CIFAR100 Classification Models: ResNet18, MobilNetV2, VGG11, and DenseNet

Authors: Alex Kramer[1], Shane Hussey[1]

[1] Northeastern University, Khorey College of Computer Science

Course: DS5220 Supervised Machine Learning | Spring 2024

Table of Contents

1. Introduction

This project explores the performance of four prominent CNN architecture - (1) ResNet18[4], (2) VGG11[3], (3) DenseNet[1] and (4) MobleNetV2[2] - on the CIFAR100 dataset[7], consisting of 60,000 32x32 color images (3 channels) with 100 different classes (600 images per class) (Table 1). By implementing and comparing these models, we aim to understand the models strengths and limitations in terms of accuracy, efficiency and ability to generalize. This report gives an overview of the models, details the setup, methodology, and findings from our experiments, shedding light on which models are not effective for specific types of image classification.

In this project, we explored several convolutional neural networks (CNNS) to tackle image classification for the CIFAR-100 dataset. Each model has its own distinct architectural innovations designed to optimize performance and efficiency. The models we looked at included ResNet18, VGG11, DenseNet, and MobileNetV2. We chose these models based on their effectiveness in handling complex image data. Below you will find a brief description of each model.

Figure 1. CIFAR-100 Image Classification Model Performance Through Time.
Taken Directly from Source - [CIFAR-100 Benchmark (Image Classification) | Papers With Code]

