Machine Learning Practical - Coursework 4

Luca Scimeca
UNN: s1344727

March 16, 2017

Contents

1 Introduction

This report describes the experiments run for *Coursework 4* of the course Machine Learning Practical at the University of Edinburgh. The coursework, second of a set of two experiments, involves the training and evaluation of Neural-Networks (NN) to perform object recognition in the CIFAR-10 and CIFAR-100 datasets. In these experiments we improve on the baseline performance of the networks designed in the first set of experiments [1], by designing more complex and suitable architectures for the recognition task at hand.

Initially, we train and evaluate a Convolutional Neural Network (CNN) architecture to recognize images in the CIFAR datasets. Thereafter, we try to improve on the baseline performance for CIFAR-100 through Multitask Learning.

2 Methods

The networks in these set of experiments are constructed and run on *TensorFlow* version 0.12.
The experiments are run on the CIFAR-10 and CIFAR-100 datasets both containing 60000 32x32x3 pictures of various objects (each as an array of 3072 elements). The images in CIFAR-10 are classified into one of 10 categories; similarly, the images in CIFAR-100 are classified into one of 100 categories, however, the latter is also divided in 20 coarse *super-classes* incorporating multiple of the 100 fine grained classes. For the purpose of these sets of experiments we train the models on a training set of 40000 images, splitting the remaining 20000 equally into a validation and test set. At training time, all models are fit with mini-batch Gradient Descent, with a batch size of 50, and are run for a variable number of epochs through the dataset. Unless otherwise specified, the weights are initialized with a GlorotUniform initialization and the error is computed as the multi-class cross entropy error (*nn.softmax_cross_entropy_with_logits* in *TensorFlow*) [1].

We compare the experimental results here reported to the best performing NN architecture in the previous set of experiments. The baseline NN architecture used for comparison consists of an input layer of 3027 (each unit consisting of a pixel in a CIFAR image), two fully connected hidden layers of respectively 400 and 200 hidden units and a final output layer of 100 units. The input and hidden layers in the network perform a non-linear *ReLu* transformation to their units, while the output applies a *SoftMax* to its affine layer, effectively performing a 1-of-10 or 1-of-100 classification of the inputs. Both hidden layers are wrapped with a *nn.dropout* layer, keeping each unit active with a *keep probability* of 0.5 [1] (see Figure 1). The results of the network are summarized in Table 1 and Table 2.

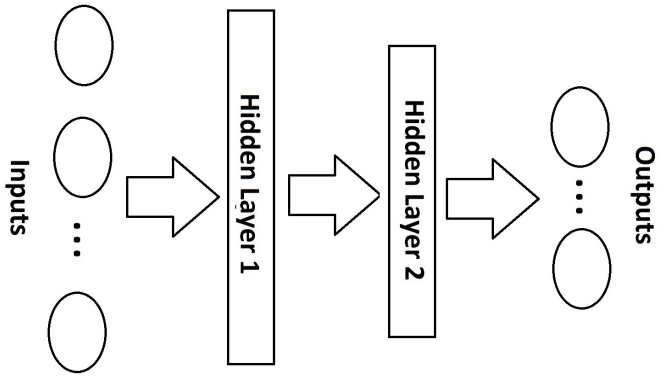


Figure 1: The Figure shows the architecture used as baseline to compare against in this set of experiments. The input and hidden layers of the network perform a *ReLu* non-linear transformation of their units, while the output layer applies a *SoftMax* to its affine layer. The units in the hidden layers use *dropout* during training, i.e. they are active with a probability of 0.5 on any given epoch.

2.1 CNN Architecture

Convolutional Neural Networks are a type of feed-forward Neural Network where we change connectivity in the attempt to account for spatial information. Each Convolutional layer in a network, in fact, will possess one or more *kernels*, each of which is used to swipe an image (or a batch) at a pre-set *stride*, and which through gradient descent will learn to recognize and activate when a particular detectable feature is observed (see [2]). To improve performance on the baseline feed-forward architecture with dropout in the first set of experiments, we design a Convolutional Neural Network to perform image classification on the datasets at hand.

As CNNs maintain spatial information (i.e. make use of the information the adjacency of pixels in an image may give) images are here reshaped from the unfolded 3072 input features, to their 32x32x3 original shape. The baseline Convolutional Neural Network designed for classification on the CIFAR-10 and CIFAR-100 datasets possesses 2 convolutional layers, 2 pooling layers, 1 normalization layer, 2 fully connected feed forward layers and an output layer. As previously mentioned, the inputs to the network need to be in their original form, thus we train the network on batches of 50x32x32x3 images.

The first layer in the CNN is a convolutional layer, which trains 100 5x5 kernels on the three (Red, Green and Blue) channels of the input images. The kernels are swept through the image at strides of 1px, and perform a *ReLu* transformation to their activation units. The convolution was here implemented through the *nn.conv2d TensorFlow* function. The non-linearity was chosen to be a *ReLu* for two reasons: one, *ReLu* units do not suffer from saturation [2] and are thus more robust to initialization and need no normalization; two, *ReLu* units have been shown to converge faster in many related applications [3].

The second layer of the architecture is a max-pooling layer, which reduces the dimensionality of the channel weights by retrieving the maximum activation in each set of three adjacent units. The layer is implemented through the *nn.max_pool TensorFlow* function which is made to swipe the channels in input at strides of 2 [4].

