COMP SCI 7318 - Deep Learning Fundamentals: CNNs for Image Classification

Viral Shailesh Panchal
Student ID: 1920193
a1920193@adelaide.edu.au

1. Introduction

One kind of artificial intelligence is a CNN, or convolutional neural network. Its primary function is picture analysis. It searches for patterns by dividing the image into tiny parts. It can identify elements in the image, such as edges or forms, thanks to these patterns. CNNs are employed in a wide range of applications, including medical image processing, self-driving cars, and facial recognition. For image categorization, there are several deep learning models available, including AlexNet, Imagenet, GoogLeNet, ResNet, LeNet, VGG16, CoCa, YOLO, and ViT. We will employ a few of these models to see which one performs the best when compared to CNNs.In picture classification, bias and data imbalance are major obstacles. Model performance can be significantly impacted by biased or subpar training data. To increase dataset quality, it is crucial to keep an eye on image relevance, mislabeled occurrences, and data amount. Convolutional neural networks (CNNs) might perform considerably worse when mislabeled or unclear. Lighting, perspective, scale, and occlusions all affect images in real-world situations. Accurate recognition and classification are difficult due to several characteristics that affect picture classification models.The CIFAR (Canadian Institute for Advanced Research) dataset will be used for this assessment. There are 60,000 images in 10 classes, with 6,000 images in each class. A total of 10,000 test photos and 50,000 training images are included. Every single one of these 32×32 pixel images are coloured. Figures 1 and 2 below shows that accuracy increases, and loss decreases with each epoch, increasing the model's overall accuracy to 69.96

2. Method Description

In this project, we will classify images using CNN,LeNet, EfficientNet, ResNet18, and ResNet50 models. We will assess each model's performance and evaluate which model has the best accuracy. For each model, we are maintaining the same epoch, learning rate, and batch size. Batch size = 50, learning rate = 0.002, and epoch = 10. The seed has been set to 100. We are switching the process from CPU to GPU in order to improve processing and performance. All of the models that we will use in this assignment are implied by the same training code. The same is true for our model's analysis. Prior to splitting and performing transformations, such normalization, for both the train and test datasets, we first execute data pre-processing. For the transformation, we first turn the images horizontally at random with a 0.5 probability. In order to differentiate a flipped object from an original one. Trims a 32x32 portion of the image at random and adds 4 pixels of padding around the edge. Applies random, maximum-change changes to hue, saturation, contrast, and brightness. By simulating changes in hue and brightness, this improves the model's ability to generalize under various circumstances. The next step makes the tensor normalized so that the mean and standard deviation of each colour channel (red, green, and blue) are 0.5. By standardizing input values, this scales each pixel value to the interval [-1, 1], This can improve model performance and stability during testing and training. We begin by splitting the data based on the CIFAR train and test datasets after the transformation is complete. We also kept num-workers at 4 to expedite the data loading.

3. MODElS

3.1. CNN

In order to begin our model definition, we define our CNN model and construct a CNN class that inherits nn.Module. Using all the required parameters and filters, we create a convolutional layer and do batch normalization. Next, we construct a fully connected layer with all required filters and parameters. Next, we establish our learning rate scheduler. After everything is finished, we train our model and analyse it to determine its accuracy. For improved processing and performance, we load the data, transfer the model to the GPU, and shift the train and test data batches to the GPU. Our CNN model is then created, trained, evaluated for correctness, and the results are saved using code

that we write. In our scenario, the function trains the CNN model using training data across ten epochs. We then create an accuracylist to keep the accuracy for each epoch. Next, we generate an accuracy list to hold the model's overall accuracy. The model is thereafter transferred to CPU for additional processing and storage. We generate two graphs with loss vs. epoch and accuracy vs. epoch after completing all the epochs and obtaining the individual accuracy and overall accuracy. As

