Pruning CNN for Faster and Efficient Classification

Shreya Chawla

CECS, Australian National University, Canberra, Australia
shreya.chawla@anu.edu.au

Abstract. CNNs are a very successful neural model but it comes at a considerable expense in both computation and storage. This paper is an effort to reduce these costs by pruning the weights of various layers by maintaining a similar accuracy. This experiment deals with binary classification of sparse historical persons by a deep convolutional neural network. The distance between extracted facial features of the image dataset are used for this research. The baseline fully connected feed-forward network achieved 80.55% average test accuracy, stored 23682 parameters and took 0.002869 seconds to train. Our proposed deep CNN without pruning achieved 79.16% average test accuracy, stored 707874 parameters and trained in 0.375123 seconds. Our model with distinctiveness pruning achieved the best test accuracy of 77.51% by storing only 707820 parameters when trained in 0.309405 seconds. We then compare these results on the basis of accuracy, time and storage costs.

Keywords: Convolutional neural network; facial features; pruning; classification; k-fold validation

1 Introduction

CNNs are one of the most popular and successful deep learning architectures. The reason for their success is that they extract meaningful features of images and convert them to lower dimensions by convolution without losing its characteristics [10]. The dataset and the task this paper solves are discussed in this section. We will see what makes this dataset and our task difficult and important in the description below.

1.1 Task

CNNs can be very huge architectures, like VGG [13], which can be difficult to store and run on small devices with limited memory like smartphones. Our goal in this paper is to optimize CNN models to train and test in lesser time and lesser memory while ensuring that our accuracy remains similar. This optimization will pave the path for bigger models to be perfected to have as few connections between neurons as possible to achieve the same task with littlest compromise in accuracy. For this we have proposed a pruning optimization whose criteria of pruning is on how distinct our neurons are. Our model acts as a dummy model with 4 hidden layers only. This is to demonstrate the difference in model's performance with and without pruning in all aspects. What makes this task difficult is our choice of dataset discussed in 1.2.

1.2 Dataset

This paper aims to compress facial features in historical human photographs by pruning to solve this classification problem efficiently [1] so that it can be implemented for other sophisticated CNNs. This paper compares the results of our proposed pruned model with a non-pruned network. The dataset consists of 36 unnamed historical images with 2 images per person only. Thus this is a sparse dataset problem, where a model has to learn from less than 5 instances (here 2) per person of an image and recognize if it is the same individual or not in a new image.

Current face recognition approaches are unable to provide sufficient interpretable information. Hence, [1] proposes features like facial feature markers (FFMs) and distances between them (in pixels) to be used for identification. The images dataset comprises 12 sets of 3 images. The sets are such that the first two out of three in a set are positively identified as being the same person, and the third is positively identified as not being a match. That is or each set of three photos, A matches B, B does not match C, and C does not match A [1]. The "distances" dataset has 36 data points from all sets such that they compare features of images (A,B), (B,C), and (C,A). Thus 3 comparisons are made for 12 sets giving us 3x12=36 examples in the "distances" dataset. The target values of these 36 examples are '1' and '0'. Here, '1' means the images are of the same person, else not depicted by '0'. Out of the 36 samples in this dataset, there are 12 instances of '1' and 24 instances of '0'. Hence, the data is imbalanced as the class distribution is not 1:1 but in a 1:2 ratio.

Table 1. Set of 14 facial markers with criteria evaluation. [1]

Facial Marker	Abbrev-iation	Criteria A	B	C	D	E	Bilateral feature / median feature
Right exocanthion	rtex	✓	✓	✓	n/a	✓	Bilateral
Right endocanthion	rten	✓	✓	✓	n/a	✓	Bilateral
Left endocanthion	lten	✓	✓	✓	n/a	✓	Bilateral
Left exocanthion	ltex	✓	✓	✓	n/a	✓	Bilateral
Nasion	nas	✓	✓	✓	✓	n/a	Median
Sub-nasale	sn	✓	✓	✓	✓	n/a	Median
Right alare	rtal	✓		✓	n/a	✓	Bilateral
Left alare	ltal	✓		✓	n/a	✓	Bilateral
Labiale superius	ls	✓		✓	✓	n/a	Median
Stomion	sto	✓		✓*	✓	n/a	Median
Labiale inferius	li	✓		✓*	✓	n/a	Median
Pogonion	pg	✓		✓*	✓	n/a	Median
Supramentale	sm	✓	✓	✓*	✓	n/a	Median
Menton	me	✓	✓	✓*	✓	n/a	Median

* Marker has potential to violate criterion C where subject has an open mouth, such as in a toothy smile. However, all photos in the dataset chosen for this research are formal photographs with closed mouths.

