COMP551: Project 4
An Empirical Analysis of SqueezeNet

Gabriele Dragotto
gabriele.dragotto@polymtl.ca

Jiaqi Liang
jiaqi.liang@polymtl.ca

Yang Yang
yang.yang7@mail.mcgill.ca

***Abstract*—Deep neural networks have become powerful tools for image recognition in recent years. In this project, we analysed SqueezeNet, a CNN-based model developed for the ImageNet classification task. We aim to reproduce the model and apply it to the image classification problem on Tiny ImageNet dataset. First, we verified the original SqueezeNet model accuracy on the full ImageNet validation dataset by exploiting the provided pre-trained model. Second, we implemented the state-of-the-art model structure from scratch and trained it on the Tiny ImageNet dataset. In order to explore the influence of various components on model robustness and performance, we designed multiple experiments on hyperparameters, including learning rate, dropout ratio, activation function, and optimizer. The best model within our experiments is the one with Leaky ReLU activation function replacing ReLU in all layers, which exceeds the baseline model at early epoch, and achieves superior top-1 accuracy 56.62% and top-5 accuracy 85.66% respectively at different epochs. Finally, we evaluated different macroarchitectures by altering layers structure and their connections. Our experimental results prove that model performances are overall stable even when the structure is altered.**

** Full computational experiments, metrics and codes are available at: https://www.comet.ml/gdragotto/squeezenet-std, and https://www.comet.ml/evelynyang94/squeezenet-adj*

I. Introduction

Deep neural networks have been widely used in recent years for various tasks, such as text analysis, singal processing, image and speech recognition [1]. Convolutional neural networks (CNNs) has become an important tool since Krizhevsky et al. [2] introduced a novel approach AlexNet architecture to solve visual recognition on the ImageNET dataset [2]. However, the increasing complexity of CNN demands high computing capacity [3]. To alleviate this problem, new architectural design strategies have emerged to improve previous models. A typical case using such an approach is SqueezeNet, which redesigns the network of Krizhevsky et al. [2].

SqueezeNet is a CNN-based architecture with $50\times$ fewer parameters than AlexNet, while maintaining AlexNet-level accuracy on ImageNet [4]. The model achieves a good balance between classification accuracy and resource usage, while also minimizing the size of trained weights. Thus, SqueezeNet is a reasonable model for us to work on, considering the limited computing time and computational resources.

The main goal of this project is to implement SqueezeNet and explore how different components affect model performance, and further to try to improve the accuracy. We first studied the structure of the SqueezeNet and exploited the pre-trained weights to verify the accuracy on the original ImageNet validation set. Due to the large size of the ImageNet dataset ($\sim$1.3TB), we performed our tests on the Tiny ImageNet dataset [5], a miniature of ImageNet dataset. Our baseline model is trained with stochastic gradient descent (SGD), and a fixed learning rate of 0.04, achieving top-1 accuracy of 55.15% and top-5 accuracy of 79.41% on the validation set. Second, we tested several different hyperparameter selections and their impact on performances, including learning rate, dropout ratio, activation function, and optimizer. Finally, we performed several changes to macroarchitecture by adding extra maxpooling between layers, removing the $Fire9$ layer, and applying residual connections in order to further explore the influence of these components.

II. Related work

CNNs have been proven to be effective in large-scale image recognition tasks. LeCun et al.

[6] proposed to use CNNs for digit recognition applications in the late 1980s. Since AlexNet won the 2012 ImageNet competition, many models with deeper network structures have been presented and achieved reasonably good performances [7]. AlexNet consists of 5 convolutional layers and 3 fully-connected layers, with batch-normalization applied after the first 2 convolutional layers. In 2014, VGGNet [8] and GoogLeNet [9] both obtained great performances in the ImageNet Large Scale Visual Recognition Challenge(ILSVRC), proving that deeper and wider networks can eventually improve the accuracy in classification tasks. However, VGGNet is computationally expensive to train, and the trained weights are stored within hundred of MBs, even though it only contains 3×3 convolutions and 2×2 pooling. He et al. [10] introduced the concept of ResNet with connections that skip over multiple layers, which is the winner of 2015 ILSVRC.

