Modular Neural Networks using Multiple Gradients

Adjorn van Engelenhoven
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
a.vanengelenhoven@student.utwente.nl

ABSTRACT

Artificial neural networks are often not able to show how they generate their results. By defining subtasks the interpretability of neural networks can be improved. Modular neural networks can use hyperparameters to define subtasks with which it should solve the problems it is trained for. However, parts of the Modular Neural Network are trained solely on the accuracy of their subtask and not the amount of useful information this subtask can provide for the entire task. In this paper the Shared Layer Modular Neural Network is proposed which could be an improvement on the standard Modular Neural Network. The accuracy could be improved by combining the layers before the output of the subtask so more relevant information of these subtasks can be combined. Furthermore, the accuracy could be improved by optimizing the weights based on a combination of the loss functions of both the subtask and the global task. In this paper, the Shared Layer Modular Neural Network is developed and tested to see if it is an improvement compared to other models. CIFAR-10 was used in the evaluation of the model. If this model is an improvement, a more accurate explainable neural network has been created which can help solve problems that are currently not completely understood.

Keywords

Modular Neural Network, Multiple Gradients, Explainable Neural Networks, Neural Network Architecture

1. INTRODUCTION

Many seemingly complex tasks in the brain can be divided into subtasks [8]. Modularity in the brain is thought to give the ability to handle different tasks with common subtasks [18]. Artificial neural networks that use weight pruning, which is similar to the brain's synaptic pruning, also seem to have high modularity [7]. However, it is unknown what problems the modules in these neural networks are solving, which is not useful when the inner workings of the neural network need to be analyzed. Modular neural networks (MNNs) are made of multiple independently trained neural networks each of which has a predefined subtask given by the designer. These clear predefined subtasks improve the transparency and explainability of the entire neural network. The individual subsystems can be separated and analyzed to understand their function which can be used to understand the entire system. Explainable neural networks are especially useful when solving a task with high accuracy of which we do not know a standardized solution. However, the training of the sub-neural networks is based only on the success of their subtask and not the success of other sub-neural networks or the whole system, which could be an oversimplification of the problem.

To complete the entire task MNNs combine the output of all sub-neural networks through some intermediary [2]. This can for instance be done through gating networks, voting schemes or other neural networks. In this paper, the focus will be on the use of neural networks as the intermediary. The intermediary network uses the output of the sub-neural networks as input to complete the entire task. Intermediary networks are trained after all the sub-neural networks are trained. This kind of MNN will from now on be called the standard MNN.

The intermediary network could also use the output of the sub-neural network's hidden layers as input. The intermediary network then needs to learn the entire task from neurons that were used to solve the subtasks. This will increase the number of parameters in the network but, as seen in this paper, it can also increase the accuracy. This type of MNN will from now on be called the Hidden Layer Modular Neural Network (HL-MNN).

In this paper, a new type of Modular Neural Network using an intermediary network is proposed. The Shared Layer Modular Neural Network (SL-MNN) is made with the intent to increase the accuracy of Modular Neural Networks while staying interpretable. The SL-MNN does not receive the output of the sub-neural network but rather it receives the output of their hidden layers. In the SL-MNN the sub-neural networks share their layers with the intermediary network. The weights in these shared layers are updated based on both the gradient of the sub-neural network and the intermediary network. These two changes were made with the idea to force the hidden layers to contain information with which is useful for both the sub-neural network and the intermediary network.

Figure 1. Example of the CIFAR-10 dataset

33rd Twente Student Conference on IT July 3rd, 2020, Enschede, The Netherlands.

The CIFAR-10 dataset is used in all experiments, an example of the dataset is given in Figure 1 [12]. Much higher accuracy can be achieved by using Convolutional Neural Networks or Residual Neural Networks [11, 9]. However, the goal of this paper is not to achieve higher accuracy than these networks but rather to improve current MNNs using only fully connected layers.

In section 2 the research questions are stated. In section 3 background information on Artificial Neural Networks and Modular Neural Networks is given. In section 4 the SL-MNN is explained in more detail. In section 5 the experiments and results are shown. In section 6 a discussion is presented and in section 7 a conclusion is made.

2. RESEARCH QUESTIONS

This research will focus on the creation and optimization of this modular neural network model, the comparison of accuracy between this model and similar models, and the significance of the intermediate outputs to the final output.

2.1 Optimization

To be able to fairly compare the models on accuracy the proposed model first needs to be optimized. This optimization can be done in two parts. The combining of multiple gradients and the definition of the subtasks. As such the following research questions (RQ) should be answered.

