COMP_SCI_7318 - Assignment 2 - CNNs for Images Classification

Hong Phuc Pham - Student ID: a1843625
The University of Adelaide
230 North Tce Australia SA 5005
hongphuc.pham@student.adelaide.edu.au

Abstract

Convolutional neural network (CNN) is a network architecture for deep learning algorithm [11], which is used explicitly for imaged recognition and processing of pixels data. In this assignment, CNNs will be used for image classification. The dataset will be worked on sign language MNIST dataset [18] to distinguish the different signs according to the alphabet. The CNN will be implemented and tested, and some tricks on computer vision will be used to enhance the accuracy. Activities are conducted to understand the mechanism of CNNs. This algorithm will be implemented with Python, using the Keras library to build the network. Some tests will be tried to seek a higher accuracy for the algorithm on the chosen dataset. The main task is the classification of different sign languages for 24 letters. Accuracy reaches about 82% for a simple fist convolution network model. With some tricks, layers adding, and layers parameters adjusting, the accuracy on testing leaps to 96%. Then, the model is applied with augmentation, and the accuracy of the test has reached absolute value.

1. Introduction

In the early stage, the precursor of CNNs or it could be told the first architecture, which required multiple network locations, must have located units - Neocogitron was introduced. In the same paper of Neocogitron [6], the concept of pooling layer, feature extraction, convolution in the neural network, recognition or classification was also proposed. The initial idea was inspired by Hubel and Wiesel work on the concept of neuron connectivity patterns forming the animal visual cortex organisation. Later in 1987, the convolutional neural networks were presented at Neural Information Processing Workshop. The data set was developed as a collection of images of sign language. The Sign Language MNIST dataset only has 24 letters sign where the letter J and Z are not counted as they are motion gestures. The data set is already split for training and testing. Images are the same size, cropped hand gestures. This report will go through the dataset overview, experiment with adding some layers, and test on different learning rates, optimiser, and batch sizes to improve the final testing performance.

2. Dataset

The original MNIST is famous for the handwritten digit image dataset. Each image represents in the format of 28x 28-pixel width and height, which is 784 pixels per sample (Figure 1). The image is stored in CSV format with labels and pixel values. This sign language image dataset is based on the American Sign Language (ASL) system developed by National Institute on Deafness and Other Communications Disorders. The deaf majorly uses this ASL system in North America, Spanish, Italian, Germany and French. Images are in grayscale with values between 0 and 255. There are 27,455 samples for training and 7172 samples for testing. Each sample recorded the gestures of different signs around the hand region of interest. Dataset only captures hand in grey, resizing with at least 50 variations. Each class has around 1000 samples (see Figure 2 for the label distribution). These modifications included 5% random pixelation, around 15% difference in contrast or brightness, along 3-degree rotation. Labels are tagged for each letter from 0 to 25. However, only 24 labels are present where the number 9 is J and 25 is the letter Z which needs the motion to describe them. The dataset can be found on Kaggle.

3. Method

The dataset is already separated into training and set. The movement set the go through the prepossessing to checking the error. One outset label was found with '200'; it has been quickly amended to the correct label instead of deleting it. The image data is hard to collect, and they have the value to augment more training data, so it was kept. The training dataset will be split again for training and validation at the rate of 8:2. Therefore, the number of samples for training, validation and testing will be 21964, 5491, and 7172.

