Yearbook dating and Geolocation prediction with CNNs

Chandana Amanchi
University of Texas at Austin
chandana@utexas.edu

Pandian Raju
University of Texas at Austin
pandian@cs.utexas.edu

Abstract

Deep networks are widely used for various machine learning and artificial intelligence tasks. Particularly in the field of computer vision, deep learning is on the rise with the usage of Convolutional Neural Networks (CNNs). CNNs are widely adopted for various image recognition tasks like image classification, object detection and semantic segmentation. There have been various architectures proposed which try to excel in these tasks, competing with each other to get the best accuracy. In this work, we try to implement some of the state-of-the-art deep networks (using CNNs) to perform image label prediction on two datasets: yearbook dating [8] and Geolocation prediction [6]. We evaluate the datasets using different architectures and different methods and we compare and analyze the results observed in different cases. We also predict the labels using an ensemble of multiple models [13] and show that we achieve the best accuracy (L1 distance) of 3.94 years for yearbook dating and 265km for geolocation prediction, compared to the individual models. Code: https://github.com/chandana44/yearbook-classification/

1. Introduction

Deep learning [17] has become a buzz word in the field of machine learning today. It has become the state-of-the-art method to do a lot of machine learning tasks and there haven been advnacements made to the field of deep learning continuously. Convolutional Neural Networks (CNNs) are a type of neural networks which contain convolutional layers which are best suited for pattern recognition in images and for visual perception. CNNs are widely used today as the default network for any image recognition task. Going back in history, CNNs were first introduced by K. Fukushima in 1980 in his work on Neocognitron [7]. It introduced a self organizing neural network model for pattern recognition.

Although CNNs were introduced in neocognitron [7], it lacked a proper framework for supervised learning which was later provided by Yann LeCun in his work on document recognition [18]. With a proper framework for supervised learning using backpropagation, CNNs started becoming the method for various image recognition tasks. Later with new advancements in the field like dropout [22], batch-normalization [12] and increased access to more GPUs, CNNs saw a break-through in the field of image recognition. The CNN network provided by A. Krizhevsky in [16] included these changes and became the game changer in the field of visual recognition. The Imagenet dataset [5] and the imagenet challenge [20] became standard measures of comparing the performance of deep networks for image recognition tasks.

Later on the line, VGG networks [21] showed how only a small convolution filter (3x3) can be used to train the CNN. Two standard networks VGG16 and VGG19 were developed (with 16 and 19 layers respectively) in an effort to go deeper in the network. ResNet [10] actually took a big leap in going very deep in the network which introduced the concept of adding the residual input from skip level connections. By the virtue of being very deep and having the residual inputs from the skip levels, ResNet achieved the best accuracy on many image recognition challenges. Recently, DenseNet [11] was proposed which concatenated inputs from every previous layer. DenseNet was also able to go very deep in the network with the insight that later layers can retain the features from earlier layers by concatenation.

Given that we have many different deep architectures trying to advance the field in different ways, there is no single architecture which is best suited for all tasks. Choosing a given architecture depends on the type of problem, nature of the dataset and desired trade-offs (like memory versus time versus accuracy, for example). In this work, we try to solve two different problems using some of these deep networks - yearbook dating and geolocation prediction. We evaluate different architectures for the tasks, tune the hyper-parameters for each architecture and try to compete one architecture against the others. We also compare and analyze the results and discuss the advantages or shortcomings of the different methods.

In addition to just evaluating the networks individually, we also ensemble the networks [13]. We see that the ensembled model achieves the highest accuracy compared to any individual model for both the tasks.

2. Datasets

2.1. Yearbook dating

The yearbook dataset[8] contains frontal-facing American high school year-book photos with labels to indicate the years those photos were taken. The dataset contains a total of 37,921 gray-scaled images of both male and female students together. The images are collected from 814 scanned yearbooks from American high schools ranging from 1905-2013 across 115 schools in 26 states. The dataset provided for this project contains 37849 of these images. It is further divided into three sub-sets for training, validating and testing the Convolutional Neural Networks. The training, validation and test datasets contain 22840, 5009 and 10000 images respectively.

