Final report on the TinyImage dataset classification

Esben Kran[1]

Abstract: This report showcases and represents reproducible code to classify the 100 classes in the TinyImage dataset, a subset of the ImageNet dataset. Three models are trained and evaluated, with a transfer learned model from the MobileNetV2 pretrained model providing the highest accuracy on the test set of 45.6%. A shallow CNN is worse than chance and a deep CNN reaches 28% accuracy.

Keywords: Machine learning, ImageNet, MobileNetV2

Introduction

Image classification is an important tool in machine learning and a classic test case for model architectures. The most famous image classification dataset is the ImageNet dataset [1] that supposedly kickstarted the deep learning revolution [2] through the participation of AlexNet [3] in the ImageNet classification challenge. In this report, we present our results to classify a subset of ImageNet, the so-called TinyImage dataset.

Methodology

The TinyImage dataset consists of 100 classes, 500 images for each class, and 5000 test images for evaluation on Kaggle[2]. Each image is a 64x64x3 tensor, i.e. 64 pixel square images with three color channels for red, green, and blue.

The device used is a Windows 11 laptop with 32GB RAM and a CUDA GTX1650Ti with 6GB dedicated memory. To run the models on the GPU, we use CuDNN [4] with Tensorflow GPU. We evaluate different models on the TinyImage dataset using an 80/20 train/validation dataset

[1] 1929452, eskchris@ucsc.edu

[2] https://www.kaggle.com/competitions/ucsc-cse-144-spring-2022-final-project/leaderboard

split. We choose not to use k-fold cross-validation purely because of limitations in computing power.

Layer types

We use Tensorflow and Keras [5] to create deep convolutional neural networks (DCNN). To create such an architecture, we use an array of layers with different properties.

1. Convolution layers

Convolution layers fit filters to the training data to generate an output. The filters use a weighted average of the neighboring pixels where the weights are the filter's parameters. The use of convolution layers were originally inspired by the visual cortex in the human brain [6] and a classic example of early-stage convolution filters are the line detectors. These are convolution filters that have learned vertical, horizontal, or diagonal weightings of their filter space and thus activates when the input pixel amplitudes correspond to e.g. the vertical line. This is also seen when the human brain does early visual processing but deeper layers might not correspond as well [7].

2. Max pool layers

The max pooling layers take a maximum value of a specific matrix of pixels (e.g. 3x3, though usually 2x2) of its input which creates a smaller image for the output. In this way, a 10x10 image becomes a 5x5 with a 2x2 filter size. This is used to create deeper representations of the input images and usually proceeds a convolution layer.

3. Batch normalization

Batch normalization reduces the problem of internal covariate shift that deep neural networks (DNNs) can have. Batch normalization (or batchnorm) normalizes (standardizes values with a mean 0 and a standard deviation of 1) the input weights to the next layer to ensure that layer parameters do not shift significantly throughout the model and make it easier to evaluate the parameter weight on the loss [8].

4. Dropout layers

Dropout layers make the connections between the previous layer and the next layer stochastically turn off. They are given a probability of turning of connections, e.g. 50%, and have no trainable parameters. The dropout layers ensure that overfitting does not happen because of overreliance on specific parameters and reduces the possible dependence on the training data.

5. Flatten layer

The flatten layer is used after all the convolution layers in a DCNN to make the two-dimensional images into one-dimensional data arrays. This makes it possible to use other types of layers on the data.

6. Global Max Pooling layers

These layers work to reduce the dimensionality of CNNs and replaces the Flatten layer and even sometimes the fully connected layer.

7. Dense layers

The dense layers are fully connected to the input data (each pixel feeds into each dense neuron) and are often used in DCNNs at the end of the process to convert the deep convolutional representation embeddings into a representation of the output classification. They are the bread and butter of neural networks and a development of the oldest neural network, the Perceptron [9].