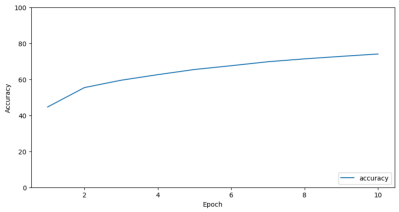

Figure 1. Figure 1: Epoch VS loss Line graph

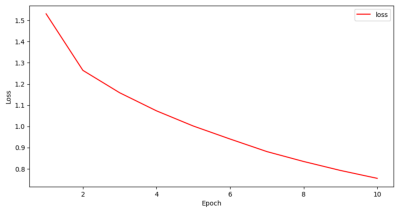

Figure 2. Epoch VS Accuracy Line graph

The above Figure 1 and 2 shows that the with every Epoch the accuracy increase and loss decrease

3.2. LeNet

LeNet is a multi-layer architecture that progressively collects and condenses information from input images. There are two fully connected layers after the input layer, first convolutional layer, first pooling layer, second convolutional layer, and second pooling layer. To improve performance, we start by building the LeNet model and then transfer it to a GPU. Using the train data for ten epochs, we invoke a function that trains the LeNet model. Next, we save each epoch's accuracy. Our LeNet model is then adjusted to run on the Test dataset. After that, the outcome is accurately stored. After the entire epoch is finished, we draw an accuracy versus epoch graph. The overall accuracy of this model is 68.06

3.3. EfficientNet

The foundation of convolutional neural networks, such as EfficientNet, is "compound scaling." This concept addresses the long-standing trade-off between model size, accuracy, and computational efficiency. The three essential parts of a neural network—resolution, depth, and

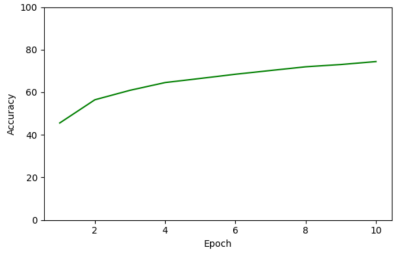

Figure 3. LeNet Model's Epoch VS loss Line graph

width—are intended to be scaled using compound scaling. We first construct a custom class that inherits nn.Module in order to implement this model. The EfficientNet-B0 model architecture is then initialized. gives back the number of input features for the last fully connected (classification) layer of the EfficientNet model. replaces EfficientNet's first fully connected layer-fc with a new classification layer designed for a 10-class classification challenge. Switch to GPU mode. The model is first trained using the train dataset, and the accuracy results for each epoch are saved in a list. The accuracy is then assessed on the test dataset, and the total accuracy is recorded in the accuracy variable. The dictionary then records the overall accuracy. The overall accuracy of this model is 60.80

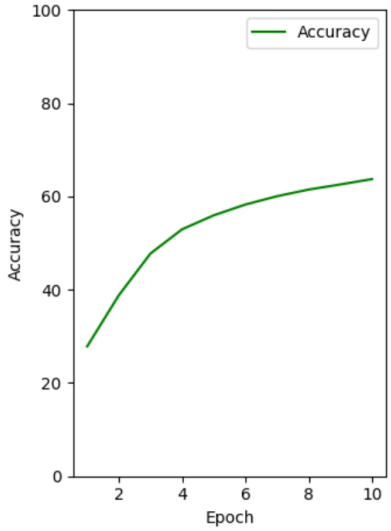

Figure 4. EfficientNet Epoch VS Accuracy Line graph

3.4. ResNet18

ResNet-18 is an 18-layer convolutional neural network. The network that has been trained for classifying photos into 1000 object categories, including several animals, keyboards, mice, and pencils. We start with creating a custom class RestNet18 which inherits nn.Module. A pretrained

ResNet-18 model that has been trained on a sizable dataset (such as ImageNet) is loaded from torchvision.models. Obtains the fully connected layer's (commonly represented as fc) amount of input features for ResNet-18. Replaces the original fully linked layer with a new classification head to modify it for a 10-class classification task. The custom ResNet-18 model for a 10-class classification is instantiated. We first train the model on the train dataset for the specified number of epoch and the result of each epoch is stored for creating the graph. We then call model-evalutaion to evaluate the trained model on the test data. This function most likely executes the model in inference mode and compares predictions to test data's true labels to determine accuracy. After storing the accuracy of the test data in the dictionary, we finally return the model to the CPU. The Epoch VS Accuracy line graph is then plotted. The overall accuracy of the model is 67.10