Following the pooling layer we add a Local Response Normalization (LRN) layer. Although *ReLu* activations theoretically do not need normalization, LRN has been shown to improve performance by a significant amount [3]. We implement the Local Response Normalization layer through the *TensorFlow* function *nn.lrn*, and we set $\alpha = \frac{0.001}{9}$ and $\beta = 0.75$, settings of the hyper-parameters shown to be working well in similar architectures (see [4]).

Following the normalization layer we build two more layers like before. The first, a convolutional layer training 100 kernels on the 100 channels in output from the normalization layer; the second, a max-pooling layer parameterized as before, which reduces the dimensionality of the 100 kernels in output further.

Finally, we implement a reshape layer through the *TensorFlow* function *reshape* which is useful in fully connecting all units of the 100 kernels in output from the second pooling layer, to a 400 dimensional Affine Layer, performing a *ReLu* transformation of its activation units. After passing from a second fully connected *ReLu* layer of 200 units, we finally classify the images in input into 1-of-10 or 1-of-100 categories, by creating a final fully connected Affine Layer of 10 or 100 units for CIFAR-10 or CIFAR-100 respectively. Figure 2 gives an overview of the designed network architecture.

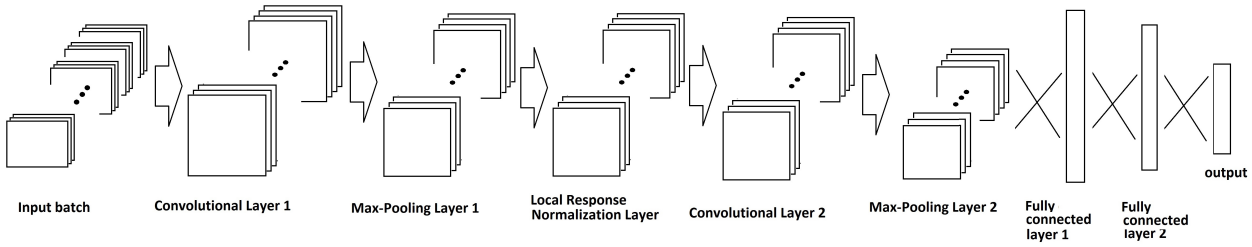


Figure 2: The Figure depicts the CNN baseline architecture developed in section 2.1. The input batches are passed through two convolutions, 2 max-pooling layers and a normalization layer before passing through 2 fully connected layers and finally classifying the inputs in 1-of-10 (CIFAR-10) or 1-of-100 (CIFAR-100) categories.

To train the network, we add an L2 regularizer to the multi-class cross entropy error on the inputs with respect to their targets. L2 regularization is a form of regularization where we try to limit the growth of the weights by including an additional term to the error to minimize i.e. :

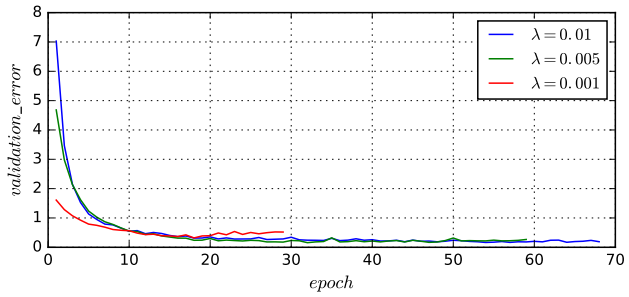
$$E_\lambda = E + \lambda \sum_{i=1}^{D} ||w_i|| \tag{1}$$

where D is the number of trainable weight parameters in the network, λ an hyper-parameter which needs setting and E_λ the new error to minimize. Intuitively larger values of λ will penalize larger weights more heavily and, in turn, regularize more. To apply L2 to the network, the error to minimize is augmented with a weighted *L2* sum of all trainable weights in the CNN (save biases), computed through the use of *nn.l2_loss* in *TensorFlow*.

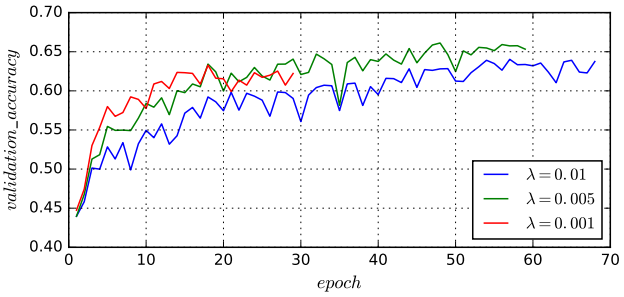
Figure 3 shows the results of the hyper-parameter search performed to choose the λ value. As clear from the Figure, when the lambda value is too low, regularization is applied only in a *weak* sense, and the network soon over-fits the data (in the figure shown as terminating the search early, due to early stopping); increasing the value of λ decreases over-fits, however, a too large value

(i.e. $\lambda = 0.01$) prevents the CNN weights to be variable enough to fit the data well, and therefor improvement on the validation set will saturate sooner. The value of lambda to minimize validation error and maximize accuracy was found to be $\lambda = 0.005$.

