Image Classification Using Convolutional Neural Networks

Group 2
Dai Qiyu 1155141616
JIANG Yun Hui 1155141677
LUO Jun Wen 1155141641
MAK Man Fung 1155141893

Contents

1 Introduction **1**

2 Data **2**

2.1 Data Collection 2

2.2 Data Transformation and Splitting 2

3 Theoretical Background of CNN **3**

3.1 Convolutional layer 3

3.2 Pooling 4

3.3 Batch Normalization 4

4 Analysis **5**

4.1 KNN 5

4.2 LeNet 5

4.3 AlexNet 6

4.4 Deep Residual Network (ResNet) 7

5 Limitations **9**

6 Results and Conclusion **10**

7 References **11**

8 Appendices **12**

1 Introduction

Nowadays, images play a crucial role in carrying online information. Given the importance of visual data and the availability of large volume datasets, image classification techniques become heavily used in the internet domains. For example, most website developers prefer to label products to improve the efficiency of users' searching behavior using keywords. Besides that, merchants can also recommend products that customers are interested in more accurately if their pictures are put in the correct categories. Therefore, a well-developed image classification system is essential in determining both users' experiences and profit of sellers.

Trained as statisticians, we tackle this problem through a data-driven approach. That is, we will train and assess machine learning classifiers on collected image data and labels. Traditional supervised machine learning methods, like k-nearest neighbors (hereinafter called "KNN") and support vector machine (hereinafter called "SVM"), are commonly used in both literature and Kaggle data science projects. Unfortunately, due to their shallow computing structures, the above methods are only suitable for solving problems with a limited number of samples. If the dataset is large, their ability to deal with complex classification problems will be insufficient (Xin & Wang, 2019).

Therefore, in recent years, the convolution neural network (CNN), one of the most popular deep learning frameworks, is gradually replacing those old works. The main reason is that CNN can be used in image processing domains with more effective and accurate predictions. Looking back at the history, CNN was originated from a design developed by LeNet and his team for recognizing handwritten digits in the 90s (Yann et al., 1998). Then, there was a huge improvement in 2012, when AlexNet was the first winner of the ImageNet challenge. Such a milestone made worldwide statisticians realize the great importance of CNNs for handling image analysis. Subsequently, VGGNet, based on AlexNet, tried to increase the depth of the architecture to reduce the error rate of classification (Yu et al., 2016). In addition, GoogLeNet invented an inception module to collect more details and features of images (Ballester & Araujo, 2016). However, in 2015, researchers found no obvious changes and even a drop in accuracy of prediction if too many layers are added for a CNN model. This motivates the presence of ResNet, which adopted a residual mapping to update the parameters effectively (Targ et al., 2016).

In the light of those aforementioned efforts, our project aims to identify and predict the categories of a scenery dataset based on three types of trainable CNN models: LeNet, AlexNet, and ResNet. Besides, for the sake of comparison, the KNN algorithm is also tested. The results of this project justify the advantages of CNN in image classification tasks over simpler traditional models. Also, our best model gives an 81.25% accuracy of prediction for a 4-categories classification problem. And anyone interested can replicate the work using our accompany code and data (click **here)**.

The remainder of this paper proceeds as follows. We describe the data source and pre-processing procedure in Section 2. Section 3 briefly introduces the theoretical

background of CNN. Section 4 contains our principal analysis. The limitations of current work, as well as potential developing directions are discussed in Section 5. Section 6 summarizes the final results and concludes.

2 Data

2.1 Data Collection

We use python code to automatically scrap the Intel images from the Unsplash website, where a lot of high-quality pictures are provided. There are 8000 in total, coming from four categories (Forest, Mountain, Ocean, and Street). The amount of each category is 2000. Some sample images are displayed in Figure 1.

It is worth mentioning that those photos are taken in different places and by different photographers. Therefore, the variation within each category can be large, leading to a non-trivial classification task.

