

Transfer Learning and Deep Neural Network Acceleration for Image Classification

Team 26: Yeqing Huang, Weihua Huang, Arik Horodniceanu, Bowen Zhang, Houjian Yu

Abstract—This project aims at performing image classifications using transfer learning [1] in deep neural networks. Pre-trained ResNet [2] models are fine-tuned and tested on the 10 Monkey Species dataset [3] and the Stanford Cars dataset [4]. Transferring pre-trained ResNet-18 model to Monkey dataset achieves over 98% testing accuracy after only one epoch of training. Meanwhile, finetuning the pre-trained ResNet-152 model achieves 92.55% top-1 accuracy and 98.14% top-5 accuracy on over 8,000 test images of the Cars dataset. Techniques including data augmentation and curriculum learning are also used to improve the performance. Nvidia’s TensorRT framework [5] is also utilized for GPU inference acceleration and reaches over 3× boost in the model’s prediction speed.

github link: <https://github.com/eebowen/transfer-learning-and-inference-acceleration.git>

Index Terms—Deep Learning, Transfer Learning, TensorRT

I. INTRODUCTION

Tremendous progress has been made in object recognition with deep convolutional neural networks, thanks to the availability of large-scale datasets. However, these networks prove to be difficult and time-consuming to be trained from scratch when the amount of available data and devices are limited. The purpose of this project is to study the use of transfer learning [1] on a relatively small dataset, during which models are pre-trained on large datasets and then fine-tuned on the custom, small datasets. Deep learning is popular in computer vision and natural language processing since it is a optimization that allows rapid progress and improved performance when modeling the task. In this project, we make use of the pre-trained VGG-16 [6] and ResNet-152 [2] model, which includes features based on ImageNet [7], a large image dataset with over 1.2 million images on 1,000 categories. We fine-tune [8] these models to prove the effectiveness of the method and explore what techniques could be used to speed up the training process or improve the final performance.

Besides the training time and performance, the model’s prediction speed is also of great concern since it decides whether the model can be used to conduct real-time prediction and be utilized on mobile devices. In this project we use NVIDIA TensorRT [5], which is a platform for high-performance deep learning inference, to speed up the prediction process. It includes a deep learning inference optimizer and run time that delivers low latency and high-throughput for deep learning inference applications. We will deploy this accelerator and compare the model’s performance before and after.

II. RELATED WORK

A. Transfer Learning

Transfer learning [1] is a machine learning technique where a model trained on one task is re-purposed on a second related task. [9] In transfer learning, we first train a base network on a general database, and then we extract the learned features or transfer them to a second target network to be trained on our specific data set and task.

For transfer learning that use image data as input, it is common to use a deep learning model pre-trained for a large and challenging image classification task such as ImageNet [7]. Convolutional neural networks are often used for this type of application. Among them, VGG [6], ResNet [10] are mostly frequently used in image classification tasks. It is argued that lower convolutional layers capture low-level image features, e.g. edges, while higher convolutional layers capture more complex details, such as body parts, faces, and other compositional features. For a new task, we can thus simply use the off-the-shelf features of a state-of-the-art CNN pre-trained on ImageNet and train a new model on these extracted features. In practice, we either keep the pre-trained parameters fixed or tune them with a small learning rate in order to ensure that we do not unlearn the previously acquired knowledge. This simple approach has been shown to achieve astounding results on an array of vision tasks [13] as well as tasks that rely on visual input such as image captioning. The ImageNet task seems to be a good proxy for general computer vision problems, as the same knowledge that is required to excel in it is also relevant for many other tasks.

As a unified framework, transfer learning tries to optimize the learning task by reusing all the knowledge we have already learned from the more difficult task. Therefore, transfer learning is gifted with the ability to increase the utilization of data. An example of successful application of transfer learning is shown by Schroff et al. [11] who employ a pre-trained model to perform face recognition and achieve state-of-art results 99.67% , outperforming the one designed by Parkhi et al. [12] in which the network is trained from scratch with an accuracy 97.27%.

