Machine Learning Optimisation in Non-Binary Classification Tasks

How does the Performance of Adam and Stochastic Gradient Descent Optimisation Algorithms Compare When Training Convolutional Neural Networks for the Purpose of Sign Language Detection?

Subject: Computer Science

Word Count: 3907

Contents

Introduction

This paper's research question is " **How Does the Performance of Adam and Stochastic Gradient Descent Optimisation Algorithms Compare When Training Convolutional Neural Networks for the Purpose of Sign Language Detection?**". The field of artificial intelligence(AI) is ever changing and in the past decade it has evolved from a purely academic pursuit to one of the most applicable technologies across many fields. Specifically, projects that leverage the capabilities of artificial intelligence for social good are becoming increasingly popular. This paper is centred upon the application of deep learning AI for sign language detection. Since this task is complex in nature, there are many different considerations to be made. One of which is the selection and customization of the most suitable optimisation algorithm - the focus of this paper. As the field of deep learning evolves, varying opinions on the most effective optimizer for computer vision tasks arise. Due to the often random and inconsistent behaviour of complex deep learning algorithms, it is impossible to know which optimisation algorithm will be the most suitable for a given task. As a result, this paper aims to not only give a detailed understanding of the theory, but also strives to provide relevant research into improving the effectiveness of AI in aiding the sign language communities.

This paper will first provide background information regarding the computational and mathematical theory necessary to understand the research question. It will then describe the experimental methodology employed along with justification for various design choices. Lastly the findings of the experiment will be analysed and discussed in order to compare the performance of the chosen optimisation algorithms.

Background

Immediately it is evident that the scope of this research is very broad. As a result, this paper shall limit its training to the equivalent signs of the alphabet characters a-z. Not only will this provide a manageable dataset to train a suitable machine learning model, but also many sign language words require complex movements and subtle gestures that would be too difficult to train a model for within the time constraints of this paper. This therefore will provide a feasible application that would be genuinely helpful in the real world.

Machine learning

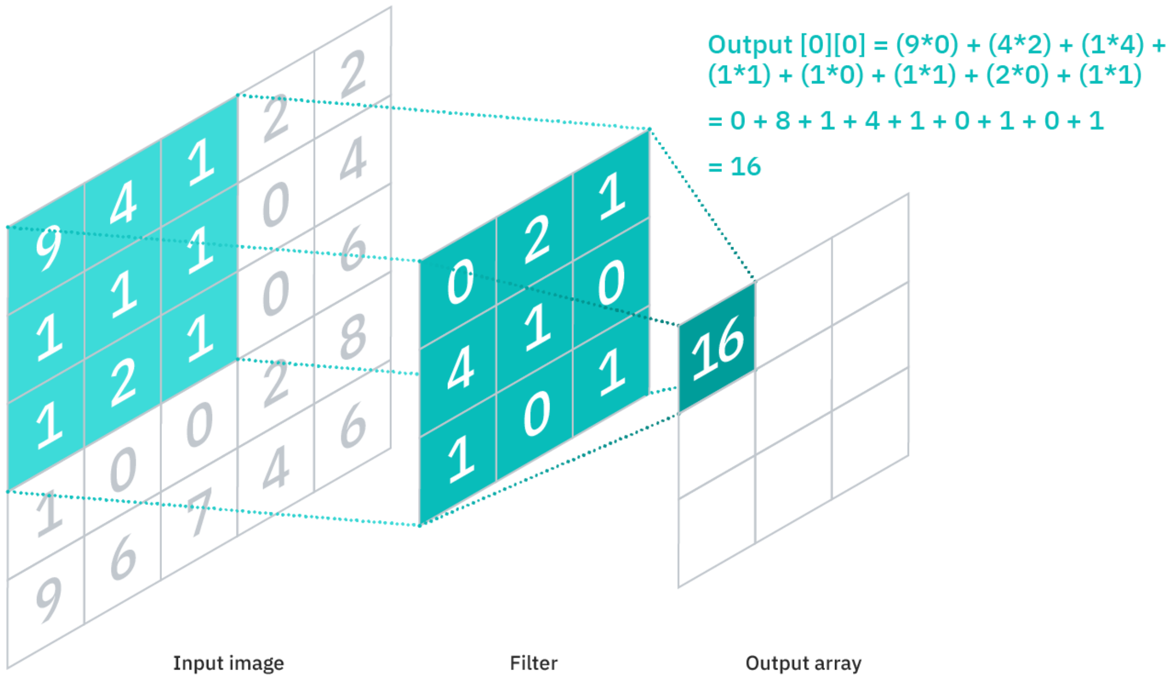
Machine learning(ML) is considered a subset of artificial intelligence whereby algorithms are used to identify patterns in data to then make predictions without direct instruction(Selig, 2022). Much like humans, the more data and time a model is given to train, the greater the accuracy of the machine learning model. Within machine learning there are 3 key categories: supervised, unsupervised and reinforcement learning. For the chosen application supervised learning is used, and shall be explained in greater detail. Supervised learning is where a labelled set of data is used to train a ML model. This type of ML is often seen in classification situations where there is a definitive correct or incorrect prediction. By training a model with a set of data along with the expected output data, the supervised algorithm will aim to find a mapping function of the input variable to the output variable. As mentioned above, the chosen application is a classification problem, yet more accurately a non binary classification problem. This is because the model must differentiate between multiple hand signs, hence the classification is non binary or ‘multi-classed’.

Convolutional neural networks

A convolutional neural network(CNN) is a specific type of neural network that has led significant growth in the machine learning field with its advancements in computer vision. Computer vision is a subset of machine learning that allows a computer to make predictions after being trained on images. A CNN generally has three layers: the convolutional layer, pooling layer, and a fully connected layer. However there are many different architectures that each have their own strengths and weaknesses in different applications.

A CNN sets itself apart from a standard neural network(NN) through its convolutional layer which allows it to "successfully capture the Spatial and Temporal dependencies"(Saha, 2018) , in other words it can be trained to understand the complexities of an image better.

