Yoga Pose Classification

Anand K S
Roll no: AM.EN.U4CSE19106
Amrita School Of Engineering,
Amritapuri

Bharath Prathap Nair
Roll no: AM.EN.U4CSE19113
Amrita School Of Engineering,
Amritapuri

Rahan Manoj
Roll no: AM.EN.U4CSE19144
Amrita School Of Engineering,
Amritapuri

***Abstract*—This project aims to classify Yoga Poses effectively using Feed-Forward Network. [To be elaborated in later phases.]**

I. Introduction

A yogasana is any action, pose, exercise that is performed with rhythmic breathing cycles to benefit both body and mind. This project aims to classify Yoga Poses effectively using Feed-Forward Network. The feed forward model is build using PyTorch framework. [To be elaborated in later phases.]

II. Dataset

The project aims at classifying different yoga poses or asanas. The model was trained using the 'Yoga Pose Image classification dataset' from Kaggle. Along with this, a few more images were also extracted from Google and other yoga-based websites using an external chrome extension. Data augmentation was performed on the dataset at later stages.

The dataset contains 1242 images classified across 10 classes signifying 10 different yoga poses/asanas, which are Svanasana, Padmasana, Bhujangasana, Utkata-konasanam, Marjarasanam, Trikonasana, Vriksasana, Uttanasana, Savasana and Tadasana which are shown in Figure 1.

III. Data Pre-processing

A stratified split was applied on the dataset, where 60% (745 images) of the data was used for training and the remaining 40% (497 images) was used for validation. As the dataset consisted of images of different size, the images were resized to a uniform magnitude of 128*128 pixels. For the network to constantly learn and update its weight, the dataset was split into batches of size 32.

IV. Methods

A sequential approach was followed to develop various feed-forward models with the goal of obtaining an optimum model.

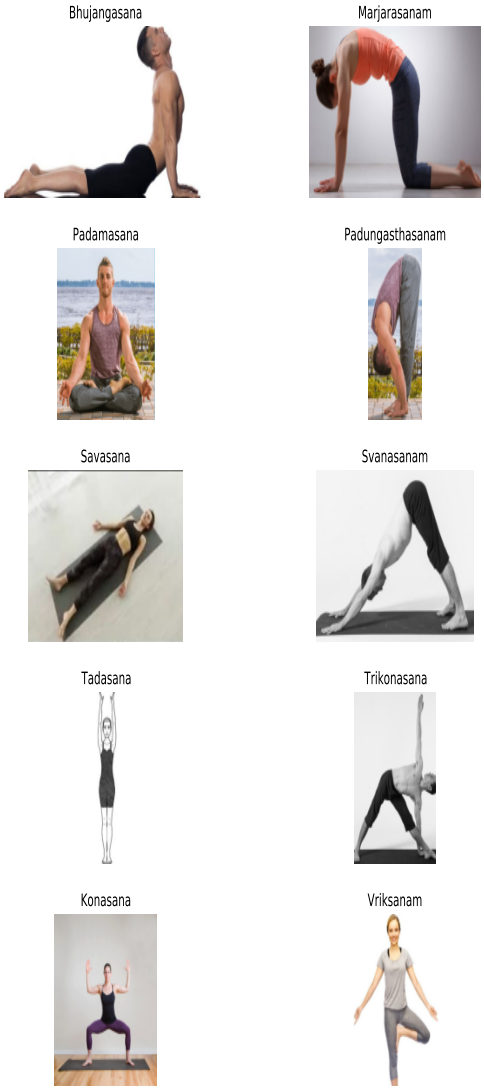

Fig. 1. Yoga Asanas

A. Step 1 - Base Model

The initial model implemented for the objective was a basic model consisting of 1 hidden layer with dimension 128. The input layer had a dimension of 3*128*128 in order to accommodate the RGB channels of each of the 128*128 sized images. The output layer had a dimension of 10, which is the number of classes in the dataset.