2.2. Geolocation

The Geolocation dataset[6] contains images from France street view with geolocation labels (latitude and longitude). The images are collected by scraping a dense sampling of panoramas of all the areas of Paris from Google street view. The given dataset contains 195010 images in total. The dataset is divided into training, validation and testing datasets with 175010, 10000, 10000 images in them respectively. The longitude of these images ranges between -0.000331 - 9.540445 and the latitude ranges between 41.390225 - 51.101005.

3. Architectures

We use the following state of the art architectures for the prediction tasks.

3.1. AlexNet

- We use the AlexNet architecture introduced in [16] for doing the classification of images for both the tasks.
- The network consists of five convolutional layers and three fully connected layers with a layer of softmax at the end.
- For the two different tasks, we set the number of classes correspondingly (118 for yearbook and 100 for geolocation) instead of the default 1000 classes in AlexNet.

3.2. VGG

- We use the VGG architecture introduced in [21] for the classification task. We try both VGG16 and VGG19 with 16 and 19 layers respectively.
- VGG16 and VGG19 consitute 5 convolutional blocks and a fully connected block at the end with different number of convolutional layers within each convolutional block.

3.3. ResNet

- For the yearbook prediction task, we used ResNet-50 and ResNet-152 introduced in [10].
- For the geolocation prediction, we used ResNet-50 in the classification approach.
- The number of classes are set to 118 for yearbook Resnets and 100 for geolocation.

3.4. DenseNet

- We use the DenseNet architecture introduced in [11] for the classification task. We try prediction using DenseNet-169 and DenseNet-121 architectures.
- We use 4 dense blocks and a growth rate of 32 for both the variants of DenseNet.

4. Implementation

4.1. Framework

We use Keras [3] framework in python to develop the deep networks using Theano [23] as the backend. Keras is a high-level neural networks API, which serves as a wrapper on top of low level neural networks libraries like TensorFlow, Theano or CNTK. It enables ease of use and fast experimentation. It also provides various state-of-the-art networks like VGG16, VGG19, DenseNet, ResNet etc as inbuilt applications. We ran all our experiments on NVIDIA Tesla K40 GPU on TACC [19] Maverick cluster.

4.2. Supervised pre-training

One of the important problems for any image classification task is the lack of abundant training data. For example, in the yearbook dataset, the training dataset consists of around 23000 images while the test dataset consists of around 10000 images which is roughly 43% of the training set. Training a deep network just on this training data might not perform well due to the lack of training. There have been techniques to tackle this problem, like augmenting the data and producing more training data, which have seemed to help to some extent.

One of the effective solutions to counter the problem of lack of training data is to do transfer learning or supervised pre-training. Using this method, the deep networks are loaded with pre-trained weights on standard dataset like the ImageNet [5] and CIFAR-10 [15] datasets instead of random initial weights. The network is then fine-tuned by using the training data. Such transfer learning has proved effective for CNNs since the first few convolutional layers in a network learn very generic features in an image like curves, edges, color gradients etc which are independent of the context of the images.

Region-based CNN (R-CNN) [9] uses such supervised pre-training to do object detection and semantic segmentation using CNNs. We do a similar pre-training. For all of our architectures, we initialize the networks with weights pre-trained on ImageNet. This is followed by fine-tuning the network using the training data of the dataset. This is done for both yearbook dating and geolocation prediction problems. Resources like [24, 25, 27] were helpful in understanding and implementing the fine-tuning.

Loading pre-trained weights of ImageNet means that the final softmax layer has default 1000 classes per the original ImageNet dataset. In order to adapt this to our specific problem, for sequential models, we pop the final classification layer and add our own softmax layer with required number of classes. For non-sequential models like ResNet and DenseNet, we add our own softmax layer to the output of the intermediate layer in the network (the layer before the fully connected layer) and train the network.