Figure 1: A sample display of Intel images (their size are different)

2.2 Data Transformation and Splitting

Before model construction, we need to rescale raw pictures to get a homogeneous input size. For KNN and SVM, considering the time to run our code, the shape of all images is converted to be $32 \times 32 \times 3$. Moreover, when researchers define the CNN architectures, they have already planned the input size and the number of layers to use. So for those models, we transform images according to the originally proposed settings. Specifically, LeNet requires $32 \times 32 \times 1$ input size while it is $224 \times 224 \times 3$ for both AlexNet and ResNet. Then, to get our data prepared for training, we normalize them to be on a scale from 0 to 1.

Lastly, we apply training-validation-test splitting to each category. The separation percentages are 70%, 15%, and 15%, respectively. We use the training dataset

for model fitting and the validation dataset for model hyperparameter selection. Afterwards, the model's predictive performance is assessed with the testing dataset.

3 Theoretical Background of CNN

CNN (Convolutional Neural Network) is a subclass of neural network, specializing in detecting patterns in images. To show its architectural features, we first explain how the convolutional layer works, which is the core of the CNN. Then, the pooling layer and batch normalization, which are also important building blocks, will be introduced.

3.1 Convolutional layer

The operations of the convolutional layer differ from the traditional fully connected layer mainly in two perspectives. Firstly, notice that the key to image recognition is to identify some critical local patterns. Therefore, instead of examining the whole image, one neuron only receives the signal from part of the input, called the receptive field. Secondly, since a particular pattern may appear in different regions, the convolutional layer applies the parameter sharing technique to increase the computational efficiency. Hence, each receptive field is dealt with a set of neurons with identical parameters. Figure 2 displays how convolutional layer works. Usually, the output of each convolutional layers is called feature map(s).

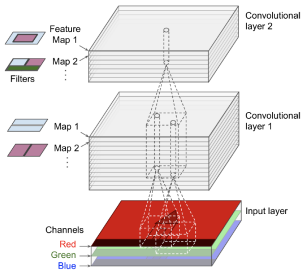


Figure 2: Convolutional Layer

Additionally, to characterize settings of receptive fields, several hyperparameters are defined.

- Kernel size is the receptive field's height and width. Notice that the depth of a receptive field is equal to the number of channels.
- Stride determines how much the receptive field shifts vertically and horizontally as the operating field of the neuron changes.
- Padding is the addition of empty pixels around the edges of an image.

Mathematically, the output function for a neuron in the convolutional layer can be summarized as

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \dots w_{u,v,k',k}$$
$$\text{where } i' = i \times s_h + u; j' = j \times s_w + v$$

3.2 Pooling

The pooling layer is usually added after the convolutional layer, whose primary goal is to shrink the result of previous layer and reduce overfitting to some extent. Notice that this operation is pre-specified rather than learned from data. Furthermore, all models in this project have used max-pooling. And for ResNet, average-pooling is also adopted.

As shown in Figure 3, max-pooling and average-pooling compute the maximum and mean for patches of a feature map respectively, which then can be used to form another downsampled (pooled) feature map. Many empirical studies have proved that this kind of translation does not significantly affect the values of outputs, as image classification does not rely on the ignored details too much(Nagi et al., 2011).

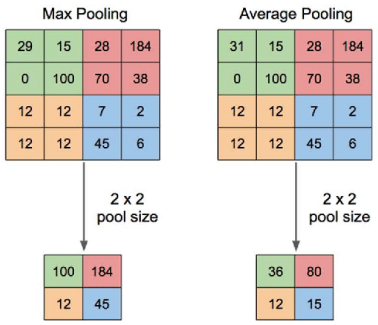


Figure 3: max-pooling

3.3 Batch Normalization

In deep neural networks, when a substantial weight is present, the output for the corresponding neuron will also be considerably larger than others. And the imbalance will continue to cascade through the remaining network. Consequently, this will cause the instability of models and even result in exploding gradient problem. Hence, the batch normalization comes.

The implementation of batch normalization consists of three steps: firstly, the output from the previous activation function is normalized; then, we multiply the normalized output by a specific parameter; lastly, another constant is added to get the final product.

Formally, denote the activation function output to be $a_1, ..., a_N$, below shows the BN operations,

$$\mathrm{BN}_{\gamma,\beta}(a_1, ..., a_N) = (\gamma\frac{a_1 - \mu}{\sigma + \epsilon}, ..., \gamma\frac{a_N - \mu}{\sigma + \epsilon}) + \beta,$$

where γ, β are the two new parameters to be trained and ϵ is a fixed constant decided before the training.