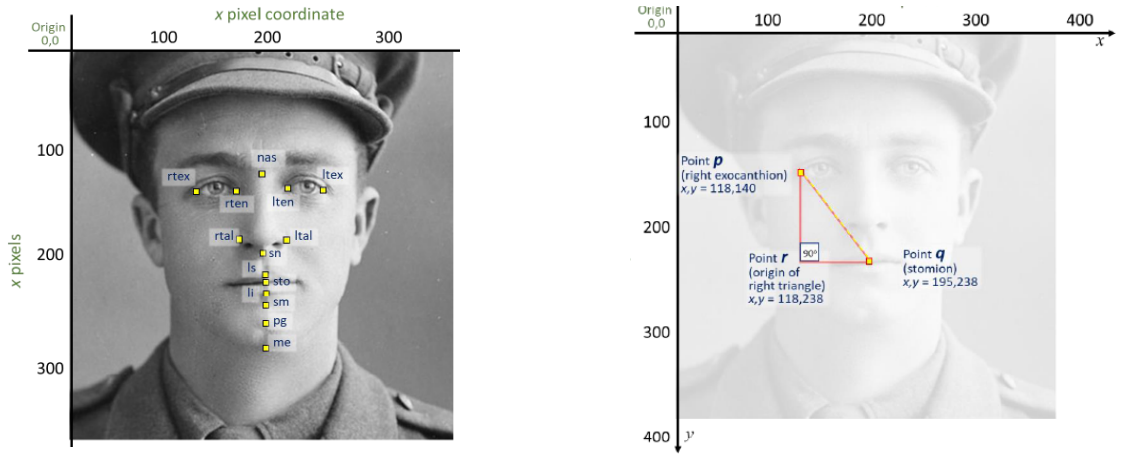
Table 1 lists the 14 facial markers and what type of features they are. Table 2 shows cosine similarity measures for distances and FFMs dataset wherein the cosine similarity for distances data is lesser than FFMs data on average. Also, the "distances" dataset gives better results than the FFMs as per [1]. Hence, we shall use distances dataset to test our models on to make the task more challenging. It also guarantees that our reported results are fair. The set of 14 FFMs are marked on Fig 1 (a) and the calculation of distances is depicted in Fig 1 (b). The "distances" dataset provides length in pixels calculated between FFM of each image with a total of 91 distances each and for the pair of images it is 182. All features are positive real numbers in the "distances" dataset.

Table 2. Cosine similarity measures for Distances and FFMs. [1]

	Distances			FFMs		
	Images A and B	Images B and C	Images A and C	Images A and B	Images B and C	Images A and C
Group 1	0.003	0.003	0.001	0.029	0.035	0.001
Group 2	0.004	0.002	0.003	0.058	0.009	0.022
Group 3	0.001	0.003	0.002	0.001	0.030	0.020
Group 4	0.002	0.001	0.002	0.015	0.001	0.019
Group 5	0.017	0.004	0.017	0.006	0.003	0.004
Group 6	0.002	0.002	0.004	0.004	0.007	0.006
Group 7	0.000	0.003	0.002	0.004	0.001	0.008
Group 8	0.073	0.070	0.007	0.031	0.005	0.027
Group 9	0.018	0.019	0.001	0.038	0.044	0.001
Group 10	0.001	0.001	0.001	0.002	0.006	0.007
Group 11	0.000	0.004	0.004	0.000	0.007	0.009
Group 12	0.001	0.006	0.004	0.000	0.045	0.048

This dataset was chosen to make the task more difficult, fair and generalizable. We are trying to improve the performance of our model by making the weights of neurons sparse. But the data itself is sparse. This will make the optimization task very challenging. Our dataset has only 36 examples but has as many as 182 dimensions. Visualizing, interpreting and analysing these many dimensions is difficult in itself. As the data is sparse, therefore we can clearly see if our model performs subpar or is actually helpful. If the outcome of our experiment is successful then this work can be generalized and easily replicated for bigger models deployed on devices with limited memory for example in [12]. We

will compare our training and testing time both. As some models nowadays require online learning, reduction in training time is of essence and lesser testing time is important for deployment.



(a) (b)

Fig. 1. (a) The FFMs are marked in one of the photographs and (b) calculating euclidean distance between FFMs at p and q pixels [1]

2 Methodology

2.1 Dataset

The binary, one-class target labels in the "distances" dataset are 1 and 0, denoting match (1) or not a match (0) respectively. Since the target variable is unbalanced, we will use stratified k-fold to get train, validation and test split which preserves the percentage of samples in each class [9]. Since the dataset is small we used cross-validation to ensure that if any biases are present in the data, it does not affect our results. The ratio of this split is shown in Table 1. The models have been trained on the training set, hyper-parameters were tuned on the validation set and tested on instances in the testing set. As the cosine similarity measures in Table 2 showed almost 0 correlation in the distances dataset, not much preprocessing was required. However, in the distances dataset, the first column of the dataset contains a dummy variable that has information about which image pairs are being compared among the three possible combinations in each set of three images (n-1 & n-2, n-2 & n-3, and n-3 & n-1). This column has been removed as a part of preprocessing as it directly gives away the target value. Since all the features of the dataset are positive, thus the gradient computed by an activation function centred at zero like ReLu could be all negative. One way to mitigate this is by normalizing data such that it is centred at zero with unit standard deviation. Another way is to use Tanh activation which is not centred at zero. [7]. Hence, the "distances" data is normalized accordingly for further use.