Another common practice followed during fine-tuning is to freeze the initial few layers of the network after loading the network with pre-trained weights. This is because the initial layers learn context-independent features of the images and we do not want to distort those weights by the training data in hand. In our experiments, we freezed almost 50-75% of the convolutional layers present in the network and train only the remaining layers.

4.3. Image preprocessing

- Since we are using pre-trained weights from the ImageNet architectures, the RGB mean of the ImageNet images is subtracted from all the images in the dataset.
- Different architectures used in the work need different dimensions for the input images. AlexNet needs 227x227 images, VGG and ResNets need 224x224 images. Hence we preprocessed the images by cropping them to the appropriate size based on the architecture being used.
- We did *not* use the data augmentation technique in the pre-processing since the images in the yearbook dataset are structured and augmenting the training data will not help in improving the accuracy.

4.4. Optimizer and loss

- We tried different optimizers and loss functions for the different architectures and also corresponding learning rates and other hyper-parameters.
- We tried SGD optimizer [18] and Adam [14] optimizer with corresponding learning rates of 1e-3 and 1e-4. We used smaller learning rates since we loaded pre-trained weights.
- We tried mean_squared_error and categorical_crossentropy losses of which, the latter seemed to work better. We also tried implementing our own loss function, which is the L1 distance between the prediction and ground truth, but it did not seem to work well as well since the L1 distance will be integers and the learning is not smooth in that case.

4.5. Ensembling models

Ensembling is the technique of combining multiple models for predicting the output. The models are typically combined by taking average or weighted average. Ensembling multiple models reduces the variance and avoids overfitting. The ensembled models typically perform better than the individual models. We ensembled different models we implemented for both the problems and got the best results.

- For the yearbook problem, we tried ensembling with all different combination of AlexNet, VGG16, VGG19, ResNet50, ResNet152, DenseNet121 and DenseNet169. For ensembling these models, we considered the following three approaches and evaluated all of them.
  - **Mean** The mean of the years predicted from all the models is calculated and rounded off.
  - **Median** The median from all the models years predictions is calculated and rounded off.
  - **Closest to Mean** The prediction closest to the mean of all models predictions is chosen as the result.
- For the geolocation task, we ensembled the regression architecture and the ResNet50 architecture. The following approaches are used for ensembling the models:
  - **Mean** The mean of longitude and latitudes predicted from the models
  - **Median** The median of longitude and latitudes predicted from the models

4.6. Regression and classification for Geolocation

Unlike the yearbook dating problem, the solution for geolocation problem is not very apparent or straightforward. We try to solve the geolocation prediction using both regession and classification models and compare the results obtained.

4.7. Geolocation as regression

Since the longitude and latitude are continuous values, we initially tried to model the geolocation prediction as a regression problem. We implemented a customized Convolutional Neural network with 4 convolutional layers and two fully connected layers. The first five layers are created with *relu* activation. Dropout and Batch normalization techniques are also used to avoid overfitting and to improve the accuracy. The output fully connected layer is two dimensional for predicting the longitude and latitude. The model is compiled with *mean squared error* loss and *RMSprop* optimizer.

4.8. Geolocation as classification

Although one might intuitively think of predicting the geolocation as a regression problem, as long as the number of classes is not infinite, one could think of it as a classificaiton problem. For this specific dataset, converting the latitudes and longitudes into XY coordinate system revealed that the difference between smallest and largest x-coordinate and the difference between smallest and largest y-coordinate was around 1.5e+6 m which is roughly 1500 km. Hence we divide the entire space in the training set as a grid of squares with length 10 units and height 10 units (100 squares totally). Now, given a latitude and longitude, the corresponding xy coordinate can be found and so the position in the grid which will be used as the classification label.