In practice, applications often require small architectures rather than very large and deep CNNs [11]. For instance, the inception architecture of GoogLeNet has a good performance with strict constraints on memory and computing power. Instead of fully connected layers, it employs average pooling on top of ConvNet, which enables elimination of a large amount of parameters [9]. It has only 5 million parameters, namely a $12\times$ reduction with respect to AlexNet [12]. SqueezeNet, proposed by Iandola et al. [4], was intended to be rolled out on mobile devices, and achieves the same accuracy level of AlexNet but with $50\times$ fewer parameters. The original SqueezeNet has a top-1 classification accuracy of 57.5% and a top-5 classification accuracy of 80.3% on the ImageNet challenge.

III. SqueezeNet

Iandola et al. [4] aimed to identify CNN architectures that have fewer parameters while maintaining a competitive accuracy, and implemented SqueezeNet pursuing the following strategies: 1) replacing 3x3 filters with 1x1 filters. 2) decreasing the number of input channels to 3x3 filters. 3) downsampling late in the network so that convolution layers have large activation maps.

According to the original paper, the Fire module of SqueezeNet is made up of a squeeze convolution layer (1×1 filters), fed into an expand layer that has a mix of 1×1 and 3×3 convolution filters (see Fig. 1). The original SqueezeNet (v1.0) starts with a standalone convolution layer ($Conv1$ with 96 filters), followed by 8 Fire modules ($Fire2-9$), and ends with a final conv layer ($Conv10$). The number of filters in each fire module increases gradually from the beginning to the end. SqueezeNet performs max-pooling with a stride of 2 after layers $Conv1$, $Fire4$ and $Fire8$, and a global average-pooling after $Conv10$ layer.

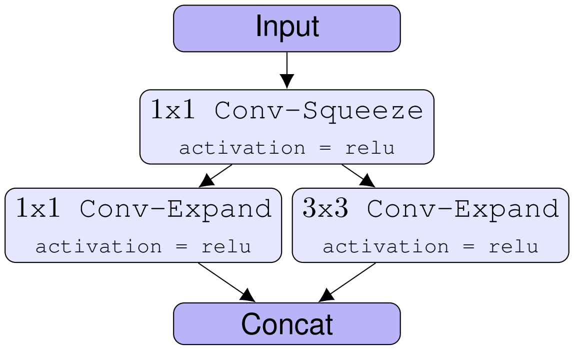

Fig. 1: Fire module structure

Taking into account our resource constraint and time requirement, we worked with SqueezeNet v1.1 which requires $2.4\times$ less computation than SqueezeNet v1.0 without diminishing accuracy [13]. The structure is shown in Fig. 2. Unlike v1.0, v1.1 starts with a standalone convolution layer with 64 filters and the pooling operations are after $Conv1$, $Fire3$, $Fire5$, and $Conv10$. In each Fire module, $s_{1\times1}$ is the number of filters in the squeeze layer, $e_{1\times1}$ is the number of 1×1 filters in the expand layer, and $e_{3\times3}$ is the number of 3×3 filters in the expand layer. The author set $s_{1\times1}$ to be less than ($e_{1\times1}$ + $e_{3\times3}$), so the squeeze layer helps to limit the number of input channels to the 3×3 filters.

IV. Dataset and pre-processing

A. Dataset

To reproduce the performance of the original SqueezeNet, we first run the pre-trained SqueezeNet model on the ILSVRC 2012 validation dataset [14]. The validation set consists of 50000 labeled images within 1000 object categories, collected from flickr and other search engines.

