Enhanced sEMG Signals Process with Two-stream CNN on Gesture Classification

A Thesis Submitted
In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

in

SIGNAL PROCESSING AND DIGITAL DESIGN

by

Sudhir Kumar
(Roll No. 2K22/SPD/09)

Under the Supervision of
Dr. Rajesh Birok
Delhi Technological University

Department of Electronics and Communication Engineering

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-110042. India

May, 2024

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

CANDIDATE'S DECLARATION

I Sudhir Kumar, Roll No – 2K22/SPD/09 student of M.Tech (Signal Processing and Digital Design) hereby certify that the work which is being presented in the thesis entitled **" Enhanced sEMG Signals Process with Two-stream CNN on Gesture Classification"** in partial fulfillment of the requirements for the award of the Degree of Master of Technology, submitted in the Department of Electronics and Communication Engineering, Delhi Technological University is an authentic record of my work carried out during the period from 2022 to 2024 under the supervision of. Dr.Rajesh Birok.

The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institute.

Candidate's Signature

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

CERTIFICATE BY THE SUPERVISOR

I certified that SUDHIR KUMAR (2K22/SPD/09) has carried out their search work presented in this thesis **"Enhanced sEMG Signals Process with Two-stream CNN on Gesture Classification"** entitled for the award of Master of Technology from the Department of Electronics and Communication Engineering, Delhi Technological University, Delhi, under my supervision. The thesis embodies results of original work, and studies are carried out by the student himself and the contents of the thesis do not form the basis for the award of any other degree to the candidate or anybody else from this or any other University/Institution.

Dr. Rajesh Birok

Associate Professor

Place:
Date:

Department of Electronics & Communication Engineering
Delhi Technological University

ABSTRACT

sEMG signals show huge potential in implementing control systems of mechatronics devices, and small muscle movements can generate enough sEMG signals to achieve desired control operations. Traditional methods on the sEMG signals process do not robustly decipher important information to distinguish subtle differences in gesture classification. This paper applied a novel deep learning method, a two-stream CNN architecture called Mario CNN, to process NinaPro DB1 data on gesture classification and achieved higher accuracy than a single stream CNN.

ACKNOWLEDGEMENT

I would like to express my deep sense of gratitude to my highly respected and esteemed guide Dr. Rajesh Birok for suggesting the topic of my Major project and for giving me complete freedom and flexibility to work on this topic. He has been very encouraging and motivating and the intensity of encouragement has always increased with time. Without her constant support and guidance, I would not have been able to complete this work. We extend my sincere thanks to all my friends who have patiently helped me directly or indirectly in accomplishing this work successfully.

Sudhir Kumar

(2K22/SPD/09)

M.Tech (Signal Processing & Digital Design)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CNN:	Convolutional Neural Network.
sEMG:	Surface Electromyographic.
DNN:	Deep Neural Network.
RNN:	Recurrent Neural Network
TD:	Time Domain
FD:	Frequency Domain
SGD:	Stochastic Gradient Descent
SVM:	Support Vector Machine
LSTM:	Long Short-Term Memory

CHAPTER 1

INTRODUCTION

One way to detect the electrical activity produced by muscle impulses is with surface electromyography (sEMG).Figure 1 shows. The amplitude patch electrodes can only detect a potential difference of around ±5 mV from the skin, even though the generated potential difference typically falls within the range of -90 mV to 30 mV. sEMG has several applications in the execution of mechatronics device control systems, and particularly the execution of fine motor control to accomplish the required control activities using sufficient signal generation. The research indicates that sEMG is frequently combined with interference waves of both high and very low frequencies (near DC), The effective sEMG signal frequency range is 10-500 Hz[1]. Hence, the signal picked up from High-pass filtering and other signal conditioning procedures are necessary for the patch electrode,(the process of straightening), low-pass filtering, and high-pass amplification. Although surface electromyography (sEMG) signals exhibit significant potential, their applications are now in a relatively nascent stage of development. The present challenges encompass the presence of monotonous signal sources, the existence of equivocal classification, and the interference caused by noise in non-ideal settings. The accuracy of measurements can be significantly influenced by several factors, such as electrode changes, limb posture, and muscle exhaustion It is important to mention that in this study, the existence of inconsistent labels can generate similar signals, however with different intensities. It is necessary to recognize that traditional machine learning classification approaches may not efficiently extract essential information. classification, and sleep stage classification. This paper studied the neural networks from related papers and applied a two-stream CNN to process sEMG signals on gesture recognition.

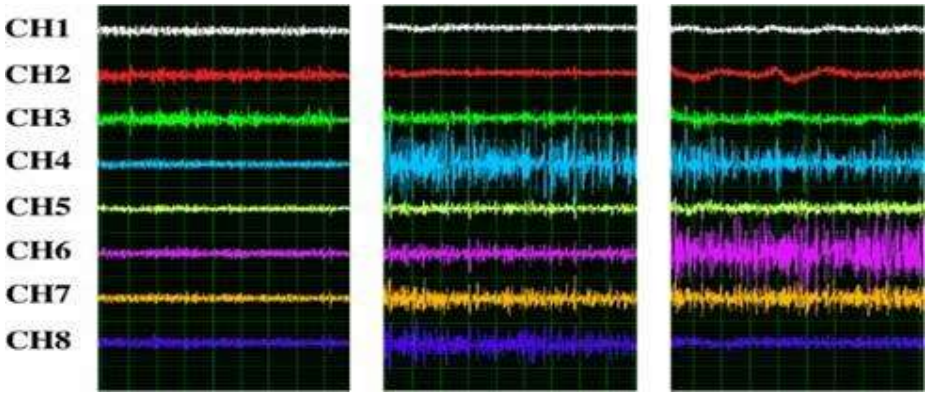


Figure 1.1: sEMG signals example for three gestures (one for each column)[2]

While sEMG signals show huge potential, the applications are relatively immature. Current problems include monotonous signal sources, ambiguous classification, and noise under non-ideal conditions. The measurements can be easily affected by many factors like electrode shifts, limb position, muscle fatigue. Meanwhile, inconsistent labels may have similar signals but with different amplitudes, and traditional machine learning classification methods do not robustly decipher important information. Therefore, researchers and scientists in recent years (especially since 2014) began to apply deep learning methods to process sEMG signals on applications, including hand gesture recognition, speech and emotion 1 classification, and sleep stage classification. This paper studied the networks from related papers and applied a novel "Mario Brothers" neural network (Mario CNN) to process sEMG signals on gesture recognition

This paper studied the networks from related papers and applied a novel "Mario Brothers" neural network (Mario CNN) to process sEMG signals on gesture recognition. The name is inspired by Mario and Luigi, who run towards the same goal in a parallel manner, as shown in Figure 2.

Figure 1.2: Mario and Ligi run in a parallel manner

CHAPTER 2

LITERATURE SURVEY

Five main deep learning architectures are commonly used in the identification of surface electromyography (sEMG) signals. The designs encompass deep feedforward neural networks (DNN),recurrent neural networks (RNN), convolutional neural networks (CNN), autoencoders (AEs), and deep belief networks (DBN) [3]. Here is a concise overview of the papers we have read that are pertinent to the current issue. Using Convolutional Neural Networks for Deep Learning in the Analysis of Electromyography Data: A Useful Tool for Classifying Movement in Prosthetic Hand Applications The user did not provide any text for rewriting.The NinaPro dataset was analysed using a modified convolutional neural network (CNN)[4] that was based on the LeNet architecture proposed by LeCun et al. in 1995. The experimental findings exhibited a precision of 66.59±6.40 percent on dataset 1.The input data was obtained using time intervals of 150 milliseconds. The architectural designThe design was split into four blocks, and each block had a different part: the traditional convolutional layer, the rectified linear unit [5](a non-linear activation function), the average pooling layer, and the softmax loss. The model is trained using stochastic methods. Gradient Descent (SGD) and other strategies, such as modifying the learning rate, utilising normalization, and incorporating data augmentation, are utilised to improve the accuracy[6]. The results suggest that using convolutional neural networks leads to higher classification accuracy compared to standard classification methods.The study named ”A Deep Neural Network Approach for Classifying 41 Hand and Wrist Movements Using Surface Electromyogram” aims to categorize different hand and wrist movements by utilizing a surface electromyogram [7]. This paper implemented DNN on NinaPro-DB5/DB7 and achieved accuracy of 93.87±1.49 %. The stride between each window was set to be smaller than the window size so as to gain better performance. The noise threshold T was set to 0.01V for DB5. The architecture of Deep Neural Network Classifier was 3 hidden layers followed by ReLU and then softmax, which included Adam optimizer, with a learning rate of 0.005 and decay of 0.00001. The authors adopted batch normalization and 20 % dropout and used a trick called Evaluation metrics in the end. Due to half rest classes in the datasets, the accuracy needed to 2 be balanced. For multi-class classification, taking the average of recall values can be generalized as the macro recall.

The current investigation utilized a parallel, multiple-scale convolutional neural network (CNN) on the NinaProDB2 dataset, leading to enhanced accuracy in comparison to a solitary CNN. We took surface electromyography (SEMG)[8] data from 12 electrodes and turned it into sEMG images that are 12x200 pixels, with a sampling rate of 2000 Hz and a time interval of 100 ms. The study discovered a specific set of properties that accurately define surface electromyography (sEMG) signals, thus minimizing the complexity of the input for the classifier. The Convolutional Neural Network (CNN)[9] is characterised by its unique architecture, which consists of two separate blocks. Block 1 employed a setup comprising five convolution layers and two maximum pooling layers. On the other hand, Block 2 showed

differences in the first three convolutional layers, where a bigger filter kernel size was used. Pooling layers were not incorporated into Block 2. The two blocks displayed parallel behaviour and were determined to be mutually independent during the feature extraction procedure. The classifier receives the merged outputs from the two blocks.The results suggest that using a larger kernel filter can result in a slight enhancement in classification accuracy

"A multi-stream convolutional neural network for sEMG-based gesture recognition in muscle-computer interface" [10] This paper constructed a multi-stream convolutional neural network to train the NinaPro dataset and achieved an accuracy of 85.0%. The authors used the divide-and-conquer classification approach on feature-space, which considered sEMG signals in each channel independently and trained them using the same CNN architecture. In the divide stage: the convolutional neural network was constructed by two convolutional layers (3*3 kernel size) and two locally-connected layers (64 non-overlapping 2D filters) along with batch normalization, Relu, and dropout. The network was trained with SGD and tested with cross-validation. In the conquer stage: all feature maps learned from each CNN were fused into a unified feature map and sent into fusion networks. Relu followed by batch normalization were applied after each fully connected layer. The results show that considering sEMG signals recorded by each channel independently is better than considering sEMG signals recorded by all channels as a whole.

CHAPTER 3

METHODS AND TECHNIQUES

3.1 Convolutional Neural Network (CNN)

[11]The models described in the aforementioned studies are applied to distinct datasets, each with its own experimental objectives and methodologies. Therefore, it is not advisable to directly compare their performances. Therefore, we first created a simple 5-layer CNN largely based on the ideas and architecture described in the published research by Atzori, M et al as a reference. The hyperparameters and layers of the code provided by malele4th on GitHub[12] were adjusted, resulting in the architecture seen in Figure 2. The architectural design has a total of seven blocks, with blocks 1 to 5 specifically designed as convolutional layers that use a non-linear activation function and average pooling. Convolutional layers employ a fixed number of filters to extract distinctive characteristics from the initial pictures, thereby generating feature maps. Pooling layers enhance ”local translational invariance” by aggregating the characteristics extracted from regions inside the feature map. In the sixth and seventh blocks of the model architecture, there are two fully connected layers incorporated, accompanied by the inclusion of Batch Normalisation and Dropout techniques. Batch Normalisation is a technique that enhances the stability of the learning process by standardising the distribution of input data, hence mitigating the issue of overfitting. The act of dropout in machine learning models serves to mitigate overfitting and enhance the generalisation capabilities of the model. The softmax classifier ultimately employs the cross-entropy loss function to compute the probability utilised for classification

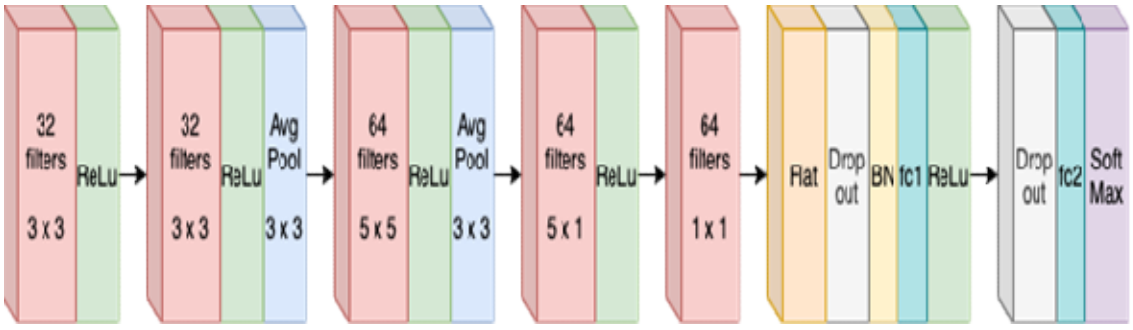


Figure 3.1: 5-layer CNN Architecture partially based on the tuning and layers from Atzori, M. et al

Based on the idea presented in the paper [13] , as shown in Figure 3, we decided to experiment with architectures with more streams, called multi-stream Mario CNN, and investigate how the number of streams affects the performances. To make the dimensions work, the hyperparameters for the pooling layers are slightly tuned while keeping the other layers the same as they are in the 2-stream CNN architecture.

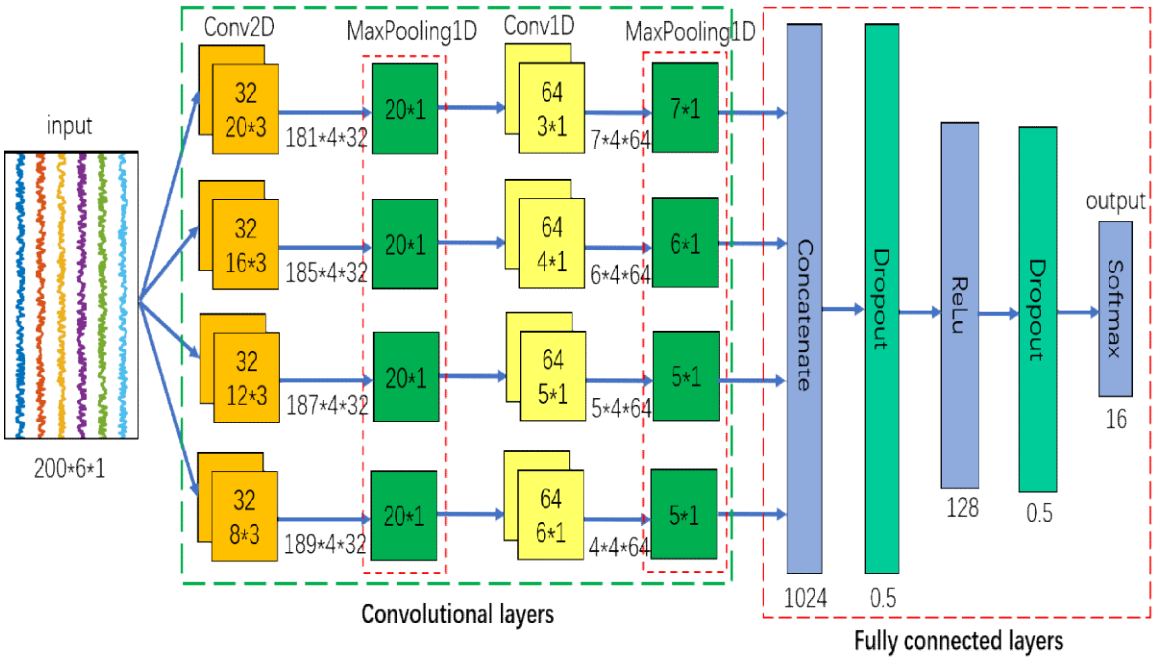


Figure 3.2: Multi-stream Convolutional Neural Network (CNN)

3.1.1.Advantage of CNN

1. Automatic Feature Extraction:- CNNs can automatically learn and extract relevant features from raw sEMG signals, eliminating the need for extensive manual feature engineering. This is particularly useful for complex and high-dimensional data like sEMG signals.
2. Spatial Hierarchies:- CNNs are adept at capturing spatial hierarchies in the data through their convolutional layers. For sEMG signals, this means they can effectively identify and leverage local dependencies and patterns in the muscle activation signals.
3. Robustness to Noise: The architecture of CNNs, with pooling layers and convolutional filters, provides a degree of robustness to noise and variability in sEMG signals. This is crucial given the potential for noise from electrode placement and muscle fatigue in sEMG recordings.
4. Scalability and Adaptability:- CNNs are scalable and can be adapted to various signal lengths and dimensionalities. They can also be fine-tuned to specific tasks such as gesture recognition or muscle activation pattern classification, making them versatile for different sEMG signal analysis applications.

