instead of training a model from scratch, use pre-trained models such as VGG, ResNet, or MobileNet as a starting point and fine-tune them for the hand gesture recognition task.
* Implement real-time detection: To make the model useful in real-world applications, implement real-time detection of hand gestures from live video streams. This can be done by optimizing the model's architecture and using efficient inference techniques.
* Improve the user interface: Develop a user-friendly interface that allows users to easily interact with the application and perform hand gestures. This can involve using a camera to capture hand gestures and displaying the recognized gesture on the screen.
* Deploy the model on mobile devices: To make the application more accessible and portable, deploy the model on mobile devices such as smartphones and tablets. This can be done by converting the model to a mobile-friendly format such as TensorFlow Lite or Core ML.
* Expand the gesture recognition capabilities: Consider adding more hand gestures to the model's repertoire to make it more versatile and useful for different applications. This can involve collecting more data for new gestures or leveraging transfer learning to recognize new gestures based on existing ones.

# **REFERENCES:**

1) https://pytorch.org/tutorials/beginner/transfer\_learning\_tutorial.html

2) https://pytorch.org/tutorials/intermediate/quantized\_transfer\_learning\_tutorial.html

3) <https://pytorch.org/tutorials/beginner/nn_tutorial.html>

4) "Real-Time Hand Gesture Recognition System for Human-Computer Interaction" by S. Hong and C. Lee in IEEE Transactions on Consumer Electronics, 2016.

5) "Hand Gesture Recognition for Human-Computer Interaction: A Review" by A. R. M. Yusoff et al. in IEEE Access, 2020.