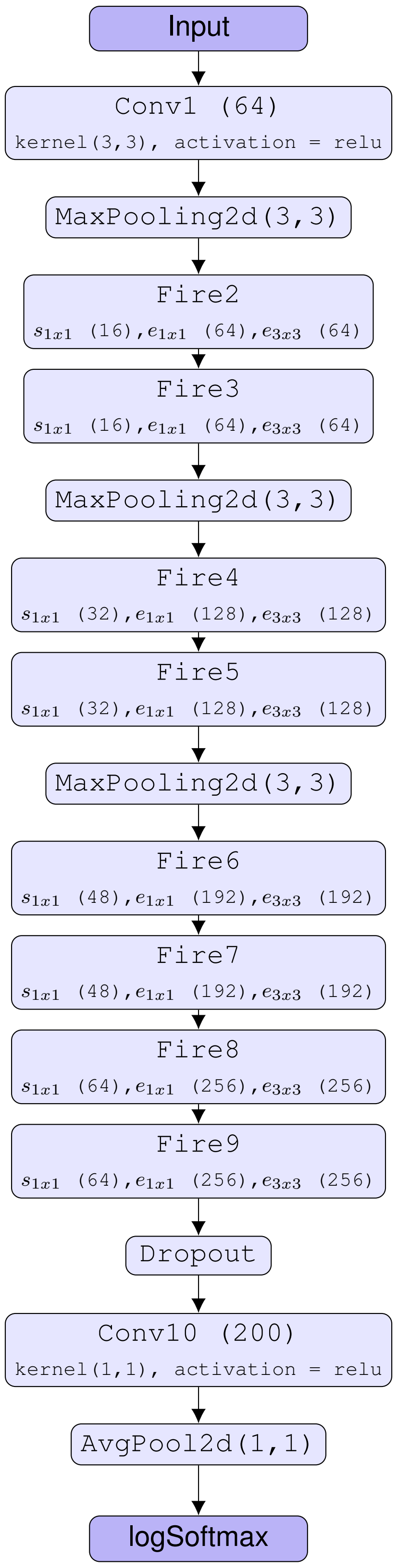
Input
Conv1 (64)
kernel(3,3), activation = relu
MaxPooling2d(3,3)
Fire2
s_{1x1} (16),e_{1x1} (64),e_{3x3} (64)
Fire3
s_{1x1} (16),e_{1x1} (64),e_{3x3} (64)
MaxPooling2d(3,3)
Fire4
s_{1x1} (32),e_{1x1} (128),e_{3x3} (128)
Fire5
s_{1x1} (32),e_{1x1} (128),e_{3x3} (128)
MaxPooling2d(3,3)
Fire6
s_{1x1} (48),e_{1x1} (192),e_{3x3} (192)
Fire7
s_{1x1} (48),e_{1x1} (192),e_{3x3} (192)
Fire8
s_{1x1} (64),e_{1x1} (256),e_{3x3} (256)
Fire9
s_{1x1} (64),e_{1x1} (256),e_{3x3} (256)
Dropout
Conv10 (200)
kernel(1,1), activation = relu
AvgPool2d(1,1)
logSoftmax

Fig. 2: SqueezeNet architecture

Since the source of training dataset is both inaccessible and too large ($\sim$ 1.3TB), we ran our experiments on the Tiny ImageNet Visual Recognition Challenge dataset [5]. It runs similar to the ILSVRC 2014. Instead of 1,000 classes in the ImageNet challenge, the Tiny ImageNet dataset has 200 classes, each of which contains 500 training images and 50 validation images. Each image has a resolution of 64×64 pixels down-sampled from 224×224. However, the details in the pictures are lost due to downsampling, which leads to an increased difficulty in recognizing small objects.

B. Pre-Processing

We applied $torchvision.transforms$ to transform images in Tiny ImageNet dataset. For the training set, we performed the following transformations: 1) frame the input images from pixels 64×64 to 227×227. 2) randomly rotate images within $(-20, 20)$ degrees. 3) horizontally flip them with a probablity of 0.5. 4) convert images to 3 dimensional tensors (3 RGB channels) with each element in the range $[0.0, 1.0]$. 5) normalize tensors with mean $[0.4802, 0.4481, 0.3975]$ and standard deviation $[0.2302, 0.2265, 0.2262]$. Here, random rotations and flips are meant to improve the robustness of the model and its predictions. For the validation set, only 1), 4), and 5) have been applied.

V. Models and Analysis

We first applied the pre-trained SqueezeNet model to the original validation set. The top-1 accuracy is 54.69% and top-5 accuracy is 78.13%. Then, we used Tiny ImageNet dataset to construct our baseline model and conducted the experiments as below.

