TPSNet: A memory efficient ResNet Architecture for Image Classification

ECE-GY-7123: Deep Learning
Project-1
Pratyaksh Prabhav Rao (pr2257), Taiyi Pan (tp2231), Sudharsan Ananth (sa6788)

Abstract

Recent advancements in deep learning have led to highly accurate image classification models. Deep residual networks have proven to show state-of-the-art performance on many image classification tasks. However, the performance is achieved at the cost of doubling the number of layers, thus significantly increasing training time. In this project, the goal is to design a novel ResNet architecture to maximize classification accuracy on the CIFAR-10 dataset under a constraint on parameter count (< 5M). Additionally, this work investigates various hyperparameters that play a major role in model accuracy. The parameters that we are allowed to optimize are - 1) *Number of residual layers*, 2) *Number of blocks in residual layer* i, 3) *Number of channels in residual layer* i 4) *Convolutional kernel size in residual layer* i, 5) *Skip connection kernel size in residual layer* i, and 6) *Average pool kernel size.* Inspired by our names, the best performing model, **TPSNet** achieves a testing accuracy **94.84** % with the total number of parameters equal to 4.96M. The proposed architecture outperforms several competitive baselines. Our code and models are available at https://github.com/taiyipan/TPSNet.

1 Introduction

Recent advancements in deep learning have led to highly accurate image classification models. Deep convolutional neural networks integrate feature extractors (series of convolution layers) and classifiers in an end-to-end fashion [1]. Network depth plays a vital role in model accuracy as the 'levels' of features can be enriched by stacking more convolution layers. However, it is observed that for deep networks, the model suffers from the vanishing/exploding gradients [2] and the degradation problem.

For a neural network with n hidden layers, n derivatives will be multiplied together. If the derivatives are large, the gradients will increase significantly as we propagate them through the model, eventually 'exploding'. This is commonly referred to as the exploding gradient problem. In contrast, if the derivatives are very small, they will decrease exponentially as we propagate the gradients through the model, until it eventually vanishes. This is referred to as the vanishing gradient problem. K. He et. al [3] argue that problem of vanishing/exploding gradients can be mitigated through normalized initialization of weights and batch normalization.

However, the main problem with deep neural networks is the degradation problem. As we increase the number of convolution layers, accuracy gets saturated and eventually starts to degrade. K. He et. al [3] propose adding identity mapping to mitigate the degradation problem. This way, a deep neural network should not produce higher training error than its shallow counterpart. The framework is called ***deep residual learning***. Fig. 1 illustrates the basic building block of the residual learning framework. The skip connections skip one of more layers and perform identity mapping where its output is added to the outputs of the stacked layers.

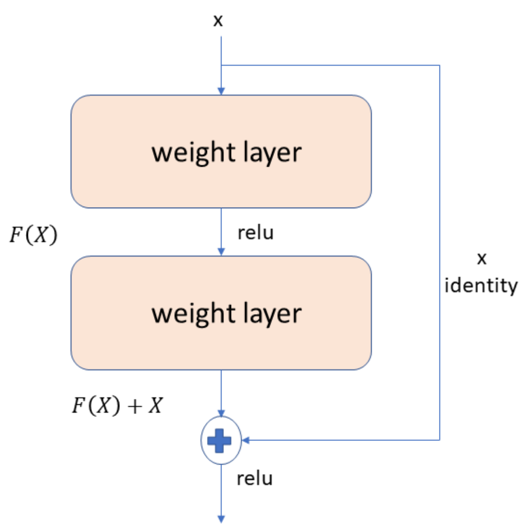

Figure 1: A basic building block of the residual learning framework.

In this project we are required to design and train a ResNet model for classifying images from the CIFAR-10 dataset. The main goal is to keep the size of the ResNet model under budget while maximizing testing accuracy. While the identity mapping is the main reason why deep residual layers perform well, it is at the same time a weakness of residual networks [4]. As the gradients are backpropagated through the network, it is possible that only a few blocks extract useful features or many residual blocks share little knowledge, thus making a small contribution. This problem is formulated as the diminishing feature reuse [5]. In this research project our main goal is to explore different techniques to tackle various problems faced during model hyperparameter tuning of ResNet models. Some of the popular techniques include but not limited to increasing the width of the ResNet blocks, adding dropout layers, increasing the resolution of the input image, etc.