5. End-to-End Learning:- CNNs support end-to-end learning, where the model can be trained directly on raw or minimally preprocessed sEMG signals. This simplifies the pipeline and potentially improves performance by leveraging raw signal information.

3.1.2.Disadvantage of CNN

1. High Computational Cost:-CNNs are computationally intensive, requiring significant processing power and memory, especially for large datasets like Ninapro DB1. Training deep networks can be time-consuming and resource-demanding, often necessitating high-end GPUs.
2. Data Preprocessing Requirements:-sEMG signals require extensive preprocessing, such as noise reduction and signal normalization, to ensure that the CNN can learn meaningful features. Poor preprocessing can lead to suboptimal performance.
   3.Sensitivity to Hyperparameters:-The performance of CNNs is highly sensitive to the choice of hyperparameters (e.g., learning rate, number of layers, filter sizes). Finding the optimal hyperparameters typically requires extensive experimentation and fine-tuning.
3. Overfitting Risk:-Due to the complexity and high capacity of CNNs, there is a significant risk of overfitting, especially with limited data or when the training set is not sufficiently representative of the test conditions. This makes the model less generalizable to new, unseen data.
4. Interpretability:-CNNs operate as "black boxes," making it difficult to interpret the learned features and understand the decision-making process. This lack of transparency can be problematic in biomedical applications where understanding the basis of a decision is crucial.
5. Feature Engineering Dependence:-While CNNs can automatically extract features from raw data, the quality of these features highly depends on the architecture design. Inadequate architecture might fail to capture important signal characteristics, leading to poor performance.
6. Dependency on Large Training Data:-CNNs generally perform better with large amounts of labeled data. For the Ninapro DB1 dataset, the size and quality of the labeled data directly influence the CNN's performance. Insufficient data can lead to poor model training and low accuracy.
7. Challenge with Temporal Dependencies:-sEMG signals have temporal dependencies that can be challenging for standard CNNs to capture effectively. Although there are techniques to address this, such as using temporal convolution layers or combining CNNs with recurrent layers (RNNs, LSTMs), they add complexity to the model.

8. Domain Expertise Requirement:-Designing an effective CNN model for sEMG analysis requires domain expertise in both deep learning and biomedical signal processing. Lack of expertise in either domain can lead to suboptimal model performance.Signal Variability:-sEMG signals can vary greatly between subjects and even within the same subject over time due to factors like electrode placement and muscle fatigue. This variability makes it challenging to develop a robust CNN model that generalizes well across different conditions.

CHAPTER 4

PROPOSED METHODOLOGY

4.1 Two-Stream Convolutional Neural Network(CNN)

Subsequently, a revolutionary two-stream architecture, referred to as Mario CNN, was developed, drawing inspiration from the concept proposed by Ding, Z. et al [14]. The network proposed by Ding, Z. et al incorporates two parallel streams, wherein the input is divided, distributed across the parallel streams, combined after 5 convolutional blocks, and thereafter processed jointly through fully connected layers at the final stage. The hyperparameters from the single-stream CNN in Figure 2 were utilised to fine-tune the two-stream CNN. Following several iterations of hyperparameter and layer tweaking, the best architecture for the two-stream CNN is presented in Figure 3. Compared with our first neural network, the two-stream CNN architecture has two stages. During the decomposition stage: we first randomly and evenly split our data into two sets and train them using the same convolutional blocks. The implemented architectures specified in this project are applied to the NinaPro-DB1-S1 dataset [15]. In the context of the NinaPro-DB1-S1 dataset, a total of 27 experimenters were selected as participants. Each experimenter was tasked with doing 10 repetitions of a set of 52 gestures. The set of 52 gestures encompassed three distinct categories of gestures, namely basic finger movements (E1), hand and wrist movements (E2), and functional grasping actions (E3). [16]The surface electromyography (sEMG) data was acquired at a sampling frequency of 100 Hz, utilising a configuration of ten electrodes positioned on the upper forearms. [17]The hand gestures were captured using a total of 22 sensors from the CyberGlove II device. Upon thorough examination of the DB1-S1 data folder, it is evident that there exist two significant datasets within each hands movement folder, namely 'sEMG' and 'stimulus'. The training picture used in our study is denoted as 'sEMG'. DB1- S1 data is a totally row time-series data set, so it's important to preprocess it before training. First, we need to drop the data of the rest stage (label 0). The preprocessed dataset is found from malele4th on GitHub

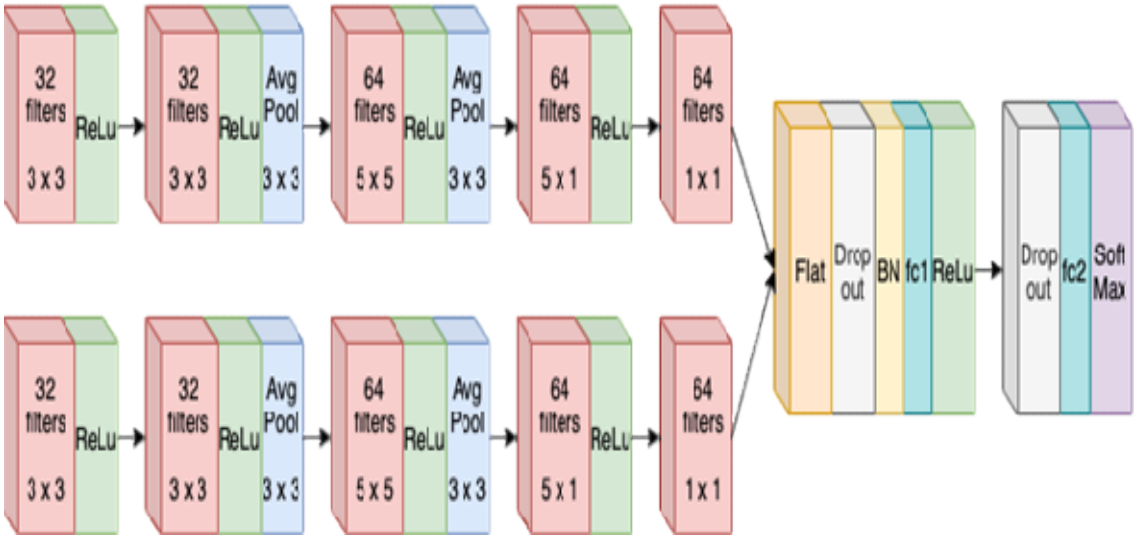


Figure 4.1:-Two-steam CNN(Convolutional Neural Network)

4.1.1.Advantage of Two-Stream CNN

1. Feature Extraction-A Two-Stream CNN can simultaneously process spatial and temporal information from sEMG signals, which is crucial for capturing the complex patterns of muscle activity.
   Spatial Stream: Captures the spatial relationships and patterns in the muscle activation, which is important for understanding the location and intensity of muscle signals.
   Temporal Stream: Focuses on the time-series aspects, extracting temporal dynamics and variations in the muscle activity over time.
2. Improved Classification Accuracy-By combining information from both spatial and temporal streams, the Two-Stream CNN can achieve higher classification accuracy compared to single-stream models. This is because it can capture a more comprehensive set of features from the sEMG signals.
3. Robustness to Noise-sEMG signals are often noisy due to factors like electrode placement, skin condition, and external interference. The Two-Stream CNN's dual focus helps in better distinguishing between relevant signal patterns and noise, thereby improving robustness.
4. Parallel Processing-The architecture of a Two-Stream CNN allows for parallel processing of spatial and temporal data, which can lead to more efficient and faster computation. This is particularly beneficial for real-time applications such as prosthetic control or rehabilitation exercises.
5. Versatility in Application-The dual-stream approach makes the model versatile for various applications, including gesture recognition, muscle fatigue analysis, an rehabilitation monitoring. It can adapt to different types of sEMG data and tasks more effectively than single-stream models. Better GeneralizationBy leveraging both spatial and temporal features, Two-Stream CNNs tend to generalize better across different subjects and conditions. This reduces the need for extensive re-training and fine-tuning when applied to new datasets or individuals.

4.1.2 Disadvantage of Two -stream CNN

1. Data Requirements:-Need for Large Datasets: To effectively train a Two-Stream CNN, a large amount of labeled data is often necessary. The NinaPro DB1 dataset may not provide enough variety or quantity of data to fully exploit the potential of a Two-Stream CNN, leading to issues like overfitting.
   Complex Data Preprocessing:-Preparing the sEMG data for both spatial and temporal streams can be complex and time-consuming. This might include synchronized recording, aligning temporal sequences, and ensuring the quality of spatial data from the electrode array.
2. Overfitting Risk:-Overfitting on Small Datasets: Given the high number of parameters in Two-Stream CNNs, there is a higher risk of overfitting, especially when the dataset is small or lacks diversity. This can be a significant issue with the NinaPro DB1 dataset if it doesn't provide sufficient variability.
   Model Generalization: While Two-Stream CNNs can generalize well with sufficient data, limited datasets like NinaPro DB1 might lead to poor generalization to new or unseen data.
3. Complexity in Model Design and Tuning:-Architectural Design: Designing an effective Two-Stream CNN involves careful consideration of how to integrate the spatial and temporal streams. This complexity can make it challenging to identify the optimal architecture without extensive experimentation.
4. Hyperparameter Tuning:- The need to tune more hyperparameters (e.g., learning rate, layer sizes, filter sizes) for both streams can complicate the model development process, requiring more expertise and time.
5. Resource Constraints:-Hardware Limitations: Deploying Two-Stream CNNs might be challenging on devices with limited computational resources, such as wearable devices used for real-time sEMG analysis. This limits the practical applicability in scenarios where computational power is constrained.
6. Energy Consumption:- Higher computational demands also translate to greater energy consumption, which can be a critical factor for battery-operated or portable devices.
   Practical Challenges with NinaPro DB1
7. Data Quality and Consistency: The NinaPro DB1 dataset, while comprehensive, might still have inconsistencies in data quality due to different recording sessions, variations in electrode placement, and subject-specific differences. These inconsistencies can impact the performance of a Two-Stream CNN.
8. Preprocessing Requirements:- Extensive preprocessing might be needed to format the NinaPro DB1 data suitably for the Two-Stream architecture, including signal normalization, artifact removal, and alignment of temporal sequences.

CHAPTER 5

EXPERIMENTAL RESULTS

This section is a discussion of the dataset used in experiments and the corresponding quantative and qualitative results.

5.1 Dataset

[18]In this project, the designed architectures mentioned are implemented on NinaPro-DB1-S1 dataset. NinaPro is a public database for research on hand gesture recognition, and the preprocessed datasets provided, DB1-DB9, have different acquisition protocols and settings. For DB1, 27 experimenters were sampled, and each experimenter repeated 10 trials of 52 gestures. The 52 gestures included three different gesture groups: Basic finger movements(E1), hand and wrist movements(E2), functional grasping movements(E3). The sEMG data was collected at a sampling rate of 100 Hz with 10 located electrodes placed on upper forearms. The hand poses were recorded by 22 sensors of CyberGlove II.

Taking a careful look at our DB1-S1 data folder, there are two important datasets in each hands movement folder: 'emg' and 'stimulus'. We take 'emg' as our training image, for 'emg' in E1, it has a shape of (101014, 10), where each of the 10 channels generates 101014 data during the processing of collecting thehand gesture data. And we take 'stimulus' as our training label, for 'stimulus' in E1, it has a shape of (101014,1), which contains 18 labels representing a rest stage and 12 basic finger movements. Each movement is around 5 seconds (500 data points).The details of our DB1-S1 data are shown in the following Table.

[19]DB1-S1 data is a totally row time-series data set, so it's important to preprocess it before training. First, we need to drop the data of the rest stage (label 0). Considering the error at the beginning and end of each movement, we select 70% of the data in the middle of each set of movements as training data. And take a 120ms (12 data points) windows to convert emg data into a 12*10 size of image. At the same time, we use a 52-dimension one-hot matrix to represent our labels. The preprocessed dataset is found from malele4th on GitHub ,with the data shape shown below in Figure 3. Table 1 The details of DB1-S1 data

emg. Shape	(101014, 10)	(142976,10)	(227493, 10)
stimulus .shape	(101014, 1)	(142976, 1)	(227493, 1)
labels and corresponding data numbers	rest stage: 0: 39063 12 movements:	rest stage: 0: 55113 17 movements:	rest stage: 0: 108449 23 movements:
	1. 5149	1. 5165	1. 5162
	2. 5174	2. 5176	2. 5188
	3. 5158	3. 5178	3. 5133
	4. 5173	4. 5169	4. 5174
	5. 5173	5. 5166	5. 5153
	6. 5170	6. 5167	6. 5132
	7. 5171	7. 5170	7. 5177
	8. 5172	8. 5177	8. 5182
	9. 5135	9. 5167	9. 5190
	10. 5137	10. 5158	10. 5182
	11. 5166	11. 5166	11. 5182
	12. 5173	12. 5170	12. 5189
		13. 5174	13. 5186
		14. 5170	14. 5191
		15. 5173	15. 5161
		16. 5155	16. 5165
		17. 5162	17. 5184
			18. 5120
			19. 5189
			20. 5202
			21. 5185
			22. 5161
			23. 5166

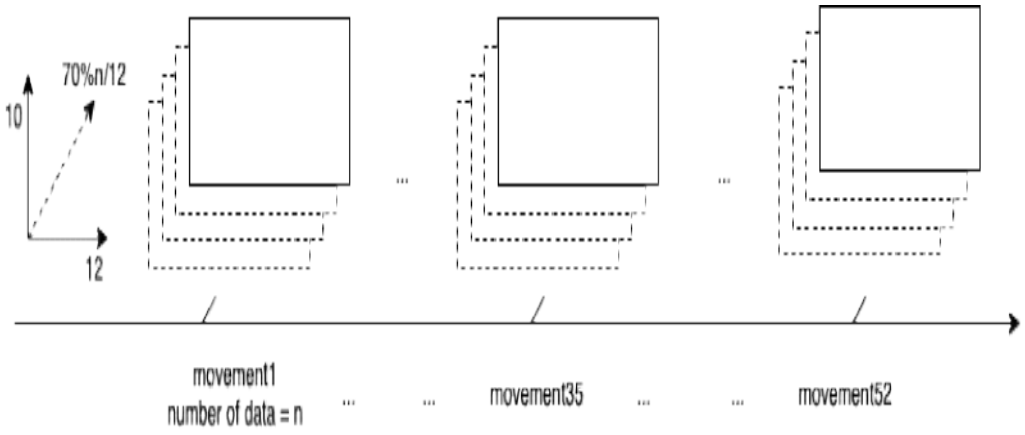
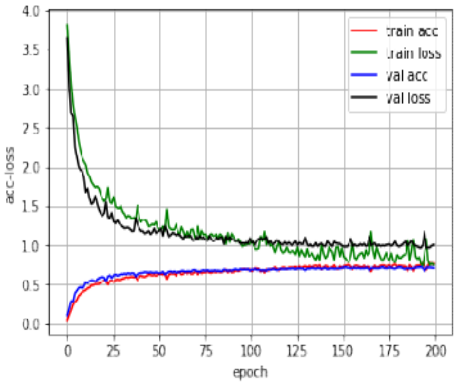
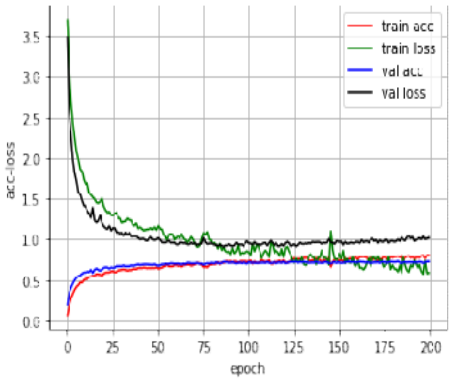


Figure 5.1: Preprocessed dataset dimensions

5.2 Results

Because the related works we referenced are not tested under the same conditions and on the same dataset, their accuracies are not eligible for direct comparisons. Therefore, we duplicated the ideas from the referenced architectures and compared them on the same dataset with the rest conditions consistent. We ran 200 Epochs for each model and have concluded the results in the tables shown below.