Convolutional layer: As mentioned, this layer is what makes this approach so effective for computer vision tasks and differentiates its performance from a standard neural network.The goal of the convolution layer is to simplify the input image to a form that is easier to compute while not losing important features. The process of convolution is a mathematical operation whereby a typically 3x3 or 2x2 matrix (a filter) is passed over the input image matrix pixel by pixel. The two matrices are multiplied together, added and the resulting number is added to a feature map describing a specific feature of the image. This process is repeated for the entire input image. Fig 1. visualises this process.



Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
Input image
Filter
Output array

Fig 1. Visualisation of convolution (Guymonahan)

<u>Pooling layer</u>: Pooling is also a mathematical process that is responsible for reducing the size of the feature map to reduce the computational load within the network, also helping to prevent overfitting.

<u>Fully connected layer</u>: The fully connected layer or otherwise known as a hidden layer is the last layer in a CNN. Up to this point the previous layers were responsible for feature extraction from the images, whereas this layer is responsible for the classification of this data. This layer is a neural network where all the neurons are connected to the next layer's neurons. While this process is both theoretically and mathematically complex, put simply, each neuron or node of the network has both an activation function and a linear function. When data passes through each node, it is first passed through the linear function which results in an output. This output is then inserted into the activation function, which then determines if the node is activated. Fig 2. illustrates this setup:

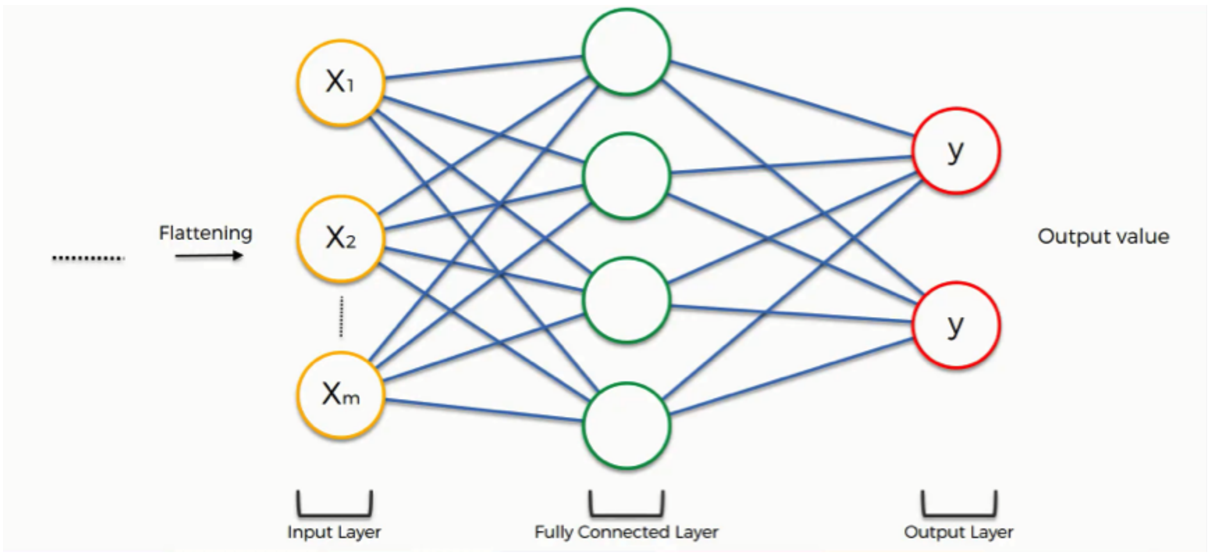


Fig 2. Visualisation of fully connected layer (SuperDataScience)

Optimisation in convolutional neural networks

The process of optimisation is where specific algorithms are used to reduce the loss of a neural network on a set of training data by adjusting its attributes e.g. weights, learning rate, e.t.c.

As a machine learning model learns, it makes decisions in order to improve its performance. The loss function essentially maps each of these decisions to their corresponding costs and is a measure of how well the model performs. A lower loss means that the model will perform better.

Optimisation algorithms can greatly improve the performance of machine learning models, making the selection of optimisers highly relevant to the final performance of the model. Since the training of complex models like CNN's can be both time consuming and resource intensive, it is of great importance to select the most suitable optimiser that results in the highest model accuracy with lowest training time and resource consumption.

The primary difficulty of optimisation of CNN's is that the error surface being optimised is non-convex. Therefore there are potentially many local minima or 'optimal points' which can cause the optimizer to settle at the local minima rather than global minima of the function. Figure 3. illustrates this.

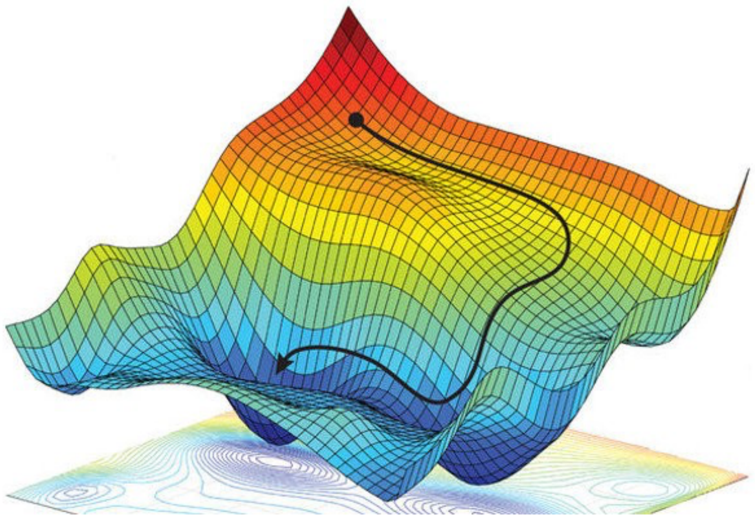


Fig 3. Visualisation of error surface optimisation (Mehdi)

While there are many different approaches to optimisation, this paper has selected two algorithms to compare:

<u>SGD</u>: Stochastic Gradient Descent is a variant of the classic gradient descent algorithm. As the name implies, in classic gradient descent the algorithm moves iteratively downwards until it reaches the minima or 0 gradient.

$$b = a - \gamma \triangledown f(a)$$

The above equation shows the mathematical equation for standard gradient descent. Where b is the next parameter of the model, a is the current parameter of the model, gamma is the learning rate and $f(a)$ is the direction of the greatest gradient. The algorithm works by initialising the parameters of the model with random values and then computes the gradient at that point. The algorithm then moves down by an amount controlled by the learning rate. This process is repeated until the algorithm determines that it has reached the minimum loss value of the curve.