Initially Tanh was taken as activation function of the hidden layer and Sigmoid function was used to activate the output layer. The optimiser used to optimise the model was Stochastic Gradient Descent with learning rate of 0.01. Cross Entropy is the loss function used here.

Upon training it was found that the training and testing accuracy remained constant. This can be observed from the model accuracy plot in Figure 2.

Various activation functions like ReLU, LeakyReLU, Softmax and LogSoftmax were used in combination with the existing optimizer as well as various optimizers including Adagrad, Adadelta, Adam, RMSprop were used in combination with the existing activation functions. It was observed that the issue of constant accuracies persisted in some combinations.

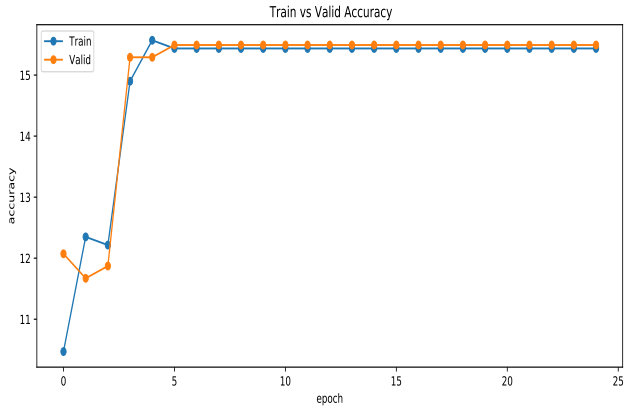

Fig. 2. Accuracy Plot for Model 1

This issue arose due to the problem of vanishing gradients in case of activation functions like Tanh and Sigmoid. In the case of ReLU and LeakyReLU, the gradient will be very small so that the learning happens at a slow rate. These factors caused Stochastic Gradient Descent's performance to be slow. This was solved by dropping them and replacing these with other alternatives. It was observed that LeakyReLU and LogSoftmax outperformed other activation functions used in hidden layer and output layer respectively. These were then subjected to various optimizers and loss functions and the details of appreciable combinations are tabulated below as well as some of them are represented in Figures 3 - 6.