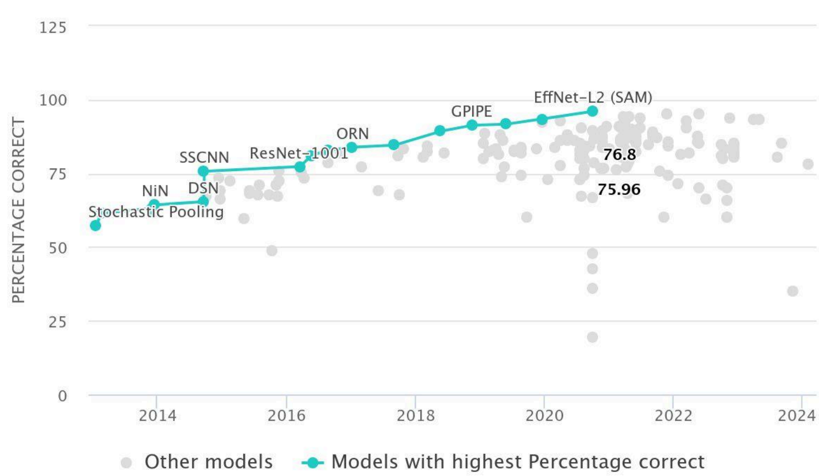


Table 1. CIFAR 100 Dataset Class Overview

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

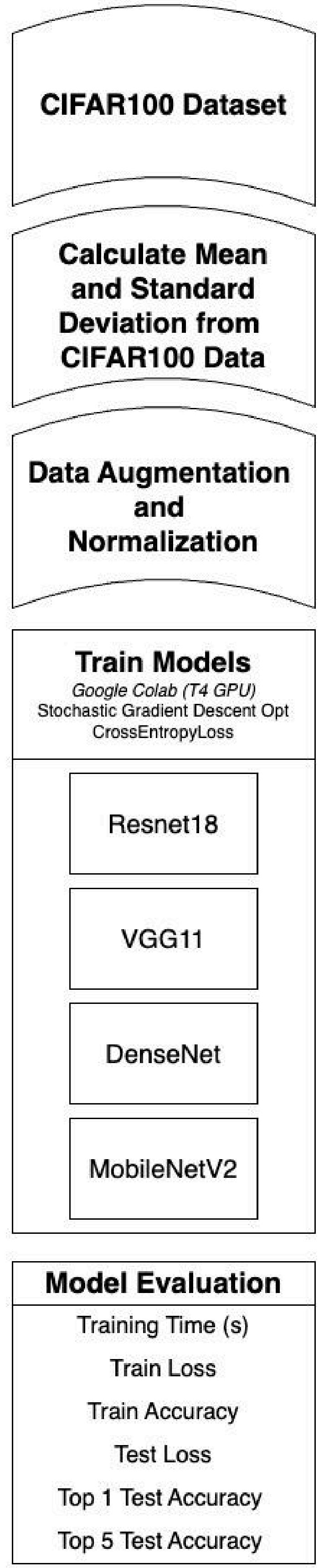
2. Methods

2.1 Data Acquisition and Preprocessing

We use the CIFAR100 dataset[7] consisting of 60,000 32x32 color images (3 channels) with 100 different classes (600 images per class) to assess performance on four different model configurations from torchvision.models: (1) ResNet18, (2) VGG11, (3) DenseNet and (4) MobleNetV2. This dataset was downloaded from https://www.cs.toronto.edu/~kriz/cifar.html and loaded into Google Drive for use with Google Colab IDE.

We initially took two approaches to image data normalization[8]: (1) use mean = [0.5, 0.5, 0.5], std = [0.5, 0.5, 0.5], and (2) calculation of the mean and std from the CIFAR100 dataset, mean = [0.507, 0.486, 0.440] [0.267, 0.256, 0.276]. When doing initial training and testing with ResNet18, we saw better test performance with the second approach, and decided to proceed with that for the remainder of training *[data not shown]*.

To reduce overfitting models to our training sample data we utilized two data augmentation approaches in which each image is would be transformed on-the-fly during model training: (1) RandomCrop(32, padding=4), and (2) RandomHorizontalFlip(). RandomCrop first pads the original images by 4 pixels on each side, changing the size from 32x32 to 40x40. After padding, it randomly crops a new 32x32 image from this padded image. RandomHorizontalFlip randomly flips the training image horizontally (mirroring along its vertical axis) with a 50% probability. These



transformations do not affect the total number of images in the dataset and each epoch will see slightly different versions of the sample image.

2.2 Model Implementation, Training and Evaluation

2.2.1 Model Selection

Four neural network model architectures from torchvision.models were selected for this image classification task: (1) ResNet18[4], (2) VGG11[3], (3) DenseNet[1] and (4) MobleNetV2[2]. These models were selected based on ease of implementation with *torchvision.models* and distinct characteristics that make them suitable for particular aspects of image classification.

ResNet18 is a variation of the Residual Network architecture that includes 18 layers. Residual Networks are known for their ‘residual connections’ that allow the input of one layer to skip some layers and be added to a later layer’s output. This can help combat the vanishing gradient problem by facilitating deeper networks.

MobileNetV2 utilizes a module called the inverted residual with linear bottleneck which uses depth wise separable convolutions which separates the convolution into a depthwise and a 1x1 pointwise convolution. The bottleneck that captures the relevant features, and utilizes shortcut connections which helps the flowing gradients directly through the network and cut down on computational costs..

VGG11 is a configuration of the Visual Geometry Group (VGG) model that consists of 11 layers (8 convolutional layers and 3 fully connected layers). VGG models are able to reduce the volume size handled by max pooling by using 3x3 convolutional layers stacked on top of eachother in increasing depths. VGG11 is able to understand complex features from images, but has high computational costs and can be slower to train.

DenseNet161 is a densely connected convolutional network (CNN) which is known by its dense connectivity pattern. For DenseNet161, each layer connects to every other layer in a feed-forward fashion. For each layer, the feature-maps of all preceding layers are used as inputs into all subsequent layers. This allows the model to alleviate the vanishing gradient problem, strengthen feature propagation, encourage feature reuse and reduce the number of parameters.

2.2.2 Model Training Optimization

The optimization function and loss function selected for all four neural network models was stochastic gradient descent (torch.optim.SGD) and cross entropy loss (torch.nn.CrossEntropyLoss()) respectively. Initial evaluation of ResNet18's stochastic gradient descent's hyperparameters caused us to converge on: learning_rate = 0.01, momentum = 0.9, and weight_decay = 0.005 [*data not shown*]. We also tested modified base models for resnet18 and densenet161 to include dropout [6] to preve

2.2.3 Model Evaluation of Functional and Analytical Performance

To evaluate the analytical and functional performance of our trained models, we employ various methods to calculate the following: Epochs Run, Per Epoch Run Time (s), Total Training Time, Max Training Accuracy, Max Top-1 Accuracy, Max Top-5 Accuracy, Epochs to Max (time to convergence). We utilize these metrics to compare optimal performance as a function of runtime between our models.