4 Analysis

4.1 KNN

We use KNN model as a starting point of classification, which is a naive non-parametric algorithm. It classifies the new data by looking at the categories of its k nearest annotated cells. Notice that the metric used to measure the distance between two vectors here is the Euclidean distance, which is defined as

$$d(X_1, X_2) = \sqrt{\sum_{j=1}^{d}(X_{1j} - X_{2j})^2}$$

where $X_1, X_2 \in R^d$. Moreover, if the neighbors of a input belong to two or more categories, the final decision should be made by taking the option of receiving more votes.

Before deciding the final model, the tuning parameter, k, should be selected to get the best predictive performance on new data points. Therefore, for a grid of k from 1 to 100, we firstly fit a KNN model using the training dataset and then record the corresponding prediction accuracy using the validation dataset. It turns out that $k = 1$ works best with a validation accuracy of 42.42%. Therefore, we evaluate this "1-NN" model on testing data, and the accuracy result is 39.92%.

Regarding the KNN classification results, there are several key remarks. On the one hand, our optimal k chosen by validation data is 1, which implies that a complex model with huge set of parameter is required for image recognition. In this sense, neural network models might be a better choice than traditional models by their enormous number of parameters. On the other hand, $k = 1$ also means 100% accuracy on the training sample. Compared with the low out-of-sample accuracy, a severe overfitting problem is present. Given these cons, we use the KNN model as a baseline to see if the CNN model can enhance the image classification work.

4.2 LeNet

The first convolutional neural network we consider is LeNet-5, one of the earliest CNN models proposed by Yann Lecun (Yann et al., 1998). It comprises seven layers. As shown in Figure 4, the layers C1, C3, and C5 are convolutional layers with 6, 16, and 120 feature maps, respectively. And layers S2 and S4 implement subsampling. Finally, layer F6 is the fully connected layer linked with the output.

Moreover, the LeNet-5 expects the input to be $32 \times 32 \times 1$ tensor. Therefore, we converted all RGB images to the corresponding grayscale version.

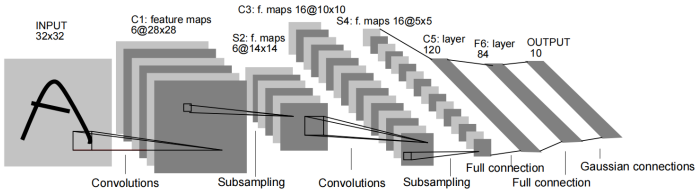


Figure 4: Architecture of LeNet-5[1]

For the LeNet-5 model and the remaining CNNs, the ReLU activation function and Adam optimizer are used. Three LeNet-5 models are trained on our data with learning rate being $1e^{-4}, 1e^{-3}, 1e^{-2},$. Since the LeNet-5 has a relatively shallow structure, 100 epochs are enough to have it converge. From the following table, $1e^{-3}$ is the best learning rate. Hence, we choose it to predict the testing dataset. An accuracy of 65.08% is achieved. Note that the visual results can be seen in Figure 9 of Appendices.

Table 1: LeNet-5: training and validation of last epoch

learning rate	training loss	training accuracy	validation loss	validation accuracy
$1e^{-4}$	0.0091	0.9955	4.7967	0.5667
$1e^{-3}$	0.4718	0.8191	0.8855	0.6542
$1e^{-2}$	0.0147	0.9950	2.4632	0.6507

Although LeNet-5 is easy to train and outperforms traditional KNN, it is old-fashioned with several disadvantages. Firstly, invented in 1998s, it is mainly designed for handling grayscale pictures. However, the images in our scenario are colorful with RGB channels. Secondly, when an overfitting problem occurs, there is no built-in mechanism to avoid that. Therefore, we are motivated to implement more advanced architectures.

4.3 AlexNet

The AlexNet architecture was proposed by Alex Krizhevsky (Krizhevsky et al., 2012). As illustrated in Figure 5, it has five convolutional layers and two fully connected layers. And The final layer is the "softmax."