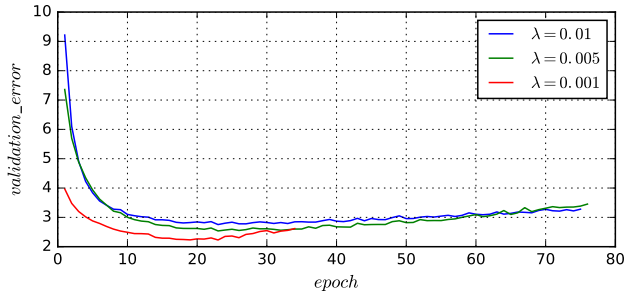
Finally, we train the network with RMSProp, an adaptive learning rule which allows us to focus less on searching the space for learning rates. During training, we set η and α to 0.0001 and 0.9 respectively, two known good hyper-parameter values for the network [1, 5].



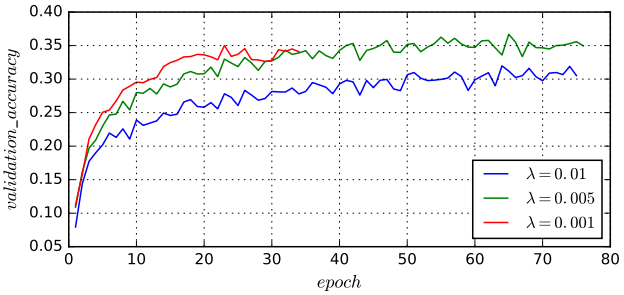
(a) CIFAR-10: validation error



(b) CIFAR-10: validation accuracy



(c) CIFAR-100: validation error



(d) CIFAR-100: validation accuracy

Figure 3: The Figures show the change in validation error (Figures (a)-(c)) and accuracy (Figures (b)-(d)) of the CNN designed in section 2.1 when trained with the RMSProp learning rule setting $\alpha = 0.9$ and $\eta = 0.0001$ (values known to work well a priory [5]) and L2 regularization with varying values of λ. We terminate training through *early stopping* on the validation accuracy, i.e. when no validation accuracy improvement is recorded over the last 15 epochs though the dataset. We find $\lambda = 0.005$ to be the hyper-parameter value which maximizes accuracy on both CIFAR validation sets.

2.2 Multitask Learning

To improve on the baseline performance on the CNN architecture previously designed, we apply Multitask Learning to CIFAR-100. In Multitask Learning, we attempt to improve the performance of the CNN by urging the network to learn, besides the 100 fine grained classes like previously, their 20 coarse counterparts. The multi-learning process is meant to allow the network to learn a representation of the data which allows it to classify an image into a class of either set, thus intrinsically working as a regularizer and forcing its units to learn more interesting and discriminatory features of the data.

To apply Multitask Learning we create two output layers in the network: one, the 100 unit affine layer to classify the data into the 1-of-100 classes; two, a 20 unit affine layer to classify the data into 1-of-20 classes. Both output layers connect to the penultimate 200 unit layer with *ReLu* non-linearity previously designed and an error is computed for both like in previous sections. We train the network by minimizing the average of the two errors, i.e:

$$E_{mul} = E_{100-class} + E_{20-class} \tag{2}$$

where E_{mul} is the error now to minimize, $E_{100-class}$ the multi-class cross entropy error of the 100 fine grained classes and $E_{20-class}$ similarly that of the 20 coarse classes. We add L2 regularization like previously to both $E_{20-class}$ and $E_{100-class}$, We thus minimize E_{mul} through gradient descent and observe the performance of the network (i.e. validate) on the 100-class dataset.

2.2.1 Multitask Learning as regularizer

Given the nature of the approach, i.e. by moving weights to classify both coarse and fine grained categories, Multitask Learning acts as a regularizer [6]. We test whether the regularizing effect of Multitask Learning is enough to prevent early over-fitting, by training the network in two different settings: one, with L2 regularization applied to its weights (keeping $\lambda = 0.005$), and two with no regularization. Subsequently, we observe and compare the performance of the CNN designed in both settings.

2.2.2 Weighted Multitask Learning and Transfer of Knowledge

When averaging the multi class cross entropy error of the two output layers for the different classes, we move in a direction in weight space which minimizes the training error for both tasks. While attractive for many reasons, this approach might not out-perform previous implementation, as learning how to predict the additional coarse classes allows the network to perform well in both classification tasks, but possibly neither one as well as if it had fine tuned its weights specifically for them. To make up for this possible drawback, we introduce two changes i.e. a weighted Multitask Learning regime, and what is known as *Transfer of knowledge*.

In the weighted Multitask Learning variation, we reformulate our error as :

$$E_{mul} = E_{100-class} + \frac{1}{2}E_{20-class} \tag{3}$$

i.e. we weight down the coarse class error ($E_{20-class}$) to bias the training step after back-propagation, towards the minimum in the weight space of the 100-class target set, and less so for the coarse classes.

The idea behind the concept of *Transfer of knowledge*, instead, is the re-use of the representation built when training a network (i.e. with Multitask Learning) to further on a similar network. Here, we monitor the validation error of the CNN on the 100 fine grained classes; when the error performance is seen not to improve for over 15 epochs, we swap the error used for back-propagation from that of Multitask Learning to the single error on 100 fine grained classes. The swapping, allows the network to overcome the potential issue previously mentioned and, after learning a useful shared representation of the data, to ultimately fine tune its weights to the classification task we are interested in.

2.2.3 Learning Swap

We attempt to overcome potential issue in Multitask Learning deriving from the inability of the network to fine tune its weights to the task we are most interested in by introducing the concept of *Learning Swap*. In *Learning Swap* we define two learning rules: one, based on the multi-class cross entropy error of the 100 fine grained classes (like in the simple CNN training); and the other, based on the weighted Multitask regime, where the error is a weighted variant of the multi-class cross entropy error of the 100 fine grained and 20 coarse classes (see equation (3)).