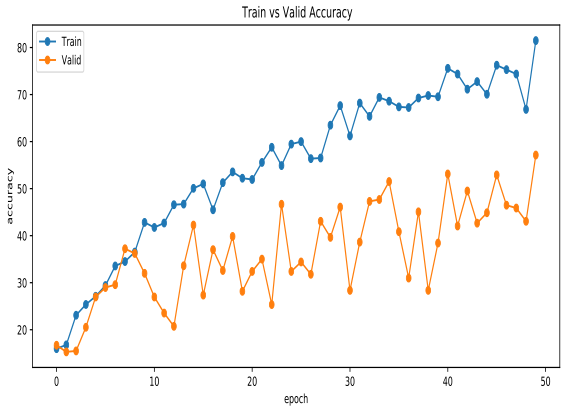

Fig. 3. Accuracy Plot for Adadelta - Cross Entropy Model

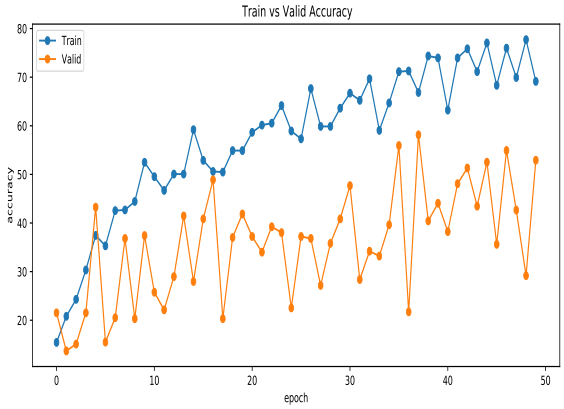

Fig. 4. Accuracy Plot for Adagrad - Cross Entropy Model

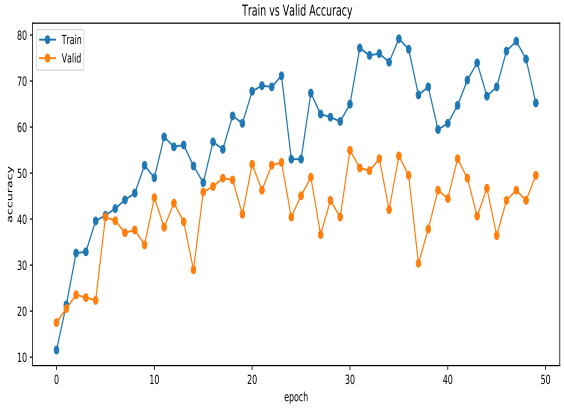

Fig. 5. Accuracy Plot for Adam - Cross Entropy Model

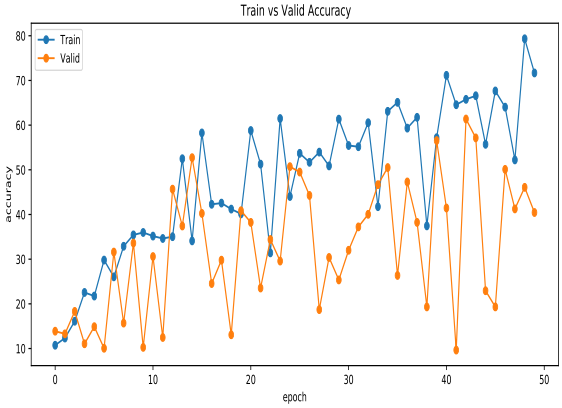

Fig. 6. Accuracy Plot for RMSprop -Cross Entropy Model

TABLE I
PERFORMANCE OF LEAKYRELU AND LOGSOFTMAX WITH VARIOUS OPTIMIZERS AND LOSS FUNCTIONS

Sl No	Hyperparameters		
	Optimizer	***Loss Function***	***Accuracy***
1	Adagrad	Cross Entropy	52.125
2	Adam	Cross Entropy	48.692
3	Adadelta	Cross Entropy	40.442
4	RMSProp	Cross Entropy	38.921
5	Adagrad	Negative Log Likelihood	50.139
6	Adam	Negative Log Likelihood	53.921
7	Adadelta	Negative Log Likelihood	40.012
8	RMSProp	Negative Log Likelihood	32.796

B. Step 2 - Increasing the number of layers

As observed from the accuracies in Table 1, the performance of RMSProp and Adadelta were comparatively low and hence they are not considered for the next step of model analysis. Further analysis was performed using Adagrad and Adam by adding more number of hidden layers and varying the dimensions. Even after performing the above modifications, no significant improvement in performance was observed.

C. Step 3 - Data Augmentation

A possible reason for low accuracies in step 2 maybe insufficient training data. To solve this issue, a strategy that can be implemented is Data Augmentation. By performing data augmentation, we add slightly modified copies of already existing data or newly created synthetic data from existing data. Thereby increasing the size of the dataset.
The augmentation techniques used are:

- Image Rotation : Rotated images in 45 degree and 300 degree inclinations.
- Shearing Images : Shearing images using Affine transformation.
- Flip images : Flips images from left to right.
- Adding Noise to Images
- Blurring images : Applied Gaussian filter to blur images.
- Increased Brightness
- Increased Contrast

After data augmentation, the number of images was increased to 6075. Even after performing these data augmentation techniques, a considerable improvement was not observed. In order to bring in better results, the number of hidden layers and number of neurons were increased. This showed better results in case of a 3 layer network, with dimension 512-1024-512 but led to an overfit condition as shown in Figures 7-10.