For the primary setting, SGD optimiser [12] will be

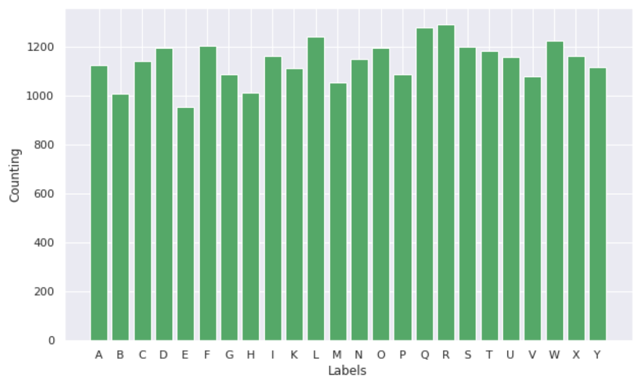

Figure 1. Class distribution of training dataset

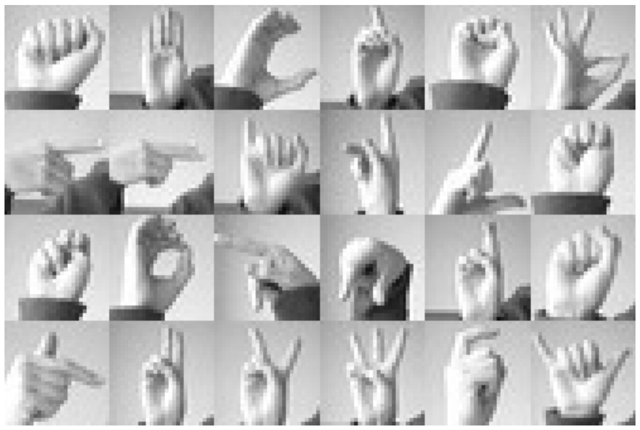

Figure 2. Data samples

used, the loss will be applied with sparse categorical cross-entropy, learning rate at 0.01, and the batch size is 32 and 15 epochs. The initial model only has two total connected layers and one output layer. The first layer has 300 nodes with ReLU activation [2], and the next one has 100 nodes also with ReLU activation. The output layer will be fixed with 25 nodes, and softmax activation [7]. The number of nodes can be set to 24 but not 23. To keep the dataset easy to manipulate and with less modification on the data label, the ordering number from 0 to 25, mapping A to Z, will be kept. If 23 nodes using for output, then the label or checking dictionary would be needed to support the relabeling. 25 or 24 output node works as the last letter Z is never in the dataset and at the end of the list; therefore, it does not affect the training or order label.

In the following three tests, 2D convolution layers [7], and max pooling [7] were gradually added to the primary model. Various values are set for the filter, kernel size, and stride [7]. The same padding, ReLU activation, is set for every hidden layer. Many filters or kernels are chosen in these values: 16, 32, 512, and 1024. Kernel size will be set in these values of 3, 5, and 7. Strides and pool size will vary between 1 and 2. A batch normalisation layer will be added in the coming text, and layer parameter adjustments will be made to earn the most accuracy.

There are two approaches for the tuning: manual changing the hyper-parameter and using the library function to test and apply the change on learning rate, batch size, loss function, and optimiser. Those values will be used as the indices for accuracy and loss comparison. The chosen architecture is illustrated in Figure 3 below.

4. Experimental Analysis

These experiments undergo several trials of model construction, tuning, and applying tricks. These tests were conducted to observe the effect of the different layers on the network. Also, to get the best of the model, tuning is used to find the best hyper-parameters for the model—some attempts with dataset tricks to levitate the performance. The experiments observed form a simple full connected layers model and then added some convolutional network layers. Next, the batch normalisation layer will be added. The tuning in a manual way and the library function will be applied later. Lastly, augmentation will also be used.

4.1. Model construction

The CNN architecture has these main components: convolution layers, pooling layers, and fully connected layer activation functions. Simple CNN structure is illustrated in Figure 4. For the construction work, layers will slowly be added or adjusted to the main model. There are three main observations in this section. The first one will test the model's performance with only top connected layers, the model with convolution layers added, and then the batch normalisation added.

