# Exploring Sophisticated Loss Functions for Early Prediction in Convolutional Neural Networks

**Anish Pimpley**
CICS
University of Massachusetts Amherst
Amherst, MA 01003
aindulkar@cs.umass.edu

**Ajinkya Indulkar**
CICS
University of Massachusetts Amherst
Amherst, MA 01003
apimpley@cs.umass.edu

## Abstract

In this project, we investigate the effects of using sophisticated cascade loss functions on early predictions in image classification tasks. We apply methods native to cascade classifiers, to guide the learning of Convolutional Neural Networks. We approach the problem by trying to identify the depth and if to place an intermediate classifier in a CNN pipeline, so as to achieve a better accuracy-cost trade-off. We use a modified AlexNet as the base model. We also extend the soft cascade and firm cascade loss functions to the multi-label case using each for training the model. We run experiments comparing the use of these modified loss functions, instead of a simple summation. We show that the firm cascade loss function achieves a noticeably better trade-off between early prediction and accuracy on our model for the MNIST and Noisy MNIST datasets across various settings for hyper parameters.

## 1 Introduction

In recent years, neural networks have become the prominent model for visual object recognition tasks. Much of this popularity is thanks to major breakthroughs[1] in Convolutional Neural Networks (CNN) and their astonishing results on image recognition datasets such as CIFAR[2] and ImageNet[3]. State of the art models[4] have matched and may have surpassed humans in the ImageNet competition. However, the performance of such models on benchmark datasets is often focused on accuracy, while paying no heed to the cost or speed of computation. This is at odds with real world tasks, where computational budget is often limited and real time results are of utmost importance. For instance, the winner of the COCO 2016 competition used an extremely expensive 'ensemble of 5 f-RCNN with Resnet and Inception-Resnet'. [1] Even the baseline used by the winner, an Inception v2 SSD [5] model, would be considered far too complex and costly for many real world applications. It is possible to define small, yet decent networks that lie within the computational and prediction time constraints imposed by the use case. However, it comes at the cost of a few points of accuracy. This is because all examples in a dataset are not uniformly as difficult to classify. This raises the question, why do we have to choose between either a complex model where we waste resources on classifying easy examples, or a simple model that mis-classifies difficult examples? Intuitively, it would make sense for a model to dynamically allot the necessary but sufficient amount of resources to classify each example.

The seminal work of Viola & Jones tried to solve this exact problem by using a boosted cascade of classifiers. Their idea of using a series of thresholded smaller classifiers has been applied to CNNs. However, they either do not leverage advancements made in cascade classifiers since the early 2000s [6] [7] or treat the individual CNNs in the model as a black box, causing extra redundant computation, or both.[8] Our project aims to address the above 2 problems. Major advancements in cascade

[1] http://image-net.org/challenges/talks/2016/GRMI-COCO-slidedeck.pdf

classifiers include the introduction of the Soft Cascade [9]and Firm Cascade [10] loss functions. Instead of using a naive sum over all individual classifier losses, the Firm and Soft cascade try to model a training time loss, that closely resembles the thresholded form of the test setup. Both firm and soft cascade were defined for binary classification.

Our main contributions are as follows:

- We extend both Soft and Firm cascade losses by casting them into a form suitable for multi-class classification.
- We exploit the internal structure of CNNs, by training a single CNN with a Fully Connected and Output layer inserted at an intermediate position between CNN layers. This avoids redundant computation by sharing weights between both the intermediate and the final classifier, and allows us to jointly train both classifiers in one go.

The project is built on top of an AlexNet inspired CNN. We evaluate our model on MNIST and Noisy MNIST datasets. We show that replacing existing loss functions with the modified Firm and Soft cascade facilitates better training. We match our baseline's accuracy, while showing noticeable reduction in cost through early predictions.

## 2 Related work

We will briefly review cascade classifiers and models inspired from them. We will also differentiate our approach from other computation sensitive approaches used in deep learning.

### 2.1 Cascade classifier loss functions

In 2001, Viola & Jones[11] introduced cascade classifiers in their landmark paper, describing them as a series of progressively more complex Adaboost classifiers. Each classifier is thresholded such that an example is propagated deeper into the cascade only if the output probability beats the threshold. However, the model was tied to Adaboost and did not rely on gradient descent or standard optimization methods for building the cascade. Our project is inspired by the original boosted cascade. However, it is implemented for a completely different model and optimization landscape.
Raykar et. al. proposed the soft cascade, where the output probability of the cascade was computed as the product of the output at every intermediate classifier. The combined formulation of the cascade's output, allowed for joint optimization of the cascade and treated each model in the cascade as a black box. This extended the cascade to be applicable to any machine learning model as long as it emitted probabilities as outputs. The product formulation came with its downsides, as each classifier was treated equally by the optimizer. The training procedure also did not model the test mode of thresholded early prediction and was agnostic to the ordering of the cascade. The probability output of the soft cascade is mathematically represented as:

$$P_*(y/x) = \prod_{i=1}^{L} p_l$$

The firm cascade, by Dadkhahi & Marlin introduces a new combination rule that allows the training loss to closely model the cascade's behavior in test mode. The probability output of the cascade is approximately equal to the output probability of the intermediate classifier where the threshold was crossed. The output of each stage is made to operate in hard decision mode by applying a normalized logistic nonlinearity to its output. In this case we utilize the linear representation of the firm cascade.The linear firm cascade is mathematically represented as follows:

$$P_*(y/x) = \sum_{i=1}^{L} \theta_l * p_l$$

$$\theta_l = \begin{cases} (1 - g_\alpha(p_l)) \prod_{k=1}^{l-1} g_\alpha(p_k), & l < L \\ \prod_{k=1}^{L-1} g_\alpha(p_k), & l = L \end{cases}$$

$$g_\alpha(p) = \frac{f_\alpha(p) - f_\alpha(0)}{f_\alpha(1) - f_\alpha(0)}$$

$$f_\alpha(p) = \frac{1}{(1 + exp(-\alpha(p - 0.5))}$$

$P_*$ is the predicted probability of the cascade. Given the form of the equation ,we expect the value of $P_*$ to lie extremely close to either 1 or 0. In the above case, the sigmoid transformation is centered at 0.5. The $g$ function represents a normalized Softmax transformation at the output of a layer and $\theta_l$ is the aforementioned coefficient for probability of layer 'l'. We use the firm cascade as the primary loss function to replace a naive linear combination of intermediate cross entropies. We also extend both the soft and firm cascade for the multi-class use case.

### 2.2 Computation/ resource efficient neural networks

Significant prior work has been done in obtaining computation efficient networks at test time by resizing the network after training, by pruning[12] or quantizing weights[13] and training smaller student networks[14] to reproduce the results of the original model. Our model fundamentally differs from these methods, as we achieve the accuracy-speed trade-off jointly, at training time itself. These methods may however, be used in conjunction with our model.
Our work most closely resembles CNN models that take inspiration from cascade classifiers. Angelova et. al. use a tiny CNN as an intermediate stage and an ALexNet like model as the final stage. This model differs from our work, in that both classifiers do not share parameters and are separately trained. Li et. al. [15] use a cascade of 6 progressively more complex CNNs. However, they too train each model individually, treating each stage as a black box and choosing thresholds via. hyper parameter optimization. Both of the above models do not share parameters between CNNs and use training criterion that do not resemble the model in test mode. Both models were primarily built for object localization and binary classification.
A few months ago, Multi Scale Densenets(MSDNets) achieved state of the art results by building a model that exploits the structure of Densenets for building a cascade classifier at multiple scales within a single neural network. MSDNets showed significant reduction in computation cost without significantly compromising on accuracy. Both our model and MSDNets share parameters across intermediate classifiers. However, MSDNets utilize a much more sophisticated and complex design for their network. They exploit dense connectivity to avoid the interference of intermediate classifiers with each other. At train time, MSDNets use a summation over the cross entropies of individual classifiers as the loss function. This is a mismatch of the training and prediction time objective. In contrast, we use both the firm and soft cascade models as alternate combination rules for the loss functions instead.

## 3 Methodology

### 3.1 Problem setup and model overview:

The problem is modeled as a multi-label classification problem. $h(x)$ is a cascade classifier that maps an example $x$ into a class label. If $f_1(x)$ and $f_2(x)$ are the predicted probabilities of both stages respectively:

$$Prediction = h(x) = (argmax_i[f(x)[i]])$$

$$f(x) = \begin{cases} f_1(x), & max(f_1(x) > threshold) \\ f_2(x), & max(f_1(x) < threshold) \end{cases}$$

In simpler terms, the intermediate classifier is responsible for classification when its confidence in a particular label exceeds a preset threshold. Else, the final classifier is responsible for classifying that particular data point.The overall pipeline of our classifier if shown in fig.1. The model is identical to a standard 2 stage cascade, with one exception. That is, the final layer parameter values of the 1st stage classifier are effectively fed as inputs to the 2nd stage. We elaborate on the design of the stages and train time structure of the model in detail in the following sections.

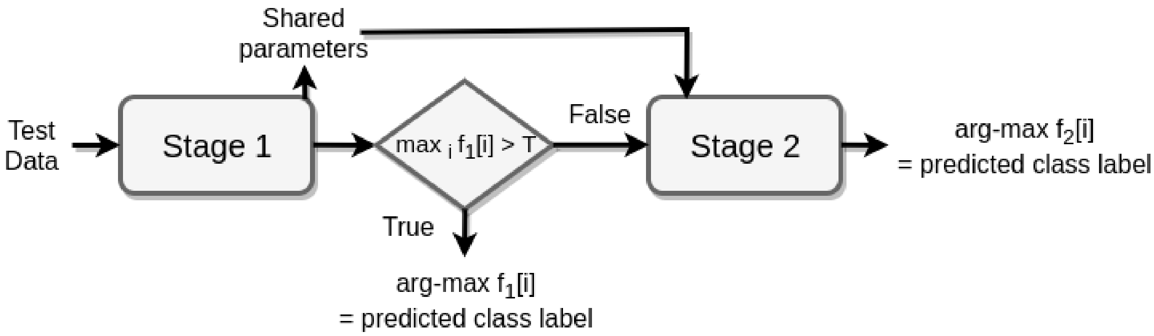

Shared parameters
Test Data
Stage 1
max $_i$ $f_1$[i] > T
False
Stage 2
arg-max $f_2$[i] = predicted class label
True
arg-max $f_1$[i] = predicted class label