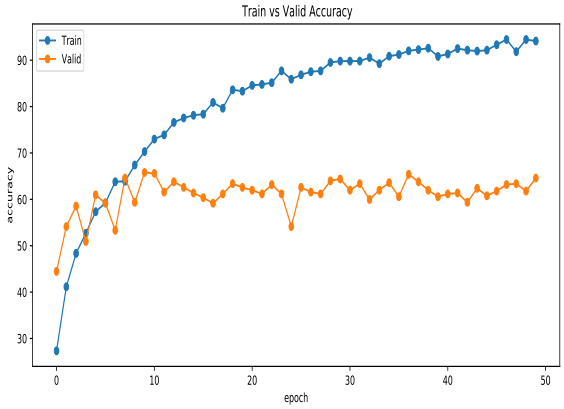

Fig. 7. Accuracy Plot for Adam -NLL Loss

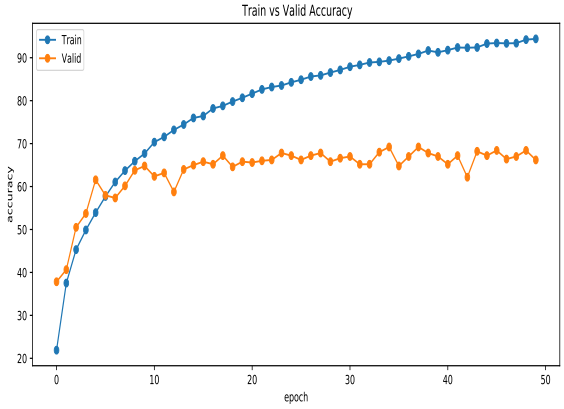

Fig. 8. Accuracy Plot for Adagrad -NLL Loss

D. Step 4 - Regularisation

In an effort to curb the overfitting encountered in step 3, we apply regularisation techniques. Regularisation is a technique used to bring in slight modification to the learning algorithm such that the model generalises better. This in turn improves the model's performance on the unseen data. The regularisation methods used here are:

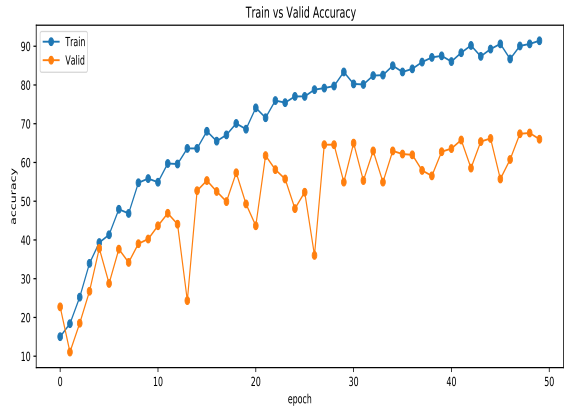

Fig. 9. Accuracy Plot for Adagrad -Cross Entropy

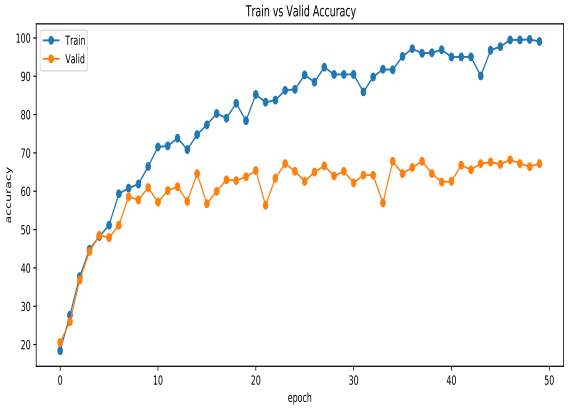

Fig. 10. Accuracy Plot for Adam -Cross Entropy

- Dropout : To select random nodes and remove them along their incoming and outgoing connections thereby capturing more randomness. The P value or the probability of how many nodes should be dropped was set to 0.5.
- L2 Regularisation : In L2 regularisation we update the cost function by adding another term known as regularisation term. It is a hyperparameter whose value is optimised for better result. The weight decay parameter was set in the PyTorch optimizer, in order to achieve this.
- Early Stopping : As an early stopping strategy, the number of epochs were reduced as we observed that no considerable improvement in training occured after 30 epochs.