Table 2: Performance Comparison between single-stream CNN and Mario CNN for 200 Epochs

	CNN	Mario CNN (two-stream)
Batch Size	256	256
Input Shape	(12, 10, 1)	(12, 10, 1)
Train Accuracy	0.805	0.877
Test Accuracy	0.817	0.884
Loss & Acc Plot		

Train/test accuracy as a function of the number of streams is plotted as shown in Figure 7.

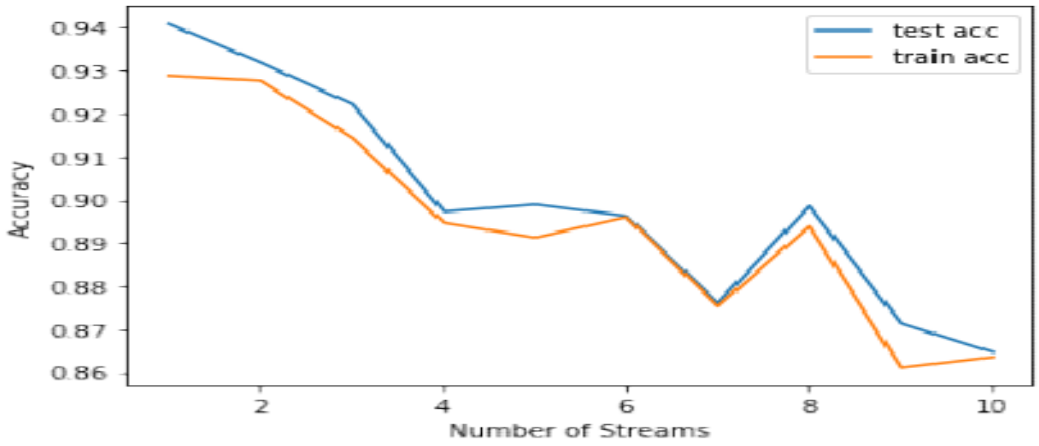
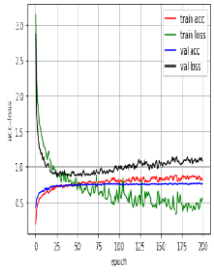
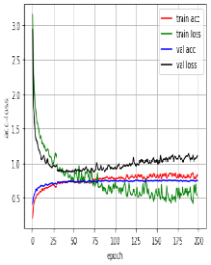
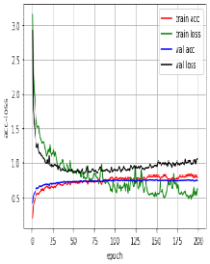
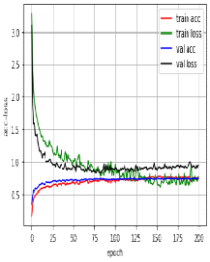
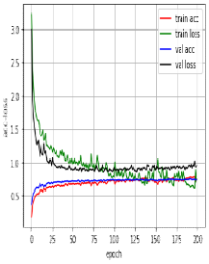
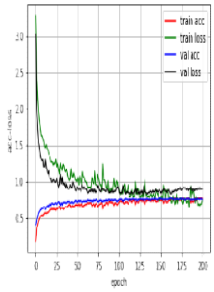
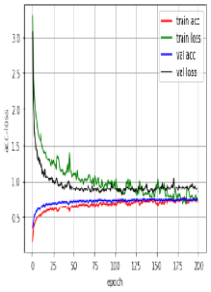
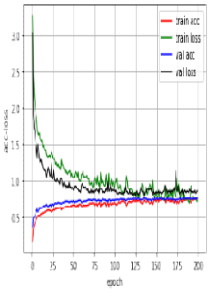
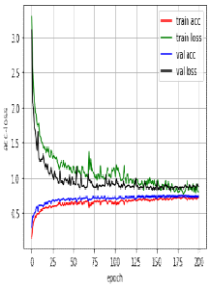
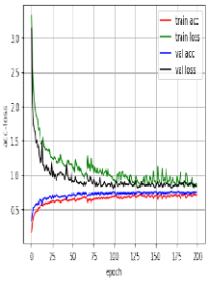


Figure 5.2: Train/test accuracy as a function of the number of streams

By increasing the number of streams from 1 to 10, given we only have 10 channels, we have observed the test accuracy increased but overfitting occurs. The loss and accuracy plots are shown below in Table 3.

Table 3: Loss and Accuracy Plots for Different Number of Streams

1-stream	2-stream	3-stream	4-stream	5-stream
				
6-stream	**7-stream**	**8-stream**	**9-stream**	**10-stream**
				

CHAPTER 6

CONCLUSION,FUTURE SCOPE AND SOCIAL IMPACT

6.1 CONCLUSION

Based on the obtained data, it is evident that the implementation of a two-stream architecture, referred to as Mario CNN, resulted in notable enhancements in performance. Using identical hyperparameters and layers, we implemented both a single-stream convolutional neural network (CNN) and a Mario CNN on a preprocessed dataset[20]. The CNN model implemented by Mario CNN has superior accuracies on both the training and test sets, achieving approximately 88contrast, the single-stream CNN model achieves accuracies of approximately 81One potential explanation is that the 10 channels exhibit varying concentrations of features. When training this data using a one-stream neural network, the resulting information obtained may be limited to a ”average” representation, potentially leading to the loss of crucial information. Alternatively, employing a dual-stream approach and subsequently merging the streams into a unified classifier may yield enhanced information extraction from surface electromyography (sEMG) data. It has been observed that only expanding the quantity of streams is not a purposeful approach, however there may be an improvement in accuracies through within each stream. Consequently, the size of pooling layers is constrained, hence causing them to encompass a surplus of inconsequential information. This, in turn, contributes to the occurrence of overfitting. The selection of suitable layer size, parameters, and the number of streams presents a significant difficulty, as it is heavily contingent upon the nature of the problem and the volume of data involved. Once the multi-stream architecture is carefully optimised, it has been demonstrated to be highly effective.

However, it should be noted that software enhancement exhibits effectiveness up to a specific threshold, beyond which it reaches a state of saturation. At this juncture, further advancements can be attained by hardware enhancements. Several studies have highlighted the effectiveness of employing highly precise sensors, such as dry-type sensors attached to the wrist [21], in enhancing the accuracy of recognition. This approach has demonstrated the ability to achieve a high recognition rate of 95 relative ease.

Figure 6.1: Changed Sensor Positions for Hardware Improvement

We can also improve our work by changing our input data. (eg: try to construct a new data feature map to improve the accuracy of sEMG classifier[11]). The existing DL studies on sEMG using data extracted fromeither time domain(TD) or frequency domain(FD). However both of these methods can only obtain partialinformation from sEMG. In order to solve this

inappropriate data processing problem, this essay proposes a fusion framework: choose a combination of TD and FD features and use it as the input data of cnn. The accuracy of the proposed method is 8.58% higher than SVM.

Furthermore, the manipulation of electrode placements has the potential to enhance the precision of recognition. The findings indicate that the placement of electrodes in close proximity to the wrist has a higher likelihood of effectively distinguishing between different movements. In a recent study conducted on individuals without physical disabilities, it was observed that the positioning of the arm had a notable impact on the efficacy of pattern classification-based myoelectric control algorithms [10]. The findings of the present study have substantiated the relevance of this effect in algorithms that rely on the preprocessing and proportional control of NinaPro. In conclusion, we have effectively constructed a novel two-stream Mario Convolutional Neural Network (CNN) and have demonstrated its superior performance compared to a single-stream CNN when evaluated on the NinaPro-DB1- S1 dataset. During this period, some reflections emerged in our thoughts. The function of kernel size in information extraction is significant. However, the relationship between hyperparameters and accuracy in this black-box technique remains unknown. Furthermore, considering that the signals emanating from various channels contain distinct information, adjusting the streams individually according to the properties of each signal, rather than maintaining uniformity across all streams, could potentially offer greater flexibility and practicality. Ultimately, it is crucial to note that drawing a conclusion that one model is superior to another just based on accuracies obtained from a single dataset is a subjective and rigid approach. In the context of an application project, the processing of surface electromyography (sEMG) signals is influenced by various aspects, including biological circumstances, environmental scenarios, data sources, and other relevant considerations. In order to enhance the scope of this project, it is advisable to regard the Mario CNN as a tool and employ it on a wider range of datasets. Additionally, it is beneficial to investigate the processing of sEMG signals as a more comprehensive statistical problem we successfully built a novel two-stream Mario CNN and it is proven to perform better than a single-stream CNN on the dataset NinaPro-DB1-S1. In the meantime, a few reflections came to our minds. Firstly, while the kernel size plays a pivotal role in information extraction, the casual relationship between hyperparameters and accuracy in this black-box procedure is still a mystery. Second, given that the signals from different channels reflect different information, tuning the streams independently based on signal characteristics rather than having all the streams the same may grant more flexibility and practicalities. Finally, and most importantly, concluding a model is "better" than another merely based on accuracies on one dataset is ambiguous and dogmatical. As an application project, sEMG signal processing is affected by many other factors, like biological conditions, scenarios, data sources, etc. To further expand this project, it is more valuable to consider this Mario CNN as a tool and apply it on more diverse datasets and study sEMG signal processing as a broader problem from a statistical perspective.

6.2 FUTURE SCOPE AND SOCIAL IMPACT

The future scope of sEMG signal analysis includes advancements in neuroprosthetics, refined rehabilitation protocols, enhanced human-computer interfaces, improved diagnostics in neuromuscular disorders, and the development of sophisticated biofeedback systems for sports and ergonomics, leveraging AI and machine learning for greater precision and personalized healthcare solutions.

REFERENCES

[1] H. Tankisi, D. Burke, L. Cui, M. de Carvalho, S. Kuwabara, S. D. Nandedkar, S. Rutkove, E. Stalberg, ˚M. J. van Putten, and A. Fuglsang-Frederiksen, "Standards of instrumentation of emg," Clinical neurophysiology, vol. 131, no. 1, pp. 243–258, 2020.

[2] R. Shioji, S.-i. Ito, M. Ito, and M. Fukumi, "Personal authentication and hand motion recognition based on wrist emg analysis by a convolutional neural network," in 2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS), pp. 184–188, IEEE, 2018.

[3] D. Buongiorno, G. D. Cascarano, I. De Feudis, A. Brunetti, L. Carnimeo, G. Dimauro, and V. Bevilacqua, "Deep learning for processing electromyographic signals: A taxonomy-based survey," Neurocomputing, vol. 452, pp. 549–565, 2021.

[4] B. Xiong, W. Chen, Y. Niu, Z. Gan, G. Mao, and Y. Xu, "A global and local feature fused cnn architecture for the semg-based hand gesture recognition," Computers in Biology and Medicine, vol. 166, p. 107497, 2023.

[5] F. Leone, C. Gentile, A. L. Ciancio, E. Gruppioni, A. Davalli, R. Sacchetti, E. Guglielmelli, and L. Zollo, "Simultaneous semg classification of hand/wrist gestures and forces," Frontiers in neurorobotics, vol. 13, p. 42, 2019.

[6] N. Abdullaev and K. S. Pashaeva, "Use of machine learning models for classification of myographic diseases," Biomedical Engineering, vol. 56, no. 5, pp. 353–357, 2023.

[7] M. Atzori, M. Cognolato, and H. Muller, "Deep learn- ¨ ing with convolutional neural networks applied to electromyography data: A resource for the classification of movements for prosthetic hands," Frontiers in neurorobotics, vol. 10, p. 9, 2016.

[8] P. Sri-Iesaranusorn, A. Chaiyaroj, C. Buekban, S. Dumnin, R. Pongthornseri, C. Thanawattano, and D. Surangsrirat, "Classification of 41 hand and wrist movements via surface electromyogram using deep neural network," Frontiers in bioengineering and biotechnology, vol. 9, p. 548357, 2021.

[9] Y. Narayan, L. Mathew, and S. Chatterji, "Semg signal classification with novel feature extraction using different machine learning approaches," Journal of Intelligent & Fuzzy Systems, vol. 35, no. 5, pp. 5099–5109, 2018.

[10] A. Vijayvargiya, Khimraj, R. Kumar, and N. Dey, "Voting-based 1d cnn model for human lower limb activity recognition using semg signal," Physical and Engineering Sciences in Medicine, vol. 44, no. 4, pp. 1297– 1309, 2021.

[11] D. Farina and A. Holobar, "Human? machine interfacing by decoding the surface electromyogram [life sciences]," IEEE signal processing magazine, vol. 32, no. 1, pp. 115–120, 2014.

[12] Y. Liu, X. Peng, Y. Tan, T. T. Oyemakinde, M. Wang, G. Li, and X. Li, "A novel unsupervised dynamic feature domain adaptation strategy for cross-individual myoelectric gesture recognition," Journal of Neural Engineering, vol. 20, no. 6, p. 066044, 2024.

[13] Z. Ding, C. Yang, Z. Tian, C. Yi, Y. Fu, and F. Jiang, "semg-based gesture recognition with convolution neural networks," Sustainability, vol. 10, no. 6, p. 1865, 2018.

[14] X. Wang, L. Tang, Q. Zheng, X. Yang, and Z. Lu, "Irdcnet: An inception network with a residual module and dilated convolution for sign language recognition based on surface electromyography," Sensors, vol. 23, no. 13, p. 5775, 2023.