The issue with standard gradient descent is that for larger datasets with possibly millions of data points and features, the algorithm needs to compute many gradients for each parameter and for each gradient many thousands or even millions of values depending on the learning rate. This is where SGD comes in. The word 'Stochastic' by definition means random. The SGD algorithm randomly picks a data point and calculates the gradient to update the parameter. In other words, rather than calculating the gradient from all of the points in the set, it simply chooses a single point, thereby reducing the computational cost by an exponential amount. However, this does come at the cost of longer convergence times and 'noisy' paths to the optimal value(GeeksforGeeks, 2022) due to the calculated gradient being only a broad approximation.

Adam: Adam is classified as an adaptive learning rate optimisation algorithm and was originally published in 2014. Adam stores different learning rates for each parameter and similar to SGD, it calculates gradients from an approximation. Adam essentially combines the two optimisation algorithms: momentum and RMSProp(root mean squared propagation) (DeepLearningAI & Ng, A. , 2017).

Momentum is a method of accelerating the gradient descent algorithm by considering the 'exponentially weighted average'(GeeksForGeeks, 2020). This can be thought of as accelerating towards the minima.

The RMSProp algorithm builds open SGD with instead adaptable learning rates for each parameter based on the calculated gradients. RMSProp also utilises what is called a decaying average so that when computing the learning rate, the algorithm will forget earlier gradients and only focus on the most recent gradients.

Adam then combines the positive attributes of these two algorithms so that it controls the momentum of the descent and learning rate of each parameter to achieve the most efficient path to the minima. Adam is seen to have high levels of performance while maintaining a low training cost, therefore making it a widely accepted default optimizer for a variety of tasks.

Despite the acceptance of Adam as the standard optimizer for most tasks, recent research has shown that SGD is sometimes seen to generalise better than ADAM at the sacrifice of training times(Zhou, 2020). More specifically, the research argues that Adam tends to converge to sharp minima, whereas SGD can escape easier from these sharp minima and find the flatter minima. This intuitively makes sense since at a flatter minima a small shift in train loss leads to a small shift in test loss, whereas a small shift in train loss at a sharper minima results in a much larger change in test loss. However, these results could have appeared due to a number of factors: poor datasets, overfitting, poor choice of architecture, e.t.c. It is also possible that these findings were highly implementation specific, therefore it is not fair to state that these findings apply to every case.

Sign language

Sign language has a number of variations that include American Sign Language, German Sign Language, Singaporean Sign Language, just to name a few. This paper shall limit itself to the A-Z character sets of the American Sign Language (ASL) due to the accessibility of suitable datasets.

Methodology

Overview of process

In this paper, a single CNN architecture will be trained on two separate A-Z datasets using the SGD and ADAM optimisation algorithms. Certain values will be measured and compared so that a conclusion can be drawn to answer the research question.

Since this paper is training on two separate datasets and utilising a CNN, there are many hyperparameters and variables that need to be considered to achieve an acceptable performance on both datasets. As a result, significant preliminary experimentation was conducted to reach a suitable methodology for this paper. The following sections detail this process and its findings. All code and implementation for the following can be found in the appendix.

Dataset

The first dataset used to train the model in this paper is an open source dataset on Kaggle titled "Sign Language MNIST". "Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions) "(*Sign Language MNIST, 2017)*. There are a total of 27,455 cases for the training set and 7172 cases for the test set of data. Fig 4. shows 10 randomly selected images from the dataset.

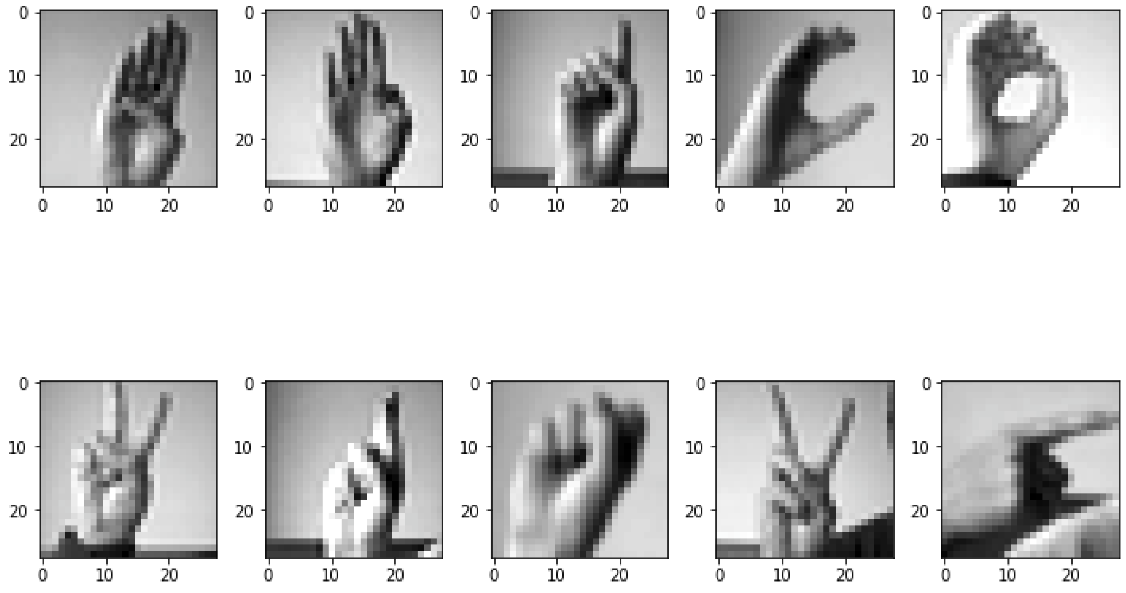


Fig 4. 10 sample 28x28 images from Sign Language MNIST dataset

The second data set is also sourced from Kaggle and is titled “ASL Alphabet”. This dataset is not only larger with a total of 87,000 images but also each image is 200x200 pixels. The higher resolution and size of this dataset will provide insight into how the size of the dataset might affect the performance of the chosen optimisation algorithms. The data did not come pre split, therefore a train test split of 7:3 was used. This was found to suitably reduce overfitting without undertraining the model. Fig 5. shows 3 randomly selected images from the dataset for reference.