**Figure 1:** Cascade classifier representation of our model

### 3.2 Base Classifier : CNN:

The internal structure of the cascade is derived from a single base classifier, a CNN. The architecture of the CNN follows heuristic guidelines for building CNNs developed over the last 5 years, primarily inspired by AlexNet, which may be considered the most widely adopted model of its type. In addition to CONV, MP, RELU and FC layers, we also use DropOut[16] and Batch Norm[17] layers, which were not present in the initial AlexNet model and are now considered essential parts of any deep learning pipeline.

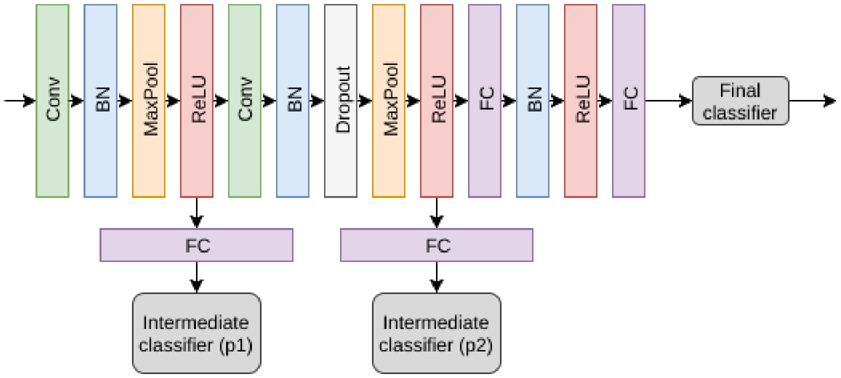


**Figure 2:** Neural Network architecture

*Nomenclature* : *CONV : Convolutional layer ; RELU : Rectified Linear Unit ; MP : Max Pool layer ; FC : Fully Connected layer ; BN : Batch Norm ; DropOut : Dropout in this particular layer.*

The model is built as a repeating series of the following blocks [ CONV ⇒ BN ⇒ DROPOUT ⇒ MP ⇒ RELU ] stacked in front of each other. In each block Dropout could be switched on or off. This stack was then succeeded by one block of [ FC ⇒ BN ⇒ RELU] and finally an [FC ⇒ SoftMax] block. The number of [ CONV ⇒ BN ⇒ DROPOUT ⇒ MP ⇒ RELU ] blocks and the state of Dropout in every block were decided by hyper-parameter optimization. For every possible configuration of dropout, we added [ CONV ⇒ BN ⇒ DROPOUT ⇒ MP ⇒ RELU ] until the accuracy of the model saturated for the training dataset. The final design of the base CNN can be seen in fig.2. In this case our dataset was MNIST.We used the same model structure for Noisy

MNIST as well.

To make the model suitable for cascade classification an intermediate [FC ⇒ SoftMax] layer is inserted between 2 blocks as seen in fig.2. We also insert a gating function at the intermediate classifier that terminates computation if the confidence values for the intermediate layer exceed the threshold. Of the intermediate classifiers shown in fig.2 only one of the intermediate classifiers will be part of the model at any given point. The figure only shows the 2 possible positions for the one intermediate classifier.

At training time, the complete cascade is trained jointly using stochastic gradient descent. The loss function utilizes a combination rule for the output of the 2 classifiers that we elaborate on in the next section.

### 3.3 Cascade loss functions: Firm and Soft cascade:

Our core contribution lies in the use of soft and firm cascade loss functions as a replacement for loss functions that are represented as a sum over individual losses.

The final layers of both the intermediate and the final classifier are SoftMax layers. Thus, for '$m$' class labels, the output to the final layer will be vector of $len(m)$, S.T. the values in that vector sum to 1. The naive model used in most modern early prediction architectures use a sum over cross entropies of individual probability outputs of each stage. This was also the loss function used in the MSDNETs paper which achieved state of the art results. Here on out we refer to this loss as MSDNET. *Note: This refers only to the loss of the MSDNET model, and not the model itself. The base model used by us is the same for all experiments.*

$V_1$ , $V_2$ = Output probability for classifiers: intermediate and final stage respectively

$\sum_{i=1}^{L} V_1[i]$ = $\sum_{i=1}^{L} V_2[i]$ = 1

The soft cascade is adapted to the multi label case by representing the out probability $P_*(y|x)$ as:

$$loss = CrossEntropy(P_*(y|x))$$

$$P_*(y|x) = elementwise \prod_{i=1}^{L} V_1[i]$$

This outputs a vector of the same length as both $V_1$ & $V_2$ with element $[i]$ equal to $V_1[i] * V_2[i]$.