Data augmentation

With the use of the ImageDataGenerator function from Keras, we rescale the pixel values within 0-1 by dividing by 255, randomly rotate within 8 degrees, randomly shift the image 10% horizontally and vertically, randomly shear (“tilt”) the image by up to 30%, randomly zoom by up to 10%, and randomly flip the image vertically and horizontally. With this method, we can expand the dataset by generating new samples. We fit the image generator to the training data and use the fit_generator method to train.

Training process

The model runs train with different batch sizes and epoch amounts as seen in the results section. The batch sizes affect training times and memory usage. For the big models (deep and MobileNetV2), a batch size of 20 was used because their memory usage is high compared to the shallow network. The loss used to evaluate the network performance for all models is the categorical cross entropy. It measures the amount of surprisal between the distributions of the model predictions and the true labels.

The learning rate for each is set to 0.01 for the non-pretrained models and 1e-5 for the MobileNetV2 transfer learning. The reasons pertain to the fact that the MobileNetV2 weights don’t need to do initial exploration of visual space at least for its earlier layers. The smaller learning rate ensures that the model searches the loss space in more detail around its already optimized area. Adam is used as the optimizer for both.

Shallow convolutional neural network

The shallow neural network is a sequential model that consists of a 2D convolution layer with 32 filters, a 2D max pooling layer with 2x2 filters, another 2D convolution layer with 64 filters, another 2D max pooling layer with 2x2 filters, a flattening layer, a dense layer with 256 fully-connected neurons, and another dense layer with 100 neurons for classification. All trainable layers (conv and dense) use ReLU activation except the last layer that uses sigmoid activation as a class classification probability measure.

Two types of the shallow CNN were tested. One without any regularization and the other with L2 regularization on all layers. The total parameter count for both models is 4,239,652.

Deep convolutional neural network with data augmentation

The deep CNN has significantly more complexity in the amount of layers and parameters. Sequentially, it consists of two sets of a Conv2D with 32 filters (without padding) and a batchnorm layer, a max pooling layer, 25% dropout, Conv2D with 64 filters, batchnorm, 25% dropout, Conv2D with 128 filters, max pooling, 25% dropout, a flattening layer, 512 neuron dense layer, batchnorm, 25% dropout, 128-neuron dense layer, a batchnorm, 25% dropout, and the output 100-neuron sigmoid activation dense layer. The total parameter count is 11,260,740.

Transfer learning on MobileNetV2

The last DCNN is an extension on the pre-trained model MobileNetV2 [10, p. 2]. We freeze all but the last four layers, add a GlobalMaxPooling layer, a 1024-neuron dense layer, and a dense 100-neuron classification layer.