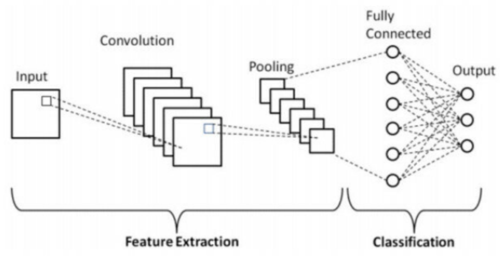

Figure 4. Simple CNN [5]

4.1.1 Full connected layers

These total connected layers are where the linear transformation of the input vector through a weight matrix is. These layers evaluate the probability relationship of the class and position of features on images (or the features) [18]. The first experiment ran swiftly with 69.44% accuracy on the test when the accuracy on the test and validation was over

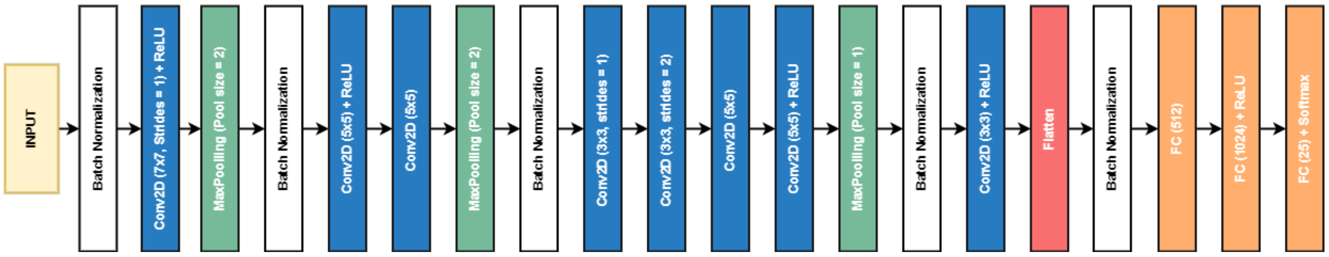

Figure 3. Chosen model architecture

90%. This current design mostly struggles with recognising the class U. Only four classes got over 90% accuracy, and more than half the class got below 50%. With this simple design, the model has learnt some features but not most of the dataset (see Figure 5 for the training performance). The accuracy between testing and training has a big gap, which indicates the over-fitting of the training. It could be because of a characteristic of total connected layers where it learns every detail with every node connected with every previous layer's node and the next layer's node. Consequently, the model learnt but lost the elasticity to identify other cases.

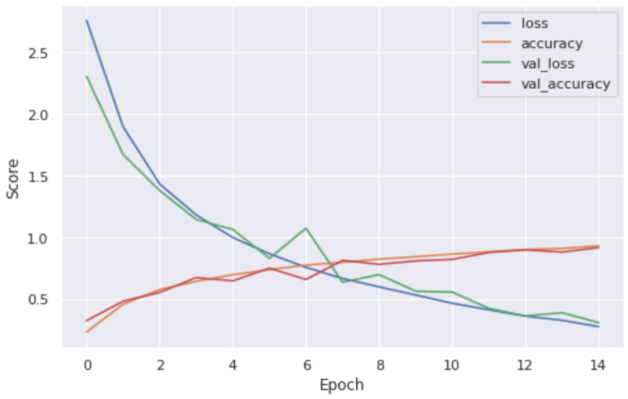

Figure 5. Loss, accuracy of training and validation on the full connected layer only model

4.1.2 Convolution layers added

The first test adds a convolutional layer and max pool into the model. Currently, the number of kernels on each layer is still set as either 32 or 16. The runtime of each epoch slightly per second increased. Training accuracy reaches a maximum, while the validation set is close to that value at 99%. Performance on the test set is not much different, with 82.48% which is a great leap from the initial complete connected layers design. Only one class got an accuracy below 50% . Class N and K are two classes that the model got struggles with.

