CSC 180-01 Intelligent Systems

Mini-Project 3: Computer Vision Using GPU and Transfer Learning

Due at 2:00 pm, Friday, October 30, 2020

Tran Ngoc Bao Huynh (Student ID: 219763298)

Ong Thao (Student ID: 219467431)

Problem Statement

With the CIFAR-10 dataset from Keras consisting of 60000 32x32 color images in 10 classes, our goal for this project is to utilize the Google GPU and transfer learning to practice image classification with the CIFAR-10 dataset. We created two CNN models (one without transfer learning and one with transfer learning) to do the image classification on the Google GPU.

Methodology

For our environment, we utilized Google Collab and enabled the GPU runtime in the backend to run our notebook. Our first step was importing the Keras CIFAR-10 dataset then splitting the dataset into training and test data. We then started implementing our CNN model with no transfer learning. For our CNN model with no transfer learning, we utilized our knowledge and best models from Project 2 with a few parameters tweaked to fit our use cases. The best CNN models from Project 2 that we used were 2 convolutional layers with a kernel size of 4x4 and 3 convolutional layers with a kernel size of 3x3 (which was our best). Additionally, from our fully connected networks in Project 2, we found relu-adam was the best activation and optimizer combination, so we continued the trend of using these two activation and optimizer functions for our CNN models with no transfer learning in this project.

Meanwhile for the CNN model with transfer learning, we implemented theVGG16 pre-trained model. In the beginning, we expected that this model would have better and higher accuracy. However, the result is surprisingly much lower compared to our non transfer learning models. We tuned the number of Dense layers, number of neurons and drop rate, but we haven’t got any higher accuracy from the transfer learning model.

Experimental Results and Analysis

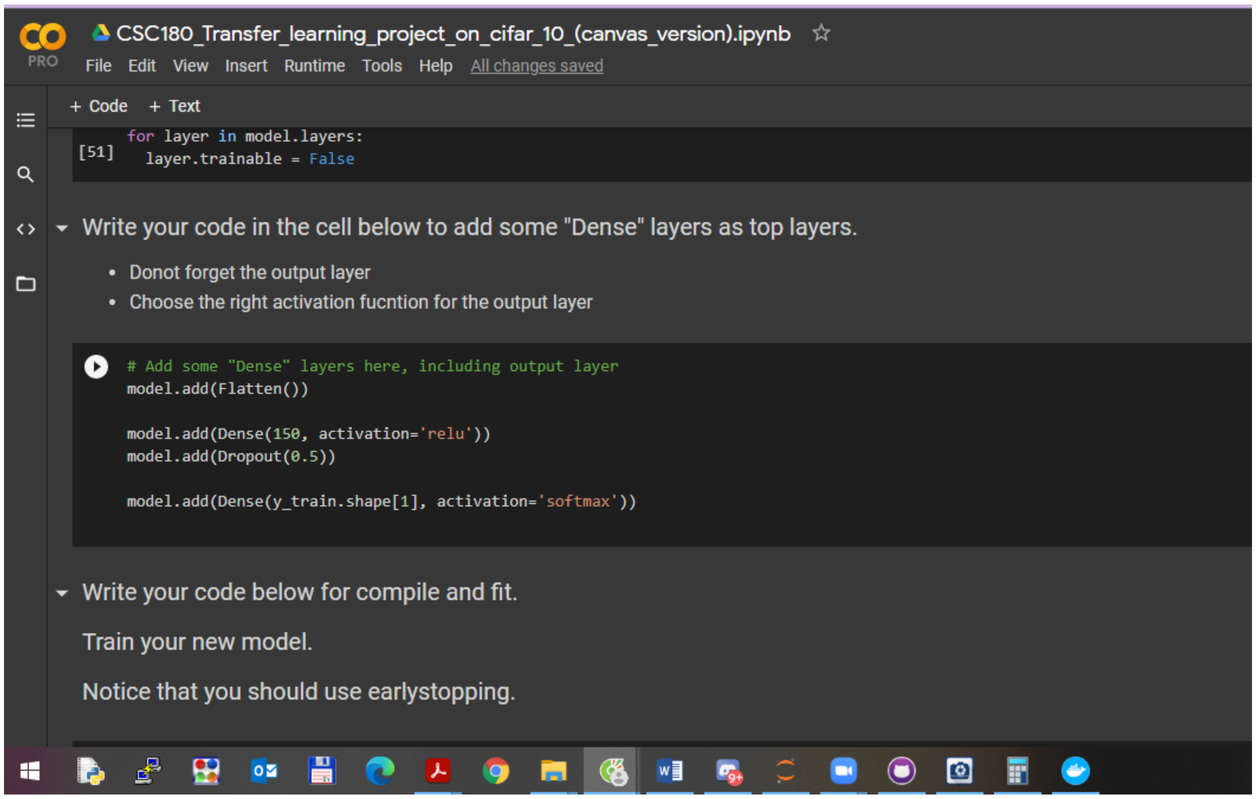
For the CNN models with no transfer learning, we found that increasing the Dropout rate from 0.1 to 0.5 actually increased the accuracy of our model from 0.74 to 0.76. This may be caused by the fact that by ignoring redundant and co-dependent neurons during the training phase with the Dropout layer, we are able to better regularize the data and train our models with more focused data and avoid overfitting to a certain degree. Furthermore, increasing the number of neurons in our Dense layer with the relu activation function from 250 to 512 to 1024 helped increase our model accuracy from 0.76 to about 0.79. This may be because with the more neurons we have, the more possible edge cases and parts of the image we are able to detect and have our models make sense of. Below are our results for our CNN models with no transfer learning:

Model:	**2 Convolutional Layers w/ 4x4**	**3 Convolutional Layers w/ 3x3**
# of images correct	⅘ randomly picked images correct	⅘ randomly picked images correct
Accuracy	0.7832	0.7877
Precision	0.7830656582239498	0.7852629821103416
Recall	0.7832	0.7877
F1-score	0.7828437553136911	0.7860135671175159

For our transfer learning VGG 16 model, we tried different numbers of Dense layers and number of neurons, but some of the scores are really bad. The best score we had for this model is: (0.6 unit lower than our best non transfer learning model above)

	Score
Accuracy	0.7222
Precision	0.7237857200126733
Recall	0.722
F1-score	0.7221205454834959

We tried smaller number, and larger number of neurons, and we got result as the screenshot below:


CO PRO
CSC180_Transfer_learning_project_on_cifar_10_(canvas_version).ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
[51] for layer in model.layers:
       layer.trainable = False
```

Write your code in the cell below to add some "Dense" layers as top layers.

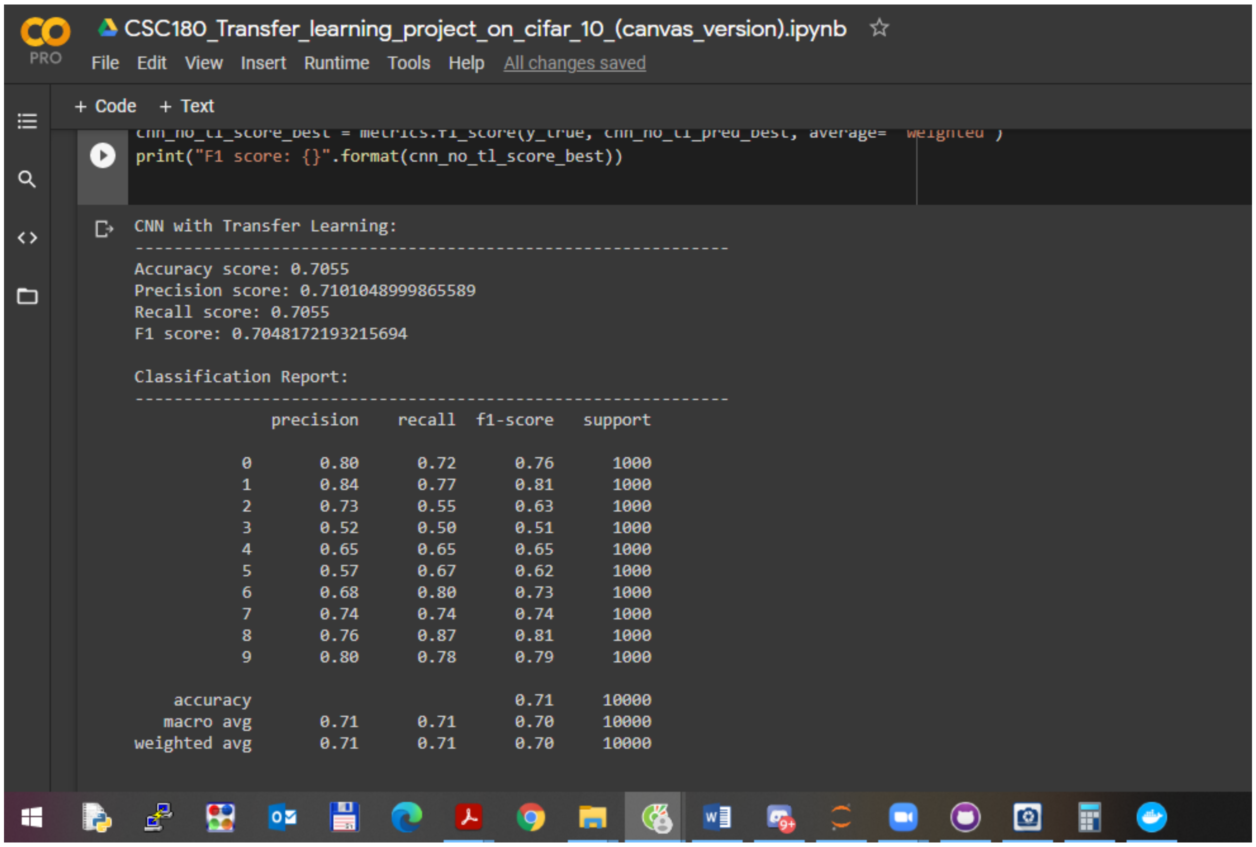
- Donot forget the output layer
- Choose the right activation fucntion for the output layer

```
# Add some "Dense" layers here, including output layer
model.add(Flatten())