3. Results

3.1 Analytical Performance of (1) ResNet18[4], (2) VGG11[3], (3) DenseNet[1] and (4) MobleNetV2[2]

We created plots of training cross entropy loss and test cross entropy loss in addition to training accuracy, top-1 accuracy, and top-5 accuracy were made for each of the following models: (1) resnet18 (2) resnet18 with dropout = 0.5 (3) mobilenet_v2 (4) VGG11 (5) DenseNet161 (6) DenseNet161 with dropout = 0.5.

Figure 2. ResNet18: Analytical Performance of standard torchvision.model.resnet18 model without dropout through 70 epochs. 20240425_202833_logs

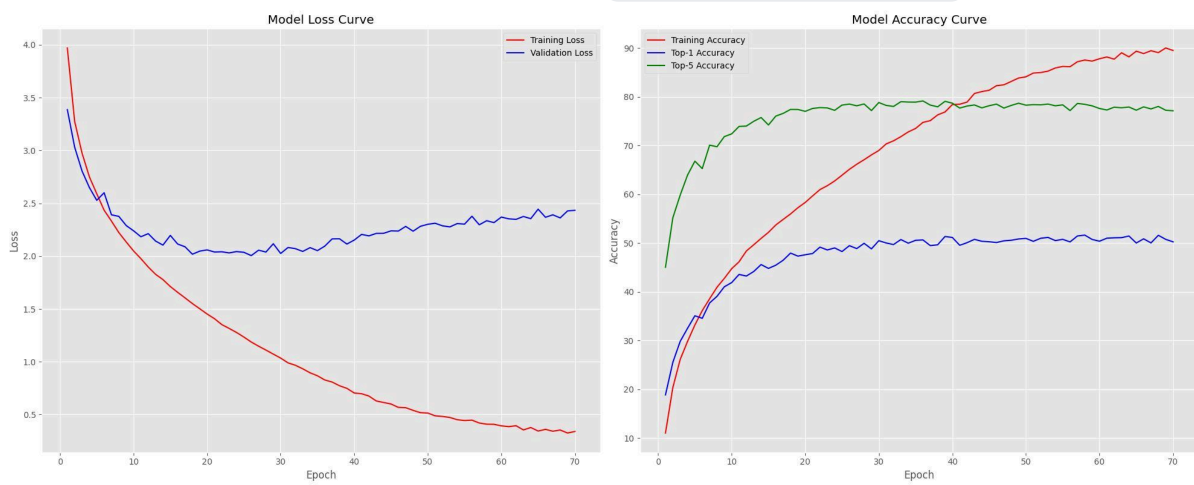


Figure 3. ResNet18: Analytical Performance of modified torchvision.model.resnet18 model with dropout = 0.5 through 70 epochs. 🗀 20240426_015532_logs