E. Step 5 - Final Modal

After subjecting the data to Data augmentation and regularisation, an improvement in the model was observed. Adam optimizer was replaced with AdamW because as Hutter points out in their paper [Decoupled Weight Decay Regularization] that the way weight decay is implemented in Adam seems to be wrong and a simple way to fix it, is by using AdamW.

The final model now uses an architecture with 3 hidden layers of dimensions 512-1024-512 using AdamW and Adagrad as optimizers with cross entropy loss and negative log likelihood loss (NLLLoss) as loss functions. LeakyReLU is used for the activation of hidden layers and LogSoftmax is used as the output activation function. He initialisation(Kaiming) was used .The results are tabulated in Table 2 and represented in Figures 11-14.

TABLE II
PERFORMANCE OF FINAL MODEL WITH VARIOUS OPTIMIZERS AND LOSS FUNCTIONS

Sl No	**Hyperparameters**		
	Optimizer	***Loss Function***	***Accuracy***
1	Adagrad	Cross Entropy	63.343
2	AdamW	Cross Entropy	60.965
3	Adagrad	Negative Log Likelihood	70.623
4	AdamW	Negative Log Likelihood	65.392

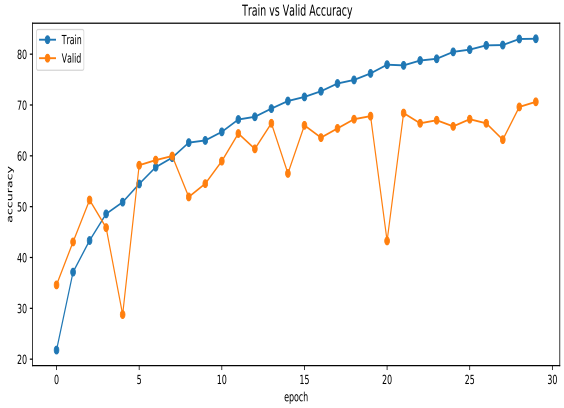

Fig. 11. Accuracy Plot for Adagrad -Negative Log Likelihood

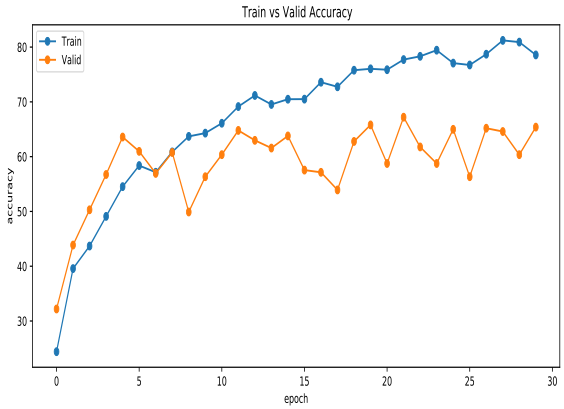

Fig. 12. Accuracy Plot for AdamW -Negative Log Likelihood

Fig. 13. Accuracy Plot for Adagrad -Cross Entropy Model

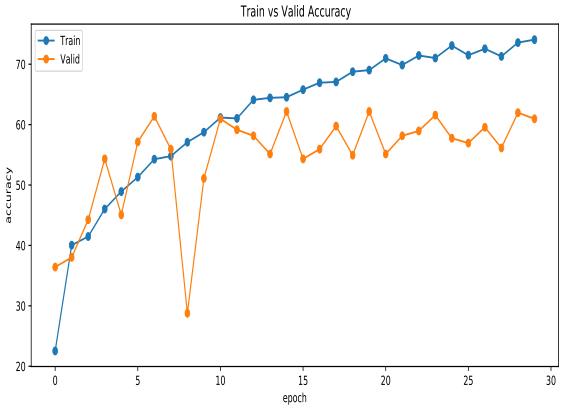

Fig. 14. Accuracy Plot for AdamW -Cross Entropy Model

```
Epoch: 1 - loss: 9.86965473606473 - accuracy: 21.789709091186523 - val_Loss: 1.8799427226185799 - val_accuracy: 34.60764694213867
Epoch: 2 - loss: 1.8744290062359 4 - accuracy: 37.1215515136718 75 - val_Loss: 1.7353754714131355 - val_accuracy: 43.058349609375
Epoch: 3 - loss: 1.6898260882922582 - accuracy: 43.340789794921875 - val_Loss: 1.5259813889861107 - val_accuracy: 51.30784606933594
Epoch: 4 - loss: 1.5635420850345068 - accuracy: 48.57569122314453 - val_Loss: 1.628514303445816 - val_accuracy: 45.87525177001953
Epoch: 5 - loss: 1.4748423099517822 - accuracy: 50.88739776611328 - val_Loss: 1.9677423909306526 - val_accuracy: 28.772636413574222
Epoch: 6 - loss: 1.3957414229710896 - accuracy: 54.46681594848633 - val_Loss: 1.347955897450447 - val_accuracy: 50.148895263671875
Epoch: 7 - loss: 1.3172546471868243 - accuracy: 57.762863159179 69 - val_Loss: 1.3295225873589516 - val_accuracy: 59.154930114746094
Epoch: 8 - loss: 1.2625924402759188 - accuracy: 59.642059326171875 - val_Loss: 1.296576648950576 8 - val_accuracy: 59.95975875854492
Epoch: 9 - loss: 1.2001941340310234 - accuracy: 62.595077514648 44 - val_Loss: 1.4292476251721382 - val_accuracy: 51.911468505859375
Epoch: 10 - loss: 1.1706021763029553 - accuracy: 63.027591705322266 - val_Loss: 1.40757604688406 - val_accuracy: 54.527164459228516