model.add(Dense(150, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(y_train.shape[1], activation='softmax'))
```

Write your code below for compile and fit.

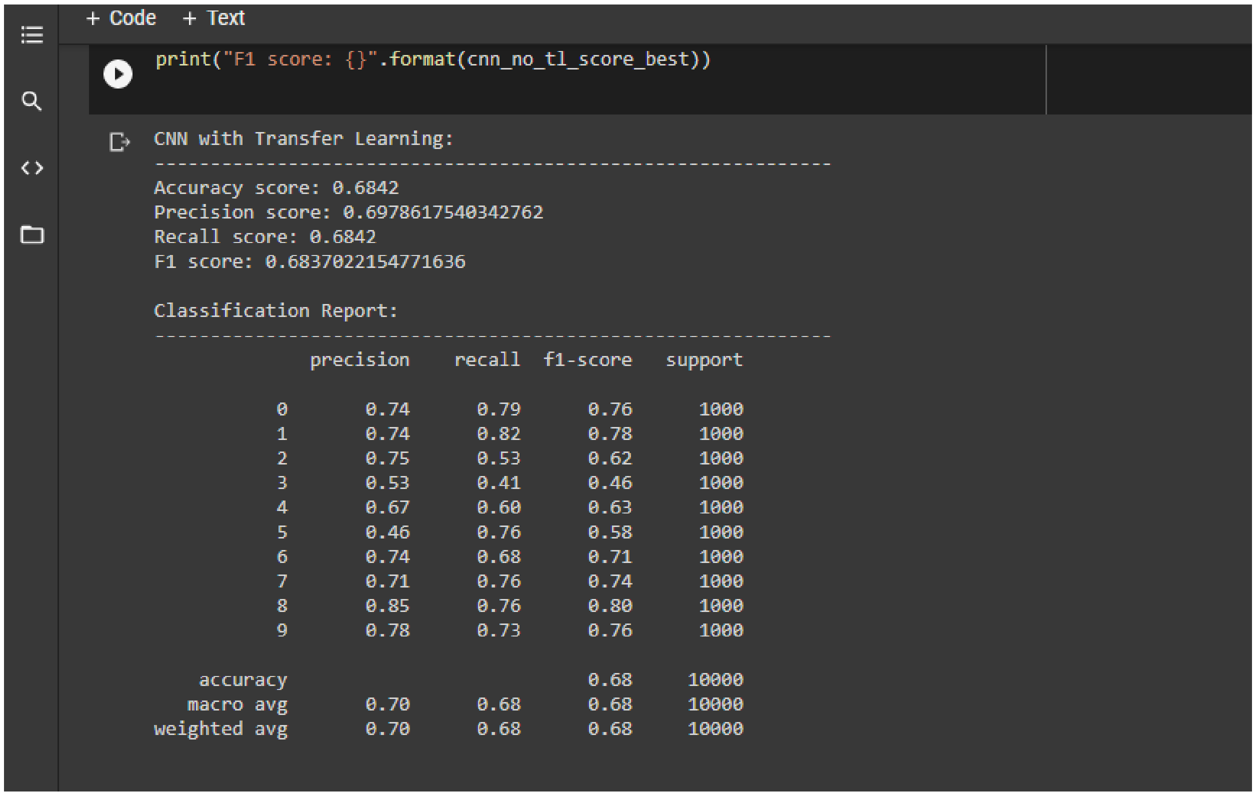
Train your new model.

Notice that you should use earlystopping.