In the case of the firm cascade, we wish for the output probability to be effectively equal to the that of the classifier which which output it. It is adapted to the multi label case by representing the out probability $P_*(y|x)$ as:

$$P_*(y|x) = g_\alpha(V_1) * V_1 + (1 - g_\alpha(V1)) * g_\alpha(V2) * V_2$$

$$loss = CrossEntropy(P_*(y|x)) + \lambda * (1 - g(V_1))$$

The function $g_\alpha$ and $f_\alpha$ use the same operations as defined in the firm cascade paper, however all multiplies are used as element-wise products, instead of scalar products. Also, instead of dropping negative cases, we are choosing whether to predict a label early or not. Thus, the above modification allows us to return a $P_*(y|x) \approx V_1$ if the 1st classifier chooses to classify the sample and $P_*(y|x) \approx V_2$ if the final classifier classifies the output. In keeping with the original firm cascade paper we choose a penalty term, that is added to the cross entropy in the loss function to loosely model the cost of computation. Here we choose $\lambda*(1-g_\alpha(V_1))$ as the penalty term. Taking cues from the firm cascade paper, we pre-train the model on our dataset and use it as a well performing initialization to train the firm cascade. We tried pre-training all of the competing models. However, we only saw significant gains for the firm cascade. The model also requires some degree of penalization to converge to a competitive model. Since the pre-trained initialization is tuned to perform well on the final classifier, we suspect that it might cause lack of gradient flow through the intermediate classifier. The introduction of a penalization term dependent on the output of the intermediate classifier might be helping guide the training of the intermediate classifier.

# 4 Dataset

The dataset we are using for our experiments is the MNIST[2] [18] database of handwritten digits. MNIST is a popular dataset in Machine Learning and is often used for comparison with state-of-art architectures. Although MNIST has fallen out of fashion, we decided to use MNIST due to its simplicity and fast train time. Whereas datasets like ImageNet take hours or days to train on expensive GPUs, MNIST takes a few minutes to train on our basic laptop computers. We believe it will be enough to show the speed-accuracy trade-off that we are trying to demonstrate. It consists of handwritten digits. There are 60,000 training samples and 10,000 test samples in the dataset.
Figure 3 shows what our samples look like. Each sample is a handwritten digit that can be from 0 to 9. The images are 28x28 and are centered and normalized. They also contain grey pixels as a result of anti-aliasing. The dataset is approximately balanced and there are roughly equal number of samples from each class. The train-test split is stratified. The dataset is stored in a binary format that is described in detail on the website. Each pixel is stored as 1 byte and the value ranges from 0 to 255. The images are greyscale, therefore they are single-channel. We will not go into the details of the format as there are popular helper methods available for reading the MNIST data. Specifically, we use the inbuilt PyTorch MNIST dataset[3]. We realized that the basic MNIST dataset was saturated in terms of performance. Therefore we also make the dataset artificially harder by adding Gaussian salt and pepper noise. The resulting examples are shown in the Figure 4

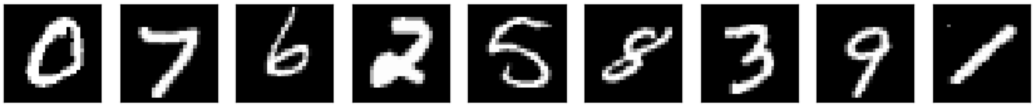


**Figure 3:** Samples from the MNIST dataset

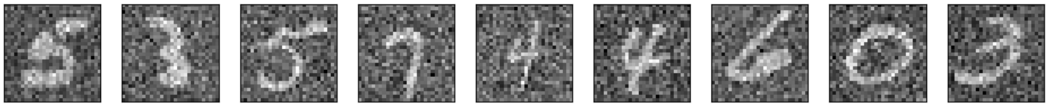


**Figure 4:** Noisy samples from the MNIST dataset

# 5 Experiments

We train the neural network shown in Figure 2. We are using an existing AlexNet like network built in PyTorch. Each convolution layer has kernel size 5 and stride 1. We train the entire network without any intermediate classifiers and call it 'Base'. For all experiments, we set threshold to 0.6 and alpha to 128 unless otherwise stated. For the firm cascade loss, we set penalty to 0.6. Note that these are just knobs that trade-off between accuracy and cost and there is no predefined single optimal trade off. Therefore, we don't do hyper parameter search for them. For each model, we train the network for a fixed number of 20 epochs. We use batches of size 64, with a learning rate of 0.001 and momentum of 0.5 with SGD.

