Assignment 01 – CIFAR100 Image Classification with Keras Report

Tina Tran

Introduction

The image classification problem is the task of assigning a label to an input image from a fixed set of categories. This is a well-documented Computer Vision task with a large variety of practical applications such as training autonomous driving systems, manufacturing, quality control, sorting, etc. In recent years, the use of deep learning has revolutionized the image classification problem and return high accuracy results for simple problems with a low number of categories.

In the previous Assignment 0, we created a simple convolutional neural network (CNN) in Keras to solve the image classification problem on the Modified National Institute of Standards and Technology (MNIST) [LeCun et al.] dataset of handwritten digits from 0 through 9. This model achieved a 92% accuracy on the testing set after 3 epochs.

For this assignment, I tested the image classification accuracy between two different model architectures on the Canadian Institute for Advanced Research, 100 classes (CIFAR100) [Krizhevsky et al.] dataset. The first architecture was the model used in Assignment 0 for the MNIST dataset while the second architecture utilizes main batch normalization (MBN) [Tung].

CIFAR100 consists of 60,000 32x32 color images. There are 100 "fine" label classes grouped into 20 "coarse" label superclasses with each of the 100 classes having 600 images each.

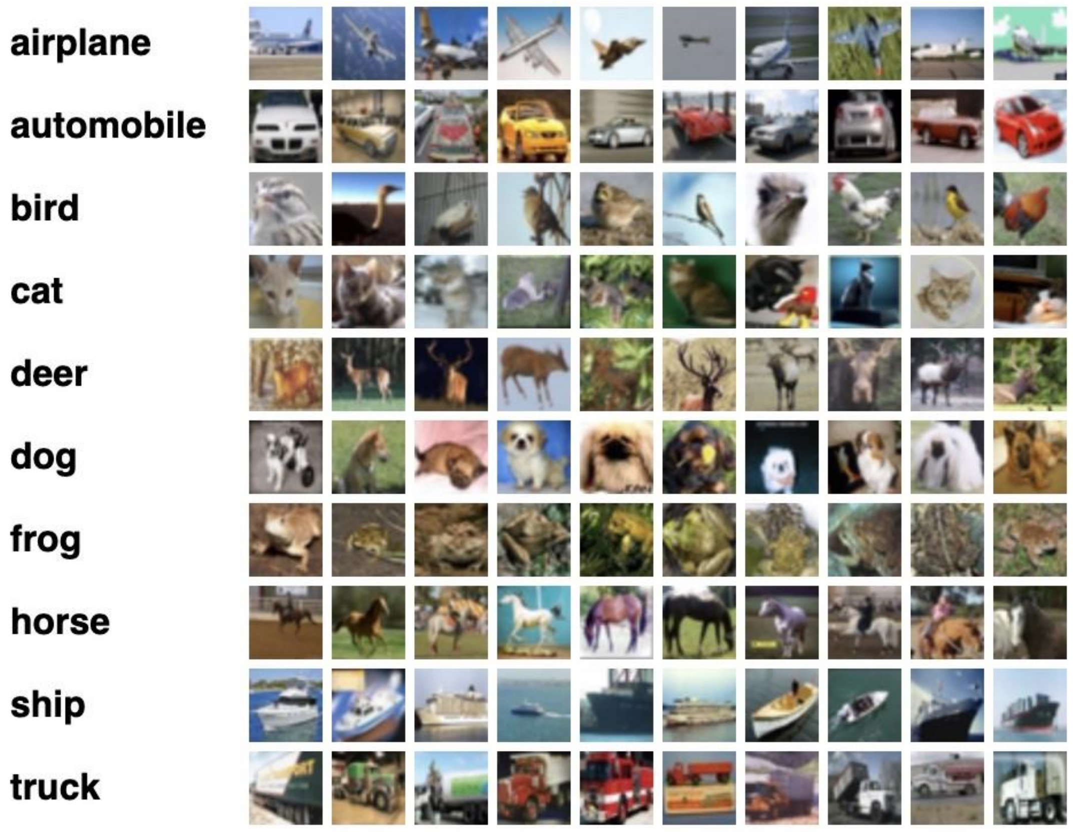


Figure 1 - example images from CIFAR100

Method

The MNIST model is a benchmark for deep learning. The images are preprocessed by scaling the values between zero and one due to the images being black and white. The labels are one-hot encoded. The MNIST model is sequential model composed of the following layers:

Layer (type)	Output Shape	Number of Parameters
conv2d (Conv2D)	(None, 26, 26, 32)	320
conv2d_1 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d (MaxPooling2D)	(None, 12, 12, 64)	0
dropout (Dropout)	(None, 12, 12, 64)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 128)	1179776
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

The MBN method contains many layers with its namesake. These layers normalize the input data, reducing the number of steps to reach the minimum during gradient descent by promoting smaller weight updates. Increasing the number of filters in convolution layers and convolution layers themselves capture the more complex patterns to be discovered to differentiate the 100 classes. The strategic placement of dropout and data augmentation create an "hourglass" model that magnifies the significance of features essential for accurate classification and minimizes otherwise. In addition, this model is simple enough to run on CPU. The MBN model is a sequential model composed of the following layers:

Layer (type)	Output Shape	Number of Parameters
conv2d (Conv2D)	(None, 32, 32, 256)	7168
batch_normalization (BatchNormalization)	(None, 32, 32, 256)	1024
activation (Activation)	(None, 32, 32, 256)	0
conv2d_1 (Conv2D)	(None, 32, 32, 256)	590080
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 256)	1024
activation_1 (Activation)	(None, 32, 32, 256)	0
max_pooling2d (MaxPooling2D)	(None, 16, 16, 256)	0
dropout (Dropout)	(None, 16, 16, 256)	0
conv2d_2 (Conv2D)	(None, 16, 16, 512)	1180160
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 512)	2048
activation_2 (Activation)	(None, 16, 16, 512)	0
conv2D_3 (Conv2D)	(None, 16, 16, 512)	2359808
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 512)	2048
activation_3 (Activation)	(None, 16, 16, 512)	0
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 512)	0
dropout_1 (Dropout)	(None, 8, 8, 512)	0
conv2d_4 (Conv2D)	(None, 8, 8, 512)	2359808
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 512)	2048

activation_4 (Activation)	(None, 8, 8, 512)	0
conv2d_5 (Conv2D)	(None, 8, 8, 512)	2359808
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 512)	2048
activation_5 (Activation)	(None, 8, 8, 512)	0
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 512)	0
dropout_2 (Dropout)	(None, 4, 4, 512)	0
conv2d_6 (Conv2D)	(None, 4, 4, 512)	2359808
batch_normalization_6 (BatchNormalization)	(None, 4, 4, 512)	2048
activation_6 (Activation)	(None, 4, 4, 512)	0
conv2d_7 (Conv2D)	(None, 4, 4, 512)	2359808
batch_normalization_7 (BatchNormalization)	(None, 4, 4, 512)	2048
activation_7 (Activation)	(None, 4, 4, 512)	0
max_pooling2d_3 (MaxPooling2D)	(None, 2, 2, 512)	0
dropout_3 (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 1024)	2098176
activation_8 (Activation)	(None, 1024)	0
dropout_4 (Dropout)	(None, 1024)	0
batch_normalization_8 (BatchNormalization)	(None, 1024)	4096
dense_1 (Dense)	(None, 100)	102500

Experiment

Both models were run on an Anaconda environment with Python version 3.7.1, Tensorflow version 2.3.0, and Keras version 2.4.3 on CPU. Although the training could have been faster since I have access to a NVIDIA GeForce RTX 3050 GPU, unfortunately I was unable to correctly setup this due to the unresolvable issue of the myriad of dependencies with the NVIDIA libraries (cudatoolkit, cuDNN, CUDA) necessitating newer versions to function and tensorflow-gpu versions greater than 2.1 being unsupported on Windows. A batch size of 256 was used and training lasted for 15 epochs. Training accuracy and loss, validation accuracy and loss, and test accuracy and loss will be recorded for both models. We will also look at examples of the images both models were able to correctly identify and incorrectly identify.