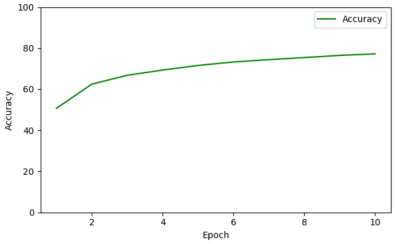

Figure 5. ResNet18 Epoch VS Accuracy Line graph

3.5. ResNet50

ResNet50 is a strong picture classification model that can produce cutting-edge outcomes when trained on huge datasets. The use of residual connections, which enables the network to learn a set of residual functions that map the input to the intended output, is one of its main advances. The convolutional layers, identity block, convolutional block, and fully connected layers are the four primary components of the ResNet50 architecture. We start with creating a custom class ResNet50 which inherit nn.Module, we load the ResNet-50 architecture pretrained on a large dataset. The pretained module is helpful because it has already learned the basic feature. We obtains the amount of input features for the ResNet-50's last fully connected (classification) layer. Install a new classification head in place of the previous fully connected layer as per our requirements. We than create a instance of ResNet-50 model and move the model process from CPU to GPU. We firstly train the model on the training dataset for the specified number of epochs. The accuracy of each epoch is stored in the list. The model is then assessed using the test data set; the overall accuracy is determined by comparing the model's predictions with the actual label. The model is sent back to the CPU once the accuracy has been stored in the dictionary, and a line graph of Epoch versus accuracy is then plotted. The overall accuracy of this model is 71.26

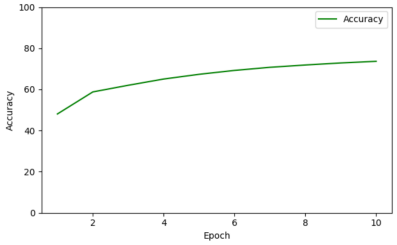

Figure 6. ResNet50 Epoch VS Accuracy Line graph

4. Comparing the accuracy of each model

Every model has been run, and the accuracy of the train and test datasets has been recorded. To compare accuracy across the several models we have trained and processed, we will use this stored accuracy to generate a histogram.

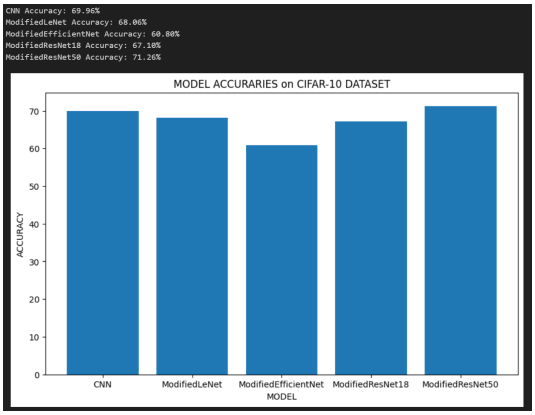

Figure 7. Models VS Accuracy Histogram

ResNet50 was the top-performing model in terms of accuracy and stability across epochs, according to a final comparison using a histogram plot. This suggests that ResNet50 is capable of extracting intricate features and generalizing on picture classification tasks. Overall, ResNet50's improved performance was probably aided by its residual connections, making it an excellent option for deep learning applications in image analysis.

5. Conclusion

Several CNN-based models, including a custom CNN, LeNet, EfficientNet, ResNet18, and ResNet50, were used in this evaluation. They were trained and assessed using the CIFAR dataset for image classification. Comparing their

accuracy and training time performance was the main objective. LeNet's accuracy was 68.06%,whereas CNN model was 69.96%. Despite its reputation for compound scaling, EfficientNet had the lowest accuracy of 60.80%, which may have been caused by its complexity in relation to the size of the dataset and the small number of training epochs. ResNet50 outperformed all models with the maximum accuracy of 71.26%, while ResNet18 obtained 67.10%. The significance of model selection in CNN-based classification tasks is highlighted by this analysis, since every architecture has

References

- https://medium.com/@ssssssppp/image-classification-using-cnn-0fad8367acfd Introduction to CNN
- https://keylabs.ai/blog/common-challenges-in-image-classification-and-solutions
- https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets
- https://blog.roboflow.com/what-is-efficientnet/
- https://au.mathworks.com/help/deeplearning/ref/resnet18.html
- https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f