During test time, the network is operated in hard mode and if the intermediate classifier returns a probability greater then threshold, we return that result without propagating it further. Intuitively, we are trading-off speed and accuracy by introducing this threshold. If our application requires fast inference at the cost of accuracy, the threshold should act as a knob that allows us to do that. This will result is cost savings in the form of computing power.

We first perform experiments on the original MNIST dataset and then on the Noisy MNIST dataset that we construct. We perform experiments on convergence of different models by checking their training loss across epochs. This should give us a general idea about the convergence of our models.

[2] http://yann.lecun.com/exdb/mnist/

[3] http://pytorch.org/docs/master/torchvision/datasets.html

The relative loss values are not comparable as they might be on different scales but the general trend should be decreasing.

We then compare the relative test accuracies of various models across epochs. This will tell us how fast each model converges relative to one another and also the final test accuracies of each model. This is important as this will tell us the performance of each model for a fixed threshold. It will also tell us how the early prediction models fare against our Base model. We expect the accuracy of the early prediction models to be less than the base model as they use fewer computational resources and predict early. We expect the Firm model to perform the best followed by Soft model and then MSDNet. This is because we believe the Firm model best encapsulates the hard-mode of the testing phase in its training phase. The other two models have a mismatch between the testing objective and the training objective, hence we expect them to perform worse than the Firm model. We compare two versions of each model where classifier is in position **p1** and in position **p2** in the network as shown in Figure 2. We expect the **p2** versions to perform better than the **p1** versions because the **p2** versions will have higher capacity in comparison to **p1** and in theory should perform better.

We then compare accuracy of Soft loss, Firm loss and MSDNet inspired loss for different values of thresholds. Intuitively, for lower values of thresholds the samples should be classified early with less accuracy and for higher values of thresholds, the samples should be propagated further into the network resulting in higher accuracy. Again, we expect the **p2** versions to perform better than the **p1** versions for the aforementioned reason.

Similarly, we compare the number of early predictions for different values of threshold. Our hypothesis is that for lower values of thresholds the network will have higher early predictions because a lower threshold signifies low confidence. In the extreme case when the threshold is 0, all samples will be predicted early i.e. at the first intermediate classifier and no samples will propagate till the end. On the other extreme, if we set threshold to 1, all samples will propagate till the end because the intermediate classifier will never predict a sample with probability 1. We expect to see a gradual decrease in early predictions as the threshold increases. Again, we expect the **p2** versions to have more early predictions because of higher capacity when compared to **p1**. We also expect the Firm model to perform better than the other two models for the aforementioned reasons. Additionally, Firm model also has a penalty parameter that forces the network to learn to predict early. The network is penalized proportional to the penalization term for samples that propagate till the end.

Additionally, we also compute the accuracy and early predictions of Firm model and see the effect of the penalty strength. We hypothesize that the number of early predictions should increase with the penalty strength because the penalty is for sample propagation. So the penalty is only applied if sample is not predicted early. In this case, it would make sense for the early prediction to increase. We also expect the accuracy to go down as we increase the penalty as the network will try to predict more and more examples as early as possible. For some examples, the shallower network might not have enough capacity to predict them correctly.

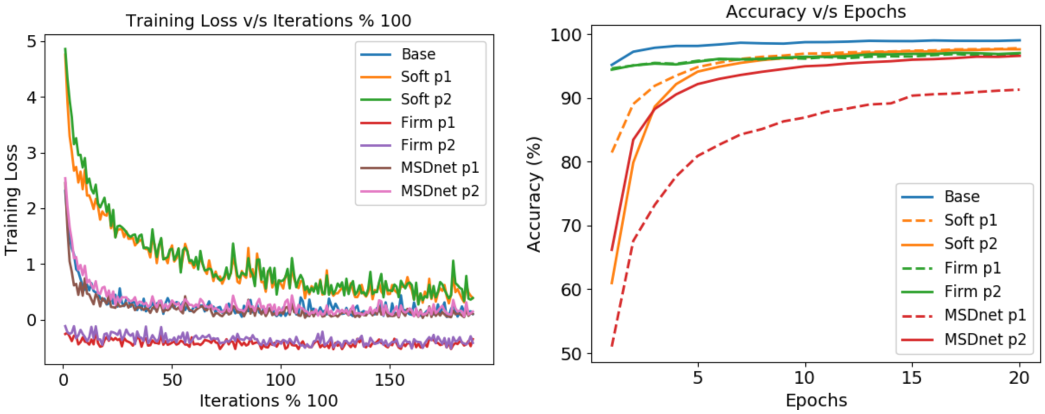
Finally, we compare the Accuracy and Early predictions for the different models for a fixed value of threshold = 0.6 to get an idea of its performance and behavior.