B. Compression

In recent years, deep convolutional neural networks have set new standards in several fields, particularly computer vision. A large obstacle faced by these advances is the considerable data storage requirement and inference times. The AlexNet [7] architecture requires 240 MB of memory to store the 60

million parameters contained in the five convolutional layers and three fully-connected layers. Another popular architecture, VGG [6], also requires significant data storage with 528 MB of memory. Due to advances in hardware in the form of GPU and CPU clusters equipped with considerable memory and computational resources, these neural networks can be trained within reasonable time and these memory requirements are not problematic to satisfy. However, in the case of mobile or embedded devices with limited computational capabilities, such resource draining deep neural networks cannot be deployed. To ameliorate this problem, we attempted to implement commonly used two compression techniques: Parameter pruning, where redundant parameters which are not detrimental to performance are removed and Quantization, where the network is compressed by reducing the number of bits required to represent each weight, or the weights are shared across neurons or layers. We attempted to implement both of these methods on the transferred domain in our problem, albeit unsuccessfully. This will be detailed more in section V.

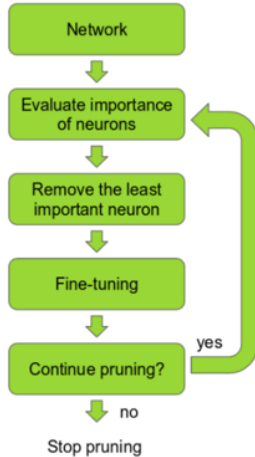


Fig. 1: Iterative pruning process.

1) *Pruning*: Pruning is the process of removing the less contributing weights to compress the network. This way, pruning methods introduce sparsity of the parameters. This high sparsity offers two advantages: Smaller data storage requirements for the pruned model and reduced inference times due lower computational complexity. Pruning was first proposed by Lecunn et al [13], where it was suggested to reduce the number of parameters by the magnitude of the Hessian of the loss function. This work reduced the parameters of the model by 60% and increased its accuracy on the MNIST dataset [14], indicating that pruning the weights reduced overfitting. However, the problem with this method is that for deep networks, calculation of the Hessian becomes unfeasible. Recent works take different approaches to pruning. Basic pruning methods remove parameters from pre-trained models with criteria such as the magnitude of the weights. Han et al [15] suggested to reduce the total number of parameters and connections in the entire network. Here, an iterative three step

procedure is given based on the magnitude of the weights: First, learn the connections, then prune the insignificant connections and finally retrain the network. L2 regularization is used to determine which weights have greater importance on the networks accuracy. The weights with magnitude less than a threshold are then set to zero, after which the network is re-trained using the sparse set of weights. This process is illustrated in figure 1.

2) *Quantization*: Another straightforward way to exploit the redundancy in deep neural networks is to reduce the bit width of its parameters. It seems that reducing the bit width of deep network from 32-bit floating point down to 8-bit floating point numbers does not impact performance significantly, even for architectures that we designed efficiently in advance, such as [16] or [17].

III. DATASET AND DATA PRE-PROCESSING

Our combined data consists of two datasets. 10 Monkey Spices dataset is for the demo purpose, which contains around 1,400 RGB images, mostly in JPEG format, of monkeys of 10 spices from Wikipedia’s monkey cladogram. Stanford cars dataset will be mainly used for the project, it contains 16,185 RGB images of 196 car classes, where the classes are typically at the level of its make, model and year.

Besides the images, both datasets provide a degree of labels and the bounding box information for the location of the target object in the image. Before used as the input of the neural network, the images are first cropped using the bounding box coordinates provided to serve as the purpose of removing the background. Since all images are re-sized into the same size when forming the mini-batches, we need to pad the images into the same shape as re-sizing without padding might lead to the problem that some images are distorted in term of their width-to-height ratio which might further result in performance degradation. For our project we choose to use zero-padding as it gives the least variance in performance.

All images are then resized to 224×224 as padded and then convert to arrays to serve as the input of the convolutional layers, note that all values in the array should be normalized into the range $[-1, 1]$ for better convergence. Data augmentation is not utilized at the first stage of the experiment and will be added when we try to improve the model’s performance. This part will be explained in detail in later sections.

IV. METHODS

We use ResNet pre-trained on ImageNet dataset. First we explain the details of convolutional neural networks (CNNs) with its implementation details. Then we illustrate the ResNet architectures and how they were adapted for the specific task using transfer learning.