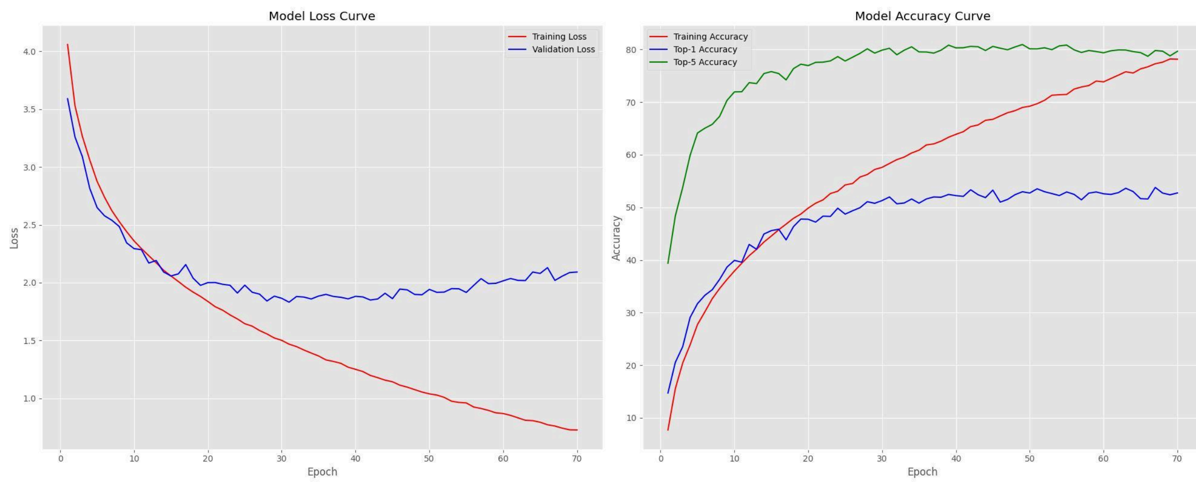


Figure 4. MobileNetV2: Analytical Performance of torchvision.model.mobilenet_v2 with standard dropout through 70 epochs. 🗀 20240425_215918_logs

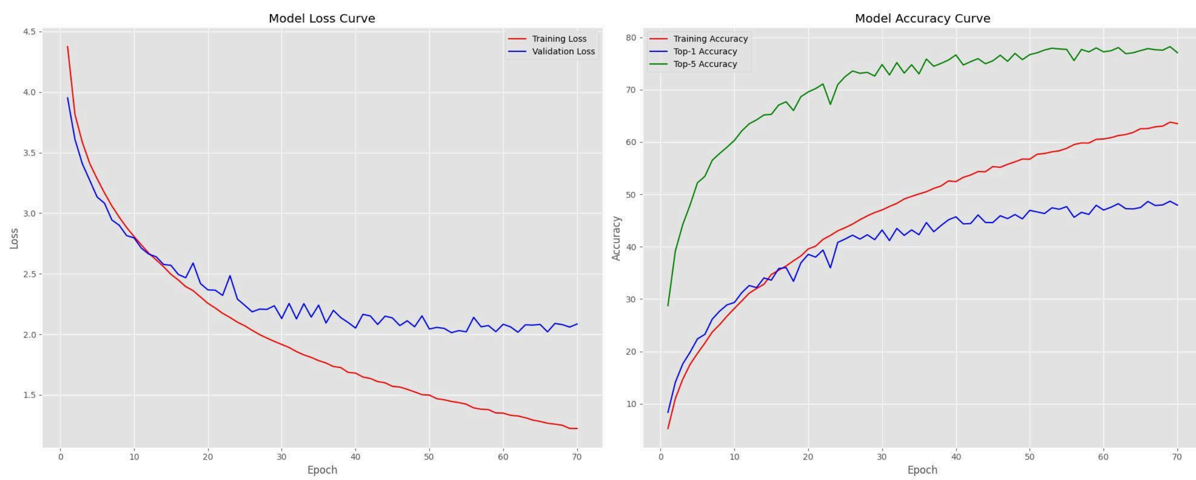


Figure 5. VGG11: Analytical Performance of torchvision.model.vgg11 with standard dropout through 40 epochs. 🗀 20240425_232158_logs