In the next experiment, some layers' kernels were raised to 512 or 1024. One full connected layer is set with 512 nodes. The average run time for each epoch six times surged. Both training and validation accuracy reaches maximum. The testing accuracy increases slightly to 86.09%. No class accuracy below 50%. Ten classes have an accuracy below 80%. This model still struggles with the K class. The number of filters helps the model recognise and distinguish different signs. That idea enabled other convolutional layers to have 512 or 1024 filters. Also, more layers were added to the model. The run time for each epoch increases dramatically with the higher filter setup. Some changes in kernel size on each layer also changed on the last test. They will be set within the values of odd numbers 3, 5, and 7. The accuracy was similar. The highest accuracy reached this stage is 86.29%.

Although having approximately the exact mechanism, what the convolutional layer brings to the model is different. Similar to the figure 4 showing, the top connected layers work on the classification while the convolutional layer extracts the features which support the model to learn and identify the class's traits and patterns. Without the convolutional layer, the model greedy learns every detail with the global likelihood of signs. That's why with the application of the convolution layers with different combinations of kernel parameters, the model reaches an acceptable result of over 80% on the testing.

4.1.3 Batch normalisation added

Even changing the layer parameters or adding or removing the layer, the model accuracy is still in the range of 86. It could be said that the model reached the bottleneck again. To come over this, another trick is applied. There are three tested model structures added with batch normalisation. These layers are mainly added after every max pooling layer and flattening layer. The last model adds one after the input. One more time, the accuracy of the testing leapt. The model gained 95.74% accuracy as the best result among the listed experience. Only the T class has an accuracy below 80%, but the performance of this class detection is too low at 79.83%. It mostly got miss classified with class X. The accuracy improved, but the training time improved, with about a third run time less than the implementation without batch normalisation layers.

Batch normalisation is widely used not only in computer vision [9,16] but also in the natural language processing [3]. This technique can be implemented on most network types like multilayer perceptrons, CNNs and recurrent neural networks. Training a deep neural network is challenging. The network is quite sensitive to the initial random weight and the learning algorithm configuration. The input distribution to layers changes after each mini-batch when weight is updated, which makes the learning algorithm might chase the moving target whose term is 'internal covariate shift' [4]. As its name suggests, this technique standardises the input to the network, applied to either the activation of the previous layer or inputs directly. It helps stabilise the learning process and significantly reduces the number of training epochs. This explains why the run time of each epoch was reduced.

4.2. Hyper-parameters tuning

On this part, FLOPs checking will be applied. FLOPs is the unit for the number of floating point operation measuring performed by a computer in a second [1]. The calculation for FLOPs is provided as `FLOPs_couter.py` file. The printed result will be the sum FLOPs of computing from convolution layers, linear layers, batch normalisation, ReLU activation function and pooling. In theory, FLOPs will be calculated by following rules:

Convolutions - FLOPs = 2x Number of Kernel x Kernel Shape x Output Shape

Fully Connected Layers - FLOPs = 2x Input Size x Output Size

Pooling Layers - FLOPs = Height x Depth x Width of an image

With a stride, FLOPs = (Height / Stride) x Depth x (Width / Stride) of an image

4.2.1 Manual tuning

On the manual experience, the model will be tested through different learning rates first. Only the spare categorical cross entropy loss function and SGD optimise function will be used on these manual tests. The best performance one will be chosen, and its learning rate will be used to test batch size variants. Five learning rate values will be used are $1e-1, 1e-2, 1e-3, 1e-4, 1e-5$. For the batch size, $16, 32, 64, and 128$ are the values that will run the tests. In this test, the model worked worst when the learning rate was set as 0.1, with the accuracy on the test set being 4.61%. The highest accuracy among them is 95% with a learning rate is 0.01 (or 1e-2), then model performance dropdown with the lower learning rate. More visualisation results are shown in the jupyter notebook file 6. If the learning curves of each model are checked, the difference will be noticed between the learning rate. The smallest values among the list have the softest learning curve but also learn quite slowly compared to others. This confirms the effect of the learning rate on the model, which is the time the model learns. Their FLOPs stay the same with the value of 1.9e+2 GFLOPs. With the chosen learning, the model worked best with batch size 32, which is not much different from the other three values. However, the FLOPs increase with the growth of batch size. Details of the manual test will be provided in the table 1.