The following are the main contributions -

1. Design a ResNet model whose parameters are less than 5M.
2. Test the model on the CIFAR-10 dataset.
3. Maximize test accuracy by optimizing various hyperparameters.

2 Methodology

Residual blocks can be defined as a function:

$$x_{l+1} = x_l + F(x_l, W_l) \tag{1}$$

where, x_{l+1} and x_l are input and output of the l-th layer of the ResNet network, F is the residual function and W are the block parameters. The ResNet consists of N residual layers, where each layer consists of one or more residual blocks. The input to the network is a tensor X_{input} with shape $< H_i, W_i, C_i >$, where H_i and W_i are the spatial dimensions and C_i is the number of channels.

Our goal is to maximize test accuracy for a given resource constraint. The hyperparameters we are allowed to change are - 1) *Number of residual layers*, 2) *Number of blocks in residual layer i*, 3) *Number of channels in residual layer i* 4) *Convolutional kernel size in residual layer i*, 5) *Skip connection kernel size in residual layer i*, and 6) *Average pool kernel size.*

The main goal is to maximize test accuracy for a given resource constraint and if formulated as an optimization problem [6] -

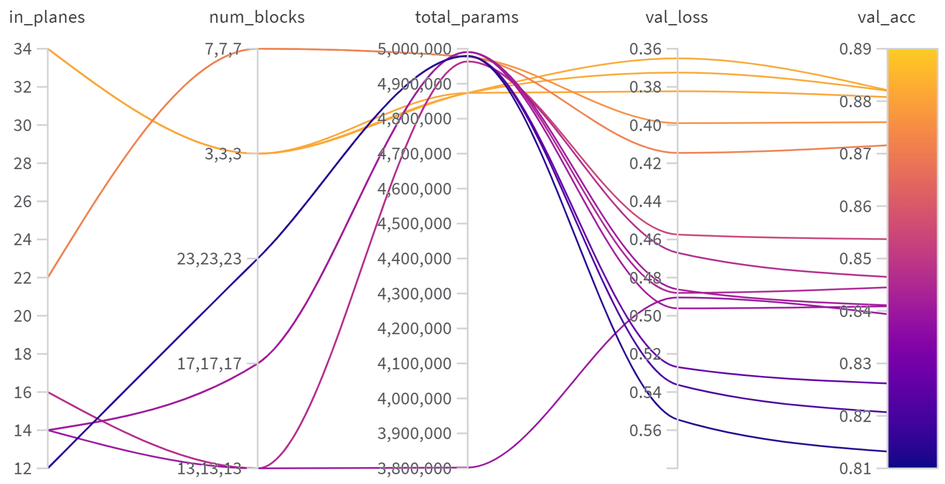

Figure 2: A basic building block of the residual learning framework.

$$
\begin{aligned}
(d^*, w^*) = arg_{<d,w>} \max Accuracy(N(d,w)) \\
s.t.\, N(d,w) = f(X_{input}) \\
\text{Memory}(N) \leq 5M
\end{aligned}
\tag{2}
$$

where, d is the number of ResNet blocks, w is the network width, and f is the ResNet model.

2.1 Network Depth

The number of convolutional layers play an important role in optimizing a deep convolutional neural network. The intuition is that deeper networks can extract richer features and generalize well on unseen data. However, that's not always the case due to the vanishing/exploading gradients problem.

2.2 Width of Residual Blocks

Another hyperparameter that has a major impact is the ReNet block width. Wider networks tend to capture intricate features and are easy to train [4]. Moreover, it is computationally more efficient to widen the layers than to increase the depth of the network as GPUs are great at running parallel computations on large tensors.

2.3 Data Augmentation

To improve the generalization capability of the model and to prevent overfitting, we utilize various data augmentation techniques. This way, more information can be extracted from the original dataset through augmentations. These augmentations increase the size of the dataset by oversampling. The techniques utilized in this project include - a) Random crop with a crop size of 32, b) Random rotation in a range (-5 degrees, 5 degrees), and c) Random horizontal flip.

2.4 Random search for hyperparameter optimization

In this project, we adopt the random search [7] technique for hyperparameter tuning. Hyperparameter tuning can make the difference between an average model and a highly accurate one. The main goal of our learning algorithm is to find a function f_θ that minimizes the pre-defined loss $L(x; f_\theta)$ over a dataset X_{train}. The learning algorithm computes f through the optimization of a set of parameters θ.