Epoch: 11 - loss: 1.1159414294220151 - accuracy: 64.71289825439453 - val_Loss: 1.3044442012906075 - val_accuracy: 58.95372390747070 7
Epoch: 12 - loss: 1.0641767186777933 - accuracy: 67.1439208984375 - val_Loss: 1.1853112541139126 - val_accuracy: 64.38631439208984
Epoch: 13 - loss: 1.0392896766403742 - accuracy: 67.68883190917969 - val_Loss: 1.2352838739752 77 - val_accuracy: 61.368209838867 19
Epoch: 14 - loss: 0.9914543234166645 - accuracy: 69.29157257080078 - val_Loss: 1.1737571991980076 - val_accuracy: 66.39839172363281
Epoch: 15 - loss: 0.9562337852659679 - accuracy: 70.78299713134766 - val_Loss: 1.3529095351696014 - val_accuracy: 56.539234161376 95
Epoch: 16 - loss: 0.9261758432501839 - accuracy: 71.58836364746094 - val_Loss: 1.146229587495327 - val_accuracy: 65.99597930908203
Epoch: 17 - loss: 0.8856932723806018 - accuracy: 72.7069320678711 - val_Loss: 1.2223378494381905 - val_accuracy: 63.581489562988 28
Epoch: 18 - loss: 0.8507910788059234 - accuracy: 74.22818756103516 - val_Loss: 1.173012100160122 - val_accuracy: 65.3923568725586
Epoch: 19 - loss: 0.8351771243980952 - accuracy: 74.92916107177734 - val_Loss: 1.1436234265565872 - val_accuracy: 67.203216552734 38
Epoch: 20 - loss: 0.8062742511431377 - accuracy: 76.21178436279297 - val_Loss: 1.1397877521812916 - val_accuracy: 67.806389892578 1
Epoch: 21 - loss: 0.7821004653260821 - accuracy: 77.92691802978516 - val_Loss: 1.6579647436738014 - val_accuracy: 43.259555816650 39
Epoch: 22 - loss: 0.7593735196760723 - accuracy: 77.79269409179688 - val_Loss: 1.1111854426562786 - val_accuracy: 68.410614257812 5
Epoch: 23 - loss: 0.7290030573095594 - accuracy: 78.74720001220703 - val_Loss: 1.1027470976114273 - val_accuracy: 66.39839172363281
Epoch: 24 - loss: 0.7174851153578077 - accuracy: 79.0902328491211 - val_Loss: 1.1205821260809898 - val_accuracy: 67.002014160156 25
Epoch: 25 - loss: 0.6823554924556188 - accuracy: 80.44742584228516 - val_Loss: 1.1609521694481373 - val_accuracy: 65.794769287109 38
Epoch: 26 - loss: 0.6649792120570228 - accuracy: 80.89485168457031 - val_Loss: 1.1256969928440094 - val_accuracy: 67.203216552734 38
Epoch: 27 - loss: 0.6446655430964061 - accuracy: 81.73004913330078 - val_Loss: 1.1118072234094143 - val_accuracy: 66.39839172363281
Epoch: 28 - loss: 0.6356304446856181 - accuracy: 81.78971099853516 - val_Loss: 1.1786682941019535 - val_accuracy: 63.179073333740234
Epoch: 29 - loss: 0.6117432898903383 - accuracy: 82.98284912109375 - val_Loss: 1.0792442075908184 - val_accuracy: 69.617706298828 12
Epoch: 30 - loss: 0.6000154681148984 - accuracy: 83.027587890625 - val_Loss: 1.0529821038795574 - val_accuracy: 70.623741149902 34
```