VALUES	ACCURACY	GFLOPs
Learing rate		
1e-1	4.61%	1.9e+2
1e-2	95.58%	1.9e+2
1e-3	94.16%	1.9e+2
1e-4	89.12%	1.9e+2
1e-5	84.09%	1.9e+2
Batch size with best learning rate		
16	95.62%	94.8
32	96.24%	1.9e+2
64	95.69%	3.79e+2
128	95.00%	7.59e+2

Table 1. Manual tuning performance with SGD optimise function and spare categorical cross entropy loss function.

4.2.2 Library function tuning

This section will use the learning rate scheduler, early stopping and hyperband tuning. The step decay concept is applied for the learning rate scheduler setup. The learning rate determines how quick and optimal the model learns on the dataset. Come on time, the model will reach the bottle net; lower the rate will let it collect the data more efficiently and sensitive in analysis. The formula for this adjustment is:

$$lr = lr_0 * (lr/2)^{\lfloor \frac{1+EpochNo.}{Epochdroprate} \rfloor} \tag{1}$$

The time for training is a concern when building the model. Too short, the model will not learn much, but too long could lead to over-fitted. This will depend on the performance of the model on the validation dataset. When the generalisation error increase, then the early stopping will be triggered. For the tuning, the library Hyperband will be used with the mentioned model design 3. The tuner will search the best combination from various values of optimiser (SGD and Adam), learning rate $(1e-1, 1e-2, 1e-3, 1e-4, 1e-5)$.

The tuner will pick up the setup with the highest accuracy on the validation dataset. Each test only runs on ten epoch training.

The tuner returns the combination of Adam optimiser with the learning rate is $1e-4$. The max accuracy on training and validation are absolute values. The model learning curves fluctuated for both loss and accuracy on training and validation. On testing, 96.63% is reached. Through the tunning progress, several combinations got the same performance result with the chosen hyper-parameters, namely Adam - 0.00001, SGD - 0.01, and SGD - 0.001. The tuner selected the first highest result, and the Adam - 0.0001 was tested earliest among others in this random tuner. More detail will be shown in the tuning progress summary of the Jupiter notebook file in the code section 6. The results are not much different from the manual tuning tests.

4.3. Tricks application

In this final section, one more implementation will be added to check whether higher accuracy could be achieved. It is common to know that computer vision tasks depend on the dataset. Consequently, more data could help the model learn more about the trait and patterns of each class. Therefore augmentation will be observed in this section. The setting for augmentation includes image shifting, zooming in or out, contrast adjusting, and brightness adjusting. These augment will apply to the training dataset, and the same model structure 3 still be used. For a quicker test process, the tuner will also be run in this experiment to test different learning rates and optimisation. Hyperparameter values were similar to the previous section. This time, the optimal parameters are SGD optimiser and learning rate 0.01, with the highest accuracy on the validation set at 95.88%. Adam optimiser with 0.0001 rates got the second highest with 93.66%. With fitting, both training and validation got high accuracy with 98.59% on training and absolute value for the validation. The testing is also called the absolute accuracy for the classification task.

The model has learned most from the dataset, and the trained data set can help the model to recognise more different signatures of each sign gesture. Therefore, augmentation has been applied to generate more data with image transformation. In this specific application above, shifting, altering brightness, and zooming are used to create more data for training. Rotation and flipping are not chosen to apply in this case which might need to be clarified for the model due to the similarity of symmetrical gestures.

In this experiment, the performance on the test set was higher than the validation set, and the test accuracy was more elevated than the validation. This could be a sign of overfitting. However, due to the augment only adding for training, the train set is more complicated and harder for the model to learn. On the other hand, the validation and test are more straightforward and already got similar cases learnt to make the prediction. Moreover, the loss on training is much higher than validation, which confirms that this model is not overfitted. They can be found in the coding file for more detail on the training performance. 6 or the illustration figure 6.