Table 3. Parameters and hyperparameters required for both models.

	BaseLineNet model	DeepCnnNet model
Parameters	Value	
Number of folds	6	6
Split of N (=36) data points into train : val : test ratio	4 : 1 : 1 (= 24 : 6 : 6)	4 : 1 : 1 (= 24 : 6 : 6)
Number of features of each sample	182	182
Batch size	same as input	same as input
Number of epochs	200	200
Loss function	Cross Entropy Loss	Cross Entropy Loss
Optimizer	Adam	Adam

Initial Learning Rate	0.005	0.005
Learning Rate Halved at epochs	10, 40	15, 40, 80
Number of Hidden layers	1 (fully-connected)	4 (3 convolutional + 1 fully connected)
Number of Neurons in all Hidden layers in a list	[128]	[512, 128, 64] (convolution layers) + [32] (fully connected layer)
Activation Function for all Hidden layers	Sigmoid	Tanh
Filter size for all Convolutional Layers	no convolution layer	[(5,), (5,), (3,)]
Output Neurons	2	2
Activation Function for Output layer	Sigmoid	Sigmoid

The batch size for all models is the same as the input. That means for training, it is 24. for validation and testing the batch size is 6. Adam optimizer is used to optimize all the models as it allows for fine-tuning of learning rate. As our dataset has a binary target, hence cross-entropy loss is used. Also, because our output is of dimension 2, we use sigmoid activation on the output layer to calculate the probability for each class.

2.2 Baseline Model

The baseline model called BaseLineNet is used to compare the results of other models and provide a baseline accuracy. It consists of an input layer, followed by one fully connected feed-forward hidden layer with Sigmoid activation followed by an output layer with sigmoid activation. The learning rate of the Adam optimizer is reduced to half after the 10th and 40th epochs for better training. The rest of the hyperparameters are in Table 3 for comparison. All these hyperparameters have been tuned by several trial and error experiments on the validation set. Fig 2 shows the model architecture.

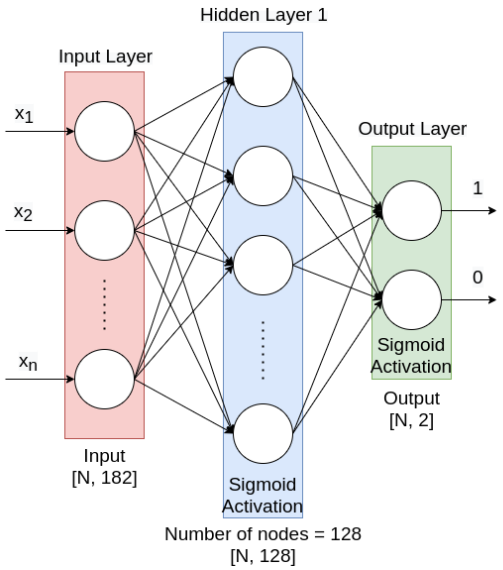


Fig. 2. The architecture for the baseline model has three layers in total with one hidden layer. The 182 in input dimension is the number of features in data. The hidden layer is fully connected to input and output layers with 128 neurons. The output layer has 2 possible outputs: 0 depicting the input information for different images or 1 indicating the photographs are of the same person.

2.3 Deep Convolutional Neural Network model

The model depicted in dataset paper [1] has been modified and is called DeepCnnNet. It has been trained, validated and tested over 6 stratified k-folds rather than 10. It has 4 hidden layers other than input and fully connected output layers. These hidden layers are composed of three 1-D convolutional layers and one fully connected feed-forward linear layer. The first fully connected hidden layer scales the feature map down to 32 dimensions and the second one reduces it to 2

(same person '1' or not '0'). Tanh activation functions are used for all hidden layers whereas Sigmoid activation is used for the output layer. Our target variable is of binary class so it makes sense to use sigmoid to get probabilities in the output layer. The model is trained using Adam optimizer wherein after the 15th, 40th and 80th epochs, the learning is reduced to half. The rest of the hyperparameters are in Table 3 for comparison. All these hyperparameters have been tuned by several trial and error experiments on the validation set. Fig 3 shows this model's architecture.