Lastly, We discuss the use of TensorRT to increase the training and testing speed of our model.

The use of Convolutional neural network (ConvNets or CNNs) has been the defining feature behind recent breakthroughs in image classification, object detection, image creation and many more areas. Without exaggeration, it can be

said that CNN alone reinvented the landscape of computer vision. Despite its alleged claim that CNN was inspired by the structure of mammalian visual cortex, it is obvious that the use of multiple convolution filters takes roots in traditional 2D image processing techniques: filter banks. Using filter banks such as Discrete Fourier transform, Discrete wavelet transform to slide though images have been established methods in image processing. Unlike these traditional filter banks that use handcrafted filters to process images, CNN adapts the filters based on the specific training data set. This data driven process is a paradigm shift that breaks away from tradition approach.

For a given input layer a number of 2 dimensional feature map or image stacked on a 3rd dimension or depth is typically fed in and a smaller rectangular kernel is convolved with the image. The explicit formulation of this process is governed by the equation below, where h is the input layer, and a is one kernel and the output is one layer of a feature map. There are multiple reasons for using this convolution operation on images. Since pixels in images typically are only related to pixels in nearby regions and independent of pixels far away, convolution gives pixel features temporal independence. i.e. extracted features are only functions of pixels nearby. Another reason for using convolution is that the output feature map is typically smaller than the original input, as expressed in the following equations for the output size relationships.

$$W_o = \frac{(W_{in}F_w + 2P)}{S} + 1 \quad (1)$$

$$h_o = \frac{(h_{in}F_h + 2P)}{S} + 1 \quad (2)$$

Where, W_o is the output width, W_{in} is the input image width, F_w is the kernel width, p is the padding size, and s is the stride used. Likewise, height is expressed in the latter equation. This decrease in size allows for a method of dimensionality reduction which is important when reducing the number of features down for large images (curse of dimensionality). One last important reason for using convolutional kernels is that the inputs used shared weights. Since a single kernel will be used for a convolution over an entire image, it greatly reduces the require number of parameters of each pixel or dimension. Given that these networks can reach parameter numbers in the order of 106 , reducing the number of weights used can reduce over fitting, and the number of required training images.

A. Network Architectures

We adopt Residual Neural Network (ResNet [2]) as our base network architecture for image classification. It won ILSVRC 2015 [18] with 3.57% top-5 error and was the first system to beat human level prediction on ImageNet [7].

It is designed to overcome the vanishing gradient problem that often plagues deep neural networks. ResNet is made up of many residual blocks as shown in the right of 2, each of which has a shortcut between input and output. This gives a direct path for error to back propagate through the entire network. By stacking up these simple building blocks, ResNet can be

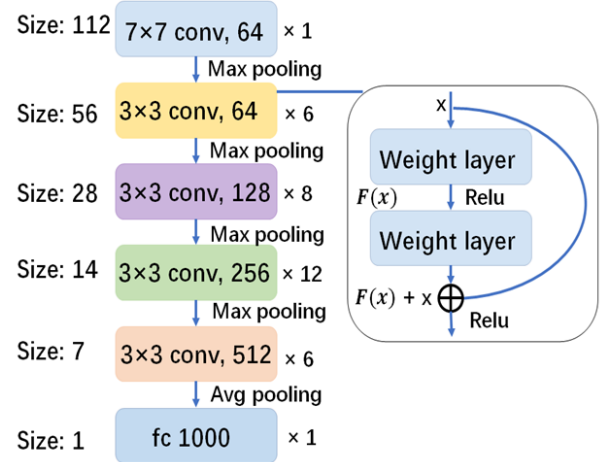


Fig. 2: ResNet used as a base for transfer learning

built with many layers, thus achieving a very deep network structure. This allows the network to learn more complicated representation and features.

We have experimented with both ResNet 18 and ResNet 152. They were pre-trained on ImageNet with 1000 classes. We kept the weights of feature extraction layers frozen and adapted the last FC layer to fit our specific classification task. By using transfer learning techniques, we were able to fine tune our ResNet networks with small databases and achieve good results.

