Computer Vision using GPU and Transfer Learning

Lucas Saechao, Ashley Thor, Noah Venethongkham

218794239, 219334909, 219660117

CSC 180 - Intelligent Systems

Project 3

April 2, 2021

Abstract

The following report describes the methodology, challenges, and the analysis of two different image classification models based on a traditional convolutional neural network approach, and a transfer-learning approach. The CIFAR-10 dataset is referenced throughout this report as it is the metric in which the predictions of both models will test against.

Problem Statement

In modern computing, and machine learning, one of the foremost interdisciplinary problems revolves around image classification and its applications. How best can we train a system in order to “see” and interpret the world around it? It is around this problem in which the field of computer vision has grown tremendously, finding usage in facial recognition, autonomous driving capability, to image generation. For this reason, our project focuses on building and comparing the performance and accuracy of two neural network models that aim to predict if an image is either of a plane, a car, a bird, a deer, a dog, a frog, a horse, a ship, or a truck.

Methodology

In order to train this dataset, several Python library functions were utilized from Scikit-learn, TensorFlow, Keras, Numpy, and Pandas. Because we are working with the CIFAR-10 dataset, our first step was to load the data into two tuples: (x_train, y_train)

and (x_test, y_test), representing a % train/test split. This data is then one-hot encoded and fed into the first pass of our notebook. We devised an algorithm to systematically test different activation functions, optimizers, and neuron counts in our model, and save the best configuration based on F1 score. When this preliminary work is finished, we are then able to build our neural network model. Our best hyperparameter configuration uses 128 and 64 neurons, and relu activation and adam optimization.

Our convolutional network (fig. 1) uses the following architecture: a input layer with 32 neurons representing a 32x32x3 shape, a 3x3 kernel with a 2x2 stride, same padding and relu activation, followed by a hidden layer with 128 neurons. The next layer is a max pooling layer with a 2x2 size, and another convolutional layer with 64 neurons, a 3x3 kernel with a 1x1 stride, followed by another max pooling layer. This model is then flattened, and followed by a fully connected layer with 512 neurons, a ‘relu’ activation layer, a subsequent 0.5 dropout layer, and a ‘softmax’ output layer with 10 neurons. This model is then compiled using categorical cross-entropy and the adam optimizer.

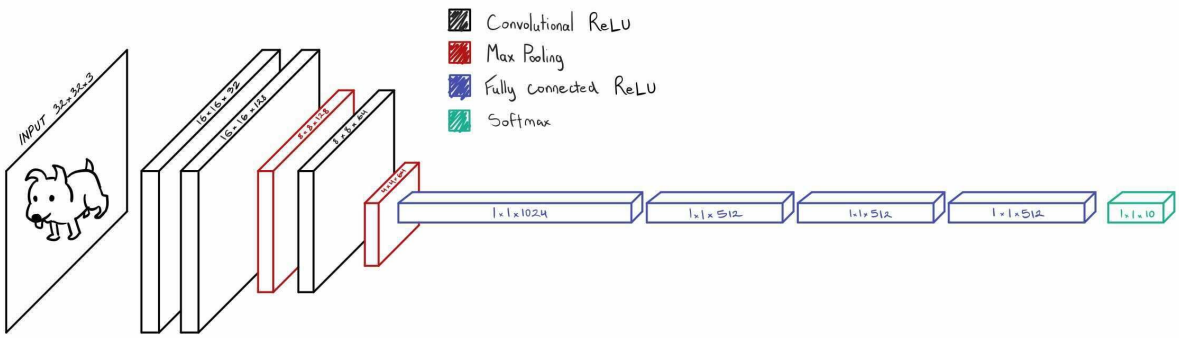

Fig. 1: Convolutional Neural Network Architecture.

Our transfer learning model uses the pre-trained VGG16 convolutional model as a base. VGG16 is previously published in “Very Deep Convolutional Networks for

Large-Scale Image Recognition”, by Karen Simonyan and Andrew Zisserman, having famously achieved 92.7% accuracy on ImageNet, a dataset consisting of 14 million images belonging to 1000 different classes. VGG16 supports images down to 48x48 as an input which conflicts with our current image set, which are of the shape 32x32, and because of this, our first step was to upsample our images. We upsampled our train and test data into 64x64 resolution images, and loaded it into a VGG16 model. After adding each layer in our VGG16 model, we froze our model weights, flattened the model, and began adding fully connected layers. We added three hidden dense layers with 512, 256, and 128 neurons, followed by a dropout layer, and our softmax layer.