[15] Y. Luo, T. Luo, Q. Xia, H. Yan, L. Xie, Y. Yan, and E. Yin, "A fusion framework to enhance semg-based gesture recognition using td and fd features," in Neural Information Processing:

28th International Conference, ICONIP 2021, Sanur, Bali, Indonesia, December 8–12, 2021, Proceedings, Part VI 28, pp. 168–175, Springer, 2021.
[16] M. Atzori and H. Muller, “The ninapro database: a ¨ resource for semg naturally controlled robotic hand prosthetics,” in 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 7151–7154, IEEE, 2015.
[17] N. Jiang, S. Muceli, B. Graimann, and D. Farina, “Effect of arm position on the prediction of kinematics from emg in amputees,” Medical & biological engineering & computing, vol. 51, pp. 143–151, 2013.
[18] J. Qi, G. Jiang, G. Li, Y. Sun, and B. Tao, “Intelligent human-computer interaction based on surface emg gesture recognition,” Ieee Access, vol. 7, pp. 61378–61387, 2019.
[19] W. Ding, G. Li, Y. Sun, G. Jiang, J. Kong, and H. Liu, “Ds evidential theory on semg signal recognition,” International Journal of Computing Science and Mathematics, vol. 8, no. 2, pp. 138–145, 2017.
[20] Y. Zhou, Y. Fang, K. Gui, K. Li, D. Zhang, and H. Liu, “semg bias-driven functional electrical stimulation system for upper-limb stroke rehabilitation,” IEEE Sensors Journal, vol. 18, no. 16, pp. 6812–6821, 2018.
[21] Y. Sun, C. Li, G. Li, G. Jiang, D. Jiang, H. Liu, Z. Zheng, and W. Shu, “Gesture recognition based on kinect and semg signal fusion,” Mobile Networks and Applications, vol. 23, pp. 797–805, 2018

APPENDICES

CNN

```
import scipy.io
import h5py
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
import keras
import keras.backend as K
from keras.layers import Input, Dense, ZeroPadding2D, Dropout,
Activation, Flatten, Conv2D, Conv1D, MaxPooling2D, AveragePooling2D,
concatenate, BatchNormalization
from keras.models import Model
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

Prepare Preprocessed Dataset

```
file = h5py.File('/content/DB1_S1_image (3).h5','r')
imageData   = file['imageData'][:]
imageLabel  = file['imageLabel'][:]
file.close()
file = h5py.File('/content/DB1_S1_image (3).h5','r')
imageData   = file['imageData'][:]
imageLabel  = file['imageLabel'][:]
file.close()
```

```
(15047, 12, 10)
(15047,)
```

```
def convert_to_one_hot(Y, C):
    Y = np.eye(C)[Y.reshape(-1)].T
    return Y
# prepare data
n = imageData.shape[0]
idx = np.random.permutation(n)
data  = imageData[idx]
label = imageLabel[idx]

data  = np.expand_dims(data, axis=3)
label = convert_to_one_hot(label, 52).T

X_train, X_test, Y_train, Y_test = train_test_split(data, label,
test_size = 0.2, random_state = 42)

print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(Y_train.shape))
print ("X_test shape: " + str(X_test.shape))
```

```
print ("Y_test shape: " + str(Y_test.shape))
X_train shape: (12037, 12, 10, 1)
Y_train shape: (12037, 52)

X_test shape: (3010, 12, 10, 1)
Y_test shape: (3010, 52)

class LossHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.losses = {'epoch':[]}
        self.accuracy = {'epoch':[]}
        self.val_loss = {'epoch':[]}
        self.val_acc = {'epoch':[]}

    def on_epoch_end(self, batch, logs={}):
        self.losses['epoch'].append(logs.get('loss'))
        self.accuracy['epoch'].append(logs.get('accuracy'))
        self.val_loss['epoch'].append(logs.get('val_loss'))
        self.val_acc['epoch'].append(logs.get('val_accuracy'))

    def loss_plot(self, loss_type):
        iters = range(len(self.losses[loss_type]))
        plt.figure()
        plt.plot(iters, self.accuracy[loss_type], 'r', label='train
acc')
        plt.plot(iters, self.losses[loss_type], 'g', label='train
loss')
        plt.plot(iters, self.val_acc[loss_type], 'b', label='val acc')
        plt.plot(iters, self.val_loss[loss_type], 'k', label='val
loss')
        plt.grid(True)
        plt.xlabel(loss_type)
        plt.ylabel('acc-loss')
        plt.legend(loc="upper right")
        plt.show()
def CNN(input_shape, classes):
    X_input = Input(input_shape)

    X = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name='conv1')(X_input)
    X = Activation('relu', name='relu1')(X)

    X = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name='conv2')(X)
    X = Activation('relu', name='relu2')(X)
    X = AveragePooling2D((3,3), strides=(2,2), name='pool1')(X)
print ("Y_train shape: " + str(Y_train.shape))
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(Y_test.shape))
X_train shape: (12037, 12, 10, 1)
```

```
Y_train shape: (12037, 52)
X_test shape: (3010, 12, 10, 1)
Y_test shape: (3010, 52)
```

```
class LossHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.losses = {'epoch':[]}
        self.accuracy = {'epoch':[]}
        self.val_loss = {'epoch':[]}
        self.val_acc = {'epoch':[]}

    def on_epoch_end(self, batch, logs={}):
        self.losses['epoch'].append(logs.get('loss'))
        self.accuracy['epoch'].append(logs.get('accuracy'))
        self.val_loss['epoch'].append(logs.get('val_loss'))
        self.val_acc['epoch'].append(logs.get('val_accuracy'))

    def loss_plot(self, loss_type):
        iters = range(len(self.losses[loss_type]))
        plt.figure()
        plt.plot(iters, self.accuracy[loss_type], 'r', label='train
acc')
        plt.plot(iters, self.losses[loss_type], 'g', label='train
loss')
        plt.plot(iters, self.val_acc[loss_type], 'b', label='val acc')
        plt.plot(iters, self.val_loss[loss_type], 'k', label='val
loss')
        plt.grid(True)
        plt.xlabel(loss_type)
        plt.ylabel('acc-loss')
        plt.legend(loc="upper right")
        plt.show()
def CNN(input_shape, classes):
    X_input = Input(input_shape)

    X = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name='conv1')(X_input)
    X = Activation('relu', name='relu1')(X)

    X = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name='conv2')(X)
    X = Activation('relu', name='relu2')(X)
    X = AveragePooling2D((3,3), strides=(2,2), name='pool1')(X)

    X = Conv2D(filters=64, kernel_size=(5,5),
strides=(1,1),padding='same', name='conv3')(X)
    X = Activation('relu', name='relu3')(X)
    X = AveragePooling2D((3,3), strides=(2,2), name='pool2')(X)
```

```
    X = Conv2D(filters=64, kernel_size=(5,1),
strides=(1,1),padding='same', name='conv4')(X)
    X = Activation('relu', name='relu4')(X)

    X = Conv2D(filters=64, kernel_size=(1,1),
strides=(1,1),padding='same', name='conv5')(X)

    X = ZeroPadding2D((0,1))(X)

    X = Flatten(name='flatten')(X)
    X = Dropout(0.5)(X)
    X = BatchNormalization(momentum=0.9)(X)
    X = Dense(128, activation='relu',name='fc1')(X)
    X = Dropout(0.5)(X)
    X = Dense(classes, activation='softmax', name='fc2')(X)

    model = Model(inputs=X_input, outputs=X, name='CNN')
    return model
model = CNN(input_shape = (12, 10, 1), classes = 52)
model.summary()
```

```
Model: "CNN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 12, 10, 1)]       0

 conv1 (Conv2D)              (None, 12, 10, 32)        320

 relu1 (Activation)          (None, 12, 10, 32)        0

 conv2 (Conv2D)              (None, 12, 10, 32)        9248

 relu2 (Activation)          (None, 12, 10, 32)        0

 pool1 (AveragePooling2D)    (None, 5, 4, 32)          0

 conv3 (Conv2D)              (None, 5, 4, 64)          51264

 relu3 (Activation)          (None, 5, 4, 64)          0

 pool2 (AveragePooling2D)    (None, 2, 1, 64)          0

 conv4 (Conv2D)              (None, 2, 1, 64)          20544

Trainable params: 142292 (555.83 KB)
Non-trainable params: 768 (3.00 KB)
_________________________________________________________________
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
history = LossHistory()
```

```
model.fit(data, label, validation_split=0.2, epochs=eps,
batch_size=256, verbose=1, callbacks=[history])

print('-----------------------------------------------------------------
---------')
preds_train = model.evaluate(X_train, Y_train)
print("Train Loss = " + str(preds_train[0]))
print("Train Accuracy = " + str(preds_train[1]))

preds_test  = model.evaluate(X_test, Y_test)
print("Test Loss = " + str(preds_test[0]))
print("Test Accuracy = " + str(preds_test[1]))
```

```
Epoch 1/200
48/48 [==============================] - 15s 21ms/step - loss: 3.8610 -
accuracy: 0.0437 - val_loss: 3.7660 - val_accuracy: 0.0625
Epoch 2/200
48/48 [==============================] - 1s 11ms/step - loss: 3.5509 -
accuracy: 0.0901 - val_loss: 3.1626 - val_accuracy: 0.1714
Epoch 3/200
48/48 [==============================] - 1s 19ms/step - loss: 3.1587 -
accuracy: 0.1550 - val_loss: 2.8182 - val_accuracy: 0.2561
Epoch 4/200
48/48 [==============================] - 1s 20ms/step - loss: 2.8967 -
accuracy: 0.2052 - val_loss: 2.5480 - val_accuracy: 0.3213
Epoch 5/200
48/48 [==============================] - 1s 16ms/step - loss: 2.6741 -
accuracy: 0.2571 - val_loss: 2.4255 - val_accuracy: 0.3462
Epoch 6/200
48/48 [==============================] - 0s 10ms/step - loss: 2.5189 -
accuracy: 0.2910 - val_loss: 2.2239 - val_accuracy: 0.3944
Epoch 7/200
48/48 [==============================] - 0s 10ms/step - loss: 2.3794 -
accuracy: 0.3305 - val_loss: 2.1168 - val_accuracy: 0.4093
Epoch 8/200
48/48 [==============================] - 0s 10ms/step - loss: 2.2431 -
accuracy: 0.3705 - val_loss: 2.2452 - val_accuracy: 0.3837
Epoch 9/200
48/48 [==============================] - 0s 9ms/step - loss: 2.2335 -
accuracy: 0.3699 - val_loss: 1.8946 - val_accuracy: 0.4794
Epoch 10/200
48/48 [==============================] - 0s 9ms/step - loss: 2.1127 -
accuracy: 0.4032 - val_loss: 1.8166 - val_accuracy: 0.4970
Epoch 11/200
48/48 [==============================] - 0s 10ms/step - loss: 1.9966 -
accuracy: 0.4313 - val_loss: 1.8048 - val_accuracy: 0.5037
Epoch 12/200
48/48 [==============================] - 0s 9ms/step - loss: 1.9593 -
accuracy: 0.4454 - val_loss: 1.7080 - val_accuracy: 0.5173
Epoch 13/200
48/48 [==============================] - 0s 9ms/step - loss: 1.8677 -
accuracy: 0.4628 - val_loss: 1.7042 - val_accuracy: 0.5246
Epoch 14/200
48/48 [==============================] - 1s 11ms/step - loss: 1.8930 -
accuracy: 0.4602 - val_loss: 1.6868 - val_accuracy: 0.5143
Epoch 15/200
```

```
48/48 [==============================] - 1s 12ms/step - loss: 1.8340 -
accuracy: 0.4760 - val_loss: 1.6319 - val_accuracy: 0.5339
Epoch 16/200
48/48 [==============================] - 1s 11ms/step - loss: 1.7815 -
accuracy: 0.4877 - val_loss: 1.5691 - val_accuracy: 0.5548
Epoch 17/200
48/48 [==============================] - 1s 12ms/step - loss: 1.7207 -
accuracy: 0.5058 - val_loss: 1.4951 - val_accuracy: 0.5738
Epoch 18/200
48/48 [==============================] - 1s 12ms/step - loss: 1.7350 -
accuracy: 0.4945 - val_loss: 1.5239 - val_accuracy: 0.5698
Epoch 19/200
48/48 [==============================] - 1s 12ms/step - loss: 1.7106 -
accuracy: 0.5028 - val_loss: 1.4787 - val_accuracy: 0.5824
Epoch 20/200
48/48 [==============================] - 1s 13ms/step - loss: 1.6163 -
accuracy: 0.5273 - val_loss: 1.4249 - val_accuracy: 0.5963
Epoch 21/200
48/48 [==============================] - 1s 11ms/step - loss: 1.6259 -
accuracy: 0.5246 - val_loss: 1.4477 - val_accuracy: 0.5917
Epoch 22/200
48/48 [==============================] - 0s 9ms/step - loss: 1.5641 -
accuracy: 0.5404 - val_loss: 1.4319 - val_accuracy: 0.5884
Epoch 23/200
48/48 [==============================] - 0s 10ms/step - loss: 1.5943 -
accuracy: 0.5325 - val_loss: 1.3863 - val_accuracy: 0.6040
Epoch 24/200
48/48 [==============================] - 0s 9ms/step - loss: 1.5423 -
accuracy: 0.5455 - val_loss: 1.4070 - val_accuracy: 0.5997
Epoch 25/200
48/48 [==============================] - 0s 10ms/step - loss: 1.5617 -
accuracy: 0.5447 - val_loss: 1.3834 - val_accuracy: 0.6047
Epoch 26/200
48/48 [==============================] - 0s 10ms/step - loss: 1.5235 -
accuracy: 0.5506 - val_loss: 1.3370 - val_accuracy: 0.6130
Epoch 27/200
48/48 [==============================] - 0s 10ms/step - loss: 1.4843 -
accuracy: 0.5589 - val_loss: 1.3012 - val_accuracy: 0.6312
Epoch 28/200
48/48 [==============================] - 0s 10ms/step - loss: 1.4534 -
accuracy: 0.5648 - val_loss: 1.2781 - val_accuracy: 0.6243
Epoch 29/200
48/48 [==============================] - 0s 10ms/step - loss: 1.4237 -
accuracy: 0.5757 - val_loss: 1.2755 - val_accuracy: 0.6352
Epoch 30/200
48/48 [==============================] - 0s 10ms/step - loss: 1.4587 -
accuracy: 0.5649 - val_loss: 1.2899 - val_accuracy: 0.6262
Epoch 31/200
48/48 [==============================] - 0s 10ms/step - loss: 1.4334 -
accuracy: 0.5741 - val_loss: 1.5137 - val_accuracy: 0.5721
Epoch 32/200
48/48 [==============================] - 0s 9ms/step - loss: 1.4811 -
accuracy: 0.5547 - val_loss: 1.3145 - val_accuracy: 0.6173
Epoch 33/200
48/48 [==============================] - 0s 9ms/step - loss: 1.3880 -
accuracy: 0.5806 - val_loss: 1.2988 - val_accuracy: 0.6282
Epoch 34/200
```

```
48/48 [==============================] - 0s 10ms/step - loss: 1.4136 -
accuracy: 0.5746 - val_loss: 1.2696 - val_accuracy: 0.6282
Epoch 35/200
48/48 [==============================] - 0s 9ms/step - loss: 1.4046 -
accuracy: 0.5825 - val_loss: 1.3322 - val_accuracy: 0.6146
Epoch 36/200
48/48 [==============================] - 0s 10ms/step - loss: 1.4506 -
accuracy: 0.5683 - val_loss: 1.3728 - val_accuracy: 0.5987
Epoch 37/200
48/48 [==============================] - 0s 9ms/step - loss: 1.3614 -
accuracy: 0.5935 - val_loss: 1.2824 - val_accuracy: 0.6306
Epoch 38/200
48/48 [==============================] - 0s 9ms/step - loss: 1.3467 -
accuracy: 0.5926 - val_loss: 1.2411 - val_accuracy: 0.6319
Epoch 39/200
48/48 [==============================] - 0s 9ms/step - loss: 1.3867 -
accuracy: 0.5926 - val_loss: 1.2626 - val_accuracy: 0.6352
Epoch 40/200
48/48 [==============================] - 0s 10ms/step - loss: 1.3714 -
accuracy: 0.5879 - val_loss: 1.1923 - val_accuracy: 0.6468
Epoch 41/200
48/48 [==============================] - 0s 9ms/step - loss: 1.3485 -
accuracy: 0.5952 - val_loss: 1.2297 - val_accuracy: 0.6415
Epoch 42/200
48/48 [==============================] - 1s 11ms/step - loss: 1.3290 -
accuracy: 0.6030 - val_loss: 1.1646 - val_accuracy: 0.6621
Epoch 43/200
48/48 [==============================] - 1s 15ms/step - loss: 1.2796 -
accuracy: 0.6119 - val_loss: 1.1753 - val_accuracy: 0.6585
Epoch 44/200
48/48 [==============================] - 1s 13ms/step - loss: 1.2908 -
accuracy: 0.6083 - val_loss: 1.2028 - val_accuracy: 0.6508
Epoch 45/200
48/48 [==============================] - 1s 13ms/step - loss: 1.3060 -
accuracy: 0.6114 - val_loss: 1.1879 - val_accuracy: 0.6508
Epoch 46/200
48/48 [==============================] - 1s 13ms/step - loss: 1.3231 -
accuracy: 0.6010 - val_loss: 1.1951 - val_accuracy: 0.6482
Epoch 47/200
48/48 [==============================] - 1s 13ms/step - loss: 1.2497 -
accuracy: 0.6223 - val_loss: 1.1597 - val_accuracy: 0.6651
Epoch 48/200
48/48 [==============================] - 1s 12ms/step - loss: 1.2235 -
accuracy: 0.6300 - val_loss: 1.1563 - val_accuracy: 0.6631
Epoch 49/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1976 -
accuracy: 0.6348 - val_loss: 1.1981 - val_accuracy: 0.6615
Epoch 50/200
48/48 [==============================] - 0s 9ms/step - loss: 1.2992 -
accuracy: 0.6112 - val_loss: 1.2062 - val_accuracy: 0.6432
Epoch 51/200
48/48 [==============================] - 0s 9ms/step - loss: 1.2788 -
accuracy: 0.6109 - val_loss: 1.1375 - val_accuracy: 0.6678
Epoch 52/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2295 -
accuracy: 0.6293 - val_loss: 1.1993 - val_accuracy: 0.6455
Epoch 53/200
```

