Sacramento State University

Project 3 - Report

CSC 180 Intelligent Systems

Due: April 2, 2021 @ 11:00am

Logan Hollmer (**ID:** 301559973)

Quinn Roemer (**ID:** 301323594)

Problem Statement:

In project 3 we were tasked with creating convolutional neural networks (CNNs) to perform image classification. The dataset used for this project is called CIFAR-10 which includes a total of 60000 RGB images that are 32x32 in size. There are a total of 10 classes of images contained in the dataset. Those being: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The first thing we would have to do would be to preprocess our data. To do this we would need to reshape the prediction data (y) and then apply one hot encoding (OHE) to this column. Once the data is prepared we would need to build, train, and test multiple CNN models. For this project, we will be using both custom CNN models and pre-built models using transfer learning. For the custom models, we will define the layer count, filter size, kernel size, etc. For the models using transfer learning, we will use the layers and weights from pre-trained models, adding our own layers on top to conform them to our data.

Methodology:

We decided to tackle this problem statement in a similar fashion to the previous two projects. Each team member worked separately on their models, using different techniques to train their models in a friendly competition to see who could create the best model. As before, each member set up semi-automatic functions for building, training, and testing models. Both team members preprocessed their data in the same way for the custom models, however, there were some differences when preparing the data for the transfer learning models which will be discussed later.

For creating the custom CNN models, Quinn created a set of custom models in which the optimization function and the optimizer could be defined. This allowed the same model to run with different activation and optimizer set-ups. All of Quinn's models made use of the usual convolution and max-pooling layers, leading into a flattening layer followed by several dense layers. This was finally topped off with a dense layer with a softmax activation function for the output. In addition to the above, some of Quinn's models also used batch normalization layers. Logan took a very similar approach, however, there were some differences. Logan did not use any batch normalization in his models, he also made much heavier use of stacking multiple convolutional layers on top of each other before adding a max-pooling layer.

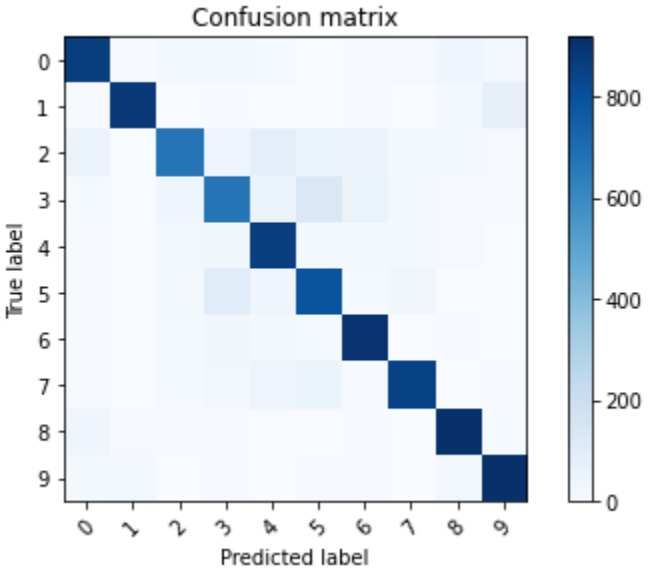
For the CNN models using transfer learning, Logan started by scaling the images up to 64x64 so that they could be accepted by the VGG model. He then loaded all the layers from the VGG model and froze them to prevent them from being adjusted during training. Next, Logan added a flattening layer and a dense layer with 10 neurons and a softmax activation function for output. Quinn did essentially the same thing however he went above and beyond in trying several different pre-trained models, these included ResNet50 and DenseNet201. For these models, he also decided to use a GlobalAveragePooling2D layer instead of a flattening layer. Finally, he also tried the VGG16 model with upsampling to 224x224 (the image resolution it was originally trained on) rather than 64x64.

Experimental Results and Analysis:

In training the CNN models from scratch, Quinn managed to create the best-performing model. This model was a custom model consisting of convolution layers followed by batch normalization layers, max-pooling layers, and dropout layers. After 4 CNN layers, a flattening layer connected the network to a fully connected network consisting of 3 layers (including the output layer). More specific details for this model can be seen in the project notebook. In Quinn's testing, this model performed best with the Relu activation function and the Adam optimizer. Overall, this model managed an F1 score of 0.8349.