Experimental Results and Analysis

One of the most striking observations we’ve made was that our model had managed to outperform the VGG16-based neural network. Figure 2 demonstrates the precision, recall, and F1-score of both models.

CIFAR-10 Model

	precision	recall	f1-score	support
0	0.76	0.84	0.80	1000
1	0.84	0.87	0.86	1000
2	0.75	0.62	0.68	1000
3	0.63	0.54	0.58	1000
4	0.70	0.74	0.72	1000
5	0.69	0.64	0.67	1000
6	0.76	0.87	0.81	1000
7	0.78	0.83	0.81	1000
8	0.85	0.85	0.85	1000
9	0.84	0.82	0.83	1000
accuracy			0.76	10000
macro avg	0.76	0.76	0.76	10000
weighted avg	0.76	0.76	0.76	10000

VGG16-Based

	precision	recall	f1-score	support
0	0.75	0.78	0.76	1000
1	0.88	0.72	0.80	1000
2	0.73	0.59	0.65	1000
3	0.56	0.48	0.51	1000
4	0.65	0.67	0.66	1000
5	0.59	0.67	0.63	1000
6	0.65	0.78	0.71	1000
7	0.70	0.79	0.74	1000
8	0.82	0.83	0.83	1000
9	0.80	0.79	0.79	1000
accuracy			0.71	10000
macro avg	0.71	0.71	0.71	10000
weighted avg	0.71	0.71	0.71	10000

Fig. 2: Comparison of CIFAR-10 Model and VGG16-Based Model

A comparison of the confusion matrix corroborates our findings as well. Our model trained from scratch correctly predicted more images of each category than the

VGG16-based model, except for a single category, dogs. However, more astoundingly, is that the VGG16-based model is able to keep pace with the model trained directly from the CIFAR-10 dataset despite being trained on an entirely different set of data. The predictions from the VGG16-based model are still within an order of magnitude of the CIFAR-10 model, achieving generally a 71% accuracy.

Task Division and Project Reflection

For this project, we are working as a team of three, and the tasks were divided accordingly. Noah and Ashley worked on preprocessing the data and set up the code to load the data, train the model, and prepare both the CNN without transfer learning and the CNN with transfer learning. They also experimented using different optimizers to see which set of settings and neuron counts would provide the greatest and most generalized model fitting for CNN without transfer learning. They made an integrated for loop that would loop through all of the different activations and optimizers in order to find out the best combination to get the most accurate results. Lucas further modified the CNNs to ensure that the results we were receiving were accurate. Lucas also printed the metrics such as model precision, recall, F1 score, and the confusion matrix. In addition, Lucas had the CNNs predict 5 random images in a series to test both models' performances. Overall, the tasks were divided accordingly and each member did their tasks well.

Challenges

By far, the biggest challenge of this project seemed to be trying to have Google Colaboratory upsample x_train and x_test to 64X64 resolution images. Google Colab kept crashing during this part of the code because the backend couldn't allocate enough ram for the free tier. Another challenge came after performing our analysis - our model at best was reaching around 71-76% accuracy and each time, it would take an hour and a half to loop through each hyperparameter configuration. No matter how we changed the hyperparameters, we could not get a score higher than 76%.

Other Insights

Computer vision was unexpectedly resource intensive compared to our last two mini-projects, resulting in generally longer hyperparameter configuration and training times. If we had more time, we could have tried to test a larger variety of hidden layers and neurons in addition to changing the hyperparameters for the CNN using transfer learning in order to produce the best model without overfitting. However, more time would be required with our current hardware specifications as it takes quite a while for the models to train. Furthermore, the amount of time it takes to test hyper parameters also takes quite a bit of time in which we were not fully able to test every combination. If given enough time, we would have liked to optimize our model to better increase our accuracy rate in predicting images.