RQ1 : How does the way of combining the gradients of the final and intermediate output influence the accuracy of the entire model?

RQ2 : Which types of subtasks can increase the accuracy of the entire model?

2.2 Accuracy

When the model has been optimized its accuracy can be compared to others models. The accuracy of the SL-MNN is of high importance since MNNs are already interpretable [2]. The SL-MNN should be compared to an MNN and HL-MNN which use the same subtasks to determine whether an improvement has been made. The SL-MNN should also be compared to a Multi-Layer Perceptron (MLP) to determine whether the adding subtasks actually improves accuracy. From this we can conclude that the following research question needs to be answered.

RQ3 : How accurate is the SL-MNN compared to a normal MLP, an MNN and HL-MNN which use the same subtasks?

2.3 Interpretability

After the SL-MNN has been trained it should be checked whether the sub-neural networks (SNNs) can explain the output of entire model. By removing a certain SNN from the SL-MNN it can be seen how much of an impact this SNN had on the final classification. Thus explaining how the SNN helps classify some input. As such the following questions should be answered.

RQ4 : How does the classification accuracy of specific classes change as different types of sub-neural networks are removed?

3. BACKGROUND

Some background knowledge of neural networks is needed to understand this paper. This background knowledge is specifically about supervised learning, neural networks in general, and modular neural networks.

3.1 Artificial Neural networks

Artificial Neural Networks (ANN) were first made as an imitation of the neurons and synapses in the brain. While ANNs do follow some of the basic principles of neurons, they are a simplified formalization of the brain. ANNs are a collection of connected neurons that are loosely based on the brain. A connection can transmit a signal from one neuron to other neurons. In ANN this signal is a number. Every connection has a weight associated with it, as learning in the ANN progresses this weight can change value. The weight of a connection can increase or decrease the signal. The value of the signal a neuron transmits, the activation, is based on the signals it receives from other neurons. The activation of a neuron is calculated using the sum of all received signals as input for an activation function. There are many activation functions with the most notable being: Rectified Linear Unit, Sigmoid, Hyperbolic Tangent, and Softmax [14, 15]. When the activation of a neuron has been calculated it then sends out signals based on this activation. A group of neurons that are not interconnected is called a layer. A neural network takes input from an input layer feeds forward signals to the other layers until it gets to the output layer.

A loss function is used to determine how far the current output is from the desired output for any given input. The result of this loss function is then used by the backpropagation algorithm to calculate a gradient. [5] This gradient can be used by an optimization algorithm to create updates to the weights such that the current output will be closer to the desired output. Some notable optimization algorithms are : Stochastic Gradient Descent, Adaptive Moment Estimation and RMSprop [19, 10, 20]. Although all of these algorithms minimize the loss through gradient descent they do so in different ways. By repeatedly using backpropagation and an optimization algorithm a neural network can be taught to give the right output for input it has never processed. At some point in this repetition the loss no longer decreases, meaning that the ANN has converged and is no longer improving.

3.2 Modular Neural Networks

Modular Neural Networks draw further from the biological inspiration of neural networks and emulate the modularization of the brain. In the brain there are many regions which all have different tasks, in a similar way neural networks can be made to divide one big task into many smaller tasks. This can be done by creating many independent sub-neural networks which solve the smaller tasks. The sub-neural networks then send the results to an intermediary which combines all results into one. The most biologically plausible intermediary would be another neural network, but other intermediaries also exist. One of those other intermediaries is a gating network. A gating network is used in a technique called Mixture of Experts (MoE) [13]. First, the experts are defined to divide the problem space, then they are trained to solve the specific problem space they have been assigned. These experts can be neural networks but they can also be other learners like: Decision Trees, Support Vector Machines, and Bayesian Networks. Then the intermediary, the gating network, is trained to decide which expert to use on which input. Designing Modular Neural Networks can have a plethora of advantages, including speed up in training and interpretability [2]. A speed-up in training can be achieved by parallelizing the training of the sub-neural networks. Interpretability is achieved since the sub-neural networks' output should match the result of the subtask. The moment this output is given to the intermediary we can de-