Using the above scheme, we train AlexNet and ResNet-50 by using the pre-trained weights on ImageNet dataset and fine-tuning on the geolocation training set. The points are converted to xy coordinates and the labels in the grid are calculated and passed as the output predictions for the model. Similarly during prediction, the corresponding box in the grid is found by using the label and the center point of the square is converted to latitude and longitude.

5. Evaluation

5.1. Yearbook dating

We measured the validation set L1 distance in years for each of the architectures individually and for the ensembled model for the evaluation. The L1 distance for each prediction is calculated by taking the absolute difference between the prediction and the ground truth. To calculate the L1 distance over the entire validation set, we sum up the L1 distances of all predictions and divide it by the number of images in the validation set. Analysis of L1 distance Vs model is presented in figure 1.

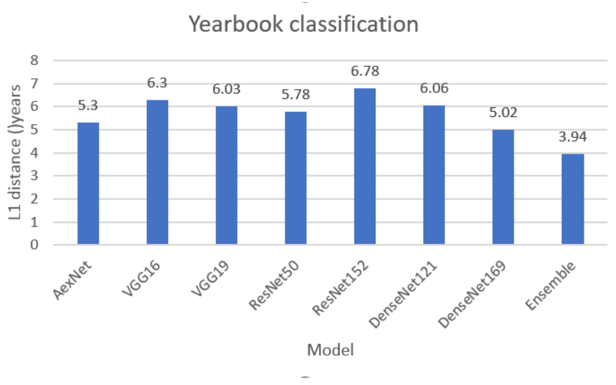

Figure 1. Yearbook L1 distance with different models

5.1.1 Best by each model

- **Alexnet:** In AlexNet, we freezed the convolutional layers after loading the pre-trained weights and trained only the fully connected layers. Since the parameters to be trained are only from the three fully connected layers, the training was quite fast. We could run 100 epochs in 11 hours and achieved an L1 distance of 5.3 years.

- **VGG16 and VGG19:** Similar to AlexNet, we freezed the initial convolutional layers in VGG16 and VGG19 to make the training faster and to not distort the pre-trained weights. We were able to achieve a mean L1 distance of 6.3 for VGG16 and 6.03 for VGG19. VGG19 was slightly better than VGG16 because it was little more deeper and hence able to track the features better.

- **ResNet-50:** Following an approach similar to AlexNet, we freezed all the convolutional layers of ResNet50 and trained only the fully connected layers. However, this did not result in a great L1 distance(8.36 in 15 epochs). Then we unfreezed some of the convolutional layers of the network allowing it to learn yearbook dataset specific features. The ResNet50 model has 177 layers in total in the implementation including all the conv, pooling, batchnormalization, dense, activation etc layers. Freezing only the first 100 layers gave an L1 distance of 6.17 in 10 epochs. Then we further tried freezing only the first 50 layers and were able to get an L1 distance of 5.78 in 6 epochs. We also experimented with the learning rate and got best performance at 0.01.

- **ResNet-152:** Since deeper ResNet architectures result in better accuracies, we also adopted the ResNet152 architecture for the yearbook problem. Similar to ResNet50 architecture, unfreezing the convolutional layers improved the accuracy of this model. The

ResNet152 model has a total of 722 layers in it. We freezed the first 500 layers and trained the remaining layers. Due to its deeper architecture, resNet152 was highly memory consuming. We had to reduce the batch size to a lower value(32) compared to the previous architectures(128) to run the model. Each training and validation epoch took around 10 hours. We had the chance to run the network for two epochs and achieved an L1 of 6.78. Training for further epochs should result in much better values.

- **DenseNet-169 and DenseNet-121:** We tried DenseNet-169 and DenseNet-121 models by loading pretrained weights from ImageNet. Since DenseNet uses a lot of memory (due to the concatenation of features), the maximum batch size it could be run it with was 16. And because of the extremely small batch size, the training was also very slow, taking around 9-10 hours to complete 1 epoch. Although it took a lot of time for training, DenseNet proved to be best architecture to provide least average L1 distance on validation set by itself. We were able to achieve a L1 distance of 5.92 in DenseNet-121 and 5.02 in DenseNet-169. We believe if we could train DenseNet-121 for more epochs, the L1 distance will decrease further.