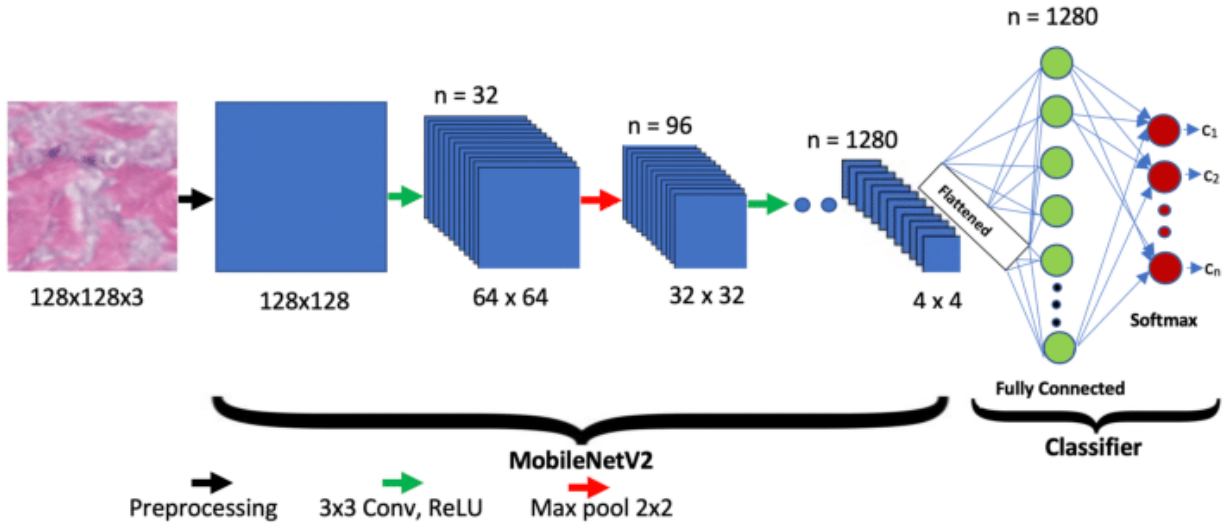

Figure 1: The original VGG16 network [10, p. 2], [11]

Results

Accuracy and loss results

The following figures show the accuracy and loss of each network type respectively. Figure 2 shows that the shallow CNN does not learn anything major and stays at chance or below performance. Figure 3 shows that the deep CNN works very well and reaches upwards of 30% performance. Running the same training process with L2 regularization reduces the performance towards chance level and the results from the regularized model are excluded as a result of its dysfunctionality.

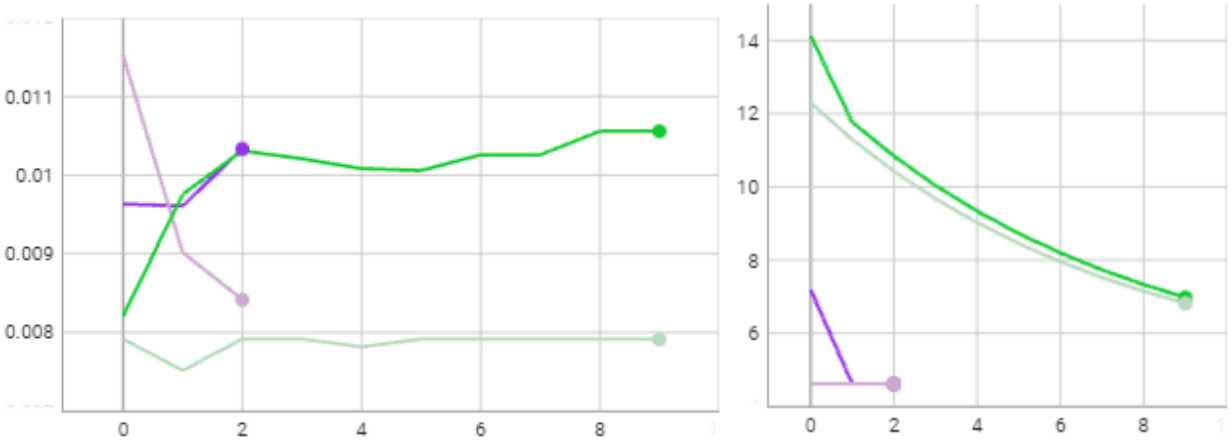

Figure 2: The shallow convolutional neural network's accuracy (left) and loss (right). It can be seen that it is never consistently above 1/100th, i.e. it is no better than random. Each color signifies the two different models run and the light version is the validation accuracy and loss. Purple = without regularization, green = with regularization.