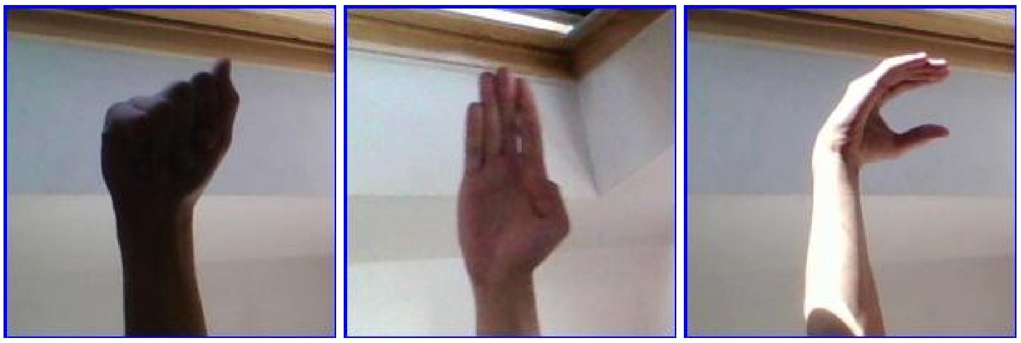


Fig 5. 3 sample 64x64x3 images from ASL Alphabet dataset

Additionally, overfitting where the model 'memorises' the training dataset leading to poor performance on the test dataset is a concern. In order to overcome this, image augmentation will be utilised. This is the process by which the dataset is artificially expanded through small transformations to better train the model. The variations chosen are as follows:

- Randomly rotate some images by 20°
- Greyscale
- Random vertical translation of images by 20% of the height
- Random horizontal translation of images by 20% of the width
- Random zooming of images by 10%

*No flips we used as this could cause the model to misclassify images as another letter.

Convolutional neural networks

In order for a valid comparison between the two optimisation algorithms to be drawn, it is important to decide on a suitable CNN architecture that trains well on both datasets. The chosen architecture has 4 convolutional layers with MaxPooling and Dropout added between them. The fully connected layer has two Dense layers. This architecture was adapted from the popular AlexNet architecture. The exact specifications of the design can be found in the appendix. This architecture was chosen because it provided sufficient complexity to classify both datasets with acceptable accuracy, while still maintaining training times in a reasonable range to be practical within the time constraints of this paper. It was also adjusted so as to prevent overfitting as much as possible.

Overfitting considerations

The program utilises a feature of the machine learning library Keras called EarlyStopping. This implements a callback method that will stop the training process if it detects that the model has reached its lowest loss or otherwise has been fully optimised. The patience of the EarlyStopping was set to 7 throughout the training of both models. This means that the model will stop training once 7 epochs(iterations) with no improvement are detected. The EarlyStopping is key to a well fit model since if the model is left to train for too long on the train data, its test loss can actually increase over time. Fig 6. shows this scenario:

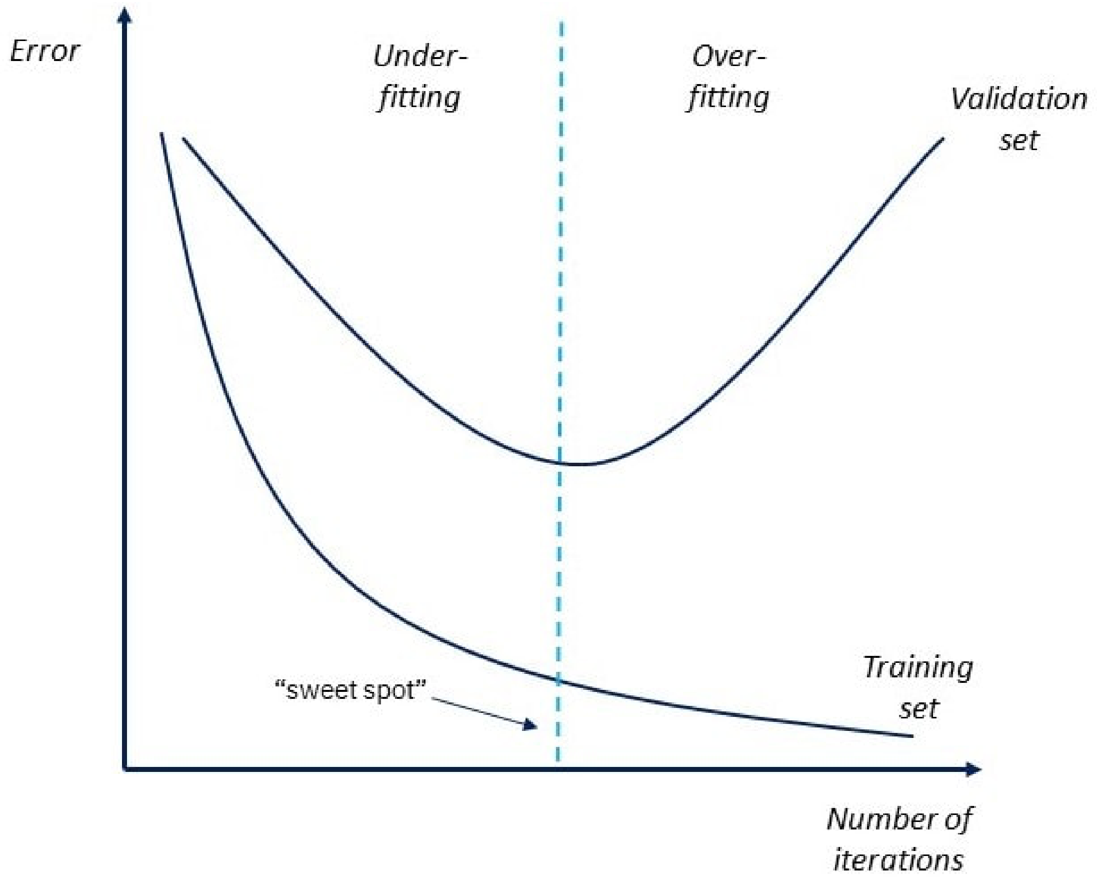


Fig 6. Loss curve visual (IBM Cloud Education)