Results

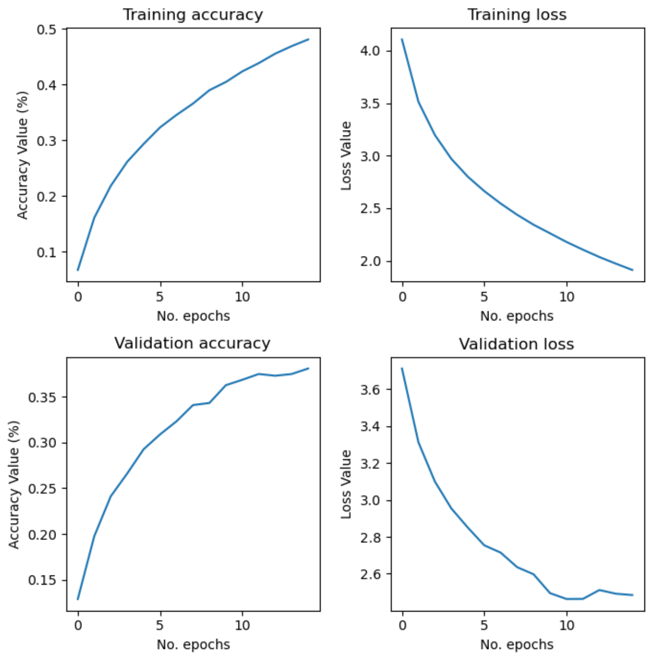


Figure 2 – MNIST training accuracy and loss (top) validation accuracy and loss (bottom)

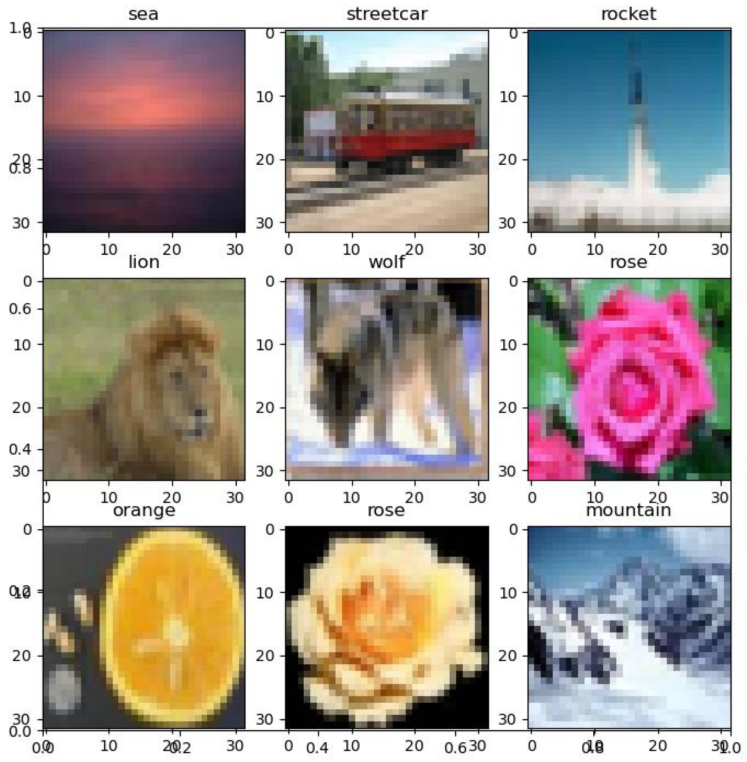
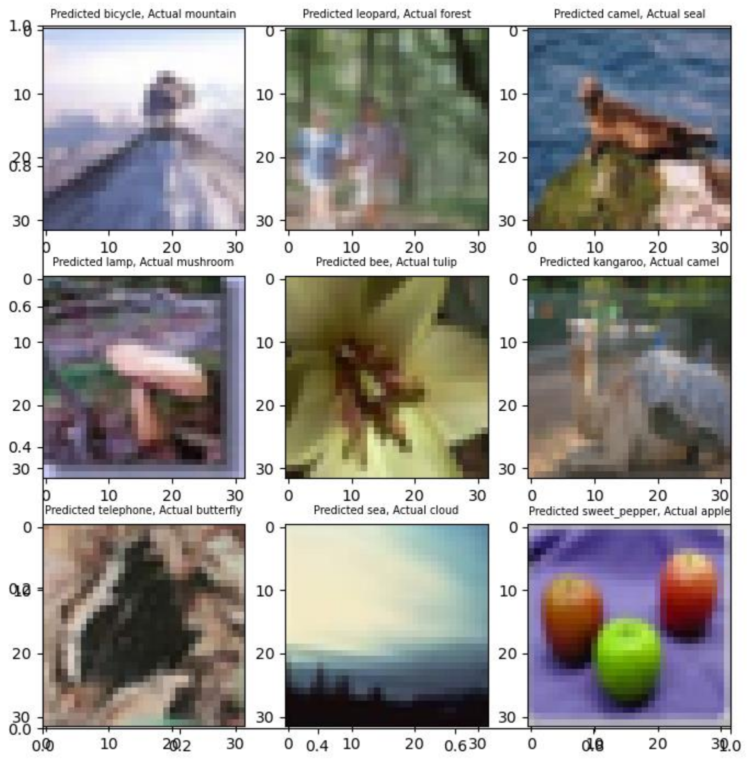


Figure 3 – MNIST correctly identified labels



Predicted bicycle, Actual mountain
Predicted leopard, Actual forest
Predicted camel, Actual seal
Predicted lamp, Actual mushroom
Predicted bee, Actual tulip
Predicted kangaroo, Actual camel
Predicted telephone, Actual butterfly
Predicted sea, Actual cloud
Predicted sweet_pepper, Actual apple

Figure 4 - MNIST incorrectly identified labels

The MNIST model achieved a test loss of 2.48 and test accuracy of 38.1% after training for about 30 minutes.