We train the CNN by swapping the two learning rules on each epoch and switch to fine tuning the network on the 100 fine grained classes, only when no validation error improvement has been observed over 15 epochs with *Learning Swap*.

2.2.4 Multitask Learning with Decay

In a final variant of Multitask Learning, we modify the sum of the errors for the two sets of targets by *decaying* the influence of the 20-coarse multi class entropy error over epochs. The idea is to initially force the weights in a point in space where is it possible to classify images into either of the two sets of targets (therefor obtaining a more useful representation of the data), and to *gradually* move the weights from said position, to one where we concentrate on classifying images into 1-of-100 classes. The error computation becomes:

$$E_{mul\_dacay} = E_{100-class} + e^{-\frac{epoch}{\alpha}} E_{20-class} \tag{4}$$

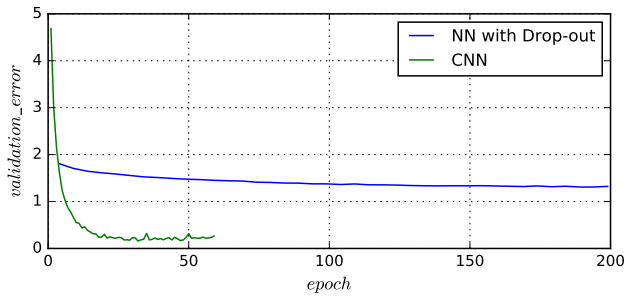
where α is an hyper-parameter that needs setting. From experiments, we observe the network reaching its top performance after ≈ 100 epochs. We therefore pick $\alpha = 50$ to a allow the decay to return values in a $\{0.9, 0.1\}$ range throughout the run. We train the CNN like before and fine tune the network to the sole 100-class task after no validation error improvement is observed over 15 epochs.

3 Results

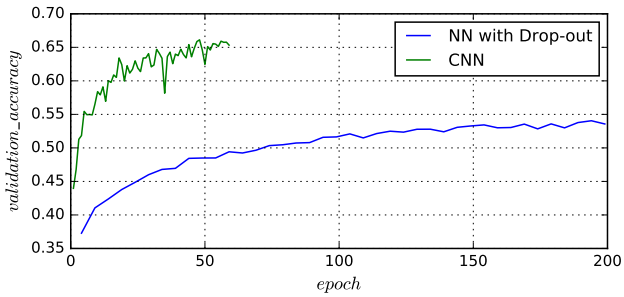
3.1 CNN architecture

We construct a CNN with 2 convolutions, 2 max-pooling layers, a normalization layer, 2 fully connected feed-forward layers and an output layer. We apply L2 regularization to the network and pick the most performing λ value in an hyper-parameter search (i.e. $\lambda = 0.005$). We train the network with RMSProp, setting the hyper-parameters to known good values and compare the performance of the CNN designed to the previous most performing feed forward NN with dropout.

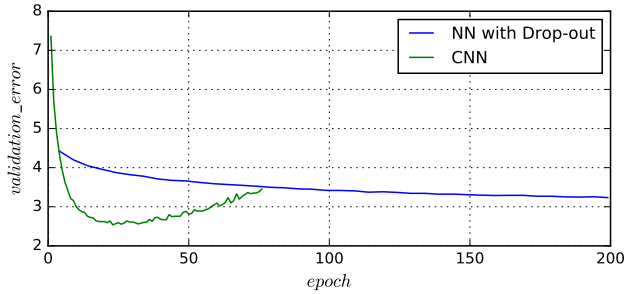
As clear from Figure 4, the CNN architecture improved greatly on the baseline performance previously achieved. The NN with dropout shows a good asymptotic behavior, reaching a top accuracy of $ACC_{cifar-10} = 0.5432$ and $ACC_{cifar-100} = 0.249599$ for CIFAR-10 and CIFAR-100 respectively. The CNN, however, reaches a much higher accuracy in both datasets, reaching $ACC_{cifar-10} = 0.6612$ and $ACC_{cifar-100} = 0.3668$ for CIFAR-100 and CIFAR-10 respectively. The large increase in performance is due to the ability of the network to extrapolate better spatially meaningful features. The increase in performance, although similar in both datasets, is most evident in CIFAR-100, where the classification accuracy observed is $\approx +50\%$ with respect to that of dropout on the simple feed forward network.