Equipment + environment

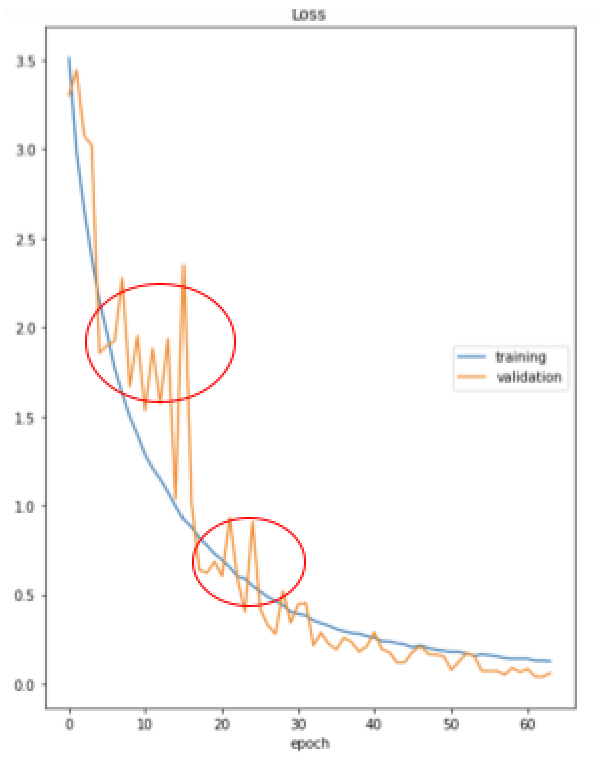
A MacBook Pro laptop running macOS Monterey version 12.3 with an Apple Silicon M1 Pro processor and 16GB of RAM was used to build, train, and test the model on through a jupyter notebook environment. All code was written using Python 3. Additionally, this paper utilises the publicly available TensorFlow library for machine learning, however, all implementations are custom. A tensorflow macOS metal plugin was also utilised to leverage the GPU in the laptop for optimal performance during the experimental process.

Measured values

- Test loss
- Train loss
- Test accuracy
- Train accuracy
- Epochs
- Training duration

Results

SGD						Adam					
Sign Language MINST			ASL Alphabet			Sign Language MINST			ASL Alphabet		
Train Loss	Test Loss	Training Duration	Train Loss	Test Loss	Training Duration	Train Loss	Test Loss	Training Duration	Train Loss	Test Loss	Training Duration
0.129	0.062	768	0.100	0.902	8989	0.121	0.035	443	0.073	0.862	5801



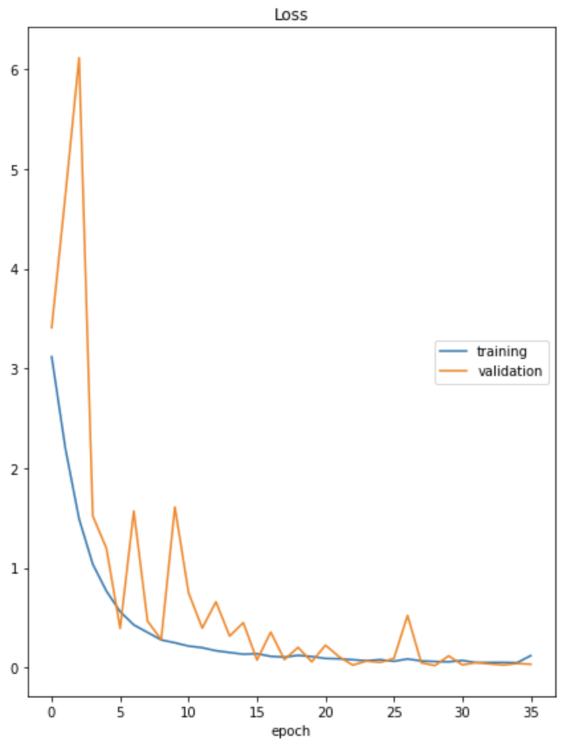
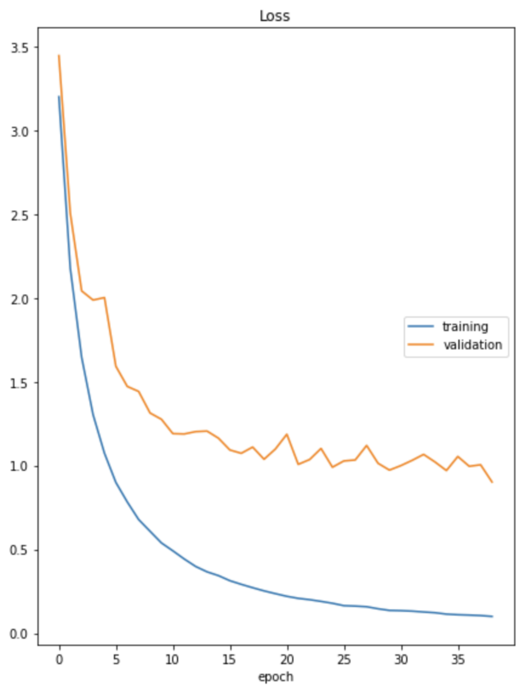


Fig 7. Small Dataset SGD **Fig 8. Small Dataset Adam**

Fig 7. and Fig 8. present the difference in training between the two optimizers. It is clear that Adam achieves a much lower loss through its first iteration and then proceeds to reach convergence at around 35 epochs. On the other hand, SGD adopts a steady descent, only reaching convergence at around 60 epochs. This is to be expected as SGD has a fixed, global learning rate, whereas Adam adapts each parameter’s learning rate leading to its faster convergence. With regards to generalisation, Adam performed with a test loss of 0.035, whereas SGD performed with a test loss of 0.062. One point to highlight is that the validation loss value is actually slightly less than the training loss towards the convergence point. This is what is called an unrepresentative validation dataset(machinelearningmastery) and it means that the test data is slightly too small to properly demonstrate the performance of the model. This isn’t particularly problematic as both optimizers were tested under the same conditions, but it is a notable trend.