A. Baseline and experiments design

Our baseline model is SN_lr1 which has the default configuration of the original SqueezeNet, except for learning rate. The batch size is 512 and the learning rate is 0.04, optimized by SGD with momentum of 0.9 and weight decay of 0.0002. In terms of dropout ratio, we started with the suggested 0.5. In order to enable our models to be trained dynamically, we added an early stop condition. In particular, after 50 non-improving (in terms of top-5 accuracy) epochs and a top-5 accuracy of at least 79%, the training process will

Model	Learning Rate	Drop-out	Activation Function	Optimi-zer	End Epoch	Best Epoch	Training Set Top-1 ImageNet Accuracy	Training Set Top-5 ImageNet Accuracy	Validation Set Top-1 ImageNet Accuracy	Validation Set Top-5 ImageNet Accuracy
SN_lr1	0.04	0.5	ReLU	SGD	146	95	54.37	80.00	55.15	79.41
SN_lr2	0.01	0.5	ReLU	SGD	201	196	58.75	86.87	47.43	75.74
SN_lr3	poly	0.5	ReLU	SGD	148	59	51.87	78.12	49.26	76.47
SN_dr1	0.04	0.25	ReLU	SGD	228	177	67.50	87.50	52.94	80.88
SN_dr2	0.04	0.75	ReLU	SGD	152	150	59.37	81.87	49.63	78.31
SN_af1	0.04	0.25	Softplus ($Conv10$)	SGD	177	144	56.87	89.37	50.74	77.21
SN_af2	0.04	0.75	Softplus ($Conv10$)	SGD	174	123	58.75	81.25	49.26	80.51
SN_af3	0.04	0.5	Leaky ReLU (Features)	SGD	146	114	60.00	85.00	54.41	82.72
SN_af4	0.04	0.5	Leaky ReLU (All layers)	SGD	211	165	61.25	86.87	55.51	85.66
SN_op1	0.04	0.5	ReLU	Adam	139	86	55.00	85.00	52.57	77.57
SN_op2	0.04	0.5	Leaky ReLU (All layers)	Adam	172	121	55.62	85.00	54.04	82.72

TABLE I: Accuracy achieved by models with various hyperparameters

be stopped. Note that some models are manually stopped, since their accuracy changes so slowly or does not increase in a long period of epochs.

The Top-1 and top-5 accuracy for SN_lr1 model on both training set and validation set is shown in Fig. 3, and the loss is illustrated in Fig. 4. Based on our baseline, we compared various models tuned by the following parameters: learning rate, dropout ratio, activation function, and optimizer. Moreover, we also changed the structure of the original model to explore the effect of some layers on the performance. Computational tests ran on 2 GPUs (*NVIDIA TITAN Xp*) with parallelized computing.

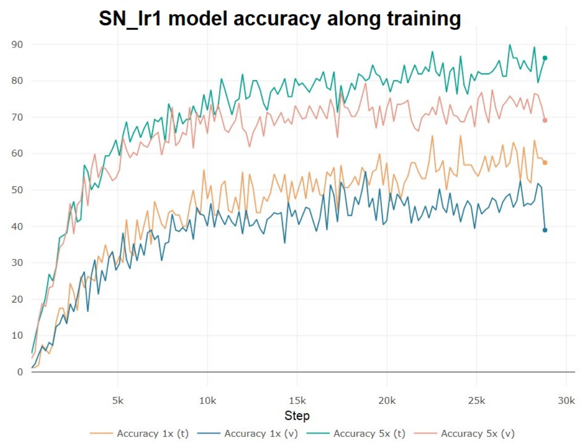

Fig. 3: Top-1 and top-5 accuracy for SN_lr1 model along training step

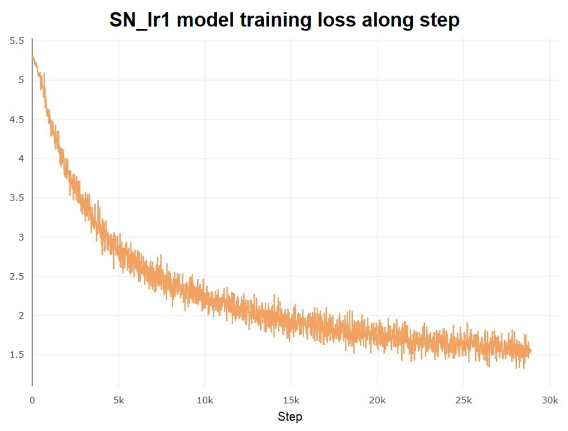

Fig. 4: SN_lr1 model training loss along training step

TABLE I shows the comparison between different hyperparameter settings. We report the best performing model according to the highest top-5 accuracy during the experiments.