In the presence of drawbacks of LeNet-5, AlexNet has a much deeper structure and more filters per layer. Therefore, it should be more capable of extracting features. Also, the dropout regularization method is employed to mitigate the overfitting problem. That is, we randomly ignore some neurons before the last

[1]Architecture of LeNet-5,[retrieved from: `https://ieeexplore.ieee.org/abstract/document/726791`,13/05/2022]

two fully connected layers. To reduce the dependency on neighboring neurons, the paper initially let the dropout rate be 0.5. However, such a high rate brings a risk of slowing the convergence rate of the algorithm and even hurting the final result (Wang et al., 2019). Therefore, we adopt a relatively low dropout rate here, which is 0.4 and 0,2, respectively.

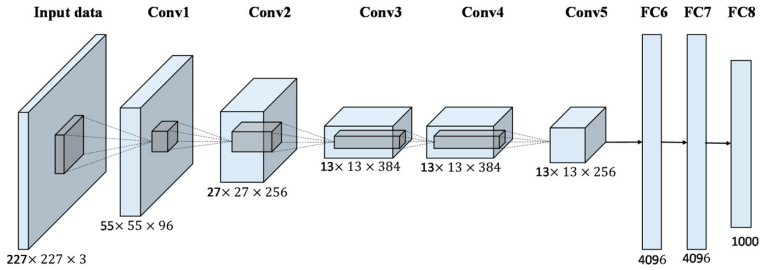


Figure 5: Overview of the AlexNet model[2]

We use a large number of epochs to ensure the convergence of AlexNet, which is 200. Then, the Adam optimizer with learning rate being $1e^{-4}, 1e^{-3}, 1e^{-2}, 1e^{-1}$ are evaluated. Their training and validation performance is displayed in Table 2. The visual comparison can be seen in Figure 10 of Appendices. The best learning rate is $1e^{-4}$, giving a prediction accuracy of 81.25% in testing, which is desirable.

Table 2: AlexNet: training and validation of last epoch

learning rate	training loss	training accuracy	validation loss	validation accuracy
$1e^{-4}$	0.0591	0.9971	0.5979	0.8167
$1e^{-3}$	0.0062	0.9971	1.1325	0.7700
$1e^{-2}$	0.0150	0.9952	1.4870	0.7475
$1e^{-1}$	0.0170	0.9939	1.5523	0.7633

To further enhance prediction accuracy, a natural idea is to simply keep stacking more layers. However, this strategy may not work well. Empirically, since our AlexNet experiment gives almost perfect training accuracy, there is not much room for improvement achieved by doing so. Theoretically, increasing depth usually causes the gradient vanishing or explosion problem(He et al., 2016). Furthermore, as the architecture of AlexNet has already been well-designed, its suitable depth and parameter structure may be ruined if we increase its depth. To tackle these issues, another CNN called ResNet is going to be introduced.

4.4 Deep Residual Network (ResNet)

ResNet is featured by residual mapping. And Figure 6 is a building block showing how it is realized. Also, the corresponding function form is given by equation (1),

[2]Overview of the AlexNet model,[retrieved from: `https://www.jianshu.com/p/a6c248505ac0`,13/05/2022]

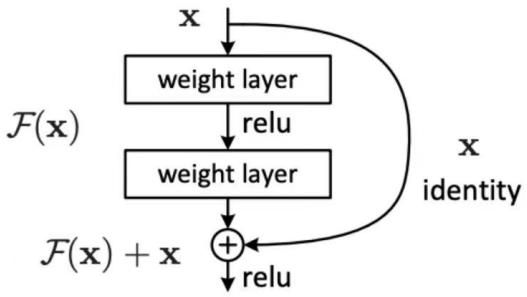


Figure 6: Building Block of a residual network

$$z^0 = x; z^l = \phi(F(W^l, z^{l-1}) + z^{l-1}) \quad l = 1, ..., L, \tag{1}$$

where F represents a few layers of the original networks. An obvious change here is a shortcut connection is added, which is represented by the curve on the right of figure 6. When the original CNN fits the actual underlying function well, the weights of layers in the residue unit would be zero, meaning that the signals would directly pass through the identity and skip the weight layers inside the unit. So, the original CNN structure would not be changed. Thus, the problem of ruining the suitable structure and causing larger training errors mentioned above is solved.