CSC180_Transfer_learning_project_on_cifar_10_(canvas_version).ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

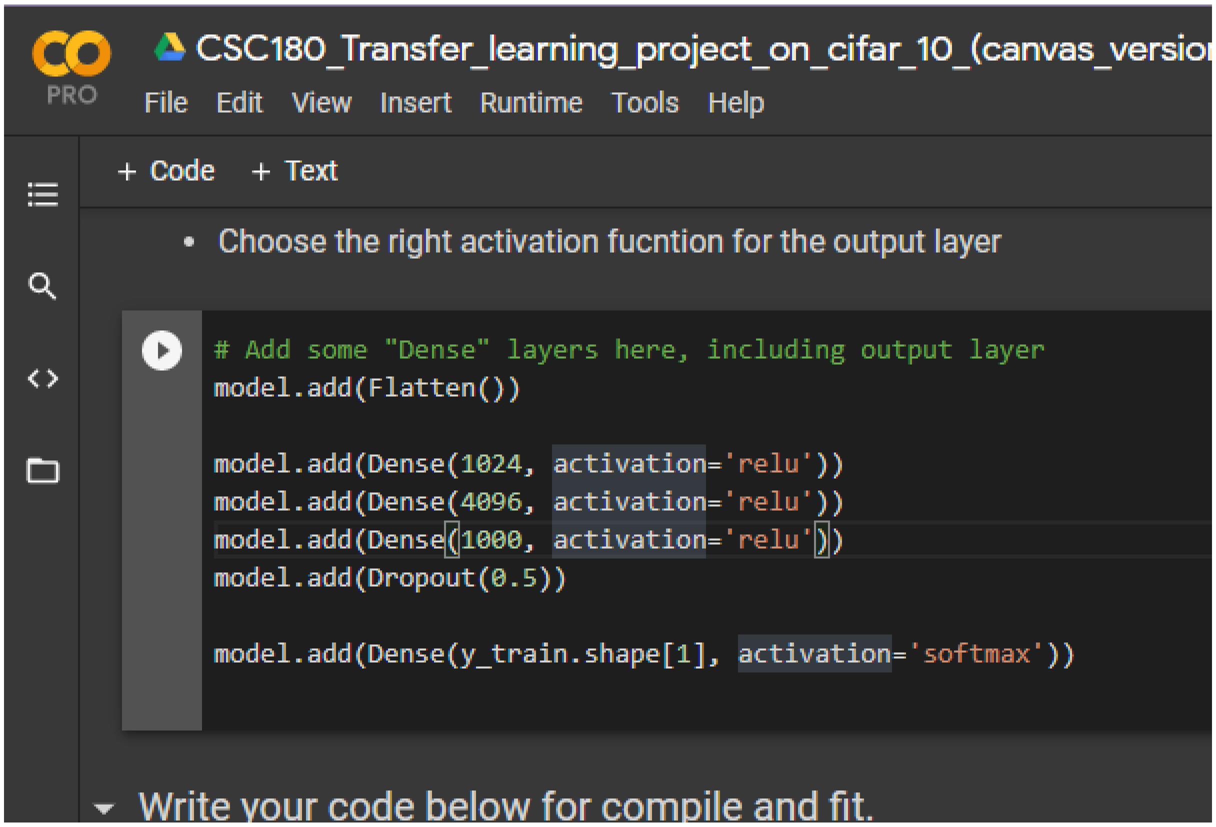
```
cnn_no_tl_score_best = metrics.f1_score(y_true, cnn_no_tl_pred_best, average='weighted')
print("F1 score: {}".format(cnn_no_tl_score_best))
```