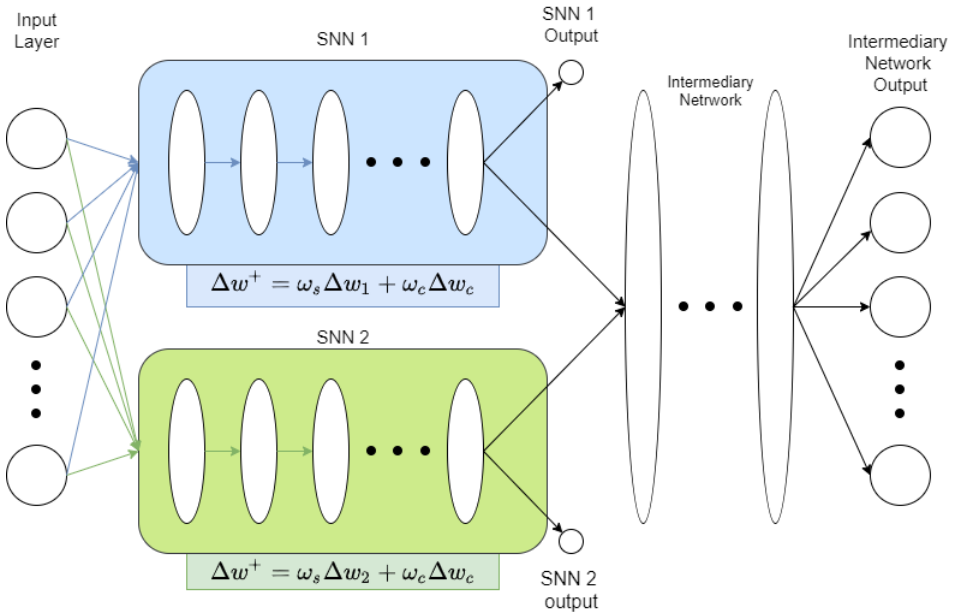

Figure 2. A visualized graph of the Shared Layer Modular Neural Network

termine how the subtasks are combined by analyzing the weights. Modularity can be designed into a neural network but it can also be achieved naturally. Weight pruning has been proved to also create a degree of modularity in neural networks. [7] This technique however does not always receive any of the previously mentioned advantages.

4. SL-MNN

In this section the Shared Layer Modular Neural Network is explained. This is done in three parts. In Figure 2 a visualization of the SL-MNN can be found. In this graph the weights that use the new gradient application are colored blue and green.

4.1 Subtasks of SNNs

One-vs-Rest and One-vs-One subtasks.

SNNs can be used to split the K-class problem non-modular neural network face into multiple subtasks that can be solved in a modular way. When it comes to image classification these subtasks can be defined as one-vs-rest (OvR) or one-vs-one subtasks (OvO) [3]. OvR SNNs are binary classifiers which distinguish one class from all remaining classes. In these SNNs the remaining classes are considered as one class. OvO SNNs are also binary classifiers but distinguish between only 2 classes and disregard all the other remaining classes. If there are K classes in a dataset then K OvR SNNs or K(K-1)/2 OvO SNNs to distinguish between all classes.

Superclass task.

Another type of SNN that is used is the superclass classifier. This SNN regards multiple classes with similar physical features as one class. In the used dataset, CIFAR-10, this SNN classifies images as either animals or vehicles. Although this SNN cannot distinguish specific classes it could decrease the number of misclassifications between superclasses which in turn could improve the final accuracy.

Class imbalance problems in SNNs.

The CIFAR-10 dataset is balanced, i.e. all classes have an equal amount of images in the dataset. However, to the SNN the dataset does seem imbalanced. An OvR SNN in CIFAR-10 would view 90% of the images as one class, thus creating an imbalanced dataset for the SNN. If no precautions are taken this usually creates SNNs that consider every image as 'the rest'. To ensure that the SNN learns all classes evenly, class weights were used in the loss function [2].

4.2 Feeding forward intermediate layers

The SNNs in any Modular Neural Network essentially compress the input image to a single value. This value represents the result of the subtask that a specific SNN is solving. The compression is could be lossy, meaning that the input that has been compressed can no longer be returned to the original input. The intermediary network of an MNN receives the output from the SNNs and creates the final output. However, because all SNNs compress the input image some important information could have been lost. Compressing the input less could result in less important information being lost. The output of an SNN is directly influenced by the layer that comes before it. This layer holds all the necessary information to create the output of the SNN. Feeding forward the output of this layer to the intermediary network the input is less compressed. The intermediary network then uses the information from these layers to create the final output. The intermediary network essentially relearns how to solve all subtasks from the hidden layers in the SNNs, but it can use the information of all SNNs to do so.

4.3 Gradient application

While normally the SNNs are trained independently it could prove useful to also train the SNNs based on the value of the context that they feed forward to the intermediary network. Here I propose a technique which uses multiple gradients to try and achieve this. Let W be all the weights in a neural network, let n be the number of SNNs and let W_k be the weights of SNN k where $k \in \{1..n\}$.