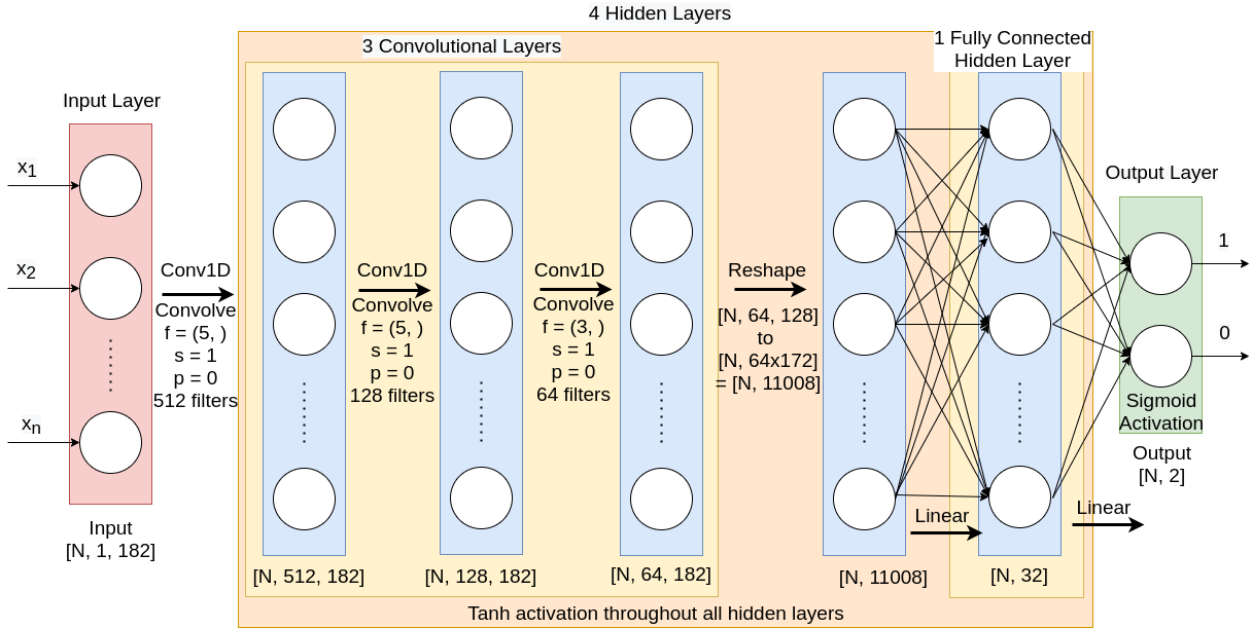


Fig. 3. The architecture for the deep CNN model has 6 layers in total with 4 hidden layers. The N inputs are convolved with 512 filters of size (4,) then 128 filters of shape (5,) and then with 64 filters of shape (3,) without padding and with stride 1. The output of the last convolution layer is reshaped then passed to the 1 fully-connected hidden layer which is then passed to the fully connected output layer. We get the output of dimension [N, 2].

2.4 Pruning the Deep CNN model

Pruning has been performed after the network has converged, and then retrained after pruning, for 200 epochs, to check the difference in the result. This retraining allows for the model to be fine-tuned now that weights are sparser. This process continues until no more weights are left to be pruned as shown in the pipeline in Fig. 4.

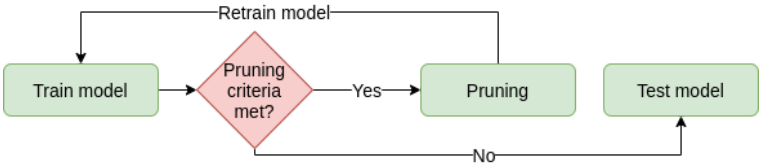


Fig. 4. Pipeline for pruning the deep CNN model based on Cosine similarity of activations. If the angle lies in a certain range, then the weights are pruned accordingly and then restrained until no such neuron is left. Thus the resulting network will be distinct.

The criteria to be pruned is cosine similarity between activation layers' outputs. If the angle between 2 activations is less than the pruning angle, for example 45 degrees, then one of the connections is pruned as they are both similar and hence redundant. However, if the angle is more than 180-pruning angle, for example 165 degrees, then the activations are complementary in which case both of these are pruned. The similar nodes perform the same task and hence removing one of them by adding its weights and biases to the kept neuron will make the architecture small and in turn, reduce time to compute. Simultaneously, the complementary activations nullify each other. Removing connections between neurons in this manner ensures distinctiveness of neurons is maintained [5]. The angle between the outputs of activations is computed using the equation below (eq 1).

$$\boldsymbol{similarity} = \text{Acos}\left((\boldsymbol{A} \,.\, \boldsymbol{B})/(||\boldsymbol{A}||x||\boldsymbol{B}||)\right) \qquad \text{eq 1}$$

The hyperparameters for pruning the model are kept the same as in the deep CNN model. The reason is that pruning is used as an optimization method. Hence, to compare better, the same model parameters as in Table 3 are used for the deep CNN model with pruning.

3 Results and Discussion

The models have been experimented on over 10 times and the average training, validation and test accuracies and losses achieved are reported across all folds for all models. These are reported in Table 4 below. Although all the results for the BaseLineNet model are better than the DeepCnnNet model, since our goal requires us to compare the accuracies, and time-space overheads of CNNs, we will proceed to prune only the deep CNN model. The time taken to train and test data using each architecture and the number of parameters required by each model is in table 5 below. The number of parameters indicates the amount of storage space required to store the model. These performance parameters are compared in Fig 5 bar plots. The results are discussed and analysed in sections 3.1, 3.2, and 3.3. Appendix A contains the learning curves of all models for all 6 folds.