```
48/48 [==============================] - 0s 10ms/step - loss: 1.2130 -
accuracy: 0.6317 - val_loss: 1.1719 - val_accuracy: 0.6508
Epoch 54/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2751 -
accuracy: 0.6136 - val_loss: 1.1437 - val_accuracy: 0.6565
Epoch 55/200
48/48 [==============================] - 0s 9ms/step - loss: 1.2748 -
accuracy: 0.6160 - val_loss: 1.1248 - val_accuracy: 0.6701
Epoch 56/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2103 -
accuracy: 0.6295 - val_loss: 1.1103 - val_accuracy: 0.6698
Epoch 57/200
48/48 [==============================] - 0s 9ms/step - loss: 1.2053 -
accuracy: 0.6319 - val_loss: 1.1745 - val_accuracy: 0.6558
Epoch 58/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2055 -
accuracy: 0.6365 - val_loss: 1.1078 - val_accuracy: 0.6658
Epoch 59/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1894 -
accuracy: 0.6358 - val_loss: 1.2078 - val_accuracy: 0.6502
Epoch 60/200
48/48 [==============================] - 0s 9ms/step - loss: 1.2996 -
accuracy: 0.6108 - val_loss: 1.1348 - val_accuracy: 0.6721
Epoch 61/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1598 -
accuracy: 0.6473 - val_loss: 1.0861 - val_accuracy: 0.6791
Epoch 62/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1164 -
accuracy: 0.6591 - val_loss: 1.1433 - val_accuracy: 0.6691
Epoch 63/200
48/48 [==============================] - 0s 9ms/step - loss: 1.2295 -
accuracy: 0.6297 - val_loss: 1.1218 - val_accuracy: 0.6718
Epoch 64/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1823 -
accuracy: 0.6433 - val_loss: 1.1551 - val_accuracy: 0.6648
Epoch 65/200
48/48 [==============================] - 0s 9ms/step - loss: 1.3179 -
accuracy: 0.6114 - val_loss: 1.1578 - val_accuracy: 0.6688
Epoch 66/200
48/48 [==============================] - 0s 9ms/step - loss: 1.2947 -
accuracy: 0.6165 - val_loss: 1.1156 - val_accuracy: 0.6764
Epoch 67/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2487 -
accuracy: 0.6288 - val_loss: 1.0919 - val_accuracy: 0.6807
Epoch 68/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1326 -
accuracy: 0.6518 - val_loss: 1.0794 - val_accuracy: 0.6854
Epoch 69/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1212 -
accuracy: 0.6600 - val_loss: 1.1641 - val_accuracy: 0.6601
Epoch 70/200
48/48 [==============================] - 1s 13ms/step - loss: 1.2254 -
accuracy: 0.6338 - val_loss: 1.1141 - val_accuracy: 0.6847
Epoch 71/200
48/48 [==============================] - 1s 13ms/step - loss: 1.1811 -
accuracy: 0.6425 - val_loss: 1.0726 - val_accuracy: 0.6850
Epoch 72/200
```

```
48/48 [==============================] - 1s 12ms/step - loss: 1.1258 -
accuracy: 0.6533 - val_loss: 1.0675 - val_accuracy: 0.6880
Epoch 73/200
48/48 [==============================] - 1s 12ms/step - loss: 1.0934 -
accuracy: 0.6662 - val_loss: 1.0638 - val_accuracy: 0.6821
Epoch 74/200
48/48 [==============================] - 1s 12ms/step - loss: 1.1579 -
accuracy: 0.6527 - val_loss: 1.0927 - val_accuracy: 0.6771
Epoch 75/200
48/48 [==============================] - 1s 12ms/step - loss: 1.0861 -
accuracy: 0.6648 - val_loss: 1.0906 - val_accuracy: 0.6837
Epoch 76/200
48/48 [==============================] - 1s 13ms/step - loss: 1.1434 -
accuracy: 0.6540 - val_loss: 1.0727 - val_accuracy: 0.6884
Epoch 77/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1108 -
accuracy: 0.6576 - val_loss: 1.1777 - val_accuracy: 0.6525
Epoch 78/200
48/48 [==============================] - 0s 9ms/step - loss: 1.2591 -
accuracy: 0.6269 - val_loss: 1.1126 - val_accuracy: 0.6741
Epoch 79/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1318 -
accuracy: 0.6546 - val_loss: 1.1306 - val_accuracy: 0.6718
Epoch 80/200
48/48 [==============================] - 0s 9ms/step - loss: 1.2041 -
accuracy: 0.6400 - val_loss: 1.0804 - val_accuracy: 0.6794
Epoch 81/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0730 -
accuracy: 0.6746 - val_loss: 1.1629 - val_accuracy: 0.6601
Epoch 82/200
48/48 [==============================] - 0s 8ms/step - loss: 1.1724 -
accuracy: 0.6433 - val_loss: 1.0518 - val_accuracy: 0.6847
Epoch 83/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0449 -
accuracy: 0.6784 - val_loss: 1.0527 - val_accuracy: 0.6937
Epoch 84/200
48/48 [==============================] - 0s 8ms/step - loss: 1.0699 -
accuracy: 0.6727 - val_loss: 1.0582 - val_accuracy: 0.6860
Epoch 85/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0764 -
accuracy: 0.6660 - val_loss: 1.0435 - val_accuracy: 0.6900
Epoch 86/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0216 -
accuracy: 0.6914 - val_loss: 1.0420 - val_accuracy: 0.6847
Epoch 87/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0391 -
accuracy: 0.6854 - val_loss: 1.0427 - val_accuracy: 0.6890
Epoch 88/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0287 -
accuracy: 0.6807 - val_loss: 1.0258 - val_accuracy: 0.7007
Epoch 89/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0130 -
accuracy: 0.6930 - val_loss: 1.0143 - val_accuracy: 0.7007
Epoch 90/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0000 -
accuracy: 0.6941 - val_loss: 1.0519 - val_accuracy: 0.6930
Epoch 91/200
```

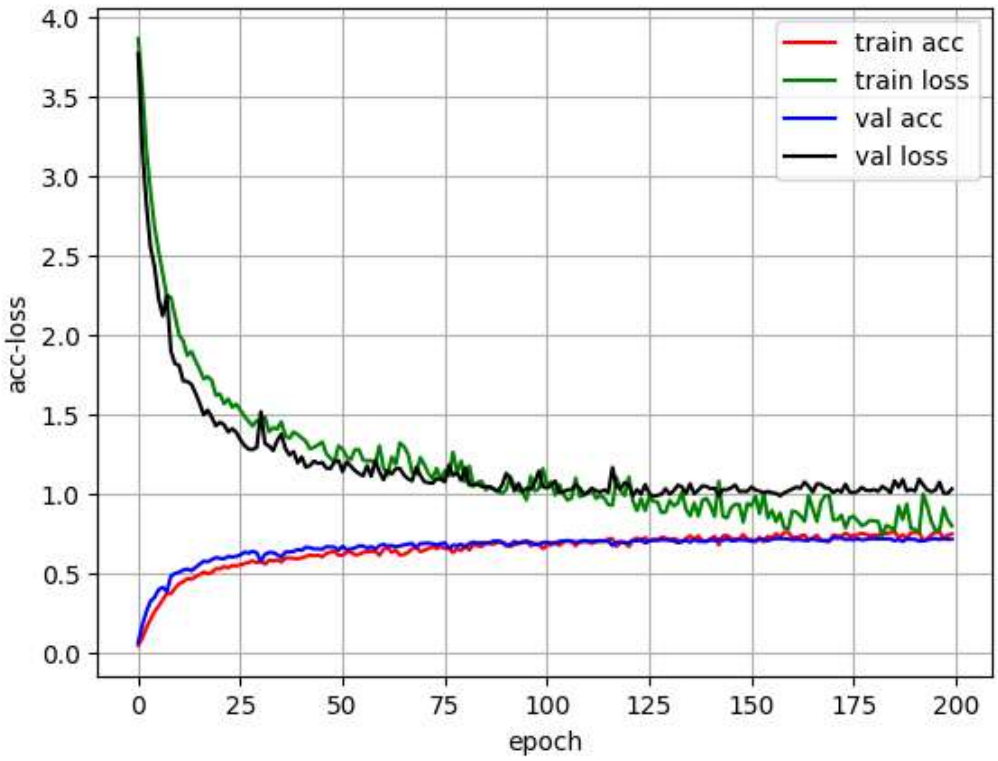
```
48/48 [==============================] - 0s 9ms/step - loss: 1.0233 -
accuracy: 0.6812 - val_loss: 1.1253 - val_accuracy: 0.6704
Epoch 92/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1071 -
accuracy: 0.6657 - val_loss: 1.0938 - val_accuracy: 0.6774
Epoch 93/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0463 -
accuracy: 0.6771 - val_loss: 1.0149 - val_accuracy: 0.6977
Epoch 94/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0175 -
accuracy: 0.6857 - val_loss: 1.0630 - val_accuracy: 0.6860
Epoch 95/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0094 -
accuracy: 0.6890 - val_loss: 1.0113 - val_accuracy: 0.6963
Epoch 96/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9596 -
accuracy: 0.7028 - val_loss: 1.0325 - val_accuracy: 0.6940
Epoch 97/200
48/48 [==============================] - 0s 8ms/step - loss: 1.1080 -
accuracy: 0.6690 - val_loss: 1.0261 - val_accuracy: 0.6937
Epoch 98/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0144 -
accuracy: 0.6853 - val_loss: 1.0696 - val_accuracy: 0.6777
Epoch 99/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0301 -
accuracy: 0.6846 - val_loss: 1.1412 - val_accuracy: 0.6777
Epoch 100/200
48/48 [==============================] - 1s 12ms/step - loss: 1.1553 -
accuracy: 0.6515 - val_loss: 1.0418 - val_accuracy: 0.6914
Epoch 101/200
48/48 [==============================] - 1s 11ms/step - loss: 1.0541 -
accuracy: 0.6748 - val_loss: 1.0341 - val_accuracy: 0.6924
Epoch 102/200
48/48 [==============================] - 1s 11ms/step - loss: 1.0731 -
accuracy: 0.6755 - val_loss: 1.0500 - val_accuracy: 0.6880
Epoch 103/200
48/48 [==============================] - 1s 11ms/step - loss: 0.9886 -
accuracy: 0.6920 - val_loss: 1.0774 - val_accuracy: 0.6877
Epoch 104/200
48/48 [==============================] - 1s 12ms/step - loss: 1.0545 -
accuracy: 0.6729 - val_loss: 1.0065 - val_accuracy: 0.6980
Epoch 105/200
48/48 [==============================] - 1s 11ms/step - loss: 0.9951 -
accuracy: 0.6945 - val_loss: 1.0102 - val_accuracy: 0.7003
Epoch 106/200
48/48 [==============================] - 1s 11ms/step - loss: 0.9577 -
accuracy: 0.7057 - val_loss: 1.0227 - val_accuracy: 0.7000
Epoch 107/200
48/48 [==============================] - 1s 11ms/step - loss: 0.9710 -
accuracy: 0.6952 - val_loss: 1.0248 - val_accuracy: 0.7000
Epoch 108/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0973 -
accuracy: 0.6704 - val_loss: 1.0527 - val_accuracy: 0.6930
Epoch 109/200
48/48 [==============================] - 0s 8ms/step - loss: 0.9788 -
accuracy: 0.6969 - val_loss: 1.0334 - val_accuracy: 0.6950
Epoch 110/200
```

```
48/48 [==============================] - 0s 8ms/step - loss: 0.9896 -
accuracy: 0.6968 - val_loss: 0.9942 - val_accuracy: 0.7040
Epoch 111/200
48/48 [==============================] - 0s 8ms/step - loss: 0.9990 -
accuracy: 0.6971 - val_loss: 1.0099 - val_accuracy: 0.6997
Epoch 112/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0236 -
accuracy: 0.6850 - val_loss: 1.0323 - val_accuracy: 0.6990
Epoch 113/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9674 -
accuracy: 0.6970 - val_loss: 1.0158 - val_accuracy: 0.7010
Epoch 114/200
48/48 [==============================] - 0s 8ms/step - loss: 0.9867 -
accuracy: 0.6983 - val_loss: 1.0094 - val_accuracy: 0.7030
Epoch 115/200
48/48 [==============================] - 0s 8ms/step - loss: 0.9700 -
accuracy: 0.6986 - val_loss: 1.0216 - val_accuracy: 0.7017
Epoch 116/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9293 -
accuracy: 0.7137 - val_loss: 1.0071 - val_accuracy: 0.6993
Epoch 117/200
48/48 [==============================] - 0s 8ms/step - loss: 0.9252 -
accuracy: 0.7150 - val_loss: 1.1629 - val_accuracy: 0.6721
Epoch 118/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1000 -
accuracy: 0.6669 - val_loss: 1.0450 - val_accuracy: 0.6920
Epoch 119/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0171 -
accuracy: 0.6885 - val_loss: 1.0300 - val_accuracy: 0.6970
Epoch 120/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9855 -
accuracy: 0.6947 - val_loss: 1.0688 - val_accuracy: 0.6884
Epoch 121/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1032 -
accuracy: 0.6651 - val_loss: 1.0533 - val_accuracy: 0.6897
Epoch 122/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9531 -
accuracy: 0.7033 - val_loss: 0.9863 - val_accuracy: 0.7060
Epoch 123/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8925 -
accuracy: 0.7189 - val_loss: 1.0245 - val_accuracy: 0.7017
Epoch 124/200
48/48 [==============================] - 0s 8ms/step - loss: 0.9675 -
accuracy: 0.7017 - val_loss: 0.9953 - val_accuracy: 0.7100
Epoch 125/200
48/48 [==============================] - 0s 8ms/step - loss: 0.8920 -
accuracy: 0.7228 - val_loss: 1.0551 - val_accuracy: 0.6884
Epoch 126/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9788 -
accuracy: 0.6970 - val_loss: 1.0004 - val_accuracy: 0.7020
Epoch 127/200
48/48 [==============================] - 0s 8ms/step - loss: 0.9488 -
accuracy: 0.7086 - val_loss: 0.9809 - val_accuracy: 0.7080
Epoch 128/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9472 -
accuracy: 0.7010 - val_loss: 0.9878 - val_accuracy: 0.7076
Epoch 129/200
```

```
48/48 [==============================] - 0s 8ms/step - loss: 0.8796 -
accuracy: 0.7250 - val_loss: 0.9951 - val_accuracy: 0.7086
Epoch 130/200
48/48 [==============================] - 0s 8ms/step - loss: 0.8827 -
accuracy: 0.7197 - val_loss: 1.0295 - val_accuracy: 0.7066
Epoch 131/200
48/48 [==============================] - 1s 11ms/step - loss: 0.9567 -
accuracy: 0.7017 - val_loss: 0.9951 - val_accuracy: 0.7100
Epoch 132/200
48/48 [==============================] - 1s 11ms/step - loss: 0.9210 -
accuracy: 0.7166 - val_loss: 1.0183 - val_accuracy: 0.7007
Epoch 133/200
48/48 [==============================] - 1s 12ms/step - loss: 0.9392 -
accuracy: 0.7065 - val_loss: 1.0517 - val_accuracy: 0.6897
Epoch 134/200
48/48 [==============================] - 1s 11ms/step - loss: 1.0084 -
accuracy: 0.6944 - val_loss: 1.0410 - val_accuracy: 0.6997
Epoch 135/200
48/48 [==============================] - 1s 12ms/step - loss: 0.8984 -
accuracy: 0.7178 - val_loss: 0.9997 - val_accuracy: 0.7033
Epoch 136/200
48/48 [==============================] - 1s 12ms/step - loss: 0.8582 -
accuracy: 0.7336 - val_loss: 1.0600 - val_accuracy: 0.6940
Epoch 137/200
48/48 [==============================] - 1s 13ms/step - loss: 0.9393 -
accuracy: 0.7108 - val_loss: 0.9978 - val_accuracy: 0.7110
Epoch 138/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8585 -
accuracy: 0.7325 - val_loss: 1.0198 - val_accuracy: 0.6983
Epoch 139/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9562 -
accuracy: 0.7020 - val_loss: 1.0475 - val_accuracy: 0.6983
Epoch 140/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9684 -
accuracy: 0.7011 - val_loss: 1.0443 - val_accuracy: 0.6970
Epoch 141/200
48/48 [==============================] - 0s 8ms/step - loss: 0.9212 -
accuracy: 0.7131 - val_loss: 1.0290 - val_accuracy: 0.7090
Epoch 142/200
48/48 [==============================] - 0s 8ms/step - loss: 0.8552 -
accuracy: 0.7312 - val_loss: 1.0331 - val_accuracy: 0.7027
Epoch 143/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0760 -
accuracy: 0.6758 - val_loss: 0.9867 - val_accuracy: 0.7106
Epoch 144/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8578 -
accuracy: 0.7321 - val_loss: 1.0250 - val_accuracy: 0.7050
Epoch 145/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8454 -
accuracy: 0.7361 - val_loss: 1.0234 - val_accuracy: 0.7126
Epoch 146/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9084 -
accuracy: 0.7170 - val_loss: 1.0342 - val_accuracy: 0.7023
Epoch 147/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9312 -
accuracy: 0.7121 - val_loss: 1.0529 - val_accuracy: 0.7027
Epoch 148/200
```