The set of weights of all SNNs are disjoint, i.e. :

$$\forall i, j \in \{1..n\} i \neq j \iff W_i \cap W_j = \emptyset$$

Let W_c be the weights of the intermediary network and let W_k^o be the weights connecting to the output layer of SNN k. Then in the proposed model the following holds

$$\forall k \in \{1..n\}, W_c \cap W_k = W_k \cap \neg W_k^o$$

All the weights in W_k except for the weights in W_k^o are in W_c. When training the SL-MNN every training step an update Δw is computed for every model's weights. So the weights in $W_c \cap W_k$ have two updates associated to them. These updates are then combined into one update

$$\Delta w^+ = \omega_s \Delta w_k + \omega_c \Delta w_c$$

Where ω_s denotes the update weight assigned to the SNNs and ω_c the update weight of the intermediary network. By choosing an ω_s and ω_c such that $\omega_s + \omega_c = 1$ it is ensured that the weights in $W_c \cap W_k$ can not experience a higher learning rate than the other weights.

5. EXPERIMENTS AND RESULTS

To answer the research questions experiments are done on the SL-MNN. As mentioned before all experiments are done using the CIFAR-10. dataset [12]. Keras and Tensorflow were used to implement and experiment on all networks [6, 1]. The used program can be found at `https://github.com/adjorn-e/SL-MNN`. The current implementation of the SL-MNN is not well optimized, accordingly only a small amount of experiments could be done. In the following experiments all models used the same hyperparameters. Stochastic Gradient Descent with Nesterov momentum was used as the optimizer [16, 19]. To ensure that the models are compared fairly the same stopping criteria were used. 10% of the training data was used as validation data. Every epoch the validation data was used to determine if the validation loss is still decreasing. If the validation loss has not decreased within 20 epochs training ends, this is called early stopping [4]. In these experiments it was chosen to use 10 OvR SNNs to create the baseline SL-MNN. If OvO SNNs were to be used then 45 SNNs would have to be created and trained. Training and managing these SNNs would take up significantly more time and as such it was chosen to use OvR SNNs instead.

5.1 Weight updates impact on accuracy

To see which weight updates are optimal four different variations of ω_s and ω_c were tested and compared. The tested weight updates were : $\omega_s = 1.0, \omega_c = 1.0$; $\omega_s = 0.5, \omega_c = 0.5$; $\omega_s = 0.3, \omega_c = 0.7$ and $\omega_s = 0.7, \omega_c = 0.3$. All variations were trained fifteen times, with each variation having the same architecture.

For $\omega_s + \omega_c > 1$ if both updates are negative or positive then this is similar as having a higher learning rate in the layers which are shared. While having different learning rates per layer is not new, having a high learning rate in early layers is generally not recommended [17].

In Table 1 it can be seen that choosing weight updates such that $\omega_s + \omega_c > 1$ holds similar characteristics to having a higher learning rate. The SL-MNNs with $\omega_s = 1.0, \omega_c = 1.0$, converge more quickly but to a slightly lower accuracy. As can be seen in figure 2, variations where $\omega_s = 0.5, \omega_c = 0.5$ and $\omega_s = 0.3, \omega_c = 0.7$ seem to converge almost identically. The model with $\omega_s = 0.7, \omega_c = 0.3$ seems to take slightly longer to converge, without any gain in accuracy of the final model. This time to convergence can also be seen in Table 1, the final accuracy of all models are quite similar, however, the accuracy of the $\omega_s = 1.0, \omega_c = 1.0$ model seems to be slightly lower than all the others.

Table 1. Epoch distribution and Accuracy distribution of SL-MNNs using different update weights

Update weights	Epoch Distribution		Accuracy Distribution	
	μ	σ	μ	σ
$\omega_s = 1.0, \omega_c = 1.0$	131.80	28.02	48.22%	0.719%
$\omega_s = 0.5, \omega_c = 0.5$	158.87	19.78	49.26%	0.330%
$\omega_s = 0.3, \omega_c = 0.7$	174.80	23.50	49.14%	0.527%
$\omega_s = 0.7, \omega_c = 0.3$	166.6	28.43	48.71%	0.725%