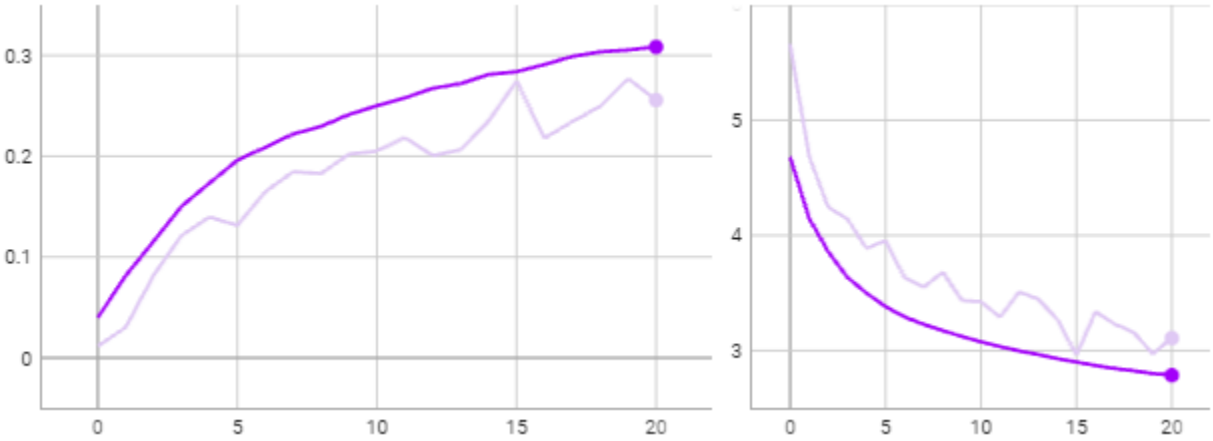

Figure 3: The deep convolutional neural network's accuracy (left) and loss (right) by epoch.

Figure 4 shows the performance of the neural network transfer learned with MobileNetV2 [10, p. 2]. This is the sequential graph for transfer learning and fine-tuning.

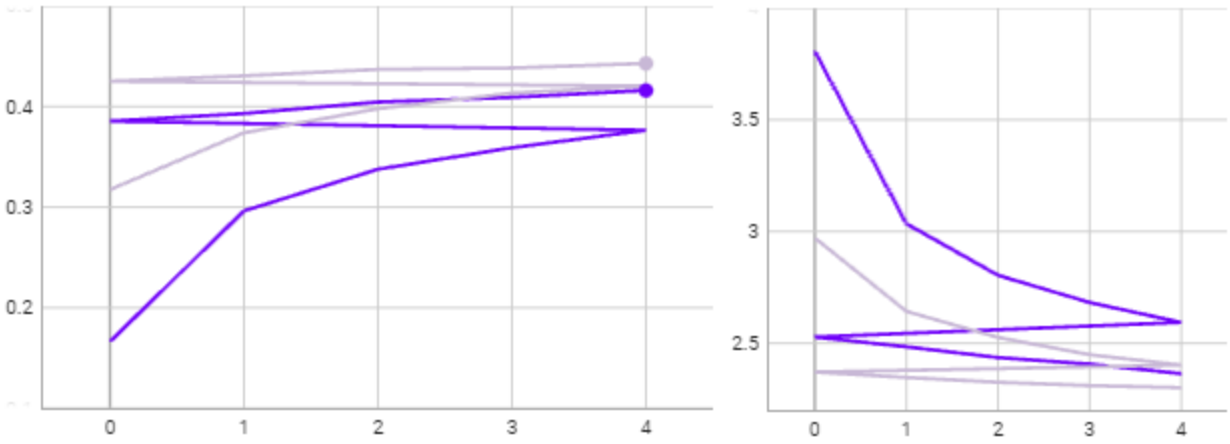

Figure 4: The transfer learning model with MobileNetV2 embeddings. Accuracy on the left and loss on the right by epoch. The graphs show two optimization steps with the same model; 1) transferring the frozen layers and training an extra top layer and 2) fine-tuning the whole model. Training set is purple and the validation set is light purple.

Table 1: Accuracy and loss on the test set

	Accuracy	Loss
Shallow CNN	0.9%	6.5
Deep CNN	28.0%	2.9
MobileNetV2 transfer	**45.6%**	**2.24**

Discussion

Which is the best model

The best model is the MobileNetV2 transferred model (see Table 1) and with more improvements, its accuracy could be advanced even further. There was no sign of plateau in learning but the models were limited artificially in training time for compute reasons. It is interesting that Figure 4 shows a higher accuracy for the validation set than the training set but it might just be an artifact of the random initialization and thereby the difficulty of the images.