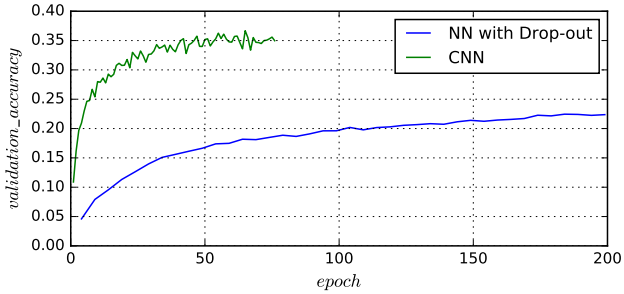
(a) CIFAR-10: validation error



(b) CIFAR-10: validation accuracy



(c) CIFAR-100: validation error



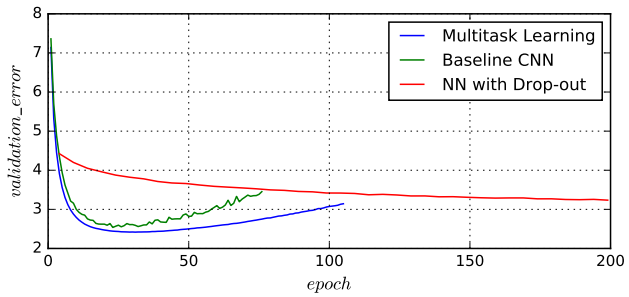
(d) CIFAR-100: validation accuracy

Figure 4: The Figures show the change in validation error (Figures (a)-(c)) and accuracy (Figures (b)-(d)) of the CNN designed in section 2.1 (green plot) against the best performing feed forward NN with dropout designed in the previous set of experiments (blue plot). The CNN training is interrupted through *early stopping*, which ceases training after no classification accuracy improvement has been observed over 15 consecutive epochs. As clear from the figures, the CNN architecture out-performs previous simple feed-forward implementation, reaching over 11% additional accuracy on both datasets.

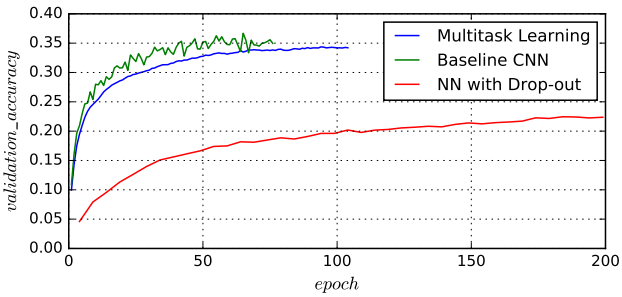
3.2 Multitask Learning

We perform Multitask Learning by training the CNN previously designed to classify images in the CIFAR-100 dataset, both in their course and fine grained category. The multi class cross entropy errors on the two sets of targets are here added, together with the L2 regularizer described in previous sections. We compare the performance of the CNN with Multitask Learning training with that of the simple previous CNN approach and that of dropout.

As clear from Figure 5, both Convolutional Neural Networks improve on the baseline performance of the simple feed-forward NN with dropout. The CNN trained with Multitask Learning shows a more *stable* learning behavior over the simple CNN, smoothly improving its weights over each epoch before over-fitting to the data (and therefor early stopping). Although Multitask Learning clearly introduces desirable properties to the CNN learning behavior, the final performance on the validation set is of $ACC_{mul} = 0.3436$, and therefor it does not improve on previous experiments.



(a) CIFAR-100: validation error



(b) CIFAR-100: validation accuracy

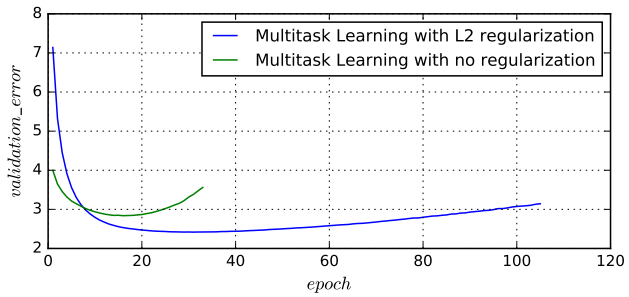
Figure 5: The figures show the performance of the simple CNN, the CNN trained with Multitask Learning and the simple feed-forward NN with dropout over a variable number of epochs (due to early stopping). The maximum validation accuracy is achieved by the simple CNN, although the CNN trained with Multitask Learning shows desirable behaviors, most preeminently, stability and a better asymptotic behavior.

The inability of the network trained with Multitask Learning to out-perform the previous CNN implementation is due to the imposition, in its learning procedures, to classify both coarse and fine grained target sets, but neither one specifically. The CNN previously trained, although incapable of classifying images in the 20 coarse targets, fine tunes its weights better to the single 100-class classification task and ultimately outperforms Multitask Learning.

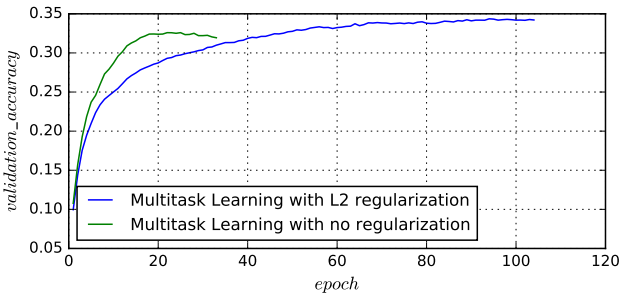
3.2.1 Multitask Learning as a regularizer

We test the ability of Multitask Learning to solely regularize the network by comparing the performance of the CNN previously designed when trained with and without L2 regularization.

Figure 6 shows how introducing L2 regularization to the CNN trained through Multitask Learning improved both classification accuracy and error. Multitask training on its own, is clearly not enough to regularize the weights of the CNN designed, which with no additional regularization over-fits the data relatively quickly (≈ 20 epochs). Even here, adding L2 regularization helps reduce over-fitting and allows the network to show a better asymptotic behavior over a larger number of epochs.
Note that here, we are not assessing the inability of Multitask Learning to act as a regularizer. Instead, we assess whether the L2 regularization previously introduced helped improving performance.