B. TensorRT

To increase the inference speed of our transfer learning model, we used TensorRT [5] developed by Nvidia. TensorRT is a deep learning platform that optimizes deep neural network models and speeds up for inference runtime to deliver a low latency and high-throughput for deep learning applications. TensorRT performs several significant transformations to optimize the neural network graph (Fig. 6). First, to prevent unnecessary computation, layers with unused output are eliminated. Next, some convolutional, bias, and ReLU layers are combined to create a single layer where possible. Then, combine the aggregated layers horizontally. This is called Horizontal layer fusion, and it enhances efficiency by mixing layers with comparable parameters that take the same source tensor and perform the same operations. By restructuring the graph, TensorRT is able to perform the operations much quickly and more efficiently without sacrificing any accuracy.

V. EXPERIMENTS AND RESULTS

All code was written in Python using the Pytorch [19] machine learning software package. Visualization was done using the Matplotlib [20] library. Unfortunately, the compression proved difficult to implement and the preliminary results we got were poor, as even for a small compression ratio the accuracy dropped significantly. Therefore, due to time constraints we abandoned that route. More relevant results are shown in this section.

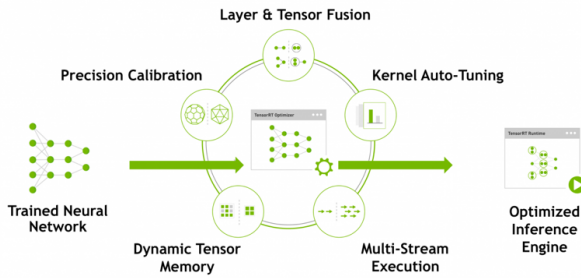


Fig. 3: Illustration of TensorRT operation

A. Evaluation Metrics

1) *Cross-entropy*: For a typical classification problem like this, it is very standard to consider the loss as the cross-entropy for K classes, which is mathematically defined as:

$$E(\mathbf{W}) = - \sum_{i=1}^N \sum_{k=1}^K d_k^i \log y_k^i \quad (3)$$

Note that here y is the network's prediction, d stands for the ground truth, the superscript i represents the i^{th} example and the subscript k stands for the k^{th} class. The prediction results generally go through the soft-max function:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K \quad (4)$$

to ensure all values for a single prediction are bounded in between $(0, 1)$ and sum up to 1 to be interpreted as the predicted probability for each class.

Using this loss, the network searches for the optimal solution such that \mathbf{W}^* such that $\nabla E(\mathbf{W}^*) = 0$.

2) *Top-1 and Top-5 accuracy*: A more direct way to access the model's performance is to measure its accuracy, which is defined as the ratio of correct predictions over all predictions.

For top-1 accuracy, we only consider the prediction with the highest probability, while for top-5 accuracy the top 5 probabilities are all considered and the prediction is defined as correct if the ground truth is contained in the 5 predictions. We introduce this concept since we have a rather large number of classes (196) and sometimes the difference between classes can be small and vague (i.e. cars with the same brand and make but different year).

B. Classification of Monkey dataset for demo purposes

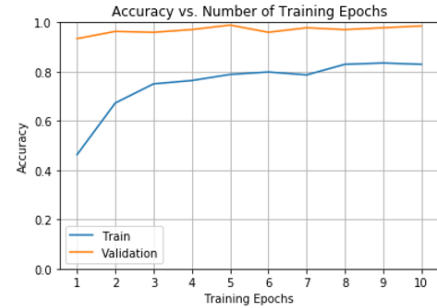
Since the Monkey dataset has relatively less number of classes, we start with using ResNet-18, which has a rather simple structure, to verify the effectiveness of transfer learning. Based on pre-trained ResNet-18 model, we deploy two methods in training: freeze-and-train & fine-tuning.

Freeze-and-train regards the original pre-trained network as a fixed feature extractor by removing the last fully-connected layer from the pre-trained model. By adding a new adapted

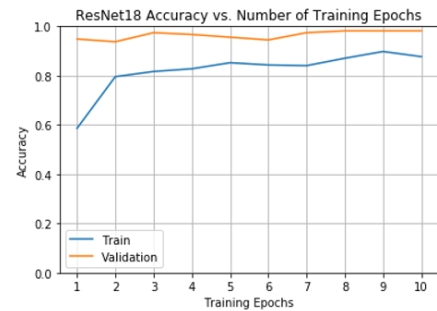
fully-connected layer to the feature extractor, this method trains the last layer by leaving the rest frozen (i.e. parameters other layers will not change during training).