- **Ensembled Model:** We took all the 7 models (*AlexNet, VGG16, VGG19, ResNet-50, ResNet152, DenseNet121, DenseNet-169*) predictions and did an exhaustive ensembling of all the combinations by following the approaches discussed in section 4.5. We got much better results with the ensembled models compared to any individual model. The mean, median and closest to mean approaches of the best ensembling(AlexNet, VGG19, ResNet50, DenseNet169 - the top four performing models) resulted in **4.13, 3.94, 4.15** years of L1 distance.

5.1.2 Training and Validation loss with epochs

Figures 2 and 3 show the training loss and validation loss (in terms of mean L1 distance for the validation set) with each epoch for different architectures. We can clearly see that the validation loss starts with a higher value (higher average L1 distance initially) but gradually comes down with increase in epochs. We also note that for some architectures like VGG16 and VGG19, the L1 distance clearly starts to increase beyond a point due to overfitting.

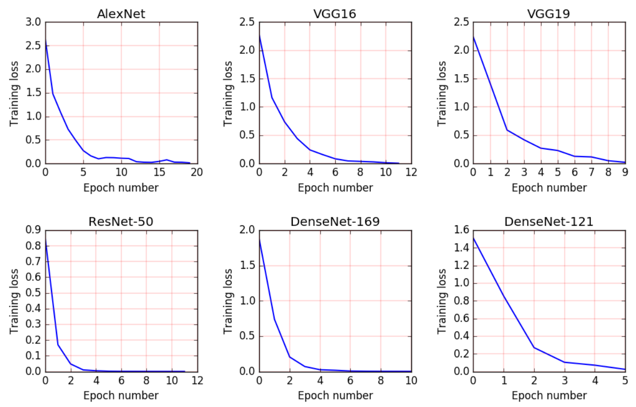

Figure 2. Yearbook training loss (categorical_crossentropy) with epochs

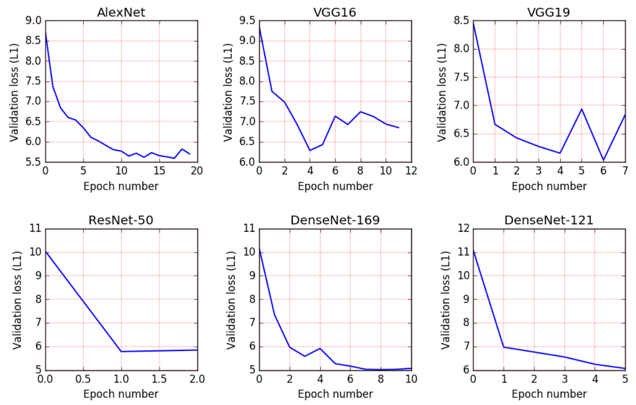

Figure 3. Yearbook Validation loss (L1 distance) with epochs

5.1.3 Training time

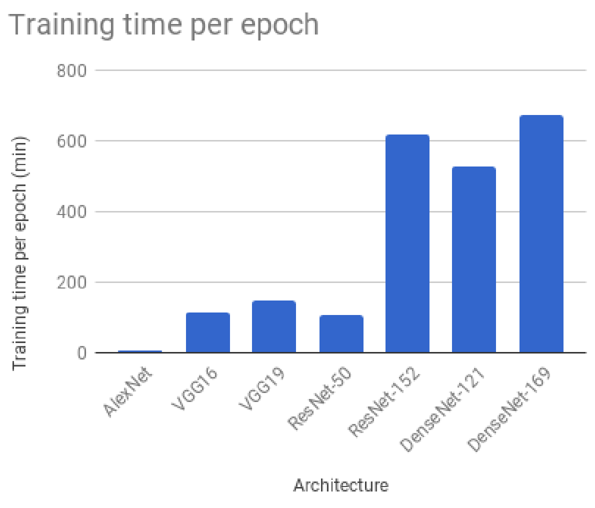

Figure 4. Training time per epoch for different architectures

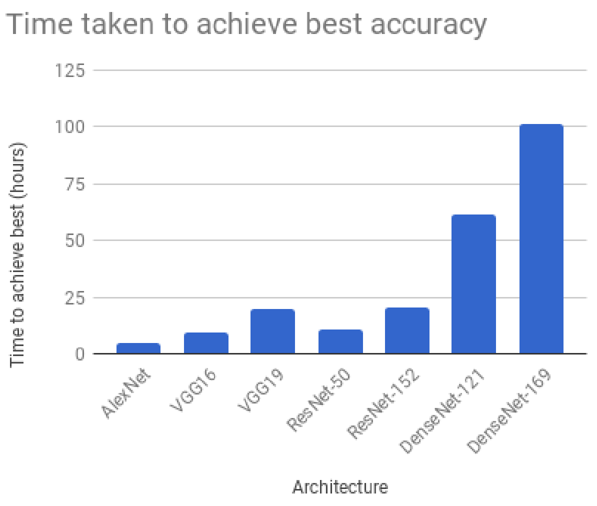

Figure 5. Training time to achieve best accuracy