On the other hand, ResNet also has an advantage of accelerating the training process. The reason is that adding skip connections among layers allows the model to update even if some layers have not begun to learn. Moreover, there are other benefits, e.g., better generalization ability, better signal propagation, and better optimization landscape.

As for the architecture, ResNet begins with a convolutional layer followed by a max-pooling layer. Then, the central part is constructed by adding a stack of the residual units to a simple CNN. Inside a residue unit, the first several layers are followed by a ReLU and batch normalization, and the last one is only a batch normalization. The output of the residue unit, which is the summation of the input identity and the output of the inner layers, is activated by ReLU. Lastly, the final part consists of a global avg pooling layer, a fully connected unit with multiple fully connected layers, and a softmax activation.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 7: several ResNet architectures

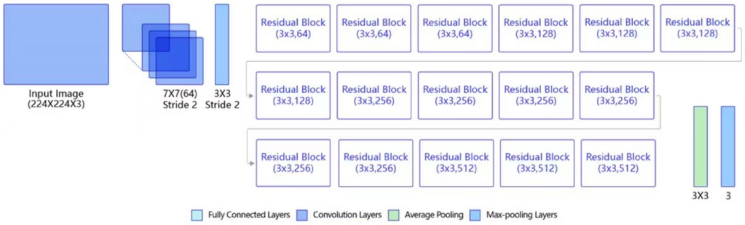


Figure 8: ResNet-34 Architecture

Two ResNet models are implemented, which are ResNet-34 and ResNet-50. Similar to the AlexNet, we optimize the two models with the learning rate being $1e^{-4}$, $1e^{-3}$, $1e^{-2}$, $1e^{-1}$, and the performance on the validation set is analyzed during each epoch. Tables 3 and 4 show the results of the last epoch for ResNet-34 and ResNet-50, from which we observe that the best learning rate is $1e^{-4}$ for both of them. Then, using the corresponding models to predict the testing data, the accuracy is 0.7192 and 0.7022 for ResNet-34 and ResNet-50, respectively.

Table 3: ResNet-50: training and validation of last epoch

learning rate	training loss	training accuracy	validation loss	validation accuracy
$1e^{-4}$	0.6638	0.7539	0.7423	0.7250
$1e^{-3}$	0.5722	0.7846	0.8464	0.6558
$1e^{-2}$	0.5183	0.8055	0.9645	0.6742
$1e^{-1}$	0.6930	0.7284	0.9227	0.6483

5 Limitations

According to the outcomes of this project, two aspects can be improved.

Firstly, the efficiency of executing code can be better. Our experiment is run on Google Colab. The time for KNN and LeNet is trivial due to their simple structure.

Table 4: ResNet-34: training and validation of last epoch

learning rate	training loss	training accuracy	validation loss	validation accuracy
$1e^{-4}$	0.6701	0.7502	0.7079	0.7442
$1e^{-3}$	0.5800	0.7782	0.7795	0.6800
$1e^{-2}$	0.5076	0.8075	0.7642	0.7300
$1e^{-1}$	0.7084	0.7250	0.9242	0.6567

However, when it comes to AlexNet and ResNet, it is evident that the computation resources randomly assigned by Google Colab are insufficient. It takes about 50 minutes to train an AlexNet model and 30 minutes to train a ResNet model. As a result, trying all learning rates causes a heavy burden on our devices and becomes time-consuming.

Secondly, more models are expected to be included. Given the time constraint, we only used three kinds of CNN models. However, apart from them, other excellent variants like VGG and GoogLeNet are worth trialing.

6 Results and Conclusion

In summary, this project used three types of CNN-LeNet, AlexNet, and ResNet to classify four classes of scenery image data scrapped from Intel. The baseline model, KNN, only gives an accuracy of 33.9% on the testing dataset. Hence, by comparison, the performance of CNN models is generally satisfactory after choosing the optimal learning rate. All of them achieved a testing accuracy between 65% $\sim$ 85% as shown in table 5. The AlexNet is the best as its accuracy is the highest, being 81.25%. Therefore, we claim that CNN models are powerful in handling image classification problems.