```
48/48 [==============================] - 0s 9ms/step - loss: 0.9308 -
accuracy: 0.7140 - val_loss: 1.0121 - val_accuracy: 0.7086
Epoch 149/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8191 -
accuracy: 0.7417 - val_loss: 1.0137 - val_accuracy: 0.7073
Epoch 150/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9367 -
accuracy: 0.7106 - val_loss: 1.0126 - val_accuracy: 0.7063
Epoch 151/200
48/48 [==============================] - 0s 8ms/step - loss: 0.9216 -
accuracy: 0.7094 - val_loss: 1.0407 - val_accuracy: 0.6977
Epoch 152/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9813 -
accuracy: 0.6956 - val_loss: 1.0213 - val_accuracy: 0.7050
Epoch 153/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8729 -
accuracy: 0.7247 - val_loss: 1.0149 - val_accuracy: 0.7106
Epoch 154/200
48/48 [==============================] - 0s 9ms/step - loss: 1.0009 -
accuracy: 0.6951 - val_loss: 1.0040 - val_accuracy: 0.7116
Epoch 155/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8636 -
accuracy: 0.7309 - val_loss: 1.0081 - val_accuracy: 0.7206
Epoch 174/200
48/48 [==============================] - 0s 9ms/step - loss: 0.7826 -
accuracy: 0.7498 - val_loss: 1.0577 - val_accuracy: 0.7086
Epoch 175/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8484 -
accuracy: 0.7347 - val_loss: 1.0087 - val_accuracy: 0.7120
Epoch 176/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8560 -
accuracy: 0.7325 - val_loss: 1.0037 - val_accuracy: 0.7076
Epoch 177/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8276 -
accuracy: 0.7413 - val_loss: 1.0137 - val_accuracy: 0.7150
Epoch 178/200
48/48 [==============================] - 0s 9ms/step - loss: 0.7957 -
accuracy: 0.7487 - val_loss: 1.0374 - val_accuracy: 0.7073
Epoch 179/200
48/48 [==============================] - 0s 8ms/step - loss: 0.8094 -
accuracy: 0.7426 - val_loss: 1.0204 - val_accuracy: 0.7096
Epoch 180/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8316 -
accuracy: 0.7332 - val_loss: 1.0169 - val_accuracy: 0.7156
Epoch 181/200
48/48 [==============================] - 0s 9ms/step - loss: 0.8265 -
accuracy: 0.7383 - val_loss: 1.0242 - val_accuracy: 0.7146
Epoch 182/200
48/48 [==============================] - 0s 8ms/step - loss: 0.8243 -
accuracy: 0.7351 - val_loss: 1.0133 - val_accuracy: 0.7140
Epoch 183/200
48/48 [==============================] - 0s 9ms/step - loss: 0.7326 -
accuracy: 0.7590 - val_loss: 1.0442 - val_accuracy: 0.7143
Epoch 184/200
48/48 [==============================] - 0s 9ms/step - loss: 0.7998 -
accuracy: 0.7469 - val_loss: 1.0275 - val_accuracy: 0.7143
Epoch 185/200
```

```
48/48 [==============================] - 0s 8ms/step - loss: 0.7857 -
accuracy: 0.7528 - val_loss: 1.0049 - val_accuracy: 0.7146
Epoch 186/200
48/48 [==============================] - 0s 8ms/step - loss: 0.7696 -
accuracy: 0.7592 - val_loss: 1.0795 - val_accuracy: 0.7017
Epoch 187/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9203 -
accuracy: 0.7211 - val_loss: 1.0382 - val_accuracy: 0.7106
Epoch 188/200
48/48 [==============================] - 0s 8ms/step - loss: 0.8357 -
accuracy: 0.7389 - val_loss: 1.0871 - val_accuracy: 0.6977
Epoch 189/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9327 -
accuracy: 0.7182 - val_loss: 1.0145 - val_accuracy: 0.7153
Epoch 190/200
48/48 [==============================] - 0s 9ms/step - loss: 0.7819 -
accuracy: 0.7528 - val_loss: 1.0260 - val_accuracy: 0.7116
Epoch 191/200
48/48 [==============================] - 0s 9ms/step - loss: 0.7739 -
accuracy: 0.7528 - val_loss: 1.0143 - val_accuracy: 0.7110
Epoch 192/200
48/48 [==============================] - 1s 11ms/step - loss: 0.7520 -
accuracy: 0.7568 - val_loss: 1.0899 - val_accuracy: 0.6997
Epoch 193/200
48/48 [==============================] - 1s 12ms/step - loss: 0.9964 -
accuracy: 0.7019 - val_loss: 1.0538 - val_accuracy: 0.7010
Epoch 194/200
48/48 [==============================] - 1s 11ms/step - loss: 0.9200 -
accuracy: 0.7140 - val_loss: 1.0216 - val_accuracy: 0.7066
Epoch 195/200
48/48 [==============================] - 1s 11ms/step - loss: 0.8230 -
accuracy: 0.7396 - val_loss: 1.0111 - val_accuracy: 0.7130
Epoch 196/200
48/48 [==============================] - 1s 12ms/step - loss: 0.7675 -
accuracy: 0.7552 - val_loss: 1.0206 - val_accuracy: 0.7163
Epoch 197/200
48/48 [==============================] - 1s 11ms/step - loss: 0.7695 -
accuracy: 0.7569 - val_loss: 1.0705 - val_accuracy: 0.7096
Epoch 198/200
48/48 [==============================] - 1s 12ms/step - loss: 0.9095 -
accuracy: 0.7181 - val_loss: 1.0019 - val_accuracy: 0.7176
Epoch 199/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8353 -
accuracy: 0.7377 - val_loss: 1.0001 - val_accuracy: 0.7110
Epoch 200/200
48/48 [==============================] - 0s 9ms/step - loss: 0.7965 -
accuracy: 0.7452 - val_loss: 1.0280 - val_accuracy: 0.7120
----------------------------------------------------------------------
--
377/377 [==============================] - 1s 3ms/step - loss: 0.6131 -
accuracy: 0.8049
Train Loss = 0.6130610108375549
Train Accuracy = 0.8048517107963562
95/95 [==============================] - 0s 5ms/step - loss: 0.6282 -
accuracy: 0.8073
Test Loss = 0.6281698942184448
Test Accuracy = 0.8073089718818665
```



TWO-STREAM CNN

```
def Mario_CNN(input_shape, classes):
    X_input = Input(input_shape)

    # Mario Brothers Split
    X1, X2 = tf.split(X_input, 2, 2)

    # Architecture 1
    X1 = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name='A1_conv1')(X1)
    X1 = Activation('relu', name='A1_relu1')(X1)

    X1 = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name='A1_conv2')(X1)
    X1 = Activation('relu', name='A1_relu2')(X1)
    X1 = MaxPooling2D((1,2), strides=(2,2), name='A1_pool1')(X1)

    X1 = Conv2D(filters=64, kernel_size=(5,5),
strides=(1,1),padding='same', name='A1_conv3')(X1)
    X1 = Activation('relu', name='A1_relu3')(X1)
    X1 = MaxPooling2D((1,2), strides=(2,2), name='A1_pool2')(X1)
```

```
    X1 = Conv2D(filters=64, kernel_size=(5,1),
strides=(1,1),padding='same', name='A1_conv4')(X1)
    X1 = Activation('relu', name='A1_relu4')(X1)

    X1 = Conv2D(filters=64, kernel_size=(1,1),
strides=(1,1),padding='same', name='A1_conv5')(X1)

    # Architecture 2
    X2 = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name='A2_conv1')(X2)
    X2 = Activation('relu', name='A2_relu1')(X2)

    X2 = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name='A2_conv2')(X2)
    X2 = Activation('relu', name='A2_relu2')(X2)
    X2 = MaxPooling2D((1,2), strides=(2,2), name='A2_pool1')(X2)

    X2 = Conv2D(filters=64, kernel_size=(5,5),
strides=(1,1),padding='same', name='A2_conv3')(X2)
    X2 = Activation('relu', name='A2_relu3')(X2)
    X2 = MaxPooling2D((1,2), strides=(2,2), name='A2_pool2')(X2)

    X2 = Conv2D(filters=64, kernel_size=(5,1),
strides=(1,1),padding='same', name='A2_conv4')(X2)
    X2 = Activation('relu', name='A2_relu4')(X2)

    X2 = Conv2D(filters=64, kernel_size=(1,1),
strides=(1,1),padding='same', name='A2_conv5')(X2)

    # Mario Brothers Reunion
    X = tf.keras.layers.Concatenate(axis=2)([X1, X2])

    X = ZeroPadding2D((0,1))(X)
    X = Flatten(name='flatten')(X)
    X = Dropout(0.5)(X)
    X = BatchNormalization(momentum=0.9)(X)
    X = Dense(128, activation='relu',name='fc1')(X)
    X = Dropout(0.5)(X)
    X = Dense(classes, activation='softmax', name='softmax')(X)

    model = Model(inputs=X_input, outputs=X, name='MarioCNN')

    return model
model = Mario_CNN(input_shape = (12, 10, 1), classes = 52)
model.summary()
```

```
Model: "MarioCNN"
__________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #
Connected to
==========================================================================================
 input_2 (InputLayer)        [(None, 12, 10, 1)]          0         []

 tf.split (TFOpLambda)       [(None, 12, 5, 1),           0
['input_2[0][0]']
                              (None, 12, 5, 1)]

 A1_conv1 (Conv2D)           (None, 12, 5, 32)            320
['tf.split[0][0]']

 A2_conv1 (Conv2D)           (None, 12, 5, 32)            320
['tf.split[0][1]']

 A1_relu1 (Activation)        (None, 12, 5, 32)           0
['A1_conv1[0][0]']

 A2_relu1 (Activation)        (None, 12, 5, 32)           0
['A2_conv1[0][0]']

 A1_conv2 (Conv2D)           (None, 12, 5, 32)            9248
['A1_relu1[0][0]']

 A2_conv2 (Conv2D)           (None, 12, 5, 32)            9248
['A2_relu1[0][0]']

 A1_relu2 (Activation)        (None, 12, 5, 32)           0
['A1_conv2[0][0]']

 A2_relu2 (Activation)        (None, 12, 5, 32)           0
['A2_conv2[0][0]']

 A1_pool1 (MaxPooling2D)     (None, 6, 2, 32)             0
['A1_relu2[0][0]']

 A2_pool1 (MaxPooling2D)     (None, 6, 2, 32)             0
['A2_relu2[0][0]']

 A1_conv3 (Conv2D)           (None, 6, 2, 64)             51264
['A1_pool1[0][0]']

 A2_conv3 (Conv2D)           (None, 6, 2, 64)             51264
['A2_pool1[0][0]']

 A1_relu3 (Activation)        (None, 6, 2, 64)            0
['A1_conv3[0][0]']
                                          A2_relu3 (Activation)
(None, 6, 2, 64)             0           ['A2_conv3[0][0]']

 A1_pool2 (MaxPooling2D)     (None, 3, 1, 64)             0
['A1_relu3[0][0]']
```

```
 A2_pool2 (MaxPooling2D)     (None, 3, 1, 64)             0
['A2_relu3[0][0]']

 A1_conv4 (Conv2D)           (None, 3, 1, 64)             20544
['A1_pool2[0][0]']

 A2_conv4 (Conv2D)           (None, 3, 1, 64)             20544
['A2_pool2[0][0]']

 A1_relu4 (Activation)        (None, 3, 1, 64)             0
['A1_conv4[0][0]']

 A2_relu4 (Activation)        (None, 3, 1, 64)             0
['A2_conv4[0][0]']

 A1_conv5 (Conv2D)           (None, 3, 1, 64)             4160
['A1_relu4[0][0]']

 A2_conv5 (Conv2D)           (None, 3, 1, 64)             4160
['A2_relu4[0][0]']

 concatenate (Concatenate)   (None, 3, 2, 64)             0
['A1_conv5[0][0]',

'A2_conv5[0][0]']

 zero_padding2d_1 (ZeroPadd  (None, 3, 4, 64)             0
['concatenate[0][0]']
 ing2D)

 flatten (Flatten)           (None, 768)                  0
['zero_padding2d_1[0][0]']

 dropout_2 (Dropout)         (None, 768)                  0
['flatten[0][0]']

 batch_normalization_1 (Bat  (None, 768)                  3072
['dropout_2[0][0]']
 chNormalization)

 fc1 (Dense)                 (None, 128)                  98432
['batch_normalization_1[0][0]'
                                                                  ]

 dropout_3 (Dropout)         (None, 128)                  0
['fc1[0][0]']
softmax (Dense)             (None, 52)                   6708
['dropout_3[0][0]']

==================================================================================
Total params: 279284 (1.07 MB)
Trainable params: 277748 (1.06 MB)
Non-trainable params: 1536 (6.00 KB)
```

```
_____model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
history = LossHistory()

model.fit(data, label, validation_split=0.2, epochs=eps,
batch_size=256, verbose=1, callbacks=[history])

preds_train = model.evaluate(X_train, Y_train)
print("Train Loss = " + str(preds_train[0]))
print("Train Accuracy = " + str(preds_train[1]))

preds_test  = model.evaluate(X_test, Y_test)
print("Test Loss = " + str(preds_test[0]))
print("Test Accuracy = " + str(preds_test[1]))
```

```
Epoch 1/200
48/48 [==============================] - 9s 22ms/step - loss: 3.6887 -
accuracy: 0.0808 - val_loss: 3.4378 - val_accuracy: 0.2216
Epoch 2/200
48/48 [==============================] - 1s 10ms/step - loss: 3.1135 -
accuracy: 0.1877 - val_loss: 2.6032 - val_accuracy: 0.3262
Epoch 3/200
48/48 [==============================] - 0s 10ms/step - loss: 2.7115 -
accuracy: 0.2801 - val_loss: 2.2404 - val_accuracy: 0.4110
Epoch 4/200
48/48 [==============================] - 0s 10ms/step - loss: 2.4768 -
accuracy: 0.3346 - val_loss: 2.0589 - val_accuracy: 0.4395
Epoch 5/200
48/48 [==============================] - 0s 9ms/step - loss: 2.2660 -
accuracy: 0.3808 - val_loss: 1.9073 - val_accuracy: 0.4897
Epoch 6/200
48/48 [==============================] - 0s 10ms/step - loss: 2.1403 -
accuracy: 0.4045 - val_loss: 1.7509 - val_accuracy: 0.5259
Epoch 7/200
48/48 [==============================] - 0s 10ms/step - loss: 2.0146 -
accuracy: 0.4307 - val_loss: 1.7139 - val_accuracy: 0.5239
Epoch 8/200
48/48 [==============================] - 0s 10ms/step - loss: 2.0149 -
accuracy: 0.4288 - val_loss: 1.6165 - val_accuracy: 0.5502
Epoch 9/200
48/48 [==============================] - 0s 10ms/step - loss: 1.8644 -
accuracy: 0.4628 - val_loss: 1.5206 - val_accuracy: 0.5807
Epoch 10/200
48/48 [==============================] - 0s 10ms/step - loss: 1.7984 -
accuracy: 0.4815 - val_loss: 1.5057 - val_accuracy: 0.5674
Epoch 11/200
48/48 [==============================] - 0s 10ms/step - loss: 1.7457 -
accuracy: 0.4923 - val_loss: 1.4245 - val_accuracy: 0.5927
Epoch 12/200
48/48 [==============================] - 0s 10ms/step - loss: 1.6677 -
accuracy: 0.5121 - val_loss: 1.4270 - val_accuracy: 0.5914
Epoch 13/200
48/48 [==============================] - 0s 10ms/step - loss: 1.6169 -
accuracy: 0.5225 - val_loss: 1.3635 - val_accuracy: 0.6156
Epoch 14/200
48/48 [==============================] - 0s 10ms/step - loss: 1.5616 -
accuracy: 0.5363 - val_loss: 1.2972 - val_accuracy: 0.6319
Epoch 15/200
```