Logan's best custom CNN model consisted of 3 convolution layers followed by a max-pooling layer. This was repeated 3 times before the network was sent into a flattening layer to be processed by a fully-connected network consisting of 3 dense layers (including the output layer) interleaved with dropout layers. This model featured the Relu activation function and the SGD optimizer. Overall, it managed an F1 score of 0.7535.

Confusion Matrix of our Best Model



Precision: 0.8349 | Recall: 0.8349 | Accuracy: 0.8349

While training our custom models we generally found that a larger number of layers performed better. This tends to suggest that there are a great many features to be extracted from each image in the dataset to help properly categorize them. In addition, the Relu activation function performed very well when paired with either the Adam or SGD optimizer.

In training our transfer learning models Quinn experimented with 3 different networks. VGG16, DenseNet201, and ResNet50. To each of these, he added 4 dense layers (including the output layer) interleaved with dropout layers. All models used the Relu activation function with the Adam optimizer. Note, all layers that were transferred over from the model were frozen. Quinn trained two versions of VGG16. One with the added top layers as previously stated, and the other with the addition of an upsampling layer at the start of the model to convert the images to 224x224, the resolution VGG16 was trained on. After training, the VGG16 model with upsampling proved to perform the best with an F1 score of 0.7389. However, the other models were not far behind.

Similarly, Logan trained a model via transfer learning using VGG16 with a few differences. For example, he added just a flattening layer and a dense layer with 10 neurons for output. He also used the SGD optimizer. This was trained on 64x64 images. In the end, it managed to achieve an F1 score of 0.69.

After creating and testing our models that incorporated transfer learning we were surprised to see them being outperformed by our custom models in almost all cases. We believe this because the weights transferred from Keras were for Imagenet. Unlike CIFAR-10 this dataset contains a great many more items to classify. In addition, these models were trained on 224x224 images. Using this resolution, Quinn saw better results with VGG16. Below you can find a table with our best models from various categories.

Table 1

Type	Average F1 Score	Activation Function	Optimizer
Custom CNN	0.8349	Relu	Adam
VGG16 with Upsampling layer	0.7389	Relu	Adam
Custom CNN	0.7347	Sigmoid	Adam
DenseNet201	0.7185	Relu	Adam
Custom CNN	0.7082	Tanh	SGD
VGG16	0.7002	Relu	Adam
ResNet50	0.4075	Relu	Adam

Table 1 displays Quinn's top models from each category trained

Task division and project reflection:

Both Logan and Quinn did the entire project themselves which included pre-processing the data, building custom CNN models, training the models, and testing them. When it comes to writing the report, both members wrote sections of it, Logan wrote the first two and the last section with Quinn writing the *Experimental Results and Analysis* Section. Quinn also compiled and edited the final version of the notebook with Logan recording the video for the presentation.

When looking back on the project we felt that it would have been very interesting to use the CIFAR-100 dataset instead of the CIFAR-10 dataset. However, we originally started with the CIFAR-10 dataset and by the time we had trained our models we simply did not have time to go back and retrain on the CIFAR-100 dataset.

Comparing the results of CIFAR-10 to CIFAR-100 would have been a very interesting exercise and a great extension to our project.

One of the major challenges we faced was issues with RAM while training our models. However, both Logan and Quinn had access to the full version of Google Colab so this issue was quickly fixed by switching into one of their High RAM environments.

Another challenge we faced was with the accuracy and recall of the models. We struggled to get a high F1 score with many of our models performing quite badly. This is most likely due in part to our inexperience with CNN's as we had to go back and adapt our models several times. While we eventually ended up with a model we feel performed relatively well, we know that there is much that could still be improved. We believe that using a custom model with even more layers would result in CNN that would get close to an F1 score of 0.9.

We also learned that there are multiple ways to get the data into the correct "shape" for a CNN. Besides the classic reshape functionality in NumPy, we found we could also use the UpSampling2D functionality in Keras. The variation of layers available for use in Keras lends itself to further research and experimentation in our future projects.

Along similar lines, we also found that we could use the GlobalAveragePooling2D layer from Keras to replace the flattening layer. Both of these layers allowed us to move into a fully connected neural network however the GlobalAveragePooling2D applies pooling layers until there is only one dimension left. This seemed to work slightly better than the flattening layer in the transfer learning models where it was incorporated.