Table 5: Test Accuracy of All Models

model	KNN	LeNet-5	AlexNet	ResNet-34	ResNet-50
optimal learning rate	NA	$1e^{-3}$	$1e^{-3}$	$1e^{-4}$	$1e^{-4}$
test accuracy	0.3992	0.6507	0.8167	0.7192	0.7250

Future studies can target using more advanced devices to run code and consider other CNN models to see if the accuracy can be further improved.

7 References

References

Ballester, P., & Araujo, R. M. (2016). On the performance of googlenet and alexnet applied to sketches. In *Thirtieth aaai conference on artificial intelligence.*

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Nagi, J., Ducatelle, F., Di Caro, G. A., Cireşan, D., Meier, U., Giusti, A., ... Gambardella, L. M. (2011). Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 ieee international conference on signal and image processing applications (icsipa)* (p. 342-347). doi: 10.1109/ICSIPA.2011.6144164

Targ, S., Almeida, D., & Lyman, K. (2016). Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*.

Wang, S., Zhou, T., & Bilmes, J. (2019). Jumpout: Improved dropout for deep neural networks with relus. In *International conference on machine learning* (pp. 6668–6676).

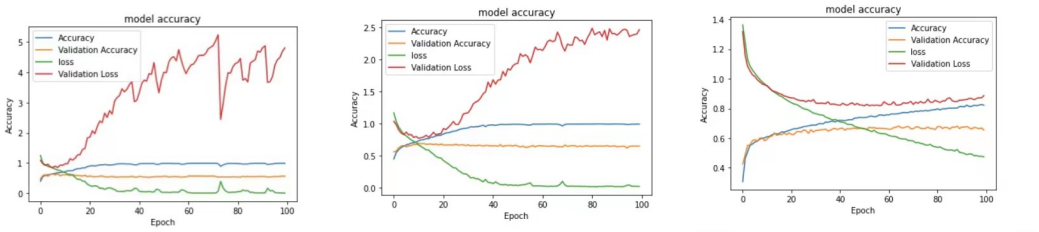
Xin, M., & Wang, Y. (2019). Research on image classification model based on deep convolution neural network. *EURASIP Journal on Image and Video Processing*, *2019*(1), 1–11.

Yann, L., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Yu, W., Yang, K., Bai, Y., Xiao, T., Yao, H., & Rui, Y. (2016). Visualizing and comparing alexnet and vgg using deconvolutional layers. In *Proceedings of the 33 rd international conference on machine learning.*