Table 4. Accuracy and loss for all architectures discussed in section 2

		BaseLineNet	DeepCnnNet without pruning	DeepCnnNet with pruning
Accuracy in percentage (average of several experiments and over different epochs)	Training	91.67	90.28	90.23
	Validation	83.33	74.99	-
	Testing	80.55	79.16	77.51
Cross Entropy Loss (average of several experiments and over different epochs)	Training	0.462422	0.450637	0.46209
	Validation	0.506855	0.558106	0.57148
	Testing	0.513476	0.567455	0.58526

Table 5. The performance overheads are reported for all models here. For the DeepCnnNet model with pruning, the training time refers to the time taken to train the model once re-training has been performed. All the results are the mean of several experiments.

	BaseLineNet	DeepCnnNet without pruning	DeepCnnNet with pruning
Training time (in seconds)	0.002303	0.406721	0.309405
Testing time (in seconds)	0.079165	0.009709	0.009752
Number of Parameters	23682	707874	707820

3.1 Baseline model

The baseline model achieves better results than expected. One possible reason is that the dataset is quite small and has only 2 target classes. Although its validation and testing loss are lower than the DeepCnnNet model, its training error is slightly more. The losses are around 0.5 with its training loss being the lowest which verifies that our baseline model's hyperparameters are well-tuned and it is not over or underfitting the data. Interestingly, the time taken by BaseLineNet to test is approximately 30 times more than training. The reason for this disparity is possibly because of the number of connections between the hidden layer and output layer being very high. The number of parameters in our non-pruned CNN is about 30 times greater than parameters in the baseline feed-forward network. This difference can be attributed to the size of our model.

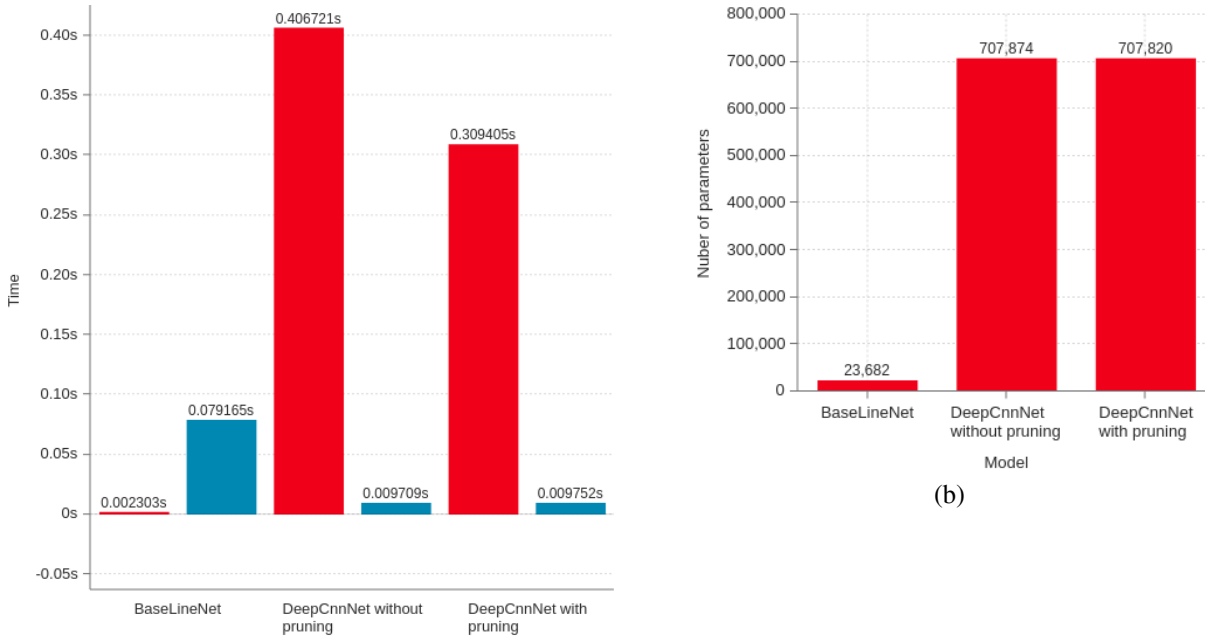


Fig. 5. (a) Comparison of time taken to train or retrain (red) and test (blue) and (b) number of parameters for the 3 models

3.2 Deep CNN model without pruning

The accuracies and losses for the deep CNN model without pruning are reported in Table 4 above. These results are comparable to the results of the dataset paper [1]. The validation accuracy being similar to the results in [1, 11]. As our model is a deep CNN implementation rather than a 2 hidden layer feed-forward network defined in [1], our model outperforms in test and training accuracies. The decrease in the number of folds from 10 to 6 also helps to generalize better. The splits in train, validation and test sets are better defined. The model has more examples to test in our proposed architecture compared to only 1-2 testing examples in [11]. The time taken to train the baseline feed-forward network is roughly 200 times lesser than the deep CNN model without pruning. This huge difference is the result of the complexity of models, where one has 4 times the hidden layers of the other.

3.3 Deep CNN model with pruning

No neurons were found to be removed at angles less than 45 degrees. The testing time as expected is lesser with pruning than without. The time to re-train the pruned CNN model is lower than that of training without pruning. The number of parameters is also lesser. The accuracy has dropped by almost 1% but that is understandably caused by sparsity of connections. We were able to achieve our goal to decrease time and space for our CNN model.