Fine-tuning is the opposite, which means not only the classifier, but also all other feature-extracting layers, are trainable. We set up this experiment for comparison since transfer learning can also achieve satisfactory results when the features extracted by the pre-trained model are useful enough for the classifier to distinguish the difference between the classes on our custom datasets.

Using Adam [21] as the optimizer, and set the learning rate to 10^{-3} , the results for both methods are shown below.



(a) ResNet-18 on Monkey dataset - freeze and train



(b) ResNet-18 on Monkey dataset - fine tuning

We see that only after one epoch of training, both methods are able to achieve over 90% validation accuracy. Both methods reach over 98% accuracy on validation data after 10 epochs, which has proved that efficient and also effective. Not many variance can be found between the two methods, which means that classification of the monkey species is rather simple and the pre-trained model has already extracted useful features for the task. We expect a larger difference in performance as we move to the harder and larger cars dataset.

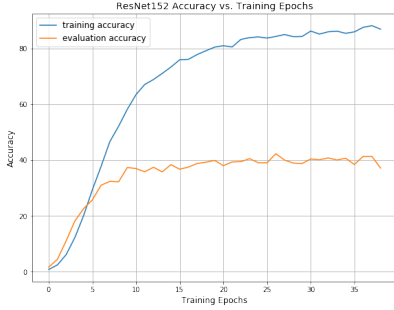
C. Classification of cars dataset

1) *Freeze and Train vs. Fine-tuning*: For this dataset we set up several experiments based on the pre-trained ResNet-152 model. We choose ResNet-152 since it has the most complicated structure in the ResNet family, thus enables it to extract more and deeper features for such a complicated task.

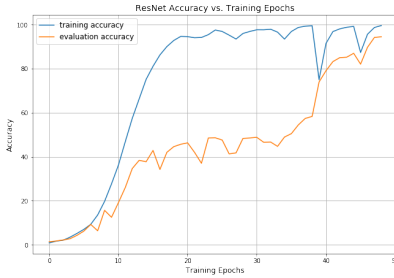
For the around 8,000 images in the training dataset, we split 70% of them for training while the rest 30% are used for validation. We first conduct freeze and train, as shown in

the figure below, after nearly 40 epochs of training, the model still struggles to get 40% on validation data while the training accuracy has reached over 80%, which indicates that the model is heavily overfitted. Hence, in the next step, we fine-tune all the layers to improve performance.

Using Adam as optimizer and learning rate of 10^{-3} , the ResNet-152 model achieves 87.04% validation accuracy after 44 epochs. Then we change the learning rate to 10^{-4} and train 4 more epochs to reach validation accuracy of 94.57%.



(a) ResNet-152 accuracy on cars dataset - freeze and train



(b) ResNet-152 accuracy on cars dataset - fine-tune

Despite impressive prediction accuracy on validation data, the model gives a rather low 82.20% accuracy when tested on the testing dataset which contains over 8,000 images. This problem indicates that the distribution of features in the testing set might be slightly differently from that of the validation set and our model is not robust enough to generalize to that change. Therefore, in the next step, we use data augmentation to further improve the performance.

2) *Data Augmentation and curriculum learning*: There exists a strong relationship between the perform of a neural network with the amount of the data provided to it. By applying data augmentation we are able to create new training data from the existing training data. To save storage space, we chose offline data augmentation which means that the new training data are generated on the fly. Initially we applied random horizontal flip, random rotate with rotating angle uniformly distribution within $[-10^\circ, 10^\circ]$, and randomly change the images saturation, brightness, contrast and hue to $\pm 20\%$ of its original value.

However, the above value doesn't deliver good performance as the validation accuracy stagnate at around 70%, which is even worse than that without data augmentation. To overcome the problem, we applied curriculum learning. This is a training approach during which we start with the easy examples and

then increase the difficulty step by step. In our case, we started training without data augmentation until training accuracy reaches nearly 99% and validation accuracy stops to increase. Then we continue training using slightly augmented data, by introducing random horizontal flip. As augmentation is introduced, the model's accuracy on training and validating would initially drop for the first few epochs, and keeps rising to higher value afterwards. We repeat the process for several times, whenever the model's training accuracy goes nearly 100% and validation accuracy stops rising, we further augment the data by introducing more distortion as we continue training.