Figure 4 and 5 show the time taken by the different architectures to complete the training and validation. Figure 4 shows the number of minutes to complete one epoch of training and one set of validation over the validation set. Figure 5 shows the number of hours taken by each architecture to achieve the best accuracy it has. Although it might not be fair to compare the time-taken head-to-head against each other since different architectures have different best accuracies, it gives an overall sense of how slow the deep networks can become.

We see that DenseNet-169 and ResNet-152 take the most amount of time among all the architectures since they are very deep and factors like memory usage limits the batch size for DenseNet. Comparing the validation loss with the number of epochs, we also find that these architectures quickly come down to a reasonable accuracy, but it takes many more epochs to achieve even better accuracies, since the learning of features becomes slow beyond a point of training.

5.2. Geolocation prediction

For evaluating the geolocation prediction, we calculated the l1 distance over the validation set for all different models. L1 distance for each prediction is calculated by finding the distance between the predicted and ground truth geolocations in kms. The overall l1 for the validation set is calculated by taking the avergae of l1 distances of all the predictions. The results of all the models are presented in figure 6.

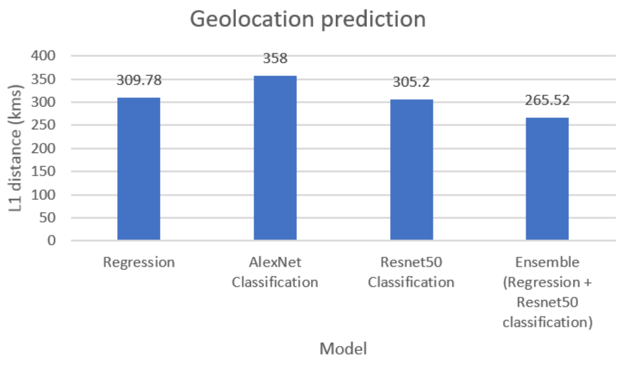

Figure 6. Geolocation L1 distance with different models

- **Regression Model:** We trained the custom Regression CNN model for the geolocation prediction without loading any pre-trained weights. Since it has only four convolutional and two fully connected layers, the training was fast. With a learning rate of 10^{-3} we could get an L1 of 345.45 in 15 epochs. Then we tried higher and lower learning rates and achieved an L1 of 309 years in 19 epochs with a learning rate of 10^{-2}.