8 Appendices

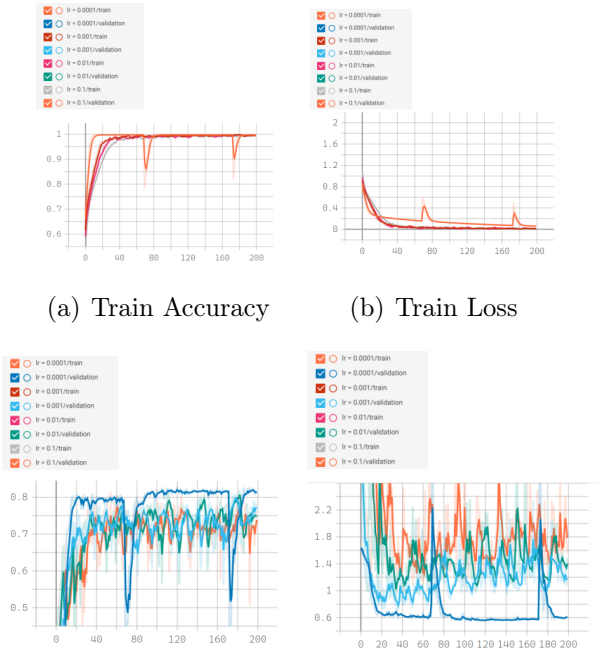
1. LeNet



(a) Learning rate=0.01 (b) Learning rate=0.001 (c) Learning rate=0.0001

Figure 9: LeNet

2. AlexNet



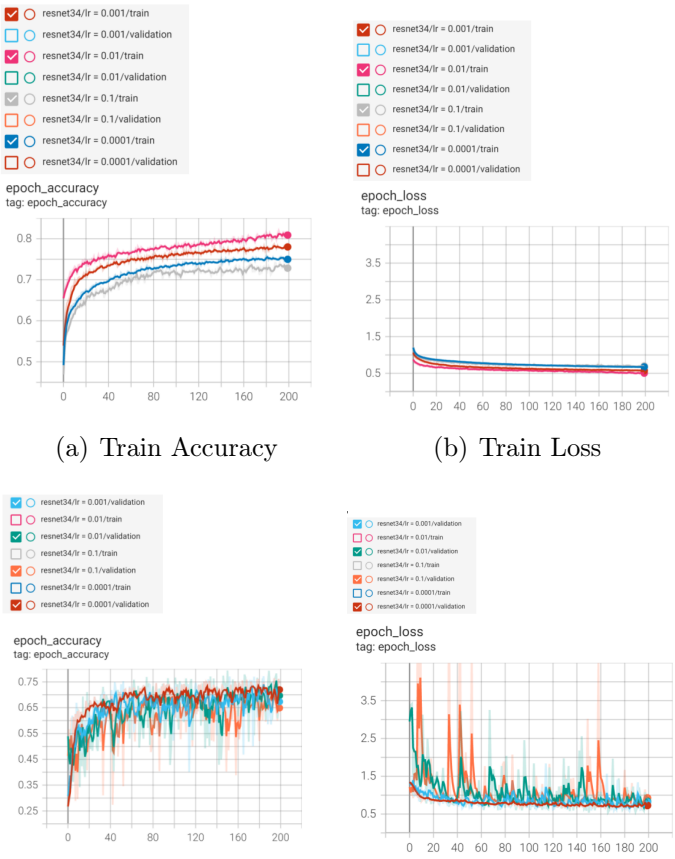
(a) Train Accuracy (b) Train Loss

(c) Validation Accuracy (d) Validation Loss

Figure 10: AlexNet

3. ResNet

- ResNet-34

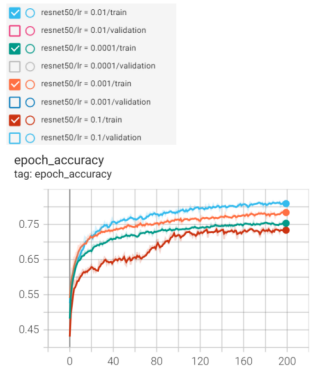


(a) Train Accuracy (b) Train Loss

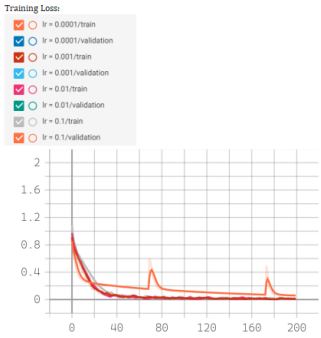
(c) Validation Accuracy (d) Validation Loss

Figure 11: ResNet-34

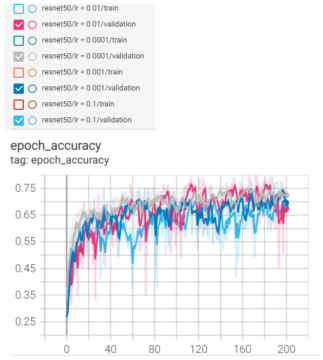
- ResNet-50



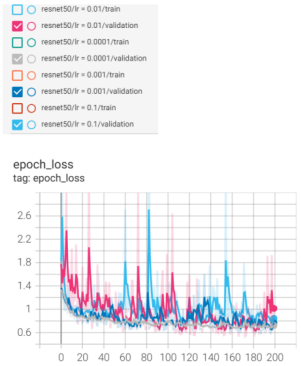
(a) Train Accuracy



(b) Train Loss



(c) Validation Accuracy



(d) Validation Loss

Figure 12: ResNet-50