The augmentation process can be summarized as: + random horizontal flip \rightarrow + random rotation and random affine \rightarrow + random brightness, contrast, saturation, hue change by 20% \rightarrow + random brightness, contrast, saturation, hue change by 50%.

The accuracy vs. training curve is shown below, note that for each sharp drop of the training accuracy indicates the start of a new around of data augmentation.

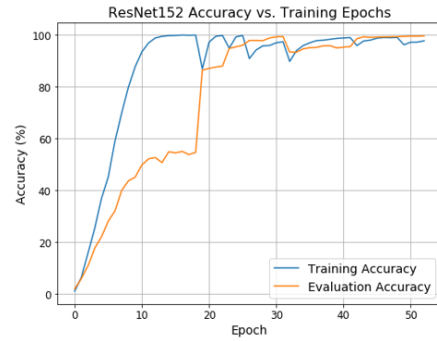


Fig. 6: Resnet accuracy on cars dataset - data augmentation & curriculum learning

By using the above techniques, we are able to further improve the testing performance.

	w/o Data Augmentation	w/ Data Augmentation
Top-1 Accuracy(%)	82.20	92.55
Top-5 Accuracy(%)	94.45	98.14

TABLE I: Comparison before and after data augmentation

The confusion matrix shown above for the fine-tuning method on Stanford Car Dataset is in Fig. 7. The diagonal entries indicates the accuracy of our classifier, which is align to the high classification precision rate.

D. the Effect of TensorRT

Generating TensorRT [5] inference engine requires trained model and parameters. First, export Resnet-152 model and weights we are using from Pytorch and load them to tensorRT. Then, Using tensorRT to create engine for this specific model and finally run the engine using same input as the Pytorch model. After some parameters tuning, the optimized engine vastly increased inference speed, as shown in Fig. 7.

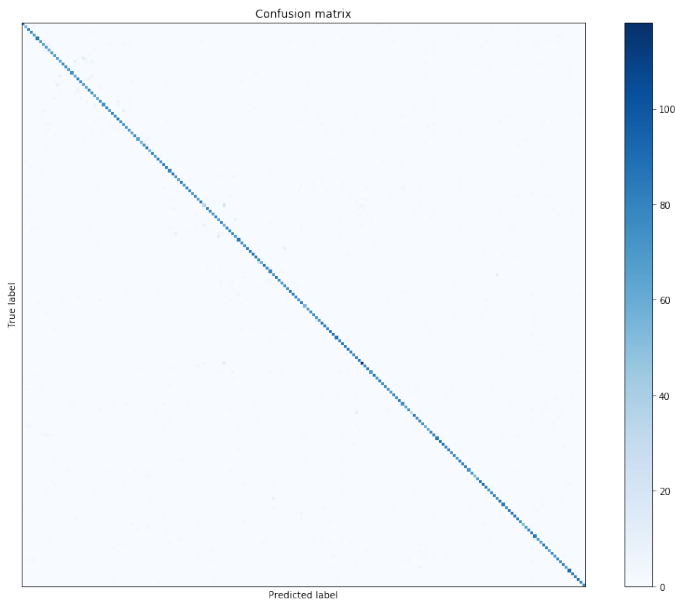


Fig. 7: The confusion matrix of the fine-tuning method on Stanford Car Dataset

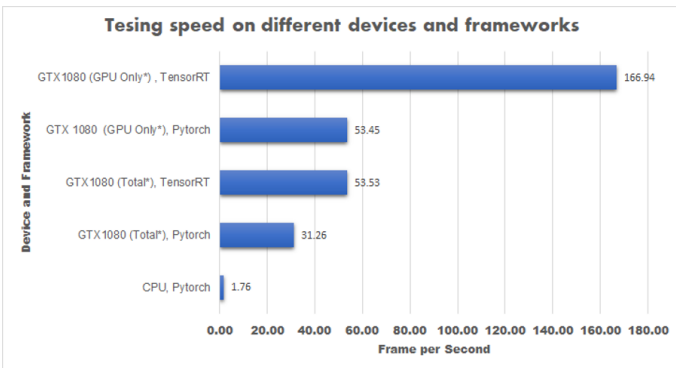


Fig. 8: TensorRT results (GPU only*: the GPU processing speed for prediction. Total*: total processing speed including loading image, preprocessing, and post processing using CPU.)