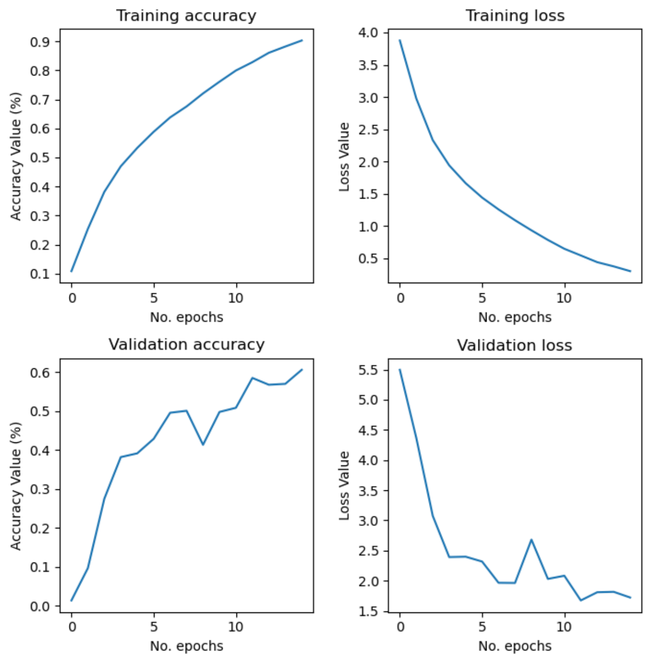


Figure 5 - MBN training accuracy and loss (top) validation accuracy and loss (bottom)