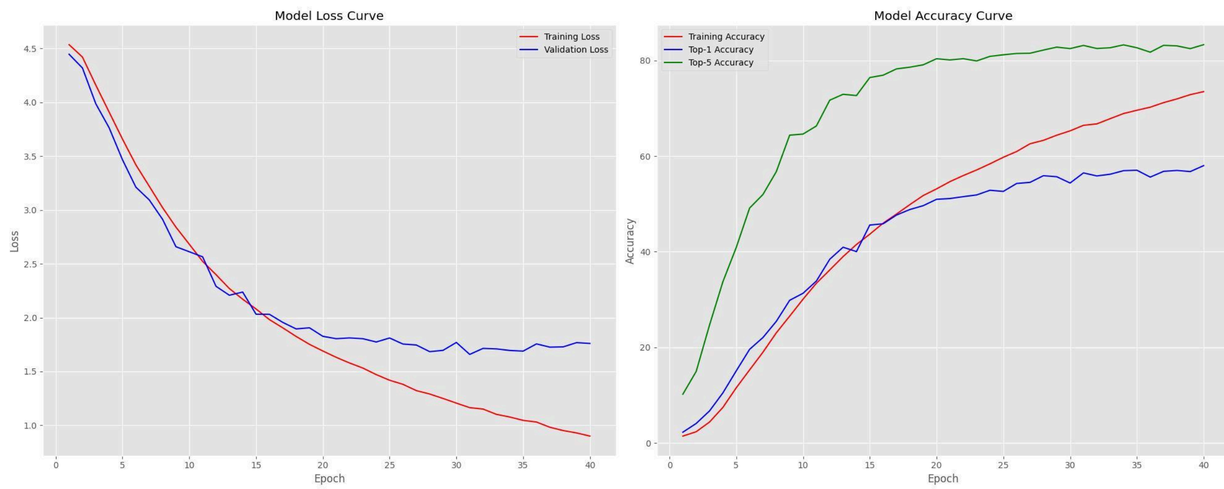


Figure 6. DenseNet: Analytical Performance of torchvision.model.densenet161 without dropout through 40 epochs. 🗀 20240425_222653_logs

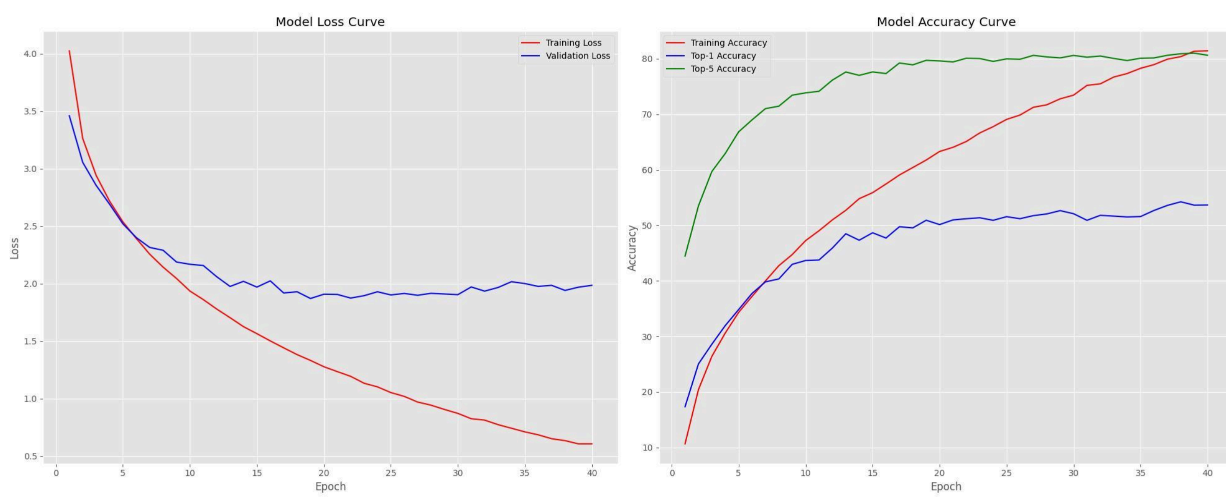
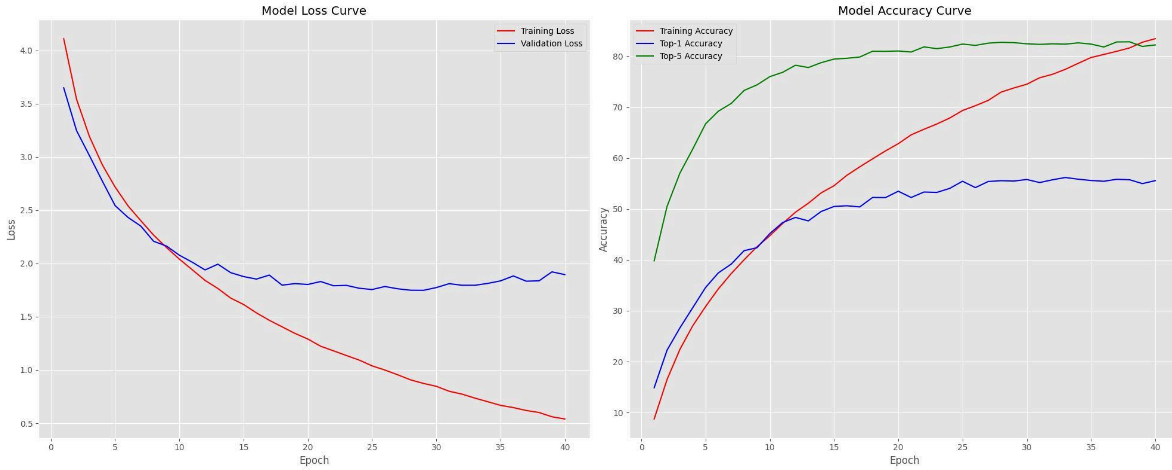