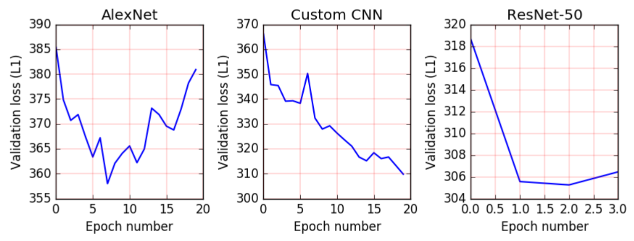

Figure 7. Geolocation Validation loss (L1 distance) with epochs

- **AlexNet Classification Model:** As discussed in Section 4.8, we tried solving the problem of geolocation prediction as a classification problem. We used AlexNet by loading pretrained weights on ImageNet and replacing the final softmax layer with our own softmax layer of 100 classification labels. In addition to that, we also freezed the initial convolutional layers of the network and we trained the model. Figure 7 shows the validation loss for all the models we used. We can see that AlexNet quickly started overfitting and the best L1 it could achieve was around 358 km.

- **ResNet-50 Classification Model:** Similar to AlexNet, we train ResNet-50 model provided by Keras to do the classification with 100 labels. We freeze the initial convolutional layers and train the network for few epochs. The L1 distance quickly came down but reached a saturation point quickly. The best we could achieve was around 305 km with ResNet-50 classification model. We believe doing data augmentation could have helped increasing the accuracy to some extent in this case (although not much in the case of yearbook dating due to the type of dataset).

- **Ensembled Model:** We ensembled the top two performing models - *Regression Model, ResNet50 Classification Model* using the mean and median approach discussed in section 4.5. The ensembling resulted in the lowest L1 of **265.52**. This is much lesser than the L1's of both the models used for ensembling.

6. Conclusion

In this work, we used different state of the art Image classification networks and their ensembled models on yearbook dating and geolocation prediction tasks. By comparing the results from all the models, we conclude that the best accuracies are achieved through the ensembling models in both the tasks- 3.94 years for yearbook dating and 265 kms for geolocation prediction. In future, we would like to combine more models in ensembling - ResNet152 & DenseNet121 for yearbook label prediction and VGG networks for geolocation prediction and study the results. We would also like to look at ways to speed up the networks by exploring GPU parallelization.

Acknowledgement

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC, GPGPU resources that have contributed to the research results reported in this paper. URL: http://www.tacc.utexas.edu

References

[1] S. Chaudhary. Cnn-keras. `https://github.com/heuritech/convnets-keras`, 2016.

[2] F. Chollet. Building powerful image classification models using very little data. `https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html`, 2016.

[3] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[4] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. Advances in neural information processing systems 2. chapter Handwritten Digit Recognition with a Back-propagation Network, pages 396–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[5] 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.

[6] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros. What makes paris look like paris? *ACM Transactions on Graphics (SIGGRAPH)*, 31(4):101:1–101:9, 2012.

[7] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.

[8] S. Ginosar, K. Rakelly, S. M. Sachs, B. Yin, C. Lee, P. Krhenbhl, and A. A. Efros. A century of portraits: A visual historical record of american high school yearbooks. *IEEE Transactions on Computational Imaging*, 3(3):421–431, Sept 2017.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[11] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[13] C. Ju, A. Bibaut, and M. J. van der Laan. The Relative Performance of Ensemble Methods with Deep Convolutional Neural Networks for Image Classification. *ArXiv e-prints*, Apr. 2017.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[15] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[17] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[19] T. U. of Texas at Austin. Texas advanced computing center. `http://www.tacc.utexas.edu`.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[23] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[24] F. Yu. A comprehensive guide to fine-tuning deep learning models in keras (part i). `https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html`, 2016.

[25] F. Yu. A comprehensive guide to fine-tuning deep learning models in keras (part ii). `https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part2.html`, 2016.

[26] F. Yu. Kaggle state farm blog. `https://flyyufelix.github.io/2016/10/11/kaggle-statefarm.html`, 2016.

[27] F. Yu. Cnn fine-tuning. `https://github.com/flyyufelix/cnn_finetune`, 2017.