Additionally, it appears that SGD struggles with some local minima(circled in Fig 7), but quickly escapes from these points and continues along its descent. Although the difference in generalisation performance is relatively small, the training duration of SGD was far greater at 789s compared to 443s on the given CNN architecture. Given these results, it is clear that the Adam optimizer is the superior algorithm for this dataset and CNN architecture. However, it is also necessary to consider how these algorithms performed on the larger, more complex dataset.



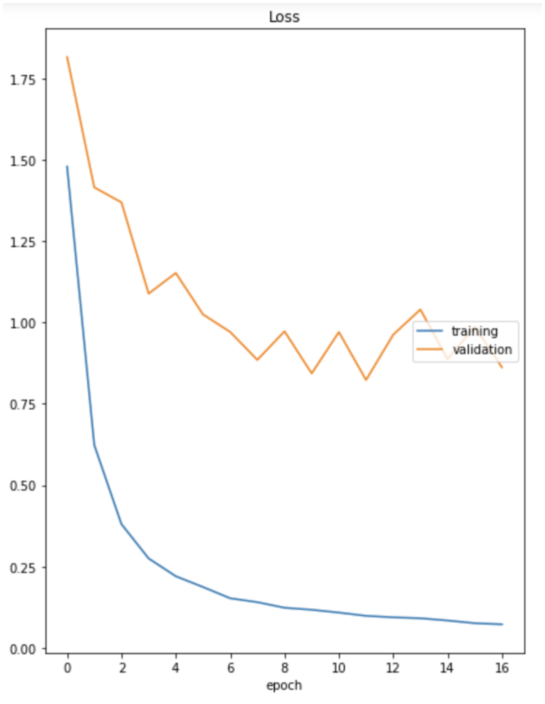


Fig 9. Large Dataset SGD **Fig 10. Large Dataset Adam**

Above, Fig 9. and Fig 10. demonstrate the loss vs epoch graphs for both optimizers on the "ASL Alphabet" dataset. Similar to the previous dataset, Adam converges in less time than SGD- reaching convergence at around 35 epochs compared to Adam's 16. The two algorithms converge to very similar loss values at 0.902 and 0.862 respectively, which prove that the loss function alone is not a reliable indicator of the model's performance. Unlike the smaller dataset, neither Adam or SGD appear to get stuck at any local minima, and descend quite evenly to the determined global minima. Furthermore, it is notable that the validation loss takes a far less ‘noisy’ path to the minima. This is likely due to the increased resolution, allowing the network to classify images with greater accuracy. It is clear that the larger validation data set removes the case in the previous dataset where the validation loss is less than the train loss.

Analysis

Overall, it is seen that both optimizers performed reasonably well, however in both datasets the Adam optimizer resulted in the lowest loss. Additionally, Adam proved to converge significantly faster than SGD, as expected. Given these two datasets there was no evidence to support the claim that SGD generalises better than Adam - at least in this context.

Although the SGD optimizer eventually converges, it is interesting to observe the initial fluctuations of the loss graph for the validation data set. The drastic fluctuations are very evident in the smaller “Sign language MNIST” dataset, and this ‘noise’ is due to the fact that each step is calculating an approximation of the gradient where an anomalous value can result in a great fluctuation in the loss curve gradient. The higher resolution of the “ASL Alphabet” dataset proved to reduce this ‘noise’ resulting in a smoother descent towards the minima. Furthermore, while the difference in training duration between SGD and Adam in the smaller dataset is relatively inconsequential, the larger dataset exhibits a 10x increase in the duration of training. The process of training this dataset with SGD was by far the most time consuming and resource intensive process. The total training time was around 2 hours and 30 minutes which pushed the available hardware to its maximum capacity. The suggestion that

SGD fails to scale on larger datasets is critical to understand as projects of increased complexity would certainly benefit from the lightweight and efficient properties of Adam.

It was also surprising to observe how on the larger dataset a higher loss value occurred at the chosen minima. Since the same model conditions were maintained in both variations of the dataset, it is likely that the images in the larger dataset proved more difficult to classify. For example, the test data may have had a particularly high number of darker low contrast images where the finer details of the hand and shape were lost, therefore raising the value of observed loss.

Conclusion

Given the experimental results presented, it can be concluded that given the conditions provided and tested, the Adam algorithm produces marginally superior performance, with significantly faster convergence speeds. There was no evidence in this implementation that suggests SGD was able to generalise better than Adam unlike select other research presented. The convergence speed becomes especially noticeable in the larger dataset where the SGD algorithm took roughly 1 hour more than Adam. Considering this model was already trained on a relatively powerful device, if this was to be extended to include data for sign language numbers for example, the dataset would likely be too large to train efficiently on the current hardware. Therefore warranting the consideration of a different approach and/or a more powerful machine.

However, as with any optimisation algorithm, there are a number of different hyperparameters that can be tweaked to achieve optimal performance. There are likely cases where SGD can be tweaked so that it achieves better performance than Adam, however, the time to precisely tune and alter the algorithm is long and ultimately not worth the time for most developers.

It should be noted that this conclusion is based upon this specific implementation. The datasets, CNN architecture and even random sampling of images is all unique to this paper, therefore its conclusions cannot be generalised too broadly to other applications and implementations.

Limitations of Study

Datasets

In machine learning a larger dataset is always preferable to achieve the highest performance model. However, as seen in this paper, there are limited publicly available datasets and the quality and variety of images in the dataset can be questionable. Furthermore, a larger dataset requires a greater computational power that simply is not available in the form factor of a laptop at this time.

Hardware

As mentioned, although the laptop used to build/train the model in this paper is quite powerful, further research would need to be conducted on a dedicated machine/server to achieve optimal results. Another approach might be to utilise transfer learning- this is where a model trained on a different dataset is taken and applied to a similar but different problem. This might reduce computational cost and allow for a larger dataset to be trained.

Extensions

In sign language most signs for words are complex gestures and thus this presents an opportunity for experimentation with how effective a CNN might be at classifying video input. This would build on the research established in this paper, however the key difference being the addition of the temporal aspect. Such a project would also likely have a strong real world impact as it would allow for gesture based signs that make up the majority of sign language words to be understood by others.