B. Learning Rate

Three different settings for the learning rate were tested. In particular: 1) SN_lr1 with a fixed learning rate 0.04. 2) SN_lr2 with a fixed learning rate 0.01, the same as the baseline rate of Mishkin et al. [15]. 3) SN_lr3 with a polynomial learning rate, linearly decreasing the base learning rate of 0.04 by $0.1^{\lfloor(\#Epochs/30)\rfloor}$, which is the method in the paper and similar to that suggested in [15].

The worst performing model is SN_lr2 (see TABLE I), which also yields the slowest convergence to the final accuracy. Its highest top-1 and top-5 accuracies are lower than model SN_lr1 respectively by 3.8% and 7.7%, and slightly lower than SN_lr3. Therefore SN_lr1 is our starting baseline with top-1 and top-5 accuracy as high as 55.1% and 79.4% at epoch 95. This result suggests that, in our case, a proper fixed learning rate can achieve moderate or even better performance than the decreasing learning rate approach (SN_lr3) proposed in SqueezeNet.

C. Dropout

To explore the influence of dropout ratio on model performance, two more dropout ratios have been applied. In particular, we tested the dropout ratio of 0.25 (SN_dr1) and 0.75 (SN_dr2) while keeping other hyperparameters the same as in the baseline (SN_lr1). Both SN_dr1 and SN_dr2 models obtain the top-1 and top-5 accuracy slightly lower than baseline (dropout ratio 0.5) when comparing results at the same epoch. SN_dr1 obtains the top-5 accuracy of 80.88%, namely higher than the baseline but at a later epoch 177 (compared to 95). Therefore, the baseline model with dropout ratio 0.5 has the best performance under the comprehensive consideration of both top-1 and top-5 accuracy, and training time.

D. Activation function

Other alternative activation functions have been considered. The SoftPlus [16] function is a smooth version of ReLU, reported to show improved performance with lesser epochs to convergence, compared to ReLU, during training in a phone recognition study [17]. We explored how the model behaves when the activation function in $Conv10$ layer changes from ReLU to Softplus, along with 2 different dropout ratios of 0.25 and 0.75. Under the dropout ratio of 0.25, SN_dr1 (ReLU) has a slightly better performance than the model SN_af1 with Softplus. However, with the dropout ratio of 0.75, Softplus (SN_af2) achieves a higher top-5 80.51% but a slightly worse top-1 accuracy, compared to SN_dr2. In our case, the results show that ReLU works better than Softplus in $Conv10$ when the dropout ratio is 0.25, and changing activation function only in $Conv10$ leads to small differences in terms of top-1 accuracy when the dropout ratio is 0.75.

Model	Activation Function	Best Epoch	Validation Set Top-1 ImageNet Accuracy	Validation Set Top-5 ImageNet Accuracy
SN_lr1	ReLU (All layers)	95	55.15	79.41
SN_af3	Leaky ReLU (Features)	76	51.84	80.51
		101	52.57	80.88
		114	54.41	**82.72**
		143	**56.25**	79.41
SN_af4	Leaky ReLU (All layers)	86	55.88	80.15
		123	56.25	81.25
		165	55.51	**85.66**
		185	**56.62**	83.09

TABLE II: Accuracy in three models with different activation functions. Here we present the observed high accuracy in SN_af3 and SN_af4 model.

With the intention to further improve model performance, we also considered ReLU variants, which theoretically improve model accuracy by constraining non-zero gradients for dealing with neuron dying issues [18]. Here, we employed Leaky ReLU, a variant proposed by Maas et al. [19] that has widely accepted to own the potential to improve model performance comparing to ReLU [18]. Another option PReLU is also offered by PyTorch, which requires to learn more parameters during training. In consideration of the limited time, we only performed our tests with Leaky ReLU.

$$LeakyReLU(x)=\max(0,x)+\alpha\cdot\min(0,x) \tag{1}$$

The Leaky ReLU activation function is computed as (1). Based on previous related researches, a model with a larger slope α will achieve better accuracy rates [20]. Therefore, we employed negative slope $\alpha = 0.2$ for Leaky ReLU function. We performed two models with Leaky ReLU replacement. One model (SN_af3) replaces ReLU with Leaky ReLU in features part, i.e., all layers except $Conv10$ layer. The other one (SN_af4) has ReLU replaced by Leaky ReLU in all layers, including Fire modules and Conv layers.