4 Conclusion and Future work

In this paper, we showed that pruning features of the input and pruning neurons of a hidden layer based on distinctiveness decrease a significant amount of parameters with a little drop in accuracy as a trade-off. With the advancement in technology, huge models with billions of parameters are deployed which could be made more efficient if pruned according to how distinct activation values are. Although the accuracy of our model needs to be improved further, we have proven that our model is able to cut down the computation and memory costs. Our proposed method is useful specifically in cases where a large model needs to be deployed on devices with small memory like mobile phones and drones enabling simpler and faster models to be trained. An example for this is Google Photos which trains and runs an AI algorithm for recognition by using user-given labels of people in images [12]. Naturally, as the connections become sparser, the task performance will progressively degrade thus choice of angle between activations is important. But this trade-off of sparsity vs. performance can reduce time and save memory.

There is plenty of room for improvement of our method. Future work involves using the proposed method together with a deep auto-encoder to further improve its efficiency. Secondly, by extending this work by using different pruning

techniques like pruning on badness factor [5] and pruning filters of CNN [6]. Pruning of connections between input layer and first hidden layer can also be tested [8]. The dataset used is unbalanced and hence handling it by resampling the dataset or generating synthetic samples could help. Also, the SMOTE technique [4] can also be tried to balance the dataset. Issues of pose (pitch, roll and yaw), resolution and camera angle are important problems to be solved. Adding pooling, regularization or batch normalization can also be tested to improve weights and features selection.

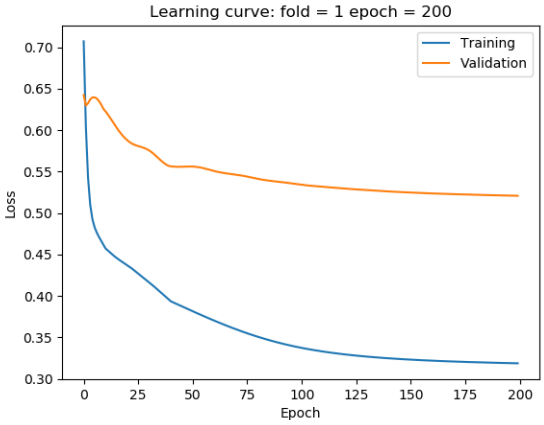
References

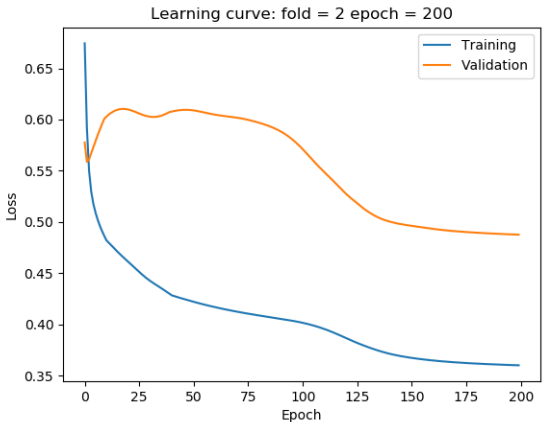
1. Caldwell, S.,: Human interpretability of AI-mediated comparisons of sparse natural person photos, CSTR-2021-1, School of Computing Technical Report, Australian National University (2021)
2. Gedeon, T.D., Harris, D.,: Progressive Image Compression, In: IJCNN International Joint Conference on Neural Networks, IEEE, col. 4, pp. 403-407 (1992)
3. Gedeon, T.D., Class Tutorials, COMP8420, College of Engineering and Computer Science, Australian National University, Canberra, ACT (2021)
4. Chawla, N. V. Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P.,: SMOTE: Synthetic Minority Over-sampling Technique, In: Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, AI Access Foundation (2002)
5. Gedeon, T.D., Harris, D.,: Network Reduction Techniques, In: Proc. Int. Conf. on Neural Networks Methodologies and Applications, AMSE, San Diego, vol. 2, pp. 25-34 (1991)
6. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.,: Pruning Filters for Efficient ConvNets, arXiv.org (2017) [Online]. Available: https://arxiv.org/abs/1608.08710. [Accessed: 22-May-2021]
7. Nwankpa, C., Ijomah, W., Gachagan, A., Marshall, S.,: Activation Functions: Comparison of trends in Practice and Research for Deep Learning, arXiv.org, (2018) [Online] Available: https://arxiv.org/abs/1811.03378 [Accessed: 29-May-2021]
8. Kawatra, A.,: Activation Functions: Neural Network pruning using hidden layers output's unit vector angles and autoencoder for feature selection, In: 1st ABCs ANU Bio-inspired Computing conference, (2018) [Online] Available: http://users.cecs.anu.edu.au/~Tom.Gedeon/conf/ABCs2018/paper/ABCs2018_paper_95.pdf [Accessed: 31-May-2021]
9. Refaeilzadeh, P., Tang, L., Liu, H.,: Cross-Validation, [Online] Available: https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_565 [Accessed: 1-June-2021]
10. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T.,: Recent advances in convolutional neural networks, In: Pattern Recognition, (2017) [Online] Available: https://www.sciencedirect.com/science/article/abs/pii/S0031320317304120 [Accessed: 01-Jun-2021]
11. Chawla, S.,: Neural Network with Pruning for Comparison of Sparse Historical People Photographs, In: 4th ABCs ANU Bio-inspired Computing conference, (2021)
12. Nieuwenhuysen, P.,: Information Discovery and Images A Case Study of Google Photos, In: IEEE Xplore (2018) [Online] Available: https://ieeexplore.ieee.org/abstract/document/8485238 [Accessed: 01-Jun-2021]
13. Simonyan, K., Zisserman, A.,: Very Deep Convolutional Networks for Large-Scale Image Recognition, In: arXiv.org, (2015) [Online] Available: https://arxiv.org/abs/1409.1556 [Accessed: 01-Jun-2021]