Figure 7. DenseNet: Analytical Performance of torchvision.model.densenet161 with dropout = 0.5 through 40 epochs. 20240426_194838_logs



3.2 Comparative Analysis of Model Performance

Table 2. Analytical And Functional Performance Characteristics for Image Classification Models.

	Resnet18		**MobileNetV2**	**VGG11**	**DenseNet161**	
Dropout Rate	0.5	0.0	0.5; 0.5	0.5; 0.5	0.5	0.0
Epochs Run	70	70	70	40	40	40
Epochs to Convergence	~40	~30	~70	~40	~30	~30
Per Epoch Run Time (s)	34.46 std=0.94	27.71 std=0.72	34.46 std=0.94	145 std=1.9	69.93 std=1.02	56.18 std=0.98
Time to Convergence (s)	~1378.4	~831.3	~2412.2	~5800	~2097.9	~1685.4
Total Training Time	1981.23	1940.06	2412.45	5896.34	2797.33	2247.58
Max Training Accuracy	78.19	90.01	63.78	76.6	83.44	81.43
Min Training Loss	0.7275	0.3254	1.22	0.779	0.5398	0.6077
Max Top-1 Test Accuracy	53.78	51.62	48.68	57.4	56.17	54.24
Max Top-5 Test Accuracy	80.94	79.14	78.21	83.6	82.83	81.02
Min Top-1 Loss	3.5891	2.0048	3.95	1.663	1.7476	1.871

4. Discussion

Our top performing model of the four tested from the Top-1 and Top-5 Accuracy perspective was VGG11 but the training process took ~7 times longer to train for a roughly 6% gain in Top-1 Accuracy (51.62 -> 57.4), and ~4% gain in Top-5 Accuracy (79.14 -> 83.6) when compared to ResNet18 with no dropout.

Adding dropout layers into ResNet18 appeared to reduce overfitting as seen in Figure 2 and Figure 3 where the loss function continued to converge an asymptote after 70 epochs

Opportunities for improvement which we did not include in our current methodology include due to time constraints:

(1) Early stopping - use this method to prevent overfitting which involves tracking our model performance metrics on a validation set at each epoch and stopping training when the performance no longer improves.

(2) k-fold cross-validation - training our models on different subsets of the data could have provided more insights into how quickly our models converge on average to help estimate the required number of epochs with more precision.

(3) More extensive hyperparameter tuning - we could use GridSearch to tune and further optimize our parameters for stochastic gradient descent and dropout. Due to training time for each model this was not feasible at this time.

References

1. Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K.Q., 2017. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4700-4708). arXiv:1608.06993
2. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520). arXiv:1801.04381
3. 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). arXiv:1512.03385
4. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
5. Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013, May). On the importance of initialization and momentum in deep learning. In International conference on machine learning (pp. 1139-1147). PMLR.
6. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.
7. Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images.
8. PyTorch Fourms - https://discuss.pytorch.org/t/understanding-transform-normalize/21730
9. https://paperswithcode.com/sota/image-classification-on-cifar-100