# 6 Results

## 6.1 Basic MNIST

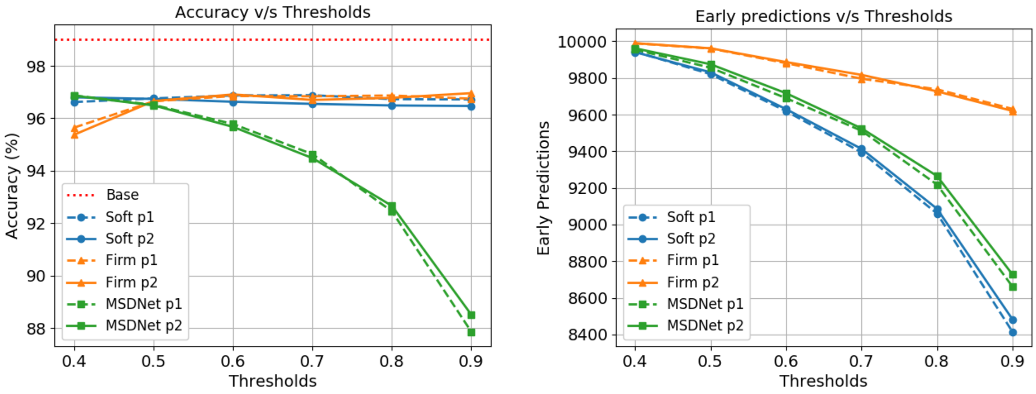
We train the network for 20 epochs and plot the training loss in Figure 5a. The results show that using our training parameters our network is converging for the different models. Although the loss values are plotted on a single plot, their relative values don't mean much since they use slightly different loss functions. The Firm models don't see much convergence because of pretraining as mentioned before. We plot the loss values for every 100 iterations of training.

Figure 5b shows the relative Test accuracy of the models during training for 20 epochs. The base model significantly outperforms our models. We see that the MSDNet **p1** has significantly lower performance than the other models. The Firm and Soft models perform reasonably well. It also confirms out hypothesis that in general **p2** models perform better than **p1** models.



**(a)** Training loss as a function of iterations % 100 **(b)** Test accuracy as a function of epochs

**Figure 5**



**(a)** Accuracy as a function of threshold **(b)** Early predictions v/s threshold

**Figure 6**

The results of our experiments on threshold with accuracy and early predictions are shown in Figure 6a and Figure 6b. Surprisingly, the MSDNet accuracy reduces as threshold increases. As expected, the number of early predictions decrease as we increase the threshold. This confirms our hypothesis that the threshold acts as a speed-accuracy threshold knob. As we increase threshold the number of early predictions decrease as expected. In terms of early predictions, Firm model performs the best followed by MSDNet and Soft model. Additionally, all **p2** models perform better than **p1** models because of higher capacity. The **p2** models also have higher early predictions than **p1** models, again because of higher capacity. Note that for threshold=1.0, we get 0 early predictions because all examples propagate to the end.

Figure 7 compares the relative performance of the models for a fixed threshold value=0.6. Note that we only compare the **p2** versions for simplicity. The **p1** versions show the same trends. Base accuracy is higher than all our early classifiers. Out of the three early classifiers, the Firm model performs the best in terms of accuracy and early predictions. Soft model has higher accuracy but lower early predictions when compared to MSDNet.

### 6.2 Noisy MNIST

We perform some of the same experiments on the Noisy version of MNIST as shown in Figure 4.