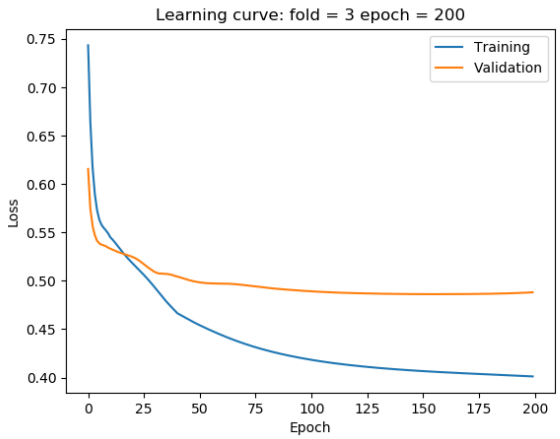
Appendix A:

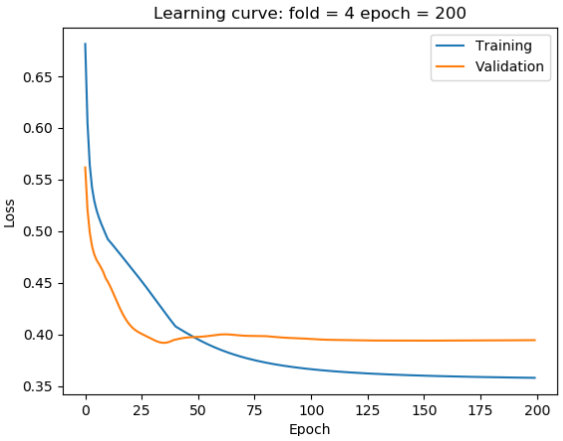
All the learning curves for both architectures across all 6 folds for 200 epochs are displayed here. These curves show that all our models are well-tuned for most folds. As training might be dependent on data for small datasets, hence it is not possible to have good learning curves for all folds. But for more than 3 out of 6 curves we get good learning curves.