In both experiments, we observed a much higher top-5 accuracy, a faster training and validation loss decay, which improves the performance compared to the baseline. As shown in TABLE II, the model with Leaky ReLU replacement before $Conv10$ (SN_af3) achieves higher top-5 accuracy as early as epoch 76, compared to our baseline, despite of lower top-1 accuracy, and further achieves the highest top-5 accuracy at epoch 114. While (SN_af3) model is not able to surpass the top-1 accuracy and top-5 accuracy of our baseline in the same time, model SN_af4 successfully exceeds the baseline markedly for both top-1 and top-5 accuracy. Specifically, the baseline SN_lr1 achieves a top-1 accuracy 55.1% and top-5 accuracy 79.4% at epoch 95 and no longer improves the accuracy. During the running time of SN_af4, the model achieves the highest top-1 accuracy 56.62% at epoch 185 and the highest top-5 accuracy 85.66% at epoch 165, which are the best values resulting from our experiments. The accuracy along training step of SN_af4 model is given in Fig. 5 and its training loss is illustrated in Fig. 6. Therefore, the Leaky ReLU is capable of countering *dead neuron* issue within SqueezeNet and gains significantly better performance, especially under circumstance of full usage of it in all layers.

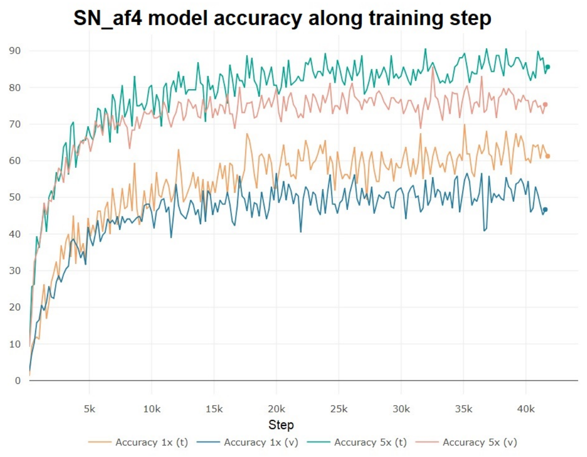

Fig. 5: Top-1 and top-5 accuracy for SN_af4 model along training step

E. Optimization

Different optimizers affect the training stage and hence the final model accuracy. Thus, we also tested Adam optimizer [21] in model SN_op1 with the same learning rate and weight decay as the baseline model. We set ϵ to 0.1 and remain the other parameters as the default of $torch.optim.Adam$. Adam yields a faster loss decay at the beginning of the training and achieves a relatively high top-1 and

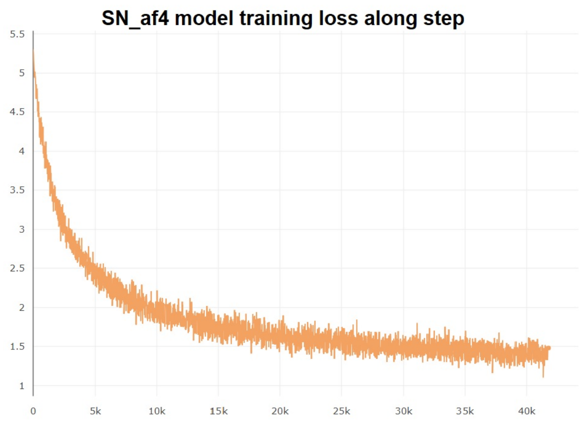

Fig. 6: SN_af4 model training loss along training step

top-5 accuracy respectively of 43.75% and 75.00% as early as epoch 59. But it is eventually stuck at a plateau in later epochs with 52.57% top-1 accuracy and 77.57% top-5 accuracy. This suggests that in our case Adam might have been caught in a local minimum avoided by the SGD. Besides, we ran another experiment with Adam and Leaky ReLU in all layers. Compared with SN_af4, the model SN_op2 with Adam underperformed SGD.