5.2 Impact of subtasks on SL-MNN accuracy

To determine which subtasks can increase the accuracy of the SL-MNN four different variation were trained. SL-MNN$_1$ has 10 OvR SNNs, this is the baseline SL-MNN to which the others are compared. SL-MNN$_2$ has 10 OvR SNNs and an additional superclass SNN that distinguishes animals and vehicles. SL-MNN$_3$ has 10 OvR SNNs and two additional OvO SNNs. The first OvO SNN classifies an image as an airplane or a ship. The second OvO SNN classifies an image as an automobile or a truck. SL-MNN$_4$ has 10 OvR SNNs, a superclass SNN and the two OvO SNNs from SL-MNN$_3$. SL-MNN$_1$ was used as baseline to see if the other variations were improvements. SL-MNN$_2$ was used to determine whether the addition of a superclass SNN is an improvement on the baseline. SL-MNN$_3$ was used to determine whether OvO SNNs can lower the misclassification rate between the two classes it classifies. SL-MNN$_4$ was used to determine whether combining all the additional SNNs can further increase the accuracy of the SL-MNN. For this experiment every single SNN has an architecture of (51x51x1), the update weights $\omega_s = 0.5$ and $\omega_c = 0.5$ were chosen.

The SL-MNNs which have the superclass SNN seem to have a higher accuracy than the others. The two OvO SNNs do not seem to positively affect the final accuracy of an SL-MNN. As seen in Figure 4, the two OvO SNNs do seem to boost the validation loss decrease in the early stages of training. It can be noticed that the SL-MNN$_4$ has the early boost of SL-MNN$_3$ and also converges to

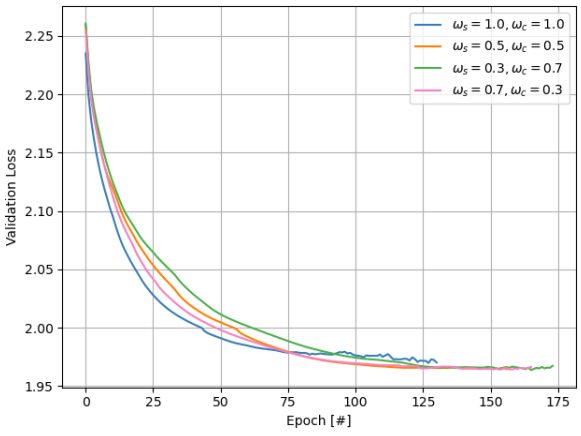

Figure 3. Average validation loss per epoch of SL-MNNs using different update weights

Table 2. Epoch distribution and Accuracy distribution of SL-MNNs using different subtasks ($N = 10$)

Variation	Parameters	Epoch Distribution		Accuracy Distribution	
		μ	σ	μ	σ
SL-MNN$_1$	1860512	164.7	29.17	49.10%	0.54%
SL-MNN$_2$	2045999	167.0	23.20	49.71%	0.26%
SL-MNN$_3$	2231486	153.0	30.85	48.79%	0.89%
SL-MNN$_4$	2416973	152.9	25.01	49.37%	0.83%

Table 3. Superclass predictions per variation ($N = 10$)

Variation	False Vehicles	True Vehicles	False Animal	True Animal
SL-MNN$_1$	722.4	3277.6	746.8	5253.2
SL-MNN$_2$	657.2	3342.8	652.8	5347.2
SL-MNN$_3$	642.3	3357.7	802.7	5197.3
SL-MNN$_4$	671.1	3328.9	637.5	5362.5

the same final accuracy of SL-MNN$_2$. Furthermore, the additional OvO SNNs do not seem to lower the misclassification rate between airplanes and ships, and automobiles and trucks. In Table 3 the superclass predictions of each model can be seen. It can be noticed that the addition of a superclass classifier lowers the amount of misclassified images between superclasses. I.e. animals and vehicles are classified as the right superclass more often.

5.3 Comparison with other methods

To determine whether the SL-MNN is an improvement, it is compared the three other neural networks: an MNN, an HL-MNN, and an MLP. The subtasks from SL-MNN$_3$ in section 5.2 were used in the comparison of all three Modular Neural Networks. For all MNNs the same SNN architecture is used (51x51x1). Then the intermediary network has the architecture (512x10), here the 512 neurons are connected to a certain layer in the SNNs. The same types of SNNs are used as well, one OvR SNN per class, and a vehicle versus animal binary classifier. The MNN has all of its SNNs trained independently. SNNs feed forward their output to the intermediary network which is then trained for the final result. [2] The Hidden Layer Modular Neural Network (HL-MNN) is an MNN but the SNNs feed forward the outputs of the layer before the final output layer. The SL-MNN works as described in section 4. The Multi-Layer Perceptron (MLP) will be a standard neural network which has no predefined modularity. The architecture of this MLP is (512x512x512x10). In Table 4 the epoch distribution and accuracy distribution in terms of the mean and standard deviation of the intermediary network can be seen. In Figure 5 the validation loss over epoch can be seen. It can be noticed that the SL-MNN