(a) CIFAR-100: validation error

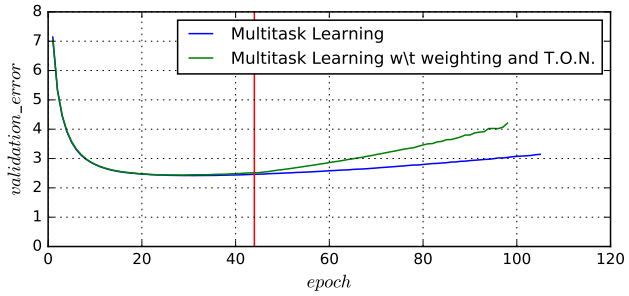


(b) CIFAR-100: validation accuracy

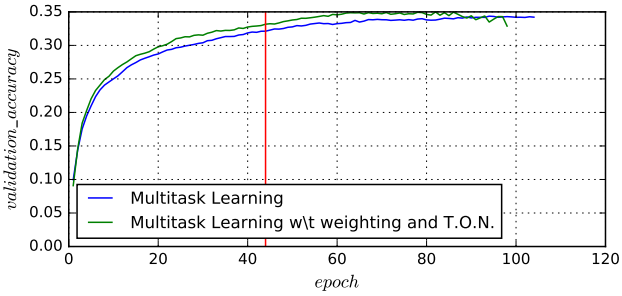
Figure 6: The figures show the classification error (Figure (a)) and accuracy (Figure (b)) of the CNN designed in section 2.1 when trained with Multitask Learning with and without L2 regularization. Although Multitask Learning helps with over-fitting, as clear from the figures, it is not enough to regularize the weights in the network. Adding L2 regularization improves the overall performance of the CNN.

3.2.2 Weighted Multitask Learning and Transfer of Knowledge

We introduce two changes to the Multitask Learning approach: we change the computation of the error to back-propagate by down-weighting the coarse class error (thus intrinsically prioritizing classifying images into the fine-grained classes), and we swap the learning rule to the one based on the fine grained classes after no improvement is observed in the validation error over 15 epochs, to try and ultimately fine tune the network for the classification task we are most interested in. Figure 7 compares the current approach to the previous (L2 regularized) simple Multitask Learning routine.



(a) CIFAR-100: validation error



(b) CIFAR-100: validation accuracy

Figure 7: The figures show the change in validation error (Figure (a)) and accuracy (Figure (b)) of the CNN trained with the weighted variant of Multitask Learning. The red line corresponds to the moment, during training, where the learning routine was swapped and the final tuning started (i.e. when no validation error improvement was observed over the last 15 epochs). As clear from the figures, the approach outperforms the previous Multitask Learning implementation.

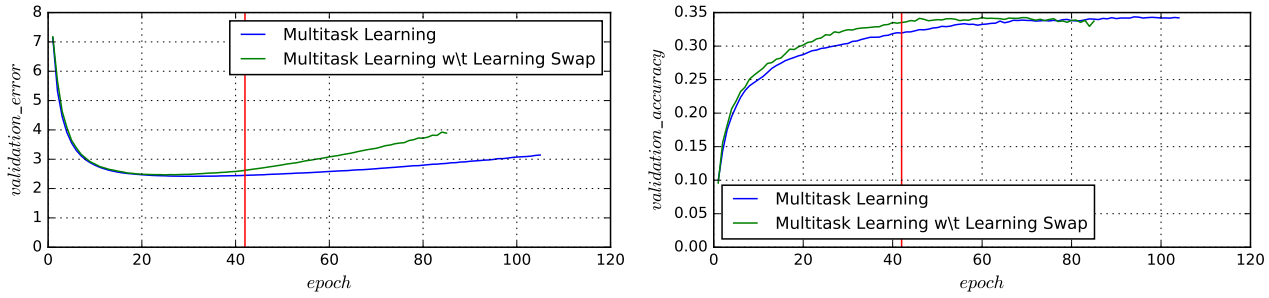
As evident in the accuracy plot (Figure 7b), the network trained with with the *weighted* variant of Multitask Learning and the final *transfer of knowledge* fine tuning performs better than the previously implemented network over all epochs, achieving a maximum validation accuracy of $ACC_{mul\_weighted} = 0.3499$ as compared to $ACC_{mul} = 0.3436$, achieved by the initial implementa-

tion. The error plot, on the other hand, shows more clearly how this method quickly over-fits, after reaching its top performance. The fast paced over-fitting (as compared its initial implementation) is due to the final swap to gradient descent updates based only on the 100 targets multi-class entropy error. The result gives further evidence of the regularizing effect of Multitask Learning.

3.2.3 Learning Swap

We introduce a variant of Multitask Learning where we *swap* the simple CNN training regime with the Multitask Learning training based on the weighted coarse and fine grained cross entropy error (see equation 3) every epoch, to try and sway the network to prioritize classification on the CIFAR-100 fine grained classes. We add the transfer of knowledge approach previously designed when no validation error improvement is observed over 15 epochs. Figure 8 compares the *Learning Swap* approach to the initial Multitask Learning implementation.

With Learning swap, the network lightly *fine tunes* its weights to perform the 100-class classification task since the first epochs, thus initially performing better than the first Multitask Learning approach. The improvement, however, does not ultimately outperform the initial implementation, due to an early over-fitting introduced by the alternating back propagation of the error with no Multitask Learning, which ultimately regularizes the weights less than in the initial approach (the plot in Figure 8a, shows clear symptoms of faster over-fitting prior the final tuning).