Transfer learning & fine-tuning

Transfer learning provides a unique opportunity to discount the amount of compute and climate impact from the models we train by using existing weights to avoid excessive training steps. These foundation models [12] provide us with strong baseline accuracies for many different tasks in e.g. language and vision. With good fine-tuning methodologies, the models can end up much better than their baseline trained contemporaries.

Limitations

Because of the GPU limitations, the epochs were not optimized and there is a chance that the models could achieve even better performance both because of learning more continually but also because of the possibility of grokking [13]. Beyond this limitation, the quality of the dataset itself has earlier been put into question [14] and because we use the 64x64 images, it can be put into question how well they represent the class they are relegated to.

Reproduce this work

Conclusion

The work is uploaded to Github at esbenkc/tinyimage-classification and the process to run it is simple. Clone the project and run `python main.py` (optionally run `pip install -r ./requirements.txt` but this will install a lot of packages). If you wish to see it in Tensorboard, run the command `tensorboard --logdir logs/fit` which will also show the runs from this report. This will reproduce the best result reported in this report, i.e. the MobileNetV2 transfer learning and fine-tuned model. This will generate the file `submission.csv`, `cnn_mobilenet_model.h5`, and several plots.

This report showcases and represents reproducible code to classify the 100 classes in the TinyImage dataset, a subset of the ImageNet dataset. Three models are trained and evaluated, with a transfer learned model from the MobileNetV2 pretrained model providing the highest accuracy on the validation and test set.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, 'ImageNet: A large-scale hierarchical image database', in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848.

[2] J. Dean, 'The Deep Learning Revolution and Its Implications for Computer Architecture and Chip Design', arXiv, arXiv:1911.05289, Nov. 2019. doi: 10.48550/arXiv.1911.05289.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, 'ImageNet Classification with Deep Convolutional Neural Networks', in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. Accessed: Jul. 09, 2020. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[4] S. Chetlur *et al.*, 'cuDNN: Efficient Primitives for Deep Learning', arXiv, arXiv:1410.0759, Dec. 2014. doi: 10.48550/arXiv.1410.0759.

[5] T. Developers, *TensorFlow*. Zenodo, 2022. doi: 10.5281/zenodo.6574269.

[6] K. Fukushima, 'Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position', *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, Apr. 1980, doi: 10.1007/BF00344251.

[7] Y. Xu and M. Vaziri-Pashkam, 'Limits to visual representational correspondence between convolutional neural networks and the human brain', *Nat. Commun.*, vol. 12, no. 1, Art. no. 1,

Apr. 2021, doi: 10.1038/s41467-021-22244-7.
[8] S. Ioffe and C. Szegedy, 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', arXiv, arXiv:1502.03167, Mar. 2015. doi: 10.48550/arXiv.1502.03167.
[9] 'The perceptron: A probabilistic model for information storage and organization in the brain. - PsycNET'. https://doi.apa.org/doiLanding?doi=10.1037%2Fh0042519 (accessed Jun. 02, 2022).
[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, 'MobileNetV2: Inverted Residuals and Linear Bottlenecks', 2018, pp. 4510–4520. Accessed: Jun. 02, 2022. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html
[11] M. Akay *et al.*, 'Deep Learning Classification of Systemic Sclerosis Skin Using the MobileNetV2 Model', *IEEE Open J. Eng. Med. Biol.*, vol. PP, pp. 1–1, Mar. 2021, doi: 10.1109/OJEMB.2021.3066097.
[12] R. Bommasani *et al.*, 'On the Opportunities and Risks of Foundation Models', *ArXiv210807258 Cs*, Aug. 2021, Accessed: Sep. 01, 2021. [Online]. Available: http://arxiv.org/abs/2108.07258
[13] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra, 'Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets', arXiv, arXiv:2201.02177, Jan. 2022. doi: 10.48550/arXiv.2201.02177.
[14] C. G. Northcutt, A. Athalye, and J. Mueller, 'Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks', arXiv, arXiv:2103.14749, Nov. 2021. doi: 10.48550/arXiv.2103.14749.