Table 4. Comparison with other methods ($N = 20$)

Model	Parameters	Epoch Distribution		Accuracy Distribution	
		μ	σ	μ	σ
MNN	1,764,971	126.50	46.43	48.20%	0.259%
HL-MNN	2,045,999	54.90	12.90	50.41%	0.227%
SL-MNN	2,045,999	157.55	23.04	49.81%	0.518%
MLP	2,103,818	160.2	23.01	51.41%	1.337%

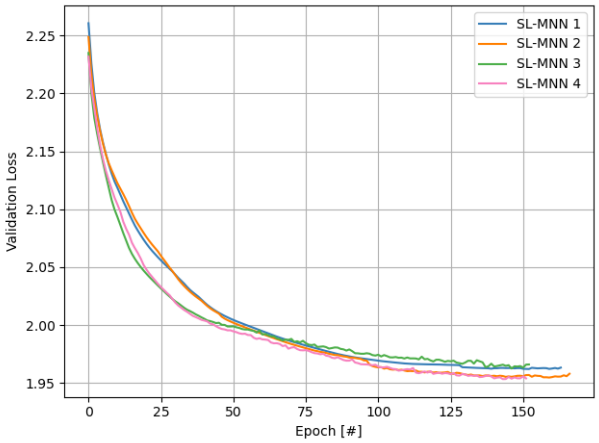

Figure 4. Average validation loss per epoch of SL-MNNs using different subtasks

has a much more stable accuracy than the MLP. But the mean accuracy of the MLP is still higher than that of the SL-MNN.

5.4 Interpretability

The gradient created by the intermediary network may overpower the gradients of the SNNs. If this were to be true then the SL-MNN would be more similar to a Multi-Layer Perceptron (MLP) and not a Modular Neural Network. To determine whether the SL-MNN is still modular the interpretability is tested. This is done by cutting specific weights that connect the SNN to the intermediary network. The neurons which have a positive weight toward the output of an SNN have all of their weights cut. This causes neurons that are supposed to activate the SNN's output to feed forward no information to the intermediary network. The effect of these cuts is analyzed through a confusion matrix to observe which classes are affected. The experiment is done on an SL-MNN which was created to compare to the other methods. Two kinds of SNNs are cut, the airplane OvR SNN and the superclass SNN. An OvR SNN is cut to show that the OvR models are still interpretable. The superclass SNN is cut to show that it is interpretable and to further support the theory that this SNN decreases misclassifications between super-