VI. Macroarchitecture Exploration

To explore the effect of model components on the macroarchitecture level, we designed several experiments and the results are given in TABLE. III.

A. Adding or Removing Layers

We conducted two experiments by adding or removing specific layers inside the model. In specific: 1) model SN_f1 by removing the $Fire9$ layer from the baseline (SN_lr1). 2) model SN_m1 by adding maxpooling layer between layer $Fire7$ and layer $Fire8$ in SN_dr1.

As we can see from TABLE. III, after removing $Fire9$ module, the model SN_f1 achieves a slightly higher top-5 accuracy of 79.78% but lower top-1 accuracy of 48.16%, compared to baseline SN_lr1. When adding a maxpooling layer between $Fire7$ and $Fire8$ modules, the model SN_m1 rapidly reaches the accuracy level of SN_dr1 in less running time, but eventually cannot surpass the performance of SN_dr1. We can deduce that adding

Model	Modification of layers	End Epoch	Best Epoch	Training Set		Validation Set	
				Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
SN_lr1		146	95	54.37	80.00	55.15	79.41
SN_f1	Remove Fire 9	271	220	64.37	85.62	48.16	79.78
SN_dr1		228	177	67.50	87.50	52.94	80.88
SN_m1	Add maxpooling between Fire 7 and Fire 8	254	157	55.62	82.50	51.47	77.57
SN_re1	Add residual connections in Fire modules 3, 5, 7, and 9 (Elementwise addition)	157	106	52.50	76.25	51.84	79.41

TABLE III: Accuracy achieved by various models

maxpooling can help accelerate the reduction of the loss, but does not help improve the final accuracy. These results suggest that removing $Fire9$ module leads to a slightly worse top-1 accuracy and a moderate top-5 accuracy (since it achieves top-1 and top-5 52.57% and 78.31% respectively at epoch 185), while adding an extra maxpooling layer between $Fire7$ and $Fire8$ causes worse performance in both top-1 and top-5 accuracy.

B. Residual connections

Inspired by Han [22], we created a residual connection in the Fire module of SqueezeNet. The Fire module with residual connection architecture is demonstrated in Fig.7. In order to implement this structure into SqueezeNet v1.1, we add residual connections to $Fire3$, $Fire5$, $Fire7$, and $Fire9$, since the input size and output size of these layers are the same. We found that the SqueezeResNet model (SN_re1 in TABLE. III) achieved a lower top-1 accuracy and the same top-5 accuracy as the model SN_dr1. The result shows that adding this residual connection architecture does not lead to significant improvement in performance within limited epochs in our case.

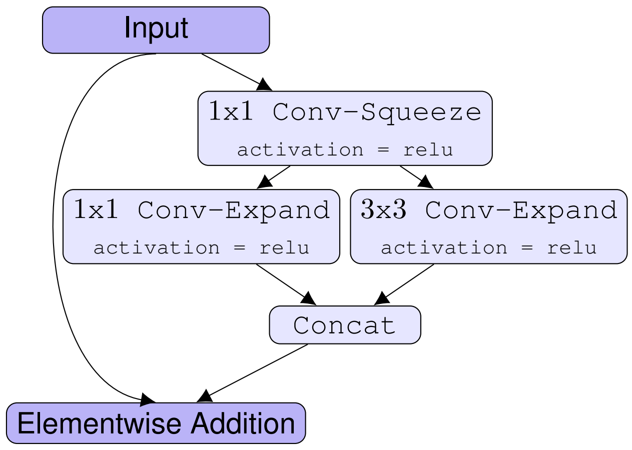

Fig. 7: Residual connection in Fire module

VII. Discussion and Conclusion

In this project, we successfully reproduced the performance of SqueezeNet on the original ImageNet validation set. We then adopted SqueezeNet v1.1 model on Tiny ImageNet dataset, a downsized version of ImageNet. We further explored SqueezeNet model robustness and improved the baseline accuracy with a different activate function.