Table 1: Random search hyperparameter tuning for 30 epochs for width factor (k) = 2

No. of residual layers (N)	No. of residual blocks in each layer	No. of channels in first conv. layer (C_1)	No. of params	Test accuracy (%)
3	[3, 3, 3]	34	4.87M	88.20 %
3	[7, 7, 7]	22	4.97M	87.16 %
3	[13, 13, 13]	16	4.96M	84.65 %
3	[17, 17, 17]	14	4.99M	84.09 %
3	[23, 23, 23]	12	4.97M	81.32 %

The random search technique involves defining a search space as a bounded domain for the model hyperparameters and randomly sample points in that domain.

We use Ray Tune [11] for optimizing the model parameters. Tune is a Python library for experiment execution and hyperparameter tuning at any scale. Tune allows us to launch a multi-node distributed hyperparameter sweep.

3 Experiment

The proposed architecture was evaluated on the CIFAR-10 dataset [8]. The dataset consists of 32x32 color images drawn from 10 different classes. The CIFAR-10 dataset consists of 50,000 train and 10,000 test images. The model was implemented with PyTorch [9]. The network parameters are optimized using stochastic gradient descent (SGD) [10] with an initial learning rate of 0.01 and a cosine annealing learning rate scheduler. This scheduling technique starts off with a large learning rate and then aggressively decreases it to a value near 0, before again increasing the learning rate. Additionally, all networks were trained with dropout inserted into residual block between conv. layers. The loss function used is cross entropy -

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c}) \tag{3}$$

were M is the total of number of classes, y is the ground truth label, and p is the predicted probability observation of class c. All experiments were conducted on an Nvidia GTX 1660 with 16 CPU threads. Additionally, the hyperparameter optimization was was conducted on the HPC with 48 CPU threads and the RTX8000 GPU. For quantitative comparison, we use the accuracy metric to evaluate the performance of our model -

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

where, TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives.

3.1 Implementation

The detailed architecture of the TPSNet consists of 3 residual layers, each with 13 individual ResNet blocks, no. of conv channels in its first layer equal to 16 and a width factor of 2. The kernel size of all conv. layers are 3. The first conv. layer in the first residual block of layer $i \geq 2$ has a stride of 2 and in all other cases the stride is 1. The skip connection of the first residual block of layer $i \geq 2$ has a stride of 2. The model was trained for 300 epochs with a batch size of 128. The average pool size is 8.

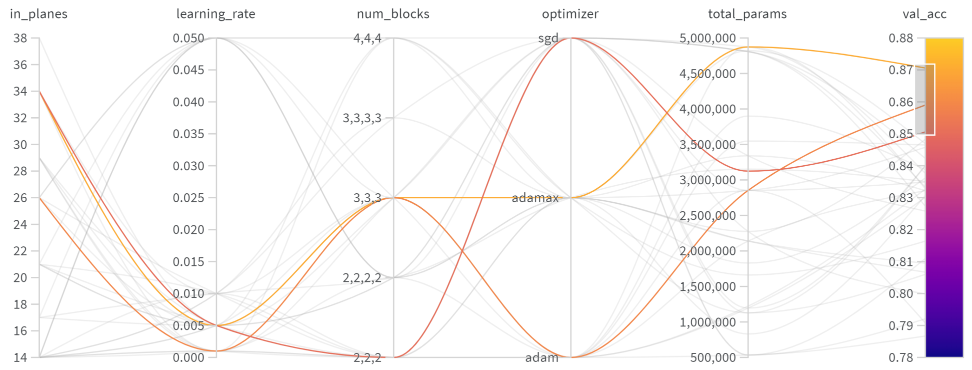

Figure 3: Visualization of random search hyperparameter tuning.