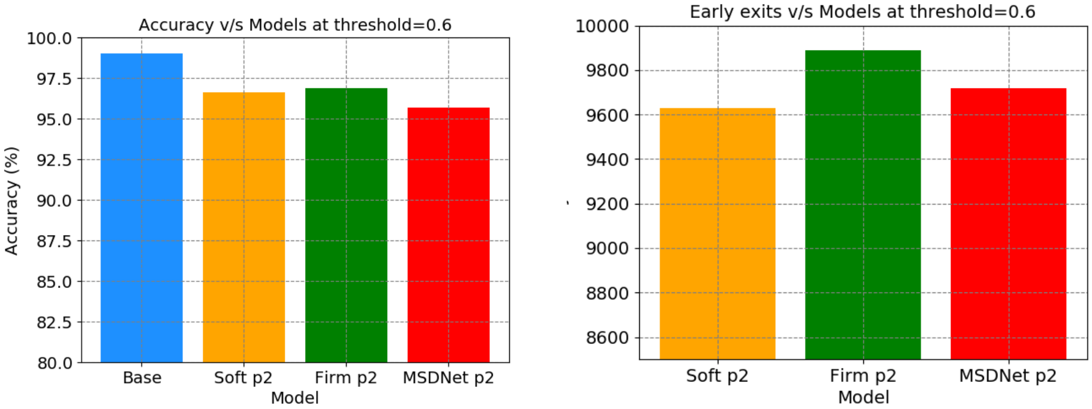


**Figure 7:** Accuracy and early predictions for our models

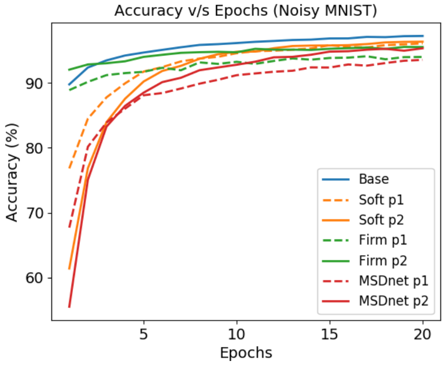
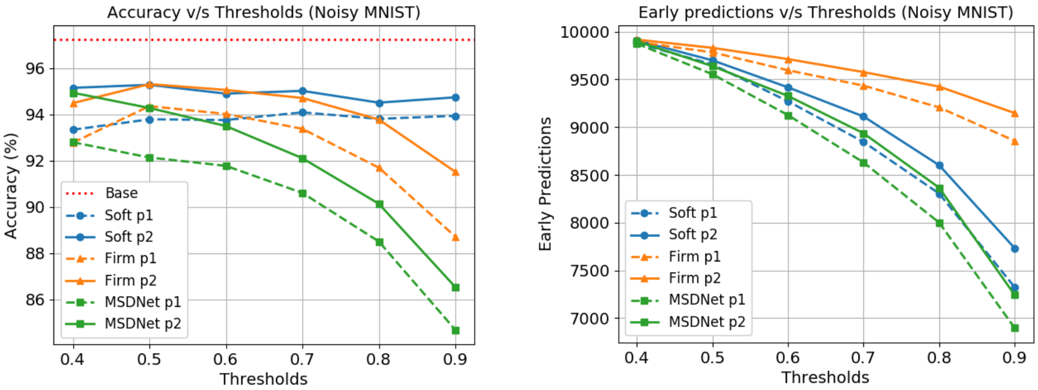


**Figure 8:** Accuracy v/s Epochs



**(a)** Accuracy as a function of threshold

**(b)** Early predictions v/s threshold

**Figure 9**

Figure 9a shows the effect of thresholds on the Accuracy of the classifiers. Here we see a more drastic difference of performance as the task becomes harder. The **p2** versions significantly outperform **p1** versions that can be attributed to the higher capacity of **p2** versions.

We also see in Figure 9b that for Noisy MNIST, the number of early predictions also drop drastically with increase in threshold. Furthermore the **p2** versions have higher early predictions compared to **p1** versions, again because of higher capacity. This shows that as the task becomes harder and harder, early prediction becomes more and more difficult and more samples are propagated till the end.

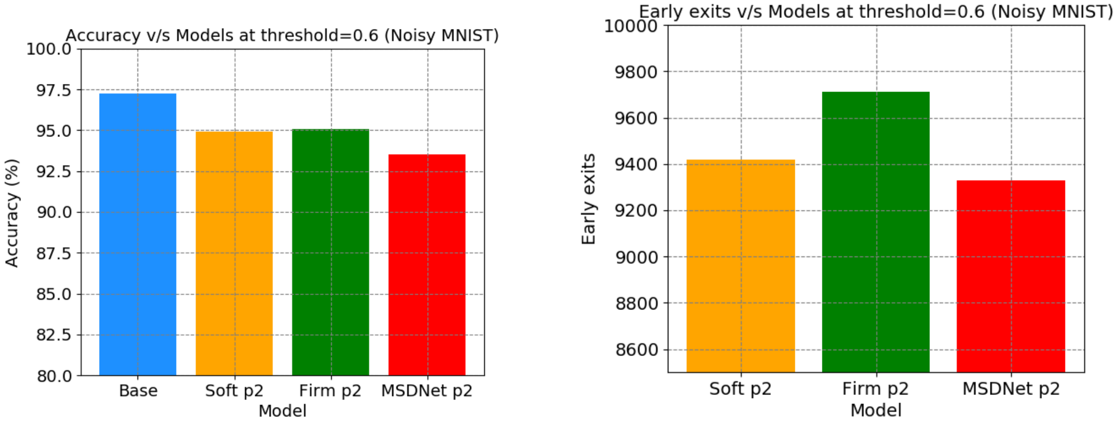


**Figure 10:** Accuracy and early predictions for our models

For a fixed threshold=0.6, Figure 10 shows the Accuracy and Early predictions for the different models. We see that the Firm model outperforms all other early prediction models in terms of accuracy and early predictions. In this case the Soft model also outperforms MSDNet in both accuracy and early predictions. This corroborates our hypothesis about the performance of the different models.