On one hand, we have tested different hyperparameter settings, including learning rate, dropout ratio, activation function, and optimizer. 1) Adjusting the dropout ratio from 0.5 to either 0.75 or 0.25 makes no significant improvements. 2) Setting a fixed learning rate of 0.04 (SN_lr1) achieves a better performance than using a linearly decreasing learning rate with base learning rate 0.04 (SN_lr3) in the original SqueezeNet model. Moreover, a lower fixed learning rate of 0.01 (SN_lr2) achieves a worse performance with longer computing time. 3) In terms of activation functions, we first changed the activation function only in the $Conv10$ layer from ReLU to Softplus, which leads to negligible changes in performance. Considering the dying neuron issue of ReLU, we employed Leaky ReLU. Leaky ReLU for all layers can significantly improve the model accuracy and shorten the training time, and finally achieve the best top-1 and top-5 accuracy of 56.62% and 85.66% in different epochs. 4) Changing the optimizer from SGD to Adam can fasten the early training, but not lead to improvement in accuracy.

On the other hand, we explored the model robustness at a macroarchitectural level by adding or removing layers and adding residual connections. Adding an extra maxpooling layer between $Fire7$ and $Fire8$ does not change significantly the accuracy, while removing the $Fire9$ layer leads to a worse model accuracy. In terms of residual connections, the approach is not able to improve the model accuracy, and slightly slow the training time.

Within constrained computational capacity and time, we have conducted extensive experiments on model components from both parametric and macroaritectural level. In further work, a detailed investigation on the impact of the α slope in the Leaky ReLU can be performed. Several variants of ReLU (eg, PReLU, SReLU) are also potential candidates for improving the accuracy. In particular, SReLU attracted interest for its promising results achieved in image classification tasks [23]. In addition, a deeper network with more Fire modules can be applied to see if they can improve the performance, although it will results in more computing time. More exploration will help find the trade-off between accuracy and computational complexity.

Statement of contribution

Gabriele Dragotto: Reproduce paper results, design experiments, and writing;

Jiaqi Liang: Design experiments, literature review, and writing;

Yang Yang: Design experiments, literature review and writing.

References

[1] Andrey V Savchenko. *Search techniques in intelligent classification systems*. Springer, 2016.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Mohammad Javad Shafiee, Francis Li, Brendan Chwyl, and Alexander Wong. Squishednets: Squishing squeezenet further for edge device scenarios via deep evolutionary synthesis. *arXiv preprint arXiv:1711.07459*, 2017.

[4] Forrest N. Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[5] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Tiny imagenet visual recognition challenge, 2016. URL https://tiny-imagenet.herokuapp.com.

[6] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[7] Hujia Yu. Deep convolutional neural networks for tiny imagenet classification. 2017.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Zhe Li, Xiaoyu Wang, Xutao Lv, and Tianbao Yang. Sep-nets: Small and effective pattern networks. *arXiv preprint arXiv:1706.03912*, 2017.

[12] Pedro Ballester and Ricardo Matsumura Araujo. On the performance of googlenet and alexnet applied to sketches. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[13] Forrest Iandola. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters, 2016. URL https://github.com/DeepScale/SqueezeNet.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A

large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[15] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of CNN advances on the imagenet. *CoRR*, abs/1606.02228, 2016. URL http://arxiv.org/abs/1606.02228.

[16] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *NIPS*, 2001.

[17] Hao Zheng, Zhanlei Yang, Wen-Ju Liu, Jizhong Liang, and Yanpeng Li. Improving deep neural networks using softplus units. pages 1–4, 07 2015. doi: 10.1109/IJCNN.2015.7280459.

[18] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018. URL http://arxiv.org/abs/1811.03378.

[19] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

[20] Xiu-Shen Wei. Must Know Tips/Tricks in Deep Neural Networks, 2015. URL http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, art. arXiv:1412.6980, Dec 2014.

[22] William B. Shen and Song Yeong Han. Exploration of the effect of residual connection on top of squeezenet a combination study of inception model and bypass layers, 2016.

[23] Xiaojie Jin, Chunyan Xu, Jiashi Feng, Yunchao Wei, Junjun Xiong, and Shuicheng Yan. Deep Learning with S-shaped Rectified Linear Activation Units. *arXiv e-prints*, art. arXiv:1512.07030, Dec 2015.