```
CNN with Transfer Learning:
------------------------------------------------------------
Accuracy score: 0.7055
Precision score: 0.7101048999865589
Recall score: 0.7055
F1 score: 0.7048172193215694

Classification Report:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.80      0.72      0.76      1000
           1       0.84      0.77      0.81      1000
           2       0.73      0.55      0.63      1000
           3       0.52      0.50      0.51      1000
           4       0.65      0.65      0.65      1000
           5       0.57      0.67      0.62      1000
           6       0.68      0.80      0.73      1000
           7       0.74      0.74      0.74      1000
           8       0.76      0.87      0.81      1000
           9       0.80      0.78      0.79      1000

    accuracy                           0.71     10000
   macro avg       0.71      0.71      0.70     10000
weighted avg       0.71      0.71      0.70     10000
```



```
print("F1 score: {}".format(cnn_no_tl_score_best))
```

```
CNN with Transfer Learning:
------------------------------------------------------------
Accuracy score: 0.6842
Precision score: 0.6978617540342762
Recall score: 0.6842
F1 score: 0.6837022154771636

Classification Report:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.74      0.79      0.76      1000
           1       0.74      0.82      0.78      1000
           2       0.75      0.53      0.62      1000
           3       0.53      0.41      0.46      1000
           4       0.67      0.60      0.63      1000
           5       0.46      0.76      0.58      1000
           6       0.74      0.68      0.71      1000
           7       0.71      0.76      0.74      1000
           8       0.85      0.76      0.80      1000
           9       0.78      0.73      0.76      1000

    accuracy                           0.68     10000
   macro avg       0.70      0.68      0.68     10000
weighted avg       0.70      0.68      0.68     10000
```


CO PRO
CSC180_Transfer_learning_project_on_cifar_10_(canvas_versio
File Edit View Insert Runtime Tools Help
+ Code + Text

- Choose the right activation fucntion for the output layer

```
# Add some "Dense" layers here, including output layer
model.add(Flatten())

model.add(Dense(1024, activation='relu'))
model.add(Dense(4096, activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(y_train.shape[1], activation='softmax'))
```

Write your code below for compile and fit.

We also tried adding more Dense Layers, adding/removing the Dropout, etc. The score did not increase, and sometimes, we got really strange results (score = 0.1). Also the running time for each epoch is much faster than expected too. It only took 7 to 8 second per epoch. With 5 randoms images, the transfer model got 3 correct and 2 wrong. From the website and description, VGG 16 has good performance and high accuracy so we think that the reason for this result may be because of the data test and learn ratio, dense layers number, neuron number, or mode. We will revisit the problem in the future to test this model again.

Task Division and Project Reflection

Our group worked on all parts of the project together and took lots of time discussing how we are going to tune our parameters and implement our model without transfer learning. We enjoyed implementing the model without transfer learning more than the model with transfer learning because the model without transfer learning enabled us to have more creative freedom and experiment more. However, we also had a great time implementing the model with transfer learning and seeing how we are able to work more efficiently if we utilize an already pre-made and optimized model to train our data.