References

Brownlee, J. (2018, December 10). *Use Early Stopping to Halt the Training of Neural Networks At the Right Time*. Machine Learning Mastery. Retrieved August 12, 2022, from https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

Brownlee, J. (2019, February 27). *How to use Learning Curves to Diagnose Machine Learning Model Performance*. Machine Learning Mastery. Retrieved August 12, 2022, from https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Brownlee, J. (2020, April 8). *4 Types of Classification Tasks in Machine Learning*. Machine Learning Mastery. Retrieved August 12, 2022, from https://machinelearningmastery.com/types-of-classification-in-machine-learning/

Brownlee, J. (2021, May 24). *Gradient Descent With RMSProp from Scratch*. Machine Learning Mastery. Retrieved August 12, 2022, from https://machinelearningmastery.com/gradient-descent-with-rmsprop-from-scratch/

DeepLearningAI & Ng, A. (2017, August 26). *Home*. YouTube. Retrieved August 12, 2022, from https://www.youtube.com/watch?v=JXQT_vxqwIs&t=248s

Doshi, S. (2019, January 13). *Various Optimization Algorithms For Training Neural Network | by Sanket Doshi*. Towards Data Science. Retrieved August 12, 2022, from https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

GeeksForGeeks. (2020, October 24). *Intuition of Adam Optimizer*. GeeksforGeeks. Retrieved August 12, 2022, from https://www.geeksforgeeks.org/intuition-of-adam-optimizer/

GeeksforGeeks. (2022, June 13). *ML | Stochastic Gradient Descent (SGD)*. GeeksforGeeks. Retrieved August 12, 2022, from https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

Guymonahan. (2021, July 12). *Convolutional Neural Network*. Retrieved August 12, 2022, from https://guymonahan.medium.com/basic-overviews-on-convolutional-neural-networks-f205180116be.

IBM Cloud Education. (2020, August 19). *What is Supervised Learning?* IBM. Retrieved August 12, 2022, from https://www.ibm.com/cloud/learn/supervised-learning

IBM Cloud Education. (2020, October 20). *What are Convolutional Neural Networks?* IBM. Retrieved August 12, 2022, from https://www.ibm.com/cloud/learn/convolutional-neural-networks

IBM Cloud Education. (2021, March 3). *What is Overfitting?* IBM. Retrieved August 12, 2022, from https://www.ibm.com/cloud/learn/overfitting

javaTpoint. (n.d.). *Supervised Machine learning*. Javatpoint. Retrieved August 12, 2022, from https://www.javatpoint.com/supervised-machine-learning

Kaggle. (n.d.). *ASL Alphabet*. Retrieved August 12, 2022, from https://www.kaggle.com/datasets/grassknoted/asl-alphabet?datasetId=23079&sortBy=commentCount

Lao, R. (2018, January 22). *A Beginner's Guide to Machine Learning | by Randy Lao*. Medium. Retrieved August 12, 2022, from https://medium.com/@randylaosat/a-beginners-guide-to-machine-learning-dfadc19f6caf

Mahendru, K. (2019, August 14). *Loss Function | Loss Function In Machine Learning*. Analytics Vidhya. Retrieved August 12, 2022, from https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/

Mehdi, R. E. (2020, July 28). *Non-Convex Optimization in Deep Learning | by Er Raqabi El Mehdi | The Startup*. Medium. Retrieved August 12, 2022, from https://medium.com/swlh/non-convex-optimization-in-deep-learning-26fa30a2b2b3

Microsoft. (n.d.). *What is machine learning?* Microsoft Azure. Retrieved August 12, 2022, from https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-machine-learning-platform/

Mishra, M. (2020, August 26). *Convolutional Neural Networks, Explained | by Mayank Mishra*. Towards Data Science. Retrieved August 12, 2022, from https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

Saha, S. (2018, December 15). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Towards Data Science. Retrieved August 12, 2022, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Selig, J. (2022, March 14). *What is Machine Learning?* Retrieved August 12, 2022, from https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwj50Piq2cL5AhVuSGwGHc_GDE8QFnoECAUQAQ&url=https%3A%2F%2Fwww.expert.ai%2Fblog%2Fmachine-learning-definition%2F&usg=AOvVaw02HC9MQ_VbBqbq4iY5w81T

Sign Language MNIST. (2017). Kaggle. Retrieved August 12, 2022, from https://www.kaggle.com/datasets/datamunge/sign-language-mnist

Soydaner, D. (2020, February). A Comparison of Optimization Algorithms for Deep Learning. *International Journal of Pattern Recognition and Artificial Intelligence*. https://www.researchgate.net/publication/339073808_A_Comparison_of_Optimization_Algorithms_for_Deep_Learning

Srinivasan, V. (2019, September 6). *Stochastic Gradient Descent — Clearly Explained !! | by Aishwarya V Srinivasan*. Towards Data Science. Retrieved August 12, 2022, from https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31

SuperDataScience. (2018, August 18). *Convolutional Neural Networks (CNN): Step 4 - Full Connection - Blogs*. SuperDataScience. Retrieved August 12, 2022, from https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection

Unsupervised Feature Learning and Deep Learning Tutorial. (n.d.). Unsupervised Feature Learning and Deep Learning Tutorial. Retrieved August 12, 2022, from http://deeplearning.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/

Zhou, P. (2020, December 6). *Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning*. Semantic Scholar. Retrieved August 12, 2022, from https://pdfs.semanticscholar.org/ba97/7e9be47be2c625365e2830478b74bb43202a.pdf

Appendix

Experimental Code for Sign Language MNIST Dataset:

Importing necessary libraries

```
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout , BatchNormalization, AveragePooling2D,MaxPoo
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import EarlyStopping
from livelossplot import PlotLossesKeras
import numpy as np
import pandas as pd
import os
from sklearn.preprocessing import MinMaxScaler
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import plot_model
from time import time
```