Table 2: Top 3 best performing models trained for 300 epochs

Baseline architectures (N)	No. of residual blocks in each layer	No. of channels in first conv. layer (C_1)	No. of params	Test accuracy (%)
Model-1	[3, 3, 3]	34	4.87M	94.02 %
Model-2	[7, 7, 7]	22	4.97M	94.24 %
TPSNet	**[13, 13, 13]**	**16**	**4.96M**	**94.84** %

4 Results

Table 1 summarizes the results of our top performing models within a pre-defined hyperparameter range. The range was defined by imposing the memory constraint (< 5M parameters). While implementing random search optimization we trained more than 500 models, thus we restricted the total number of epochs to 30. However, the top 3 models were later retrained for 300 epochs. Figure 3 illustrates the effect of different selection of hyperparametrs. It is observed that a 3 layer ResNet model outperforms a 2 and 4 layer model. Hence, while training our final model, we restrict the number of layers to 3. Furthermore, we can see that an odd number of residual blocks in each layer performs considerably better than the even number of residual blocks. Finally, it was observed that dropout decreases test error, thus preventing overfitting.

While increasing the width of the network increases the no. of parameters, it improves the performance across residual networks of different depth. It is interesting to note that while the shallow layers performed better than the deep networks when trained for 30 epochs, when trained for 300 epochs, the deeper network outperformed them. Table 2 indicates the testing performance of our top 3 models obtained after hyperparameter tuning. It is observed that our best model, TPSNet, achieved a test accuracy of 94.84% with the lowest training loss reaching 0.0009. Figure 4. and Figure 5. illustrate the the training curves for TPSNet. Models with comparable number of parameters turned out to give more or less the same test accuracy.

5 Conclusion

In this project, we designed a novel ResNet architecture to maximize classification accuracy on the CIFAR-10 dataset while ensuring that the total number of parameters are less than 5M. We implement a random search approach for hyperparameter optimization. Our best performing architecture, TPSNet achieved a test accuracy of 94.84 %.

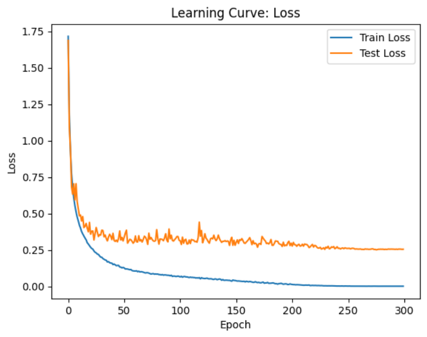

Figure 4: Loss vs Epoch for our best performing ResNet architecture.

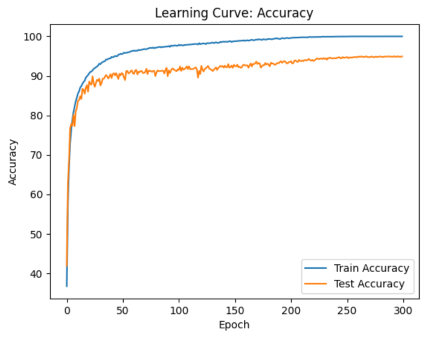

Figure 5: Accuracy vs Epoch for our best performing ResNet architecture.

References

[1] M. D. Zeiler and R. Fergus. *Visualizing and understanding convolu- tional neural networks.* In ECCV, 2014.

[2] Y. Bengio, P. Simard, and P. Frasconi. *Learning long-term dependencies with gradient descent is difficult*. IEEE Transactions on Neural Networks, 5(2):157–166, 1994. TELOS/Springer–Verlag.

[3] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[4] Zagoruyko, Sergey, and Nikos Komodakis. "Wide residual networks." arXiv preprint arXiv:1605.07146 (2016).

[5] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. CoRR, abs/1505.00387, 2015.

[6] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." International conference on machine learning. PMLR, 2019.

[7] Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of machine learning research 13.2 (2012).

[8] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2012.

[9] PyTorch. https://pytorch.org/.

[10] Deng, Li, Li, Jinyu, Huang, Jui-Ting, Yao, Kaisheng, Yu, Dong, Seide, Frank, Seltzer, Michael, Zweig, Geoff, He, Xiaodong, Williams, Jason, et al. Recent advances in deep learning for speech research at microsoft. ICASSP 2013, 2013.

[11] Liaw, Richard, et al. "Tune: A research platform for distributed model selection and training." arXiv preprint arXiv:1807.05118 (2018).

ACKNOWLEDGEMENT

We would like to express our thanks to the people who's discussion helped us through the project. We are grateful to Prof. Siddharth Garg, Prof. Arsalan Mosenia and the teaching assistant Ezgi Ozyilkan for their nonstop support. Lastly, we would like to extend our special thanks to the teaching team for giving us this opportunity to work on these assignments and projects. They were extremely helpful and pertinent to understanding the concepts.