VI. DISCUSSION

From results above, we see that our method worked well for classification, both for the easier task of the Monkey dataset and the harder task of the Cars dataset. Fine-tuning did not have much effect on the results, despite our expectation for some improvement. However, we think this is because both of our targets were inside the original Imagenet [7] dataset so the domain adaptation was not difficult. We theorize that if we had done this for images not in the original training set (e.g. medical images), this would not have been so, as fine-tuning has been shown to perform better than freeze-and-train in these contexts.

TensorRT also performed well, successfully reaching a relatively high accuracy for cars dataset compared with the state of art. The inference speed after optimization is $30\times$ faster than CPU and $1.7\times$ faster than the GPU without optimization.

GPU processing speed is $3\times$ faster than the one without this optimization. As mentioned, unfortunately we could not get the compression methods to run well. There are several possible reasons for this: 1. We used Pytorch, which is not the friendliest environment for neural network compression. TensorFlow [22] is a machine-learning package which supports methods like quantization as a built-in function. Also, the hyper-parameters in these schemes were difficult to optimize. Choosing the threshold in pruning is a tricky matter, and even a high one would yield poor results.

VII. CONCLUSION AND FUTURE WORK

We can conclude from the results that for moderately closely related problems, transfer learning works very well. TensorRT also performs well, acting as a plug-and-play device which speeds up the performance of our model significantly.

If we had more time to implement this project, we are confident that we would have been able to make the compression schemes mentioned above work. There are recent papers ([23] [24] [25]) that show very impressive results in both the pruning and the quantization avenues of research. The above are mostly implemented in the original domain, and applying these methods to the transfer domain can prove to be an interesting direction of research. We may pursue these directions in future classes. Also, implementing compression methods in addition to using TensorRT seems like an interesting direction, enabling to be ran on mobile or embedded devices in real time.

VIII. ACKNOWLEDGMENT

The team would like to express our gratitude to Professor Peter Gerstoft and teaching assistants Harshul Gupta, Siva Prasad Varma and Ruixian Liu for a professional and informative course in ECE 228.

IX. CONTRIBUTIONS

- Bowen, Houjian: Code, poster
- Yeqing: Code, report
- Weihua, Arik: report.

REFERENCES

- [1] L. Y. Pratt, "Discriminability-based transfer between neural networks," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. Morgan-Kaufmann, 1993, pp. 204–211. [Online]. Available: <http://papers.nips.cc/paper/641-discriminability-based-transfer-between-neural-networks.pdf>
- [2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: <http://arxiv.org/abs/1512.03385>
- [3] Kaggle, "<https://www.kaggle.com/slothkong/10-monkey-species>," *Kaggle*, 2019.
- [4] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.
- [5] NVIDIA, "<https://developer.nvidia.com/tensorrt>."
- [6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.
- [7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [8] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *CoRR*, vol. abs/1411.1792, 2014. [Online]. Available: <http://arxiv.org/abs/1411.1792>
- [9] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct 2010.
- [10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [11] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.
- [12] O. M. Parkhi, A. Vedaldi, A. Zisserman *et al.*, "Deep face recognition." in *bmvc*, vol. 1, no. 3, 2015, p. 6.
- [13] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.
- [14] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [15] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.
- [16] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.
- [17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
- [18] 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.
- [19] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.
- [20] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <http://tensorflow.org/>
- [23] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.
- [24] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016. [Online]. Available: <http://arxiv.org/abs/1611.06440>
- [25] Y. Zhao, X. Gao, D. Bates, R. Mullins, and C. Xu, "Efficient and effective quantization for sparse dnns," *CoRR*, vol. abs/1903.03046, 2019. [Online]. Available: <http://arxiv.org/abs/1903.03046>