(a) CIFAR-100: validation error

(b) CIFAR-100: validation accuracy

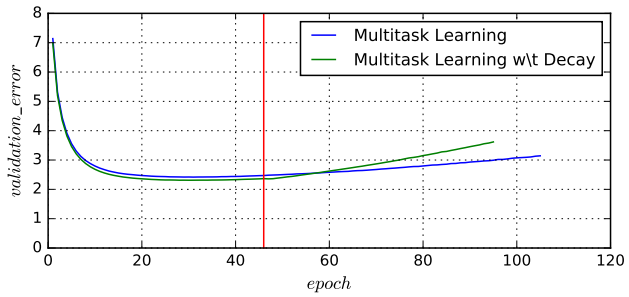
Figure 8: The figures show the change in validation error (Figure (a)) and accuracy (Figure (b)) of the CNN trained with the *Learning Swap* variant of Multitask Learning. The red line corresponds to the moment, during training, where the learning routine was swapped and the final tuning started (i.e. when no validation error improvement was observed over the last 15 epochs). *Learning Swap* accounts for an initial improvement over the Multitask Learning routine, but eventually reaches a similar performance to that of the simple Multitask Learning implementation.

3.2.4 Multitask Learning with Decay

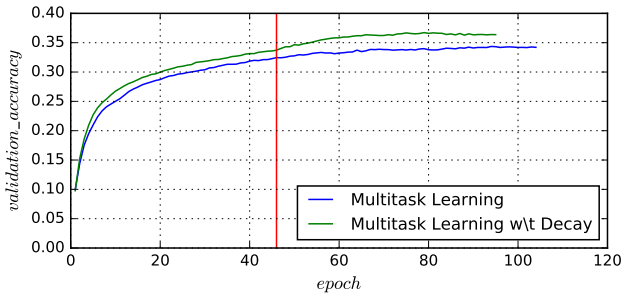
We introduce a final variant of Multitask Learning where we decay the influence of the 20-coarse multi class entropy error in Multitask Learning over the epochs. Figure 9 reports the performance of the decay routine when compared to initial Multitask Learning approach.

The *decay* variant accounts for a better performance of the network over all epochs where Multitask Learning was applied to the learning routine of the network (Figure 9b, left side of the red line). Moreover, after the learning swap for the final tuning (epoch 46), the network accuracy improves

yet again (Figure 9b, right side of the red line) reaching a top accuracy of $ACC_{mul\_decay} = 0.3672$, the highest achieved in all experiments.



(a) CIFAR-100: validation error



(b) CIFAR-100: validation accuracy

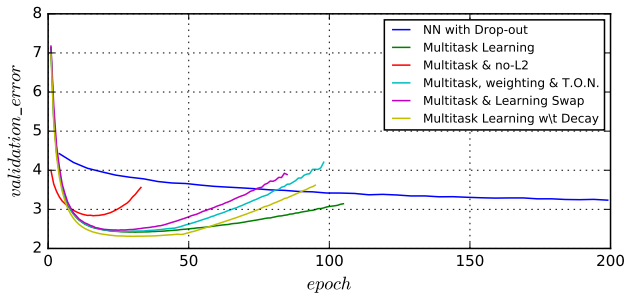
Figure 9: The figures show the change in validation error (Figure (a)) and accuracy (Figure (b)) of the CNN trained with the *decay* variant of Multitask Learning. The red line corresponds to the moment, during training, where the learning routine was swapped and the final tuning started (i.e. when no validation error improvement was observed over the last 15 epochs). As clear from the accuracy performance, the CNN trained with the *decay* variant of Multitask Learning performs overall strikingly better than the simple Multitask Learning implementation, reaching a validation accuracy of $ACC_{mul\_decay} = 0.367200$.

The gradual decay of the *pull* in weight space generated by learning the 20-coarse classes, besides the fine grained 100 classes in CIFAR 100, successfully managed to initially push the weights towards a more generic representation of the data and progressively shift the focus of the network into classifying only the 100 fine grained classes, effectively out-performing all previous experiments.

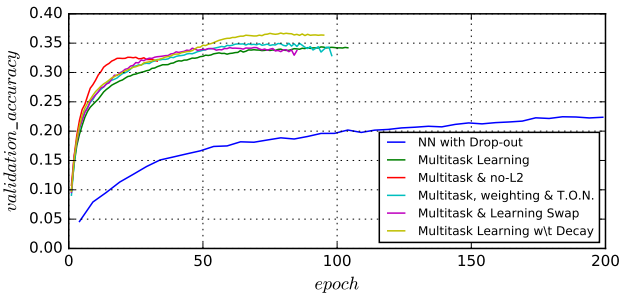
4 Summary and Discussion

To improve on the CIFAR-10/CIFAR-100 image classification performance of the baseline feed-forward NN implemented in the first set of experiments, we construct a CNN with 2 convolutions, 2 max-pooling layers, a normalization layer, 2 fully connected feed-forward layers and an output layer. We train the network through RMSProp, shown to give stable performances over previous experiments [1, 2, 7], and add L2 regularization to prevent early over-fitting. The CNN designed easily out-performs the previous simple NN with dropout, reaching a better validation accuracy and error on the datasets ($ERROR_{cifar-10} = 0.97811$, $ACC_{cifar-10} = 0.6612$, $ERROR_{cifar-100} = 2.537402$, $ACC_{cifar-100} = 0.3668$).