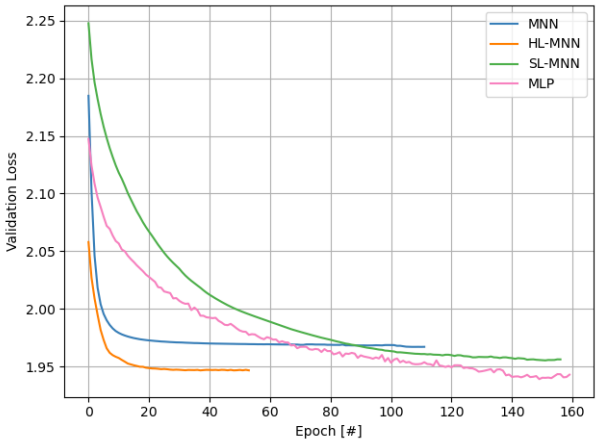

Figure 5. Average validation loss per epoch of multiple methods

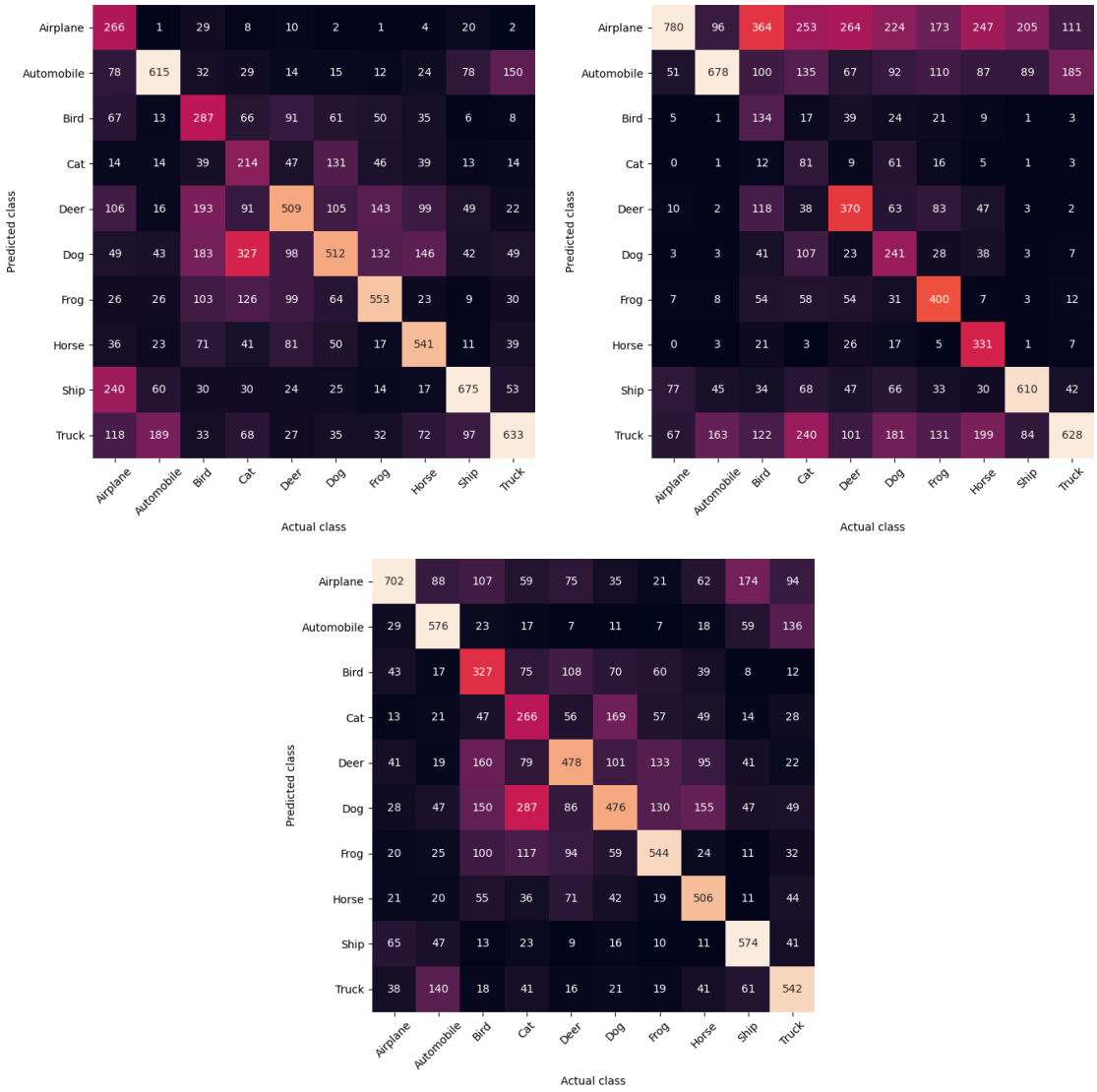

Figure 6. Confusion matrices of an SL-MNN, before cutting any weights (bottom), after cutting the airplane OvR SNN (top left), and after cutting the superclass SNN (top right)

classes. Cutting the weights of the airplane OvR SNN causes airplanes to be misclassified as other classes more often, see Figure 6. This suggests that the OvR models are interpretable. Cutting the weights of the superclass classifier causes vehicles to be misclassified as animals many more times, see figure 6. This further suggests the superclass classifier is interpretable and decreases the number of misclassifications. The HL-MNN and SL-MNN were also compared to determine if there was any major difference in the impact of cuts. The impact of cutting weights seems higher in SL-MNN than in HL-MNN. However, not enough data was gathered on this to statistically differentiate the two.

6. DISCUSSION

While the SL-MNN has a lower accuracy than the HL-MNN it has other benefits. The SL-MNN allows for real time performance measurements of the final model while the HL-MNN first needs to have all of its SNNs trained before the performance of the final model can be determined. The SL-MNN could also be better with different architectures and hyperparameters. The current implementation is not optimized so only a small amount of experiments could be done in the time alotted for this paper. Various other methods that could improve the accuracy of the SL-MNN are further mentioned below.