Retrieve data and split into test/train labels and images for the Sign Language MNIST dataset

```
train_df = pd.read_csv("/Users/jsc/Downloads/archive/sign_mnist_train/sign_mnist_train.csv")
test_df = pd.read_csv("/Users/jsc/Downloads/archive/sign_mnist_test/sign_mnist_test.csv")
x=train_df.iloc[:,1:].values
y=train_df.iloc[:,0]

norm=MinMaxScaler()
norm.fit(x)
transnorm=norm.transform(x)

x=x.reshape(-1,28,28,1)
y=to_categorical(y,num_classes=25)

x_train,x_valid,y_train,y_valid=train_test_split(x,y,test_size=0.3,random_state=4)
```

Add random image augmentation to combat overfitting

```
train_datagen = ImageDataGenerator(rescale=1./255,
                        rotation_range=20,
                         height_shift_range=0.2,
                         width_shift_range=0.2,
                         horizontal_flip=False,
                         zoom_range=0.10)

train_datagen.fit(x_train)
```

Compile the specified CNN model

```
chosenOptimizer = 'sgd' # variable to change sgd or adam

model=Sequential()
#Convolutional layer
model.add(Conv2D(128,(5,5),input_shape=(28,28,1),activation='relu',name='conv1'))
model.add(BatchNormalization())
model.add(Conv2D(128,(5,5),activation='relu',name='conv2'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),name='max1'))
model.add(Dropout(0.3))

model.add(Conv2D(64,(3,3),activation='relu',name='conv3'))
model.add(BatchNormalization())
model.add(Conv2D(64,(3,3),activation='relu',name='conv4'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),name='max2'))
model.add(Dropout(0.3))

model.add(Conv2D(32,(3,3),activation='relu',name='conv5'))
model.add(BatchNormalization())
model.add(Dropout(0.3))
#Fully connected layer
model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(25,activation='softmax'))

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

Train model and plot loss vs epochs graph

```
from time import time
es = EarlyStopping(monitor='val_loss', patience=5)

start = time()

history=model.fit(train_datagen.flow(x_train, y_train, batch_size = 128),
validation_data = train_datagen.flow(x_valid, y_valid),epochs = 100,callbacks=[es1,PlotLossesKeras()])

print(time()-start)
```

Experimental Code for ASL Alphabet Dataset:

Retrieve data for ASL Alphabet dataset and split into train and validation sets. Image augmentation was also applied

```
data_dir = "/Users/jsc/Downloads/archive-2/asl_alphabet_train/asl_alphabet_train"
target_size = (64, 64)
target_dims = (64, 64, 3) # add channel for RGB
n_classes = 29
val_frac = 0.3
batch_size = 128

data_augmentor = ImageDataGenerator(
                                    rotation_range=20,
                                    height_shift_range=0.2,
                                    width_shift_range=0.2,
                                    horizontal_flip=False,
                                    zoom_range=0.10,
                                    validation_split=val_frac)

train_generator = data_augmentor.flow_from_directory(data_dir, target_size=target_size, batch_size=batch_size,
                                                     shuffle=True, subset="training")
val_generator = data_augmentor.flow_from_directory(data_dir, target_size=target_size, batch_size=batch_size,
                                                   subset="validation")
```

Compile the specified CNN model

```
chosenOptimizer = 'sgd' # variable to change sgd or adam


model=Sequential()
#Convolutional layer
model.add(Conv2D(128,(5,5),input_shape=(64,64,3),activation='relu',name='conv1'))
model.add(BatchNormalization())
model.add(Conv2D(128,(5,5),activation='relu',name='conv2'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),name='max1'))
model.add(Dropout(0.3))

model.add(Conv2D(64,(3,3),activation='relu',name='conv3'))
model.add(BatchNormalization())
model.add(Conv2D(64,(3,3),activation='relu',name='conv4'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),name='max2'))
model.add(Dropout(0.3))

model.add(Conv2D(32,(3,3),activation='relu',name='conv5'))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(29,activation='softmax'))

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

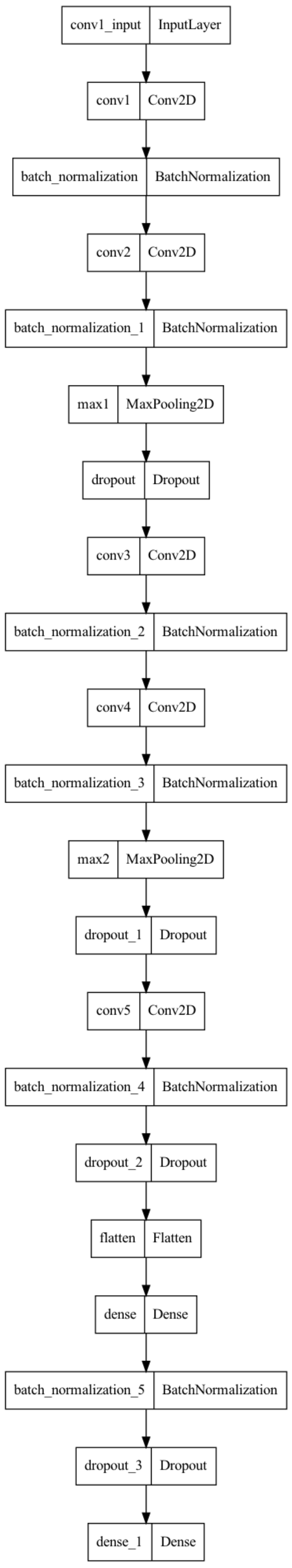
Train model and plot loss vs epochs graph

```
from time import time

es = EarlyStopping(monitor='val_loss', patience=5)

start = time()
history=model.fit(train_generator, batch_size = 128,
 validation_data = val_generator,epochs = 100,callbacks=[es,PlotLossesKeras()])
print(time()-start)
```

Visual of chosen CNN architecture:


conv1_input InputLayer
conv1 Conv2D
batch_normalization BatchNormalization
conv2 Conv2D
batch_normalization_1 BatchNormalization
max1 MaxPooling2D
dropout Dropout
conv3 Conv2D
batch_normalization_2 BatchNormalization
conv4 Conv2D
batch_normalization_3 BatchNormalization
max2 MaxPooling2D
dropout_1 Dropout
conv5 Conv2D
batch_normalization_4 BatchNormalization
dropout_2 Dropout
flatten Flatten
dense Dense
batch_normalization_5 BatchNormalization
dropout_3 Dropout
dense_1 Dense