With Multitask Learning we introduce desirable properties to the CNN learning behavior, although we show the inability of the simple Multitask Learning approach to improve on previous results. We modify the approach by: differently weighting the double learning process, introducing the related concept of *transfer of knowledge*, and introducing the novel concepts of *Learning Swap* and *decay* for Multitask Learning. Figure 10 compares the validation error and accuracy of each of the different approaches.



(a) CIFAR-100: validation error



(b) CIFAR-100: validation accuracy

Figure 10: The figures show the validation error (Figure (a)) and accuracy (Figure (b)) of the different architectures and approaches designed throughout the report.

Evident in the Figures 10a and 10b is the strikingly better performance of the CNNs as compared to simple NN for image classification, which reach an overall lower classification error and higher accuracy. The improvement is mainly due to the type of network implemented. CNNs, in fact, are capable of taking advantage of pixel adjacency and spatially significant features, which in turn improves classification accuracy by a large margin.

Weighting the coarse class influence to the error used for back propagation in the network and/or swapping the Multitask Learning rule to that based on the single 100 class classification, helps biasing the network towards classifying the 100-class targets since the start, effectively reaching better performances in the first 30 to 40 epochs than the simple Multitask Learning approach. However, the continuous diminished *pull* of the coarse classes during training reduces the regularizing effect of the approach and performance drops before surpassing the initial CNN training routine.

Multitask Learning with *decay* attempts to get the best of both worlds by initially pushing the state of the network to one capable of classifying images into both the 100 fine grained and the 20 coarse grained classes, and decaying the pull of the coarse classes exponentially as the epoch number increases, so to *gradually* tune the network towards the fine grained classes. The approach out-performs all previous approaches, effectively achieving the highest classification accuracy over all previous experiments. Table 2 summarizes the results in this paper.

5 Future Work

The concept of Multitask Learning is a powerful one and can be useful in a varied number of applications. In this set of experiments we implemented Multitask Learning variations on the CIFAR-100 dataset, making use of the given 20 coarse classes labels. For future experiments, it would be worth exploring the improvement Multitask Learning could have on CIFAR-10. More interestingly, it should be possible to use the inverse approach; i.e. label the images in CIFAR-10 into $N >> 10$ classes, and improve the 10 (coarse) class classification accuracy by inducing the network to learn more fine grained descriptions of images.

In the Multitask Learning *decay* variant, we set the α hyper-parameter to 50, by picking a value that returns reasonable weightings over the observed average number of epochs it takes the CNN designed to converge. Although sensible, it is worth exploring whether a more systematic approach

to setting the α hyper-parameter results in an increase in the classification performance (e.g. an hyper-parameter space search). Moreover, it would be interesting to compare the exponential decay implemented, to other forms of decay, which would change the weighting distribution of the 20-coarse multi class entropy error over the epochs.

In this experiment, we limit ourselves so relatively shallow Convolutional Neural Network, favoring the study of the variation introduced by the various approaches, to reaching state of the art performances in the datasets. The approaches here described, however, can be applied to any network almost independently of its layout or depth. For this reason, it would be interesting to reproduce the current results on deeper and/or different architectures, and assess the generality of the described Multitask Learning variants in other frameworks.

References

[1] L. Scimeca. Machine learning practical - coursework 3. 2 2016.

[2] L. Scimeca. Machine learning practical - coursework 2. 12 2016.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*.

[4] TensorFlow. Convolutional neural networks.
url: `https://www.tensorflow.org/tutorials/deep_cnn`
last checked: March 16, 2017, February 2017.

[5] S. Renals. Machine learning practical lecture 4.
url: `http://www.inf.ed.ac.uk/teaching/courses/mlp/2016/mlp04-learn.pdf`
last checked: March 16, 2017, October 2016.

[6] Ramtin Mehdizadeh Seraj. Multi-task learning.
url: `https://www.cs.ubc.ca/~schmidtm/MLRG/Multi-task%20Learning.pdf`
last checked: March 16, 2017, January 2014.

[7] L. Scimeca. Machine learning practical - coursework 1. 11 2016.

Table 1: The table summarizes the results obtained on CIFAR 10 on this second set of experiments, as compared to the best performing baseline NN developed in previous experiments [1]. The models are ordered in descending order of validation accuracy (i.e. from most to least performing).

Model	**Minimum Validation Error**	**Maximum Validation Accuracy**	**Average Training Time (sec/epoch)**	**Epochs to Convergence (Early stopping)**
CNN with simple training	0.978111	0.661200	902.17	59
Simple Feed-Forward NN with Dropout	1.30684	0.5432	65.11	193

Appendix I: Tables

Table 2: The table summarizes the results obtained on CIFAR 100 on this second set of experiments, as compared to the best performing baseline NN developed in previous experiments [1]. The models are ordered in descending order of validation Accuracy (i.e. from most to least performing).

Model	**Minimum Validation Error**	**Maximum Validation Accuracy**	**Average Training Time (sec/epoch)**	**Epochs to Convergence (Early stopping)**
CNN with Multitask Learning & Decay	2.311161	0.367200	1083.82	95
CNN with simple training	2.537402	0.366800	896.33	76
CNN with weighted Multitask Learning & Transfer of knowledge	2.437121	0.349900	1149.65	98
CNN with Multitask Learning	2.417679	0.343600	1009.45	105
CNN with Multitask Learning & Learning swap	2.467854	0.342500	1183.17	86
CNN with Multitask Learning & No regularization	2.837735	0.325900	932.19	33
Simple Feed-Forward NN with Dropout	3.101271	0.249600	78.32	193