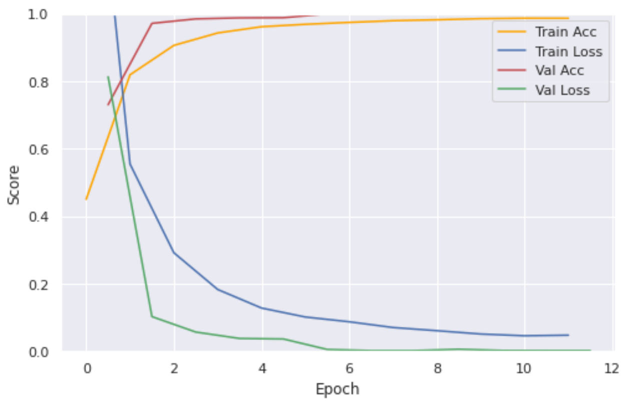

Figure 6. Augmenting implementation performance on training and validation

5. Discussion

In this assignment, there are several experiments have been conducted and observed. They include gradual adding layers, parameters alternation, tuning and augmentation. Four main Keras layers - Conv2D, MaxPooling, BactchNormalization, Dense are used to build the model. Different layers parameters like: kernel size, number of kernel, stride, pool size (for max pooling) are tested. Hyper parameters like optimiser, learning rate, batch size are also brought on. To extend the training dataset, more images are generated by zooming, changing contrast, brightness adjusting, and shifting.

In the first part of experiments, the work on the layers stack up and adjusting the parameter. The first effect of these features could be easily notice that the accuracy is increased when more layers added to the design. Base on the the point has been mentioned in the experimental discussion, the hidden layers are used for features extraction. That means the good structures can help the model extract good core features and learn the data which improve the performance of model. However, with greedy build that more than the sufficient need will cause the overfitting and drag the accuracy down [7]. Different layer parameters are also changed and observed. As the image features do not have a same properties, there are various characteristic search like larger size of filter, separated chain features, different number of filters to scan on general feature of data.

Hyperparameters are important factor in training and should be consider as they also affected the performance of the model. As on this experiment, what has been done is more aligned on seeking the higher accuracy then the autonomy of the model is not deeply focused. Might the batch size in tunning test still small compare to the datasize then they do not show the significant difference. The large batch size will make the model stuck with the local minima which cause the lost of the generalization [10].But too small size will not guaranteed that model converge to the global optima which will bound around that area.

Learning rate directly affect how fast the algorithm learns and will the cost function minimised. The optimal value is not easily found, it also depend on which optimise function is used. Either higher or lower than optimal values, the model will got stuck in learning, and the cost function will go high as well. Also not every states need the same learning rate, to get the model learn more, the scheduling has been applied to model to adjust the rate base on the period of training. Optimsers also give the different effects. ADAM learns fast and finds solution with big weight, which might make it not achieve high test accuracy and not generalising with large learning rate [8]. While in the most learning with SGD optimiser, it converges slower than ADAM and need higher learning rate than Adam. SGD more locally unstable which help it better to escape the sharp minima to the flat minima which make it better in generalization performance [19].

The augmentation is an extra test, which confirm the importance of dataset. The richness of it has made a great leap on the training. The model has learnt the objects in different conditions, environment, and might some has a bit cropped by the shifting. Only small shifting has been done as the dataset quite sensitive and can be mistaken without or be alter with auto filling function of shifting. Augmentation only add on the training not the validation or testing. Although the model has been adding the early stopping to prevent the overfitting, but it can not be denied that augmentation also contribute to the overfitting blocking. With more variant data, the model will not to focus into data detail which may of them could be noise.