```
48/48 [==============================] - 0s 10ms/step - loss: 1.5222 -
accuracy: 0.5515 - val_loss: 1.2735 - val_accuracy: 0.6432
Epoch 16/200
48/48 [==============================] - 0s 10ms/step - loss: 1.5252 -
accuracy: 0.5476 - val_loss: 1.3188 - val_accuracy: 0.6189
Epoch 17/200
48/48 [==============================] - 0s 10ms/step - loss: 1.5413 -
accuracy: 0.5410 - val_loss: 1.2519 - val_accuracy: 0.6389
Epoch 18/200
48/48 [==============================] - 0s 9ms/step - loss: 1.4539 -
accuracy: 0.5644 - val_loss: 1.2353 - val_accuracy: 0.6449
Epoch 19/200
48/48 [==============================] - 1s 13ms/step - loss: 1.4378 -
accuracy: 0.5719 - val_loss: 1.1955 - val_accuracy: 0.6488
Epoch 20/200
48/48 [==============================] - 1s 12ms/step - loss: 1.4406 -
accuracy: 0.5713 - val_loss: 1.1868 - val_accuracy: 0.6542
Epoch 21/200
48/48 [==============================] - 1s 13ms/step - loss: 1.3940 -
accuracy: 0.5819 - val_loss: 1.1795 - val_accuracy: 0.6535
Epoch 22/200
48/48 [==============================] - 1s 15ms/step - loss: 1.3691 -
accuracy: 0.5928 - val_loss: 1.1664 - val_accuracy: 0.6565
Epoch 23/200
48/48 [==============================] - 1s 13ms/step - loss: 1.3755 -
accuracy: 0.5898 - val_loss: 1.1443 - val_accuracy: 0.6601
Epoch 24/200
48/48 [==============================] - 1s 13ms/step - loss: 1.3285 -
accuracy: 0.5988 - val_loss: 1.1421 - val_accuracy: 0.6791
Epoch 25/200
48/48 [==============================] - 1s 13ms/step - loss: 1.3756 -
accuracy: 0.5845 - val_loss: 1.1326 - val_accuracy: 0.6601
Epoch 26/200
48/48 [==============================] - 0s 10ms/step - loss: 1.3003 -
accuracy: 0.6047 - val_loss: 1.1460 - val_accuracy: 0.6621
Epoch 27/200
48/48 [==============================] - 0s 10ms/step - loss: 1.3637 -
accuracy: 0.5898 - val_loss: 1.1029 - val_accuracy: 0.6711
Epoch 28/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2477 -
accuracy: 0.6183 - val_loss: 1.0928 - val_accuracy: 0.6648
Epoch 29/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2602 -
accuracy: 0.6148 - val_loss: 1.0971 - val_accuracy: 0.6777
Epoch 30/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2318 -
accuracy: 0.6240 - val_loss: 1.0616 - val_accuracy: 0.6847
Epoch 31/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2424 -
accuracy: 0.6216 - val_loss: 1.0580 - val_accuracy: 0.6844
Epoch 32/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2035 -
accuracy: 0.6309 - val_loss: 1.0782 - val_accuracy: 0.6807
Epoch 33/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2113 -
accuracy: 0.6325 - val_loss: 1.0477 - val_accuracy: 0.6917
Epoch 34/200
```

```
48/48 [==============================] - 1s 11ms/step - loss: 1.1820 -
accuracy: 0.6396 - val_loss: 1.0782 - val_accuracy: 0.6811
Epoch 35/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2267 -
accuracy: 0.6253 - val_loss: 1.0555 - val_accuracy: 0.6804
Epoch 36/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2334 -
accuracy: 0.6261 - val_loss: 1.0727 - val_accuracy: 0.6887
Epoch 37/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2498 -
accuracy: 0.6159 - val_loss: 1.0383 - val_accuracy: 0.6920
Epoch 38/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2815 -
accuracy: 0.6128 - val_loss: 1.0571 - val_accuracy: 0.6841
Epoch 39/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1727 -
accuracy: 0.6397 - val_loss: 1.0486 - val_accuracy: 0.6993
Epoch 40/200
48/48 [==============================] - 0s 10ms/step - loss: 1.2200 -
accuracy: 0.6304 - val_loss: 1.0580 - val_accuracy: 0.6767
Epoch 41/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1616 -
accuracy: 0.6423 - val_loss: 1.0291 - val_accuracy: 0.6990
Epoch 42/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1324 -
accuracy: 0.6527 - val_loss: 0.9967 - val_accuracy: 0.7033
Epoch 43/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1106 -
accuracy: 0.6580 - val_loss: 1.0016 - val_accuracy: 0.7113
Epoch 44/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1057 -
accuracy: 0.6580 - val_loss: 0.9957 - val_accuracy: 0.7090
Epoch 45/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1427 -
accuracy: 0.6463 - val_loss: 1.0196 - val_accuracy: 0.7013
Epoch 46/200
48/48 [==============================] - 1s 14ms/step - loss: 1.0892 -
accuracy: 0.6630 - val_loss: 1.0048 - val_accuracy: 0.7066
Epoch 47/200
48/48 [==============================] - 1s 13ms/step - loss: 1.0937 -
accuracy: 0.6594 - val_loss: 1.0296 - val_accuracy: 0.6880
Epoch 48/200
48/48 [==============================] - 1s 14ms/step - loss: 1.1321 -
accuracy: 0.6534 - val_loss: 0.9920 - val_accuracy: 0.7040
Epoch 49/200
48/48 [==============================] - 1s 16ms/step - loss: 1.0290 -
accuracy: 0.6777 - val_loss: 1.0003 - val_accuracy: 0.7013
Epoch 50/200
48/48 [==============================] - 1s 15ms/step - loss: 1.0927 -
accuracy: 0.6618 - val_loss: 0.9901 - val_accuracy: 0.7040
Epoch 51/200
48/48 [==============================] - 1s 14ms/step - loss: 1.0807 -
accuracy: 0.6657 - val_loss: 0.9834 - val_accuracy: 0.7023
Epoch 52/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1077 -
accuracy: 0.6546 - val_loss: 0.9896 - val_accuracy: 0.7060
Epoch 53/200
```

```
48/48 [==============================] - 0s 10ms/step - loss: 1.0898 -
accuracy: 0.6608 - val_loss: 0.9876 - val_accuracy: 0.7050
Epoch 54/200
48/48 [==============================] - 0s 10ms/step - loss: 1.0711 -
accuracy: 0.6662 - val_loss: 0.9739 - val_accuracy: 0.7070
Epoch 55/200
48/48 [==============================] - 0s 10ms/step - loss: 1.0554 -
accuracy: 0.6658 - val_loss: 0.9942 - val_accuracy: 0.7020
Epoch 56/200
48/48 [==============================] - 0s 9ms/step - loss: 1.1327 -
accuracy: 0.6515 - val_loss: 1.0154 - val_accuracy: 0.6917
Epoch 57/200
48/48 [==============================] - 0s 10ms/step - loss: 1.1185 -
accuracy: 0.6518 - val_loss: 0.9620 - val_accuracy: 0.7086
Epoch 58/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9873 -
accuracy: 0.6892 - val_loss: 0.9760 - val_accuracy: 0.7070
Epoch 59/200
48/48 [==============================] - 0s 10ms/step - loss: 1.0146 -
accuracy: 0.6795 - val_loss: 0.9482 - val_accuracy: 0.7150
Epoch 60/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9967 -
accuracy: 0.6850 - val_loss: 0.9348 - val_accuracy: 0.7213
Epoch 61/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9420 -
accuracy: 0.7002 - val_loss: 0.9350 - val_accuracy: 0.7213
Epoch 62/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9403 -
accuracy: 0.7035 - val_loss: 0.9408 - val_accuracy: 0.7243
Epoch 63/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9185 -
accuracy: 0.7097 - val_loss: 0.9815 - val_accuracy: 0.71
Epoch 64/200
48/48 [==============================] - 0s 10ms/step - loss: 1.0444 -
accuracy: 0.6738 - val_loss: 0.9562 - val_accuracy: 0.7166
Epoch 65/200
48/48 [==============================] - 0s 10ms/step - loss: 1.0335 -
accuracy: 0.6748 - val_loss: 0.9798 - val_accuracy: 0.7163
Epoch 66/200
48/48 [==============================] - 0s 10ms/step - loss: 1.0285 -
accuracy: 0.6762 - val_loss: 0.9751 - val_accuracy: 0.7100
Epoch 67/200
48/48 [==============================] - 0s 10ms/step - loss: 1.0509 -
accuracy: 0.6703 - val_loss: 0.9539 - val_accuracy: 0.7186
Epoch 68/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9573 -
accuracy: 0.6924 - val_loss: 0.9413 - val_accuracy: 0.7140
Epoch 69/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9467 -
accuracy: 0.7021 - val_loss: 0.9307 - val_accuracy: 0.7249
Epoch 70/200
48/48 [==============================] - 0s 9ms/step - loss: 0.9035 -
accuracy: 0.7086 - val_loss: 0.9749 - val_accuracy: 0.7166
Epoch 71/200
48/48 [==============================] - 0s 10ms/step - loss: 1.0191 -
accuracy: 0.6788 - val_loss: 0.9467 - val_accuracy: 0.7169
Epoch 72/200
```

```
48/48 [==============================] - 1s 12ms/step - loss: 0.9477 -
accuracy: 0.7017 - val_loss: 0.9605 - val_accuracy: 0.7130
Epoch 73/200
48/48 [==============================] - 1s 14ms/step - loss: 0.9007 -
accuracy: 0.7135 - val_loss: 0.9461 - val_accuracy: 0.7209
Epoch 74/200
48/48 [==============================] - 1s 13ms/step - loss: 1.0585 -
accuracy: 0.6703 - val_loss: 0.9670 - val_accuracy: 0.7113
Epoch 75/200
48/48 [==============================] - 1s 13ms/step - loss: 1.0215 -
accuracy: 0.6770 - val_loss: 0.9540 - val_accuracy: 0.7136
Epoch 76/200
48/48 [==============================] - 1s 13ms/step - loss: 0.9895 -
accuracy: 0.6829 - val_loss: 0.9518 - val_accuracy: 0.7213
Epoch 77/200
48/48 [==============================] - 1s 13ms/step - loss: 0.9138 -
accuracy: 0.7058 - val_loss: 0.9429 - val_accuracy: 0.7246
Epoch 78/200
48/48 [==============================] - 1s 13ms/step - loss: 0.9879 -
accuracy: 0.6879 - val_loss: 0.9563 - val_accuracy: 0.7209
Epoch 79/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8851 -
accuracy: 0.7165 - val_loss: 0.9515 - val_accuracy: 0.7176
Epoch 80/200
48/48 [==============================] - 0s 10ms/step - loss: 1.0296 -
accuracy: 0.6782 - val_loss: 0.9638 - val_accuracy: 0.7146
Epoch 81/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8760 -
accuracy: 0.7144 - val_loss: 0.9368 - val_accuracy: 0.7203
Epoch 82/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9633 -
accuracy: 0.6969 - val_loss: 0.9569 - val_accuracy: 0.7193
Epoch 83/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9151 -
accuracy: 0.7101 - val_loss: 0.9369 - val_accuracy: 0.7266
Epoch 84/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8776 -
accuracy: 0.7153 - val_loss: 0.9295 - val_accuracy: 0.7256
Epoch 85/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8423 -
accuracy: 0.7272 - val_loss: 0.9455 - val_accuracy: 0.7236
Epoch 86/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9699 -
accuracy: 0.7013 - val_loss: 0.9521 - val_accuracy: 0.7173
Epoch 87/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8877 -
accuracy: 0.7152 - val_loss: 0.9450 - val_accuracy: 0.7196
Epoch 88/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9732 -
accuracy: 0.6919 - val_loss: 0.9439 - val_accuracy: 0.7103
Epoch 89/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8517 -
accuracy: 0.7250 - val_loss: 0.9557 - val_accuracy: 0.7140
Epoch 90/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9303 -
accuracy: 0.7086 - val_loss: 0.9477 - val_accuracy: 0.7193
Epoch 91/200
```

```
48/48 [==============================] - 0s 10ms/step - loss: 0.9039 -
accuracy: 0.7069 - val_loss: 0.9450 - val_accuracy: 0.7262
Epoch 92/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9729 -
accuracy: 0.6940 - val_loss: 0.9599 - val_accuracy: 0.7143
Epoch 93/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8729 -
accuracy: 0.7147 - val_loss: 0.9156 - val_accuracy: 0.7233
Epoch 94/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8032 -
accuracy: 0.7356 - val_loss: 0.9148 - val_accuracy: 0.7289
Epoch 95/200
48/48 [==============================] - 1s 10ms/step - loss: 0.8085 -
accuracy: 0.7350 - val_loss: 0.9372 - val_accuracy: 0.7246
Epoch 96/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8878 -
accuracy: 0.7178 - val_loss: 0.9313 - val_accuracy: 0.7282
Epoch 97/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8352 -
accuracy: 0.7290 - val_loss: 0.9397 - val_accuracy: 0.7243
Epoch 98/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8274 -
accuracy: 0.7326 - val_loss: 0.9337 - val_accuracy: 0.7186
Epoch 99/200
48/48 [==============================] - 1s 11ms/step - loss: 0.8433 -
accuracy: 0.7279 - val_loss: 0.9505 - val_accuracy: 0.7203
Epoch 118/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8489 -
accuracy: 0.7278 - val_loss: 0.9390 - val_accuracy: 0.7342
Epoch 119/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7649 -
accuracy: 0.7497 - val_loss: 0.9616 - val_accuracy: 0.7216
Epoch 120/200
48/48 [==============================] - 0s 10ms/step - loss: 0.9693 -
accuracy: 0.6969 - val_loss: 0.9403 - val_accuracy: 0.7176
Epoch 121/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7701 -
accuracy: 0.7509 - val_loss: 0.9334 - val_accuracy: 0.7256
Epoch 122/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7663 -
accuracy: 0.7462 - val_loss: 0.9385 - val_accuracy: 0.7249
Epoch 123/200
48/48 [==============================] - 1s 10ms/step - loss: 0.7988 -
accuracy: 0.7401 - val_loss: 0.9256 - val_accuracy: 0.7336
Epoch 124/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7096 -
accuracy: 0.7664 - val_loss: 0.9435 - val_accuracy: 0.7336
Epoch 125/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8600 -
accuracy: 0.7240 - val_loss: 0.9484 - val_accuracy: 0.7256
Epoch 126/200
48/48 [==============================] - 1s 11ms/step - loss: 0.9166 -
accuracy: 0.7072 - val_loss: 0.9241 - val_accuracy: 0.7306
Epoch 127/200
48/48 [==============================] - 1s 13ms/step - loss: 0.7618 -
accuracy: 0.7553 - val_loss: 0.9614 - val_accuracy: 0.7282
Epoch 128/200
```