Fig. 15. Train-Test Accuracies and Loss during each Epoch

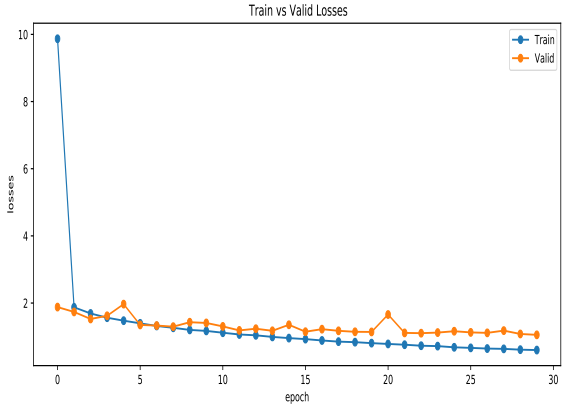

Fig. 16. Loss Function of Final Model

V. Inference

From Table 2, it can be noticed that Adagrad optimizer outperforms AdamW and Negative Log Likelihood gives better results when compared to Cross Entropy Loss Function. This is also evident in model that uses both Adagrad optimizer and Negative Log Likelihood loss function which gives the best result of 70.623% accuracy. The activation function used for the hidden layers is LeakyReLU and LogSoftmax is used as the output activation function.

References

[1] Ilya Loshchilov, Frank Hutter , "Decoupled Weight Decay Regularization," Published on 14 Nov 2017 (v1), last revised 4 Jan 2019 (this version, v3)

[2] Manisha Verma, Sudhakar Kumawat, Yuta Nakashima, Shanmuganathan Raman , "Yoga-82: A New Dataset for Fine-grained Classification of Human Poses" , Published on 22 Apr 2020

[3] Kothari, Shruti, "Yoga Pose Classification Using Deep Learning" (2020). Master's Projects. 932. DOI: https://doi.org/10.31979/etd.rkgu-pc9k

[4] Shubham Jain, April 19, 2018 - Analytics Vidhya, "An Overview of Regularization Techniques in Deep Learning", www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/.

[5] Shruti Saxena, "Yoga Pose Image classification dataset", www.kaggle.com/shrutisaxena/yoga-pose-image-classification-dataset