6. Code

The code for the CNNs implementation, testing, PDF coding file, and data set are at following Git hub repository: `https://github.com/hongphuc-pham/CNNs_for_image_classification`

7. Limitation

The images in the dataset only have one channel; therefore, they are not tested on the pre-trained model and got transfer learning due to their different channels. Therefore, theoretically, they are deployable; however, the accuracy might not guarantee, or it will require more processing to support the transfer. Their model structure can be reused and opened for further study. The preprocessing (augmentation) is tested with the shown value. Various combinations could be applied for this model, or some augmentation can be added, like adding noise and adjusting contrast.

Besides the batch normalisation technique for improvement, there is another approach which is drop out [14]. This regularisation technique randomly selects neurons and ignores them. Therefore, in some interactions, the squared norm of weights will shrink down. The network will try to tune and find the features specified for target classes through the way the network learns. Some drops out will help the model to avoid strictly sticking to the specialisation, which could lead to overfitting. This technique was not applied in the experiments. It could give out similar results to batch normalisation. But it might conflict to combine both of them [4].

There are several other techniques like different activation functions (ELU, PReLU, ...), normalisation (unit normalisation, layer normalisation), and regularisation (alpha dropout, Gaussian dropout, activity regularisation, ...) that were not tested in this assignment. However, they could deliver the same result or better performance when put in the right place.

Aside from building own CNNs, there is another approach called transfer learning. Simply put, it takes the large dataset-trained model and transfers the model's knowledge to a smaller dataset. Several famous pre-trained models include the VGG family [13], ResNet family [9], EfficientNet family [17], and Inception family [15]. With this application, much work could be saved. However, as the purpose of this assignment is to discover and study about CNNs then, it does not use. Their performance could be compared with their design CNNs. Therefore, it could be initially better without tuning or augmenting.

Undeniably, this dataset is more superficial than other sets, which is why it can reach absolute accuracy on the test. The model might work well on this dataset but might be uncertain on others. This dataset has a blank background, with all white. In contrast, the CIFAR dataset, for example, will have more noise due to the background complexity. It also could be the resolution problem or the environmental condition.

8. Conclusion

Several experiments have been gone through. These tests include full connected layers test only, adding up with convolution network, batch normalisation layer, manually or automatically tuning, and augmentation. The accuracy on the test set starts from 67.80% with the initial model design and reaches absolute value with the different implementations of augmentation. Without the augmentation, several hyper-parameters are set up for similar performance with

Adam - 0.00001 or 0.0001, SGD - 0.01 or 0.001. When using augmentation, the best hyperparameters for the model are SGD and learning rate at 0.01, batch size 32. The more layers models have, the more dedicated parameter adjustment and layer arranging to extract the most from the dataset. Dataset richness is also an essential factor in computer vision tasks. That's why model set-up and configuration are critical to task performance for accuracy, the computing resource economy, and time-saving, as shown in the experiments.

References

[1] Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008. 4

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018. 2

[3] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 173–182, New York, New York, USA, 20–22 Jun 2016. PMLR. 4

[4] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 173–182, New York, New York, USA, 20–22 Jun 2016. PMLR. 4, 6

[5] Naciye Celebi, Tze-Li Hsu, and Qingzhong Liu. A comparison study to detect seam carving forgery in jpeg images with deep learning models. *Journal of Surveillance, Security and Safety*, 3, 08 2022. 2

[6] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130, 1988. 1

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 2, 5

[8] Aman Gupta, Rohan Ramanath, Jun Shi, and S. Sathiya Keerthi. Adam vs. sgd: Closing the generalization gap on image classification, 2021. 6

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 4, 6

[10] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016. 6

[11] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015. 1

[12] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. 1

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 6

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014. 6

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. 6

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 4

[17] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. 6

[18] Tecperson. Sign language mnist, Oct 2017. 1, 2

[19] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven C. H. Hoi, and Weinan E. Towards theoretically understanding why SGD generalizes better than ADAM in deep learning. *CoRR*, abs/2010.05627, 2020. 6