```
48/48 [==============================] - 1s 13ms/step - loss: 0.8048 -
accuracy: 0.7396 - val_loss: 0.9544 - val_accuracy: 0.7143
Epoch 129/200
48/48 [==============================] - 1s 12ms/step - loss: 0.8353 -
accuracy: 0.7345 - val_loss: 0.9214 - val_accuracy: 0.7352
Epoch 130/200
48/48 [==============================] - 1s 13ms/step - loss: 0.7940 -
accuracy: 0.7410 - val_loss: 0.9445 - val_accuracy: 0.7252
Epoch 131/200
48/48 [==============================] - 1s 13ms/step - loss: 0.7855 -
accuracy: 0.7485 - val_loss: 0.9445 - val_accuracy: 0.7289
Epoch 132/200
48/48 [==============================] - 1s 13ms/step - loss: 0.7268 -
accuracy: 0.7592 - val_loss: 0.9471 - val_accuracy: 0.7259
Epoch 133/200
48/48 [==============================] - 1s 12ms/step - loss: 0.7252 -
accuracy: 0.7572 - val_loss: 0.9364 - val_accuracy: 0.7309
Epoch 134/200
48/48 [==============================] - 1s 10ms/step - loss: 0.6841 -
accuracy: 0.7735 - val_loss: 0.9680 - val_accuracy: 0.7332
Epoch 135/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7878 -
accuracy: 0.7464 - val_loss: 0.9435 - val_accuracy: 0.7319
Epoch 136/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7366 -
accuracy: 0.7594 - val_loss: 0.9493 - val_accuracy: 0.7299
Epoch 137/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7039 -
accuracy: 0.7689 - val_loss: 0.9685 - val_accuracy: 0.7292
Epoch 138/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8043 -
accuracy: 0.7464 - val_loss: 0.9664 - val_accuracy: 0.7336
Epoch 139/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7239 -
accuracy: 0.7653 - val_loss: 0.9595 - val_accuracy: 0.7349
Epoch 140/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8058 -
accuracy: 0.7396 - val_loss: 0.9389 - val_accuracy: 0.7369
Epoch 141/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7398 -
accuracy: 0.7601 - val_loss: 0.9634 - val_accuracy: 0.7342
Epoch 142/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6877 -
accuracy: 0.7723 - val_loss: 0.9869 - val_accuracy: 0.7269
Epoch 143/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8046 -
accuracy: 0.7397 - val_loss: 0.9771 - val_accuracy: 0.7216
Epoch 144/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7891 -
accuracy: 0.7459 - val_loss: 0.9595 - val_accuracy: 0.7286
Epoch 145/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7859 -
accuracy: 0.7459 - val_loss: 0.9641 - val_accuracy: 0.7332
Epoch 146/200
48/48 [==============================] - 1s 10ms/step - loss: 0.8409 -
accuracy: 0.7282 - val_loss: 0.9521 - val_accuracy: 0.7289
Epoch 147/200
```

```
48/48 [==============================] - 0s 10ms/step - loss: 0.7062 -
accuracy: 0.7687 - val_loss: 0.9619 - val_accuracy: 0.7326
Epoch 148/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7556 -
accuracy: 0.7549 - val_loss: 0.9882 - val_accuracy: 0.7266
Epoch 149/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7996 -
accuracy: 0.7477 - val_loss: 0.9621 - val_accuracy: 0.7256
Epoch 150/200
48/48 [==============================] - 0s 10ms/step - loss: 0.8592 -
accuracy: 0.7234 - val_loss: 0.9632 - val_accuracy: 0.7229
Epoch 151/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7608 -
accuracy: 0.7443 - val_loss: 0.9661 - val_accuracy: 0.7269
Epoch 152/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6865 -
accuracy: 0.7710 - val_loss: 0.9850 - val_accuracy: 0.7239
Epoch 153/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6815 -
accuracy: 0.7748 - val_loss: 0.9767 - val_accuracy: 0.7326
accuracy: 0.7437 - val_loss: 0.9932 - val_accuracy: 0.7312
Epoch 159/200
48/48 [==============================] - 1s 14ms/step - loss: 0.6705 -
accuracy: 0.7800 - val_loss: 1.0220 - val_accuracy: 0.7259
Epoch 160/200
48/48 [==============================] - 1s 13ms/step - loss: 0.8632 -
accuracy: 0.7226 - val_loss: 0.9922 - val_accuracy: 0.7223
Epoch 161/200
48/48 [==============================] - 1s 11ms/step - loss: 0.7958 -
accuracy: 0.7451 - val_loss: 0.9923 - val_accuracy: 0.7193
Epoch 162/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6953 -
accuracy: 0.7725 - val_loss: 0.9508 - val_accuracy: 0.7336
Epoch 163/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7078 -
accuracy: 0.7651 - val_loss: 0.9914 - val_accuracy: 0.7249
Epoch 164/200
48/48 [==============================] - 0s 9ms/step - loss: 0.7439 -
accuracy: 0.7553 - val_loss: 0.9763 - val_accuracy: 0.7276
Epoch 165/200
48/48 [==============================] - 1s 11ms/step - loss: 0.7623 -
accuracy: 0.7499 - val_loss: 1.0140 - val_accuracy: 0.7130
Epoch 166/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7967 -
accuracy: 0.7371 - val_loss: 0.9609 - val_accuracy: 0.7226
Epoch 167/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7451 -
accuracy: 0.7578 - val_loss: 0.9673 - val_accuracy: 0.7252
Epoch 168/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6773 -
accuracy: 0.7751 - val_loss: 0.9751 - val_accuracy: 0.7279
Epoch 169/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6263 -
accuracy: 0.7890 - val_loss: 0.9746 - val_accuracy: 0.7352
Epoch 170/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6701 -
accuracy: 0.7771 - val_loss: 0.9969 - val_accuracy: 0.7236
Epoch 171/200
```

```
48/48 [==============================] - 0s 9ms/step - loss: 0.7395 -
accuracy: 0.7599 - val_loss: 1.0013 - val_accuracy: 0.7209
accuracy: 0.7682 - val_loss: 0.9921 - val_accuracy: 0.7362
Epoch 185/200
48/48 [==============================] - 1s 14ms/step - loss: 0.7465 -
accuracy: 0.7549 - val_loss: 1.0040 - val_accuracy: 0.7269
Epoch 186/200
48/48 [==============================] - 1s 14ms/step - loss: 0.7079 -
accuracy: 0.7700 - val_loss: 0.9859 - val_accuracy: 0.7249
Epoch 187/200
48/48 [==============================] - 1s 13ms/step - loss: 0.6058 -
accuracy: 0.7973 - val_loss: 1.0128 - val_accuracy: 0.7266
Epoch 188/200
48/48 [==============================] - 1s 11ms/step - loss: 0.7111 -
accuracy: 0.7669 - val_loss: 1.0204 - val_accuracy: 0.7243
Epoch 189/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6535 -
accuracy: 0.7883 - val_loss: 1.0250 - val_accuracy: 0.7262
Epoch 190/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6310 -
accuracy: 0.7894 - val_loss: 1.0130 - val_accuracy: 0.7236
Epoch 191/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6393 -
accuracy: 0.7931 - val_loss: 0.9946 - val_accuracy: 0.7369
Epoch 192/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6045 -
accuracy: 0.8001 - val_loss: 1.0099 - val_accuracy: 0.7346
Epoch 193/200
48/48 [==============================] - 1s 11ms/step - loss: 0.6173 -
accuracy: 0.8001 - val_loss: 1.0217 - val_accuracy: 0.7385
Epoch 194/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6285 -
accuracy: 0.7917 - val_loss: 1.0074 - val_accuracy: 0.7309
Epoch 195/200
48/48 [==============================] - 0s 10ms/step - loss: 0.5553 -
accuracy: 0.8108 - val_loss: 1.0472 - val_accuracy: 0.7329
Epoch 196/200
48/48 [==============================] - 0s 10ms/step - loss: 0.5985 -
accuracy: 0.8014 - val_loss: 1.0746 - val_accuracy: 0.7233
Epoch 197/200
48/48 [==============================] - 0s 10ms/step - loss: 0.7101 -
accuracy: 0.7746 - val_loss: 1.0410 - val_accuracy: 0.7276
Epoch 198/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6393 -
accuracy: 0.7884 - val_loss: 1.0289 - val_accuracy: 0.7326
Epoch 199/200
48/48 [==============================] - 1s 11ms/step - loss: 0.6331 -
accuracy: 0.7944 - val_loss: 1.0188 - val_accuracy: 0.7292
Epoch 200/200
48/48 [==============================] - 0s 10ms/step - loss: 0.6045 -
accuracy: 0.8005 - val_loss: 1.0536 - val_accuracy: 0.7309
377/377 [==============================] - 2s 4ms/step - loss: 0.4105 -
accuracy: 0.8846
Train Loss = 0.4105406403541565
Train Accuracy = 0.8846058249473572
95/95 [==============================] - 1s 5ms/step - loss: 0.4466 -
accuracy: 0.8794
Test Loss = 0.4465959370136261
```

```
Test Accuracy = 0.8794019818305969
dic = {}
dic[1] = 1
dic[2] = 2
dic[3] = [3, 3, 4]
dic[4] = [2, 2, 3, 3]
dic[5] = 5
dic[6] = [1, 1, 2, 2, 2, 2]
dic[7] = [1, 1, 1, 1, 2, 2, 2]
dic[8] = [1, 1, 1, 1, 1, 1, 2, 2]
dic[9] = [1, 1, 1, 1, 1, 1, 1, 1, 2]
dic[10] = 10

def Mario_CNN_plus(input_shape, classes, n_of_streams, dic):
    X_input = Input(input_shape)
    split_rules = dic[n_of_streams]

    # Mario Brothers Split
    Xs1 = tf.split(X_input, split_rules, 2)
    Xs2 = []
    i = 0

    # Multiple Streams
for X in Xs1:
      i += 1
      X = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name= (str(i) + '_conv1'))(X)
      X = Activation('relu', name=(str(i) + '_relu1'))(X)

      X = Conv2D(filters=32, kernel_size=(3,3),
strides=(1,1),padding='same', name=(str(i) + '_conv2'))(X)
      X = Activation('relu', name=(str(i) + '_relu2'))(X)
      X = MaxPooling2D((1,1), strides=(1,1), name=(str(i) +
'_pool1'))(X)

      X = Conv2D(filters=64, kernel_size=(5,5),
strides=(1,1),padding='same', name=(str(i) + '_conv3'))(X)
      X = Activation('relu', name=(str(i) + '_relu3'))(X)
      X = MaxPooling2D((1,1), strides=(1,1), name=(str(i) +
'_pool2'))(X)

      X = Conv2D(filters=64, kernel_size=(5,1),
strides=(1,1),padding='same', name=(str(i) + '_conv4'))(X)
      X = Activation('relu', name=(str(i) + '_relu4'))(X)

      X = Conv2D(filters=64, kernel_size=(1,1),
strides=(1,1),padding='same', name=(str(i) + '_conv5'))(X)
```

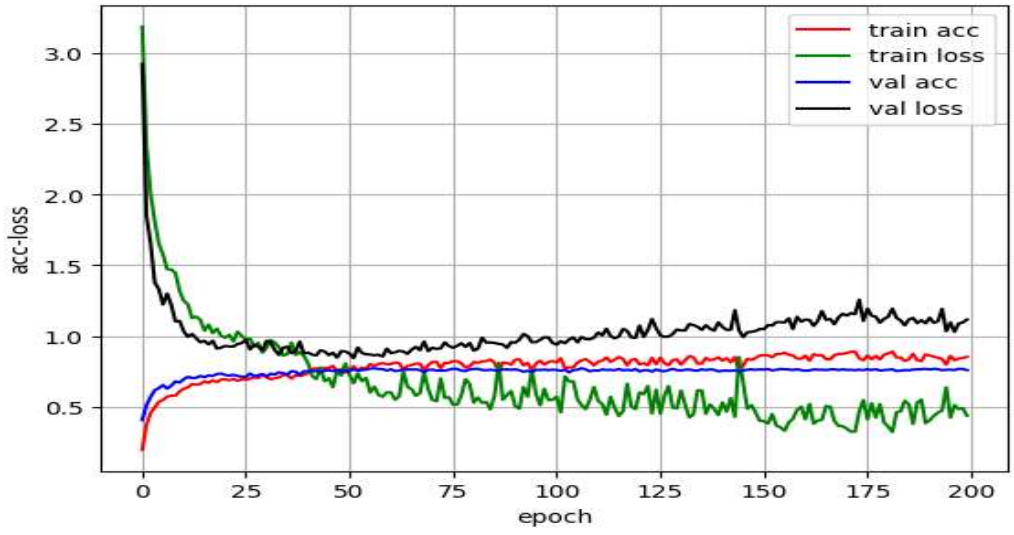
```
        Xs2.append(X)

    # Mario Brothers Reunion
    X = tf.keras.layers.Concatenate(axis=2)(Xs2)

    X = ZeroPadding2D((0,1))(X)
    X = Flatten(name='flatten')(X)
    X = Dropout(0.5)(X)
    X = BatchNormalization(momentum=0.9)(X)
    X = Dense(128, activation='relu',name='fc1')(X)
    X = Dropout(0.5)(X)
    X = Dense(classes, activation='softmax', name='softmax')(X)

    model = Model(inputs=X_input, outputs=X, name='MarioCNN')

    return model
train_acc = []
test_acc = []
for j in range(1, 11):
  model = Mario_CNN_plus(input_shape = (12, 10, 1), classes = 52,
n_of_streams = j, dic=dic)
  model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
  history = LossHistory()
  model.fit(data, label, validation_split=0.2, epochs=eps,
batch_size=256, verbose=1, callbacks=[history])
  preds_train = model.evaluate(X_train, Y_train)
  print("Train Accuracy = " + str(preds_train[1]))
  train_acc.append(preds_train[1])
  preds_test  = model.evaluate(X_test, Y_test)
  print("Test Accuracy = " + str(preds_test[1]))
  test_acc.append(preds_test[1])
  history.loss_plot('epoch')
```



LIST OF PUBLICATION

[1] Sudhir Kumar, Rajesh Birok ".Enhanced sEMG Signals Process with Two-Stream CNN on Gesture Classification" has been accepted for Publication in the15th IEEE International Conference on Computing ,Communication and Networking Technologies(ICCNT),IIT Mandi,India.

Gmail SUDHIR KUMAR <sudhirkumardos999@gmail.com>

15th ICCCNT 2024 submission 2440

1 message

15th ICCCNT 2024 <15thicccnt2024@easychair.org> Mon, May 27, 2024 at 7:38 AM
To: Sudhir Kumar <sudhirkumardos999@gmail.com>

"Dear Authors,
Paper ID:2440
Title: Enhanced sEMG Signals Process with Two-stream CNN on Gesture Classification

Congratulations! Your paper got accepted.

Similarity/Plagiarism Index: 11.2%

1. Work flow is good with sufficient results.
2. Provide a comparison table of proposed method with the state-of-the-art methods.
3. In table I, contents in the images are hard to read. Split the table such that, the content are clearly visible.
4. Does the accuracy value remains constant after 200 epochs?
5. Separate section for 'Results and Discussion' and Conclusion can be given. Conclusion can be precise and clear.
6. Plagiarism in the abstract should be reduced.

Author affiliation and paper should be in IEEE conference template
(https://www.ieee.org/conferences/publishing/templates.html)

***Complete the registration process immediately after receiving this email in order to prepare the presentation schedule (https://15icccnt.com/register/index.php).

For making payments (Indian Authors), using following bank account
Name of the Bank: Axis Bank
Account Name: ICCCNT Foundation
Account Name: 920020042183777 (savings bank account)
IFSC Code: UTIB0002811
Branch: Singanallur, Tamil Nadu, India

For authors outside India :
Paypal ID will be send soon (Kindly mention your Paper Id while making payments).

Fee Details: The fee details are mentioned in the following link https://15icccnt.com/registration.php

Similarity/Plagiarism Index should be below 5%.
Kindly update your manuscript based on the above comments and also update the same in the Easychair login at the same paper ID.
We will inform you of any further updates or changes required, if any.

Best regards,
15th ICCCNT 2024"

PAPER NAME
Two sream cnn method.docx

AUTHOR
Sudhir Kumar

WORD COUNT
4671 Words

CHARACTER COUNT
26958 Characters

PAGE COUNT
16 Pages

FILE SIZE
862.0KB

SUBMISSION DATE
May 26, 2024 2:19 PM GMT+5:30

REPORT DATE
May 26, 2024 2:19 PM GMT+5:30

7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- **3% Internet database**
- **4% Publications database**
- **Crossref database**
- **Crossref Posted Content**
- **database3% Submitted Works database**

Excluded from Similarity Report

- **Bibliographic material**
- **Quoted**
- **materialSmall Matches (Less then 8 words)**

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

PLAGIARISM VERIFICATION

Title of the Thesis_____

_____

Total Pages _____ Name of the Scholar_____

Supervisor (s)

(1)_____

(2)_____

(3)_____

Department_____

This is to report that the above thesis was scanned for similarity detection. Process and outcome is given below:

Software used: _____ Similarity Index: _____, Total Word Count: _____

Date: _____

Candidate's Signature **Signature of Supervisor(s)**