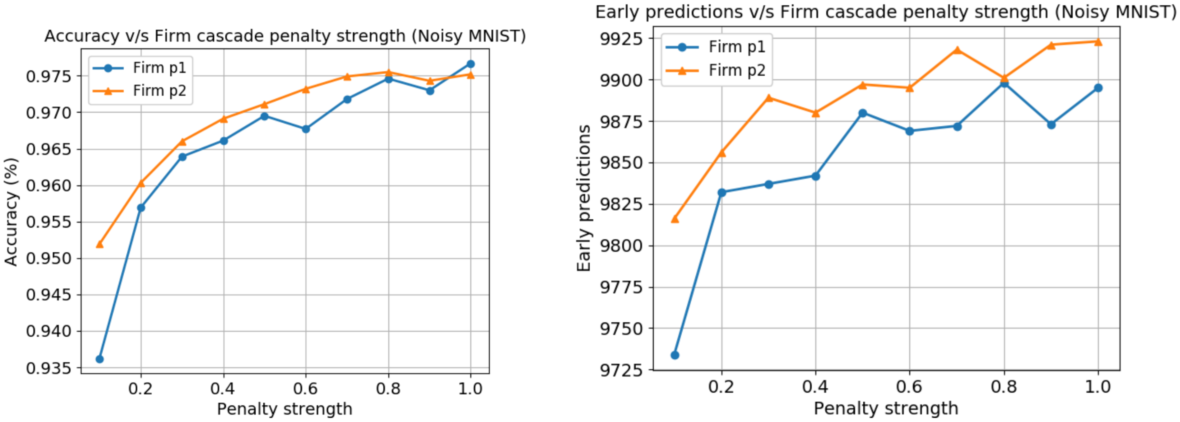


**Figure 11:** Accuracy and early predictions for Firm model v/s Penalty

Figure 11 shows the accuracy and the early predictions of the Firm model as a function of the Penalty strength with threshold=0.6. The accuracy increases as the Penalty increases which is counter-intuitive as one would imagine that increasing the penalty forces early predictions and results in lower performance. As expected, however, the **p2** version performs better than the **p1** version for both accuracy and early predictions. The early predictions, on the other hand increase as penalty increases which is expected because the penalty acts as a knob that controls the speed of inference.

## 7 Discussion and Conclusions:

We presented a novel combination of the firm-cascade model integrated with a CNN, that was cast into a multi-label classification problem with early exits. The design of our model was based around allowing maximal reuse of parameters, joint training of the cascade and a training objective that closely modeled a cascade's operation at test time. We showed that our modifications to the firm and soft cascade models preserved their behavioral traits from the binary case. Keeping the Model pipeline and dataset simple, allowed us to conduct a range of experiments in a controlled and well documented environment. The results show robustness of the firm cascade model in different settings, as it significantly outperform the MSDNet objective of a simple summation and edges out the soft cascade more often than not. Previous literature has largely resorted to treating CNNs used in cascades as black boxes and implemented algorithms largely based on the stock Viola & Jones cascade. We show that there there potential gains to be made if the cascade is directly integrated into the CNN and if sophisticated training objectives such as the firm and soft cascade are used instead of naive sums of Cross entropies. Finally, we were also able to confirm behavioral trends of early predictions wrt. position of the intermediate classifier and their respective thresholds.

We also noticed a few limitations of using such an approach. Sharing parameters means that our classifiers always trades off between accuracy and early prediction. Thus, we consistently saw a non-negligible drop in accuracy for all cascades over the baseline. The Multi-Scale Dense approach in MSDNets avoids this problem by using dense skip connections and multiple output routes.Secondly, both soft cascade and firm cascade add additional tuning parameters can be completely avoided in a sum of losses. Lastly, our experiments were run on a limited model and a relatively top heavy dataset on the accuracy front. It is yet to be seen how well such models scale to deeper networks, more stages and more difficult datasets.

Given the boundary pushing results of MSDNets, we hope to adapt both the firm and soft cascade to be applied to the MSDNet architecture and conduct robust experiments on it. The complexity of the model meant it took up a day to a train on personal commodity hardware for baseline datasets, making it for unfeasible for this project. We would also have liked to use the energy estimation tool[19] from work at MIT's Eyeriss project, to accurately model the savings made by early predictions.

## 8 Acknowledgements:

The authors would like to thank Prof. Ben Marlin for helpful guidance on this project. The project was inspired by work done by the authors under the guidance of Prof. Ben Marlin during their Summer 2017 independent study.

## References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[5] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. 2017.

[6] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense convolutional networks for efficient prediction. *arXiv preprint arXiv:1703.09844*, 2017.

[7] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648, 2016.

[8] Anelia Angelova, Alex Krizhevsky, Vincent Vanhoucke, Abhijit S Ogale, and Dave Ferguson. Real-time pedestrian detection with deep network cascades.

[9] Vikas C Raykar, Balaji Krishnapuram, and Shipeng Yu. Designing efficient cascaded classifiers: tradeoff between accuracy and cost. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 853–860. ACM, 2010.

[10] Hamid Dadkhahi and Benjamin M Marlin. Learning tree-structured detection cascades for heterogeneous networks of embedded devices. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1773–1781. ACM, 2017.

[11] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518 vol.1, 2001.

[12] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990.

[13] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *stat*, 1050:9, 2015.

[15] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5325–5334, 2015.

[16] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[17] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[18] Yann LeCun, Corinna Cortes, and Christopher JC Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[19] Vivienne Sze, Tien-Ju Yang, and Yu-Hsin Chen. Designing energy-efficient convolutional neural networks using energy-aware pruning. 2017.