6.1 Fitting and fine-tuning SNNs

The SNNs of the MNN and HL-MNN are trained independently before the intermediary network is trained. On all SNNs early stopping is used to not overfit the SNNs, after the SNNs are trained their weights are frozen [4]. This means that these weights are no longer being updated. The SL-MNN does use an early stopping mechanism for any of the SNNs which can cause the underfitting and overfitting. Freezing the earlier layers in a model also generally results in a higher accuracy, this is known as fine-tuning. The SL-MNN does not do this either and as such does not gain the benefits of this technique. Thus two techniques that are used in the MNN and HL-MNN are not being used in the SL-MNN. These techniques could be implemented using a custom early stopping mechanism. All SNNs would have their validation losses tracked, if the validation loss has not decreased within a certain amount of epochs then the weight updates of the SNN are no longer applied. Only the weight updates of the inter-

mediary network are still being applied. After a certain amount of epochs, the layers in the SNN are frozen and no updates can be applied anymore. The intermediary network cannot stop its training until all SNNs are frozen and its validation loss has not decreased within a certain amount of epochs. Using this custom early stopping and fine-tuning mechanism the SL-MNN could have a higher accuracy than the HL-MNN.

6.2 Update rules

Both updates are in the same direction when $\Delta w_s > 0 \wedge \Delta w_s > 0$ or $\Delta w_s < 0 \wedge \Delta w_s < 0$. When this holds the loss for both the intermediary network and the corresponding SNN are lowered by the resulting update. But the moment that the weights are not in the same direction, i.e. $\Delta w_s > 0 > \Delta w_c \vee \Delta w_s < 0 < \Delta w_c$, the resulting update only decreases the loss for either the intermediary network or the SNN. In the next training step, the weight update of the model corresponding to the opposite direction could become more extreme, if this is repeated many times an equilibrium could be reached. $\Delta w = \omega_s \Delta w_s + \omega_c \Delta w_c = 0$ In this equilibrium neither model will have this weight converge to a value which minimizes their loss. Some mechanisms could be made to undermine this effect.

7. CONCLUSION

The Shared Layer Modular Neural Network allows for the simultaneous training of both the Sub-Neural Networks and the intermediary network. When trying new architectures for a model, the SL-MNN can instantly give an idea of how good the current model is performing. The standard MNN and HL-MNN first need to have their SNNs trained before it is possible to determine how the final model is performing. The update weights do not seem to significantly impact the accuracy of the SL-MNN as long as the sum of the update weights is equal to one. The addition of a superclass SNN decreases the number of misclassifications between superclasses, resulting in a higher accuracy. The addition of OvO SNNs give an early boost to the validation loss but do not result in a higher final accuracy. The SL-MNN does not result in a higher accuracy for the CIFAR-10 dataset than an HL-MNN and MLP. Cutting the weights from SNNs to the intermediary network has the expected effect on the classification accuracy, making the SL-MNN interpretable.

In future work, the effect that the weight updates have on the SL-MNN can be further investigated. More experiments can be done to determine whether the effect of cutting SNNs in the SL-MNN has a bigger impact than in other MNNs. Furthermore, a custom early stopping mechanism could be implemented to further increase the accuracy of the SL-MNN.

8. REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] R. Anand, K. Mehrotra, C. K. Mohan, and S. Ranka. Efficient classification for multiclass problems using modular neural networks. *IEEE Transactions on Neural Networks*, 6(1):117–124, 1995.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[4] R. Caruana, S. Lawrence, and L. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPSâ00, page 381â387, Cambridge, MA, USA, 2000. MIT Press.

[5] Y. Chauvin and D. E. Rumelhart, editors. *Backpropagation: Theory, Architectures, and Applications*. L. Erlbaum Associates Inc., USA, 1995.

[6] F. Chollet et al. Keras. `https://keras.io`, 2015.

[7] D. Filan, S. Hod, C. Wild, A. Critch, and S. Russell. Neural networks are surprisingly modular. *ArXiv*, abs/2003.04881, 2020.

[8] J. A. Fodor. *The Modularity of Mind*. MIT Press, 1983.

[9] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism, 2018.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[11] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby. Big transfer (bit): General visual representation learning, 2019.

[12] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[13] S. Masoudnia and R. Ebrahimpour. Mixture of experts: A literature survey. *Artif. Intell. Rev.*, 42(2):275–293, Aug. 2014.

[14] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, Madison, WI, USA, 2010. Omnipress.

[15] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[16] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[17] B. Singh, S. De, Y. Zhang, T. Goldstein, and G. Taylor. Layer-specific adaptive learning rates for deep networks, 2015.

[18] P. Sterling and S. Laughlin. *Principles of neural design*. January 2015.

[19] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[20] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.