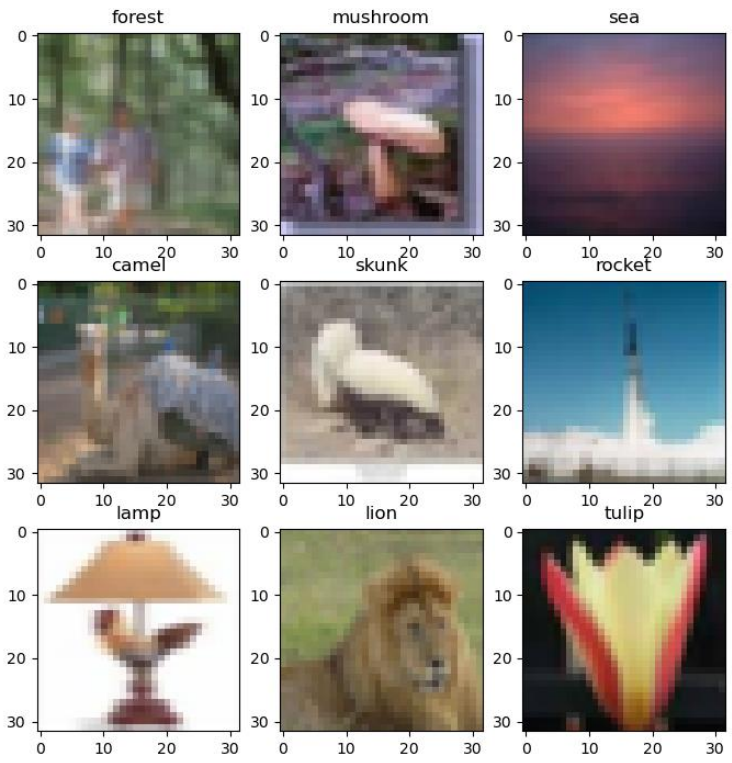


Figure 6 - MBN correctly identified labels

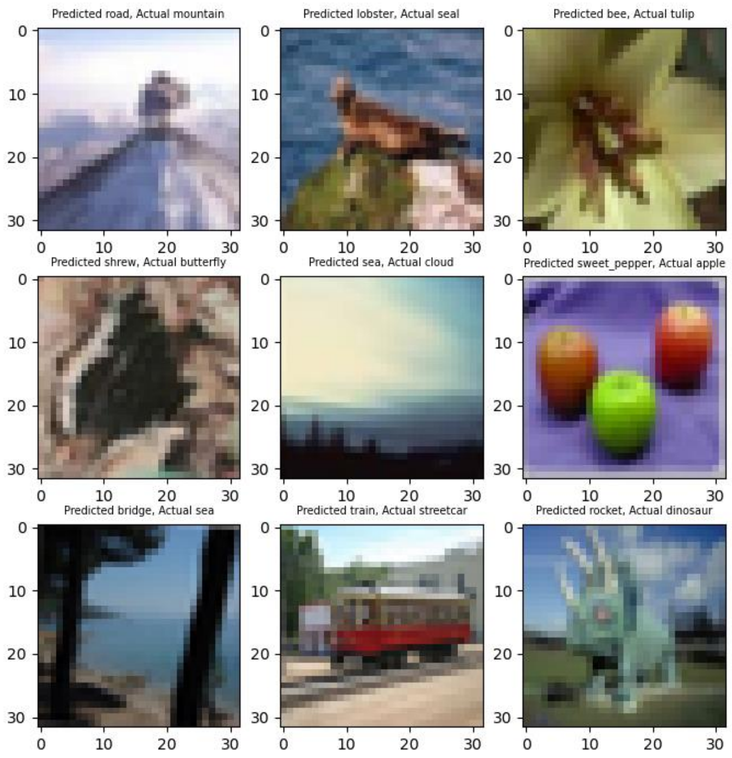

Predicted road, Actual mountain
Predicted lobster, Actual seal
Predicted bee, Actual tulip
Predicted shrew, Actual butterfly
Predicted sea, Actual cloud
Predicted sweet_pepper, Actual apple
Predicted bridge, Actual sea
Predicted train, Actual streetcar
Predicted rocket, Actual dinosaur

Figure 7 - MBN incorrectly identified labels

The MBN model achieved a test loss of 1.72 and a test accuracy of 61% after training for about 8 hours.

Analysis/Conclusion

Both models had similar images that were correctly and incorrectly identified. One significant difference would be the image labeled “forest” with two people walking. The MNIST model incorrectly identified this image as “leopard” likely correlating the feline animal with the green color of the forest and mistaking the people as a leopard due to the light color contrasting the forest and resembling the color of a leopard’s fur. The MBN model was able to correctly identify the image as forest. A similar phenomenon can be seen in the “camel” labeled image incorrectly identified as “kangaroo”. The increased number of layers in the MBN model along with the magnifying of inherent pixels to each class was likely the reason it was able to correctly identify these images over the MNIST model and achieve a much higher test accuracy, a 22.9% increase from the MNIST model.

According to paperswithcode.com, the best percentage accuracies that convolutional neural networks have achieved on CIFAR100 is a 70.18% with the Very Deep Convolutional Networks for Large-Scale Image Recognition (VGG-16) [Simonyan and Zisserman] and 67.7% with the model Deep Convolutional Neural Networks as Generic Feature Extractors (DCNN+GFE) [Hertel et al.]. At first, I attempted to replicate the DCNN+GFE model instead of using the Main Batch Normalization method, however I ran into an issue with the resizing. DCNN+GFE resizes the 32x32 images into 227x227 and my machine did not have enough system memory to allocate. VGC-16 also requires this resizing thus implanting this model was also not possible. I found and decided to use the MBN method since it kept the image size small.

Other models that have been used include ResNet, Transformer, EfficientNet, and a pre-trained CLIP. These models have achieved much better performance on CIFAR100. It is apparent from the experiment that it is difficult for CNNs to be able to extract the correct features from a small pixel size. Further testing is needed on larger models utilizing the resizing method to provide a greater number of trainable parameters.

References

L. Hertel, E. Barth, T. Kaster, T. Martinetz. (2017). *Deep Convolutional Neural Networks as Generic Feature Extractors*.

A. Krizhevsky. (2009). *Learning Multiple Layers of Features from Tiny Images*.

Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. (1998). *Gradient-based learning applied to document recognition*. Proceedings of the IEEE 1998.

K. Simonyan, A. Zisserman. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*.

L. Tung. (2021). *A Simple CNN model in Keras without transfer learning achieves 67% accuracy on test set of CIFAR-100*. https://github.com/LeoTungAnh/CNN-CIFAR-100

Image Classification on CIFAR-100. https://paperswithcode.com/sota/image-classification-on-cifar-100