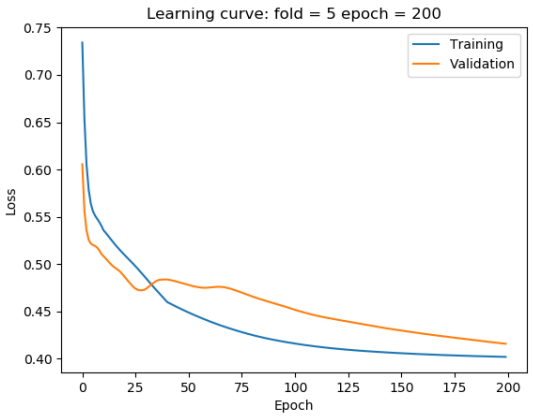
1. Baseline - Feed-forward network

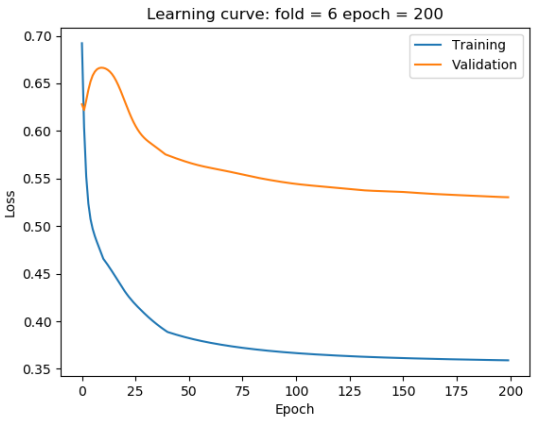




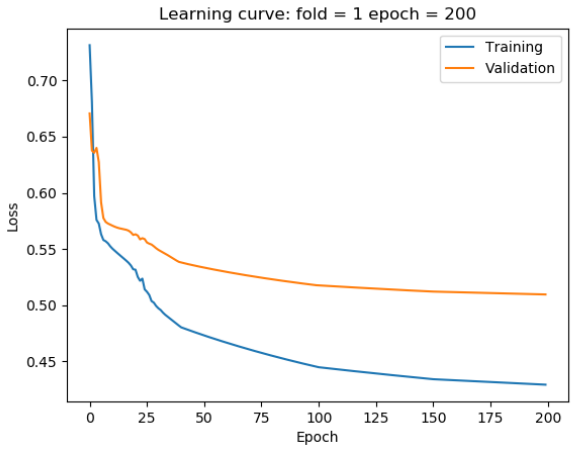


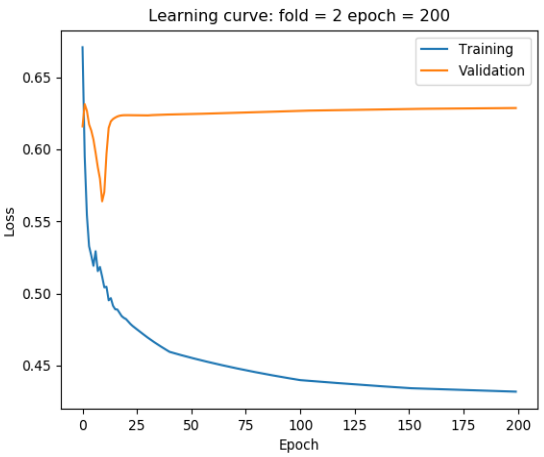


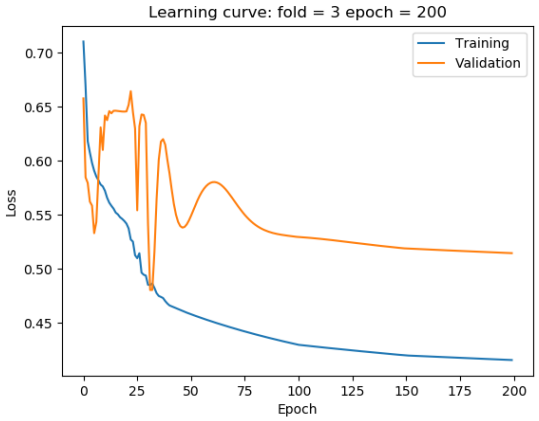


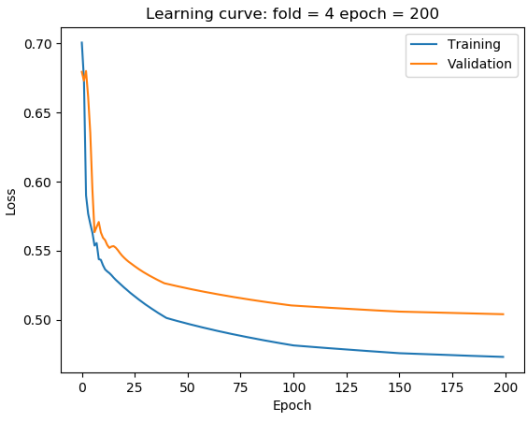


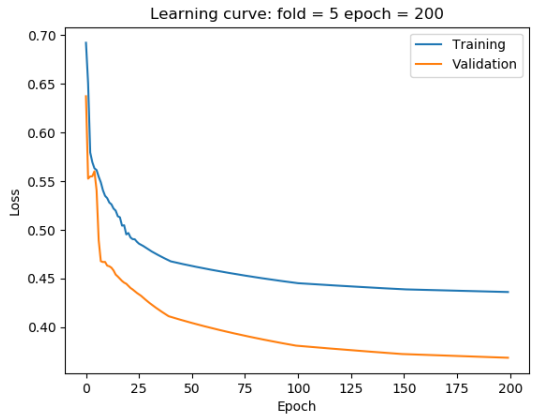
2. Deep CNN without pruning

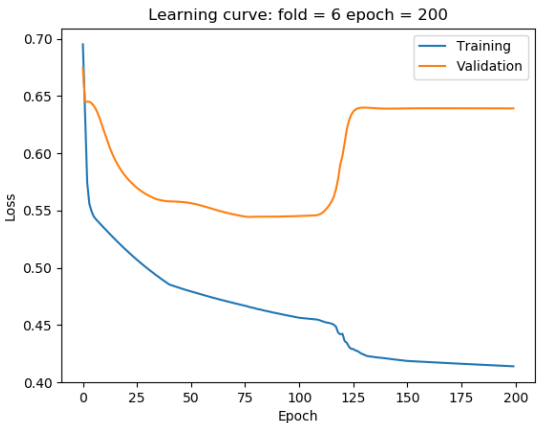





Learning curve: fold = 3 epoch = 200
Training
Validation
Loss
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0 25 50 75 100 125 150 175 200


Learning curve: fold = 4 epoch = 200
Training
Validation
Loss
Epoch
0.70
0.65
0.60
0.55
0.50
0 25 50 75 100 125 150 175 200


Learning curve: fold = 5 epoch = 200
Training
Validation
Loss
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0 25 50 75 100 125 150 175 200


Learning curve: fold = 6 epoch = 200
Training
Validation
Loss
Epoch
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0 25 50 75 100 125 150 175 200