Deep Patel

Quality Control Project Report

Video Demo - https://youtu.be/l0urnBij-W0

Problem

The purpose of this project is to find a solution on how large image data sets can be sorted and classified using machine learning. This purpose derives from the idea of how AI can be used specifically in the manufacturing industry for quality control to automatically detect defects. This project explores a dataset from the Pepsico company which produces potato chips. Some potato chips during the production process become too burnt and are deemed defective and cannot be packaged. Since there is a large volume of potato chips that are processed, it is nearly impossible to be able to manually sort the defective from the non-defective. This is where AI comes in as a possible solution method to sort these chips.

Approach

I attempted to solve this problem using binary classification. After doing some research (resources to be found in ./docs/README.md in class GitHub), I found that using binary classification would be my best approach to take. Binary classification allows for the use of machine learning to classify images between two categories, which in my case is ‘defective’ and ‘non-defective’.

I came across multi-class classification which essentially is binary classification, but can be used with multiple classes. Since my problem only calls for two classes, I concluded that this would not be viable method to use as a secondary approach. Instead, I turned to using two different libraries and comparing them. The two libraries I used were Keras from Tensorflow and PyTorch. Although I used two different libraries, the workflow was essentially the same: I would load the dataset, setup the CNN architecture, train the model, and evaluate it.

Evaluation

The purpose of my project is to evaluate the viability of my models so that they could be used during the manufacturing process. The only way a model like this would be used in a manufacturing process if it was very accurate. A manufacturer would not want to use a model if the accuracy was low, possibly classifying products defective when they're not and vice versa. This in turn leads to waste and higher cost. Below are the results of my evaluations, outputting the percentage of success of correctly identifying a chip of either being defective or non-defective.

Keras

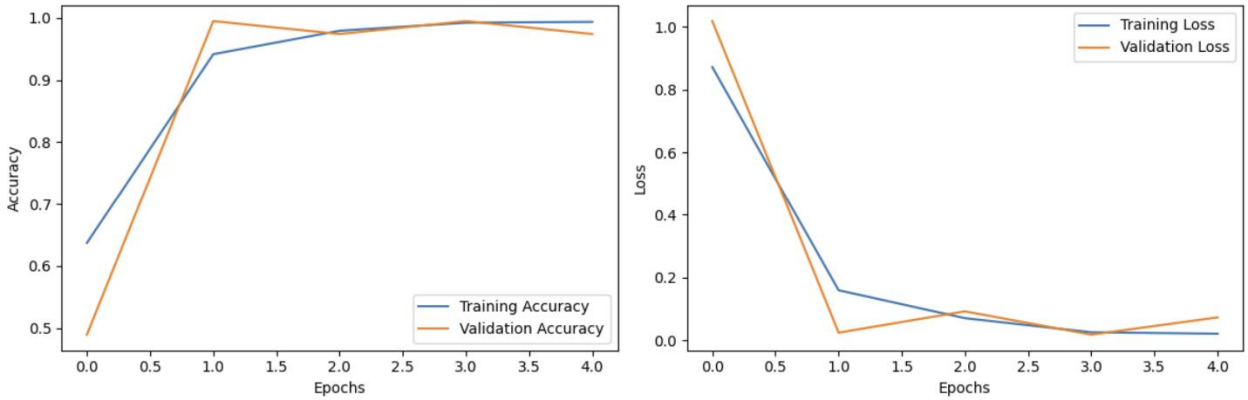
Implementation can be found in ./code/keras-classification.ipynb.

```
Epoch 1/5
25/25 [==============================] - ETA: 0s - loss: 0.8714 - accuracy: 0.6372
Epoch 1: val_accuracy improved from -inf to 0.48958, saving model to model_checkpoint.h5
C:\Users\deepk\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site
  saving_api.save_model(
25/25 [==============================] - 92s 4s/step - loss: 0.8714 - accuracy: 0.6372 - val_loss: 1.0179 - val_accuracy: 0.4896
Epoch 2/5
25/25 [==============================] - ETA: 0s - loss: 0.1597 - accuracy: 0.9415
Epoch 2: val_accuracy improved from 0.48958 to 0.99479, saving model to model_checkpoint.h5
25/25 [==============================] - 79s 3s/step - loss: 0.1597 - accuracy: 0.9415 - val_loss: 0.0246 - val_accuracy: 0.9948
Epoch 3/5
25/25 [==============================] - ETA: 0s - loss: 0.0708 - accuracy: 0.9792
Epoch 3: val_accuracy did not improve from 0.99479
25/25 [==============================] - 79s 3s/step - loss: 0.0708 - accuracy: 0.9792 - val_loss: 0.0926 - val_accuracy: 0.9740
Epoch 4/5
25/25 [==============================] - ETA: 0s - loss: 0.0260 - accuracy: 0.9922
Epoch 4: val_accuracy did not improve from 0.99479
25/25 [==============================] - 78s 3s/step - loss: 0.0260 - accuracy: 0.9922 - val_loss: 0.0184 - val_accuracy: 0.9948
Epoch 5/5
25/25 [==============================] - ETA: 0s - loss: 0.0214 - accuracy: 0.9935
Epoch 5: val_accuracy did not improve from 0.99479
25/25 [==============================] - 77s 3s/step - loss: 0.0214 - accuracy: 0.9935 - val_loss: 0.0733 - val_accuracy: 0.9740
```

The above is a sample output from training the model using Keras. After each epoch, the loss and the accuracy are printed. Once the first epoch is completed, the results are saved to model_checkpoint.h5. This file is used not only as storage for my results, but is also used to load the results of the model after training is complete. In the subsequent epochs, if the accuracy is improved from that of the previous, these values overwrite the existing values. The purpose of overwriting and saving the results is to be able to load the model with the best accuracy found so that the model can be used without retraining and the results can be used for any subsequent actions. In the case that the epoch fails to conclude in a better accuracy, it outputs that the accuracy did not improve and will move on to the next epoch; only the best result is saved. The

cell containing *model.load_weights(checkpoint_filepath*) can be run to load the checkpoint file and retrieve values. The accuracy and loss can then be printed after. Below are the results (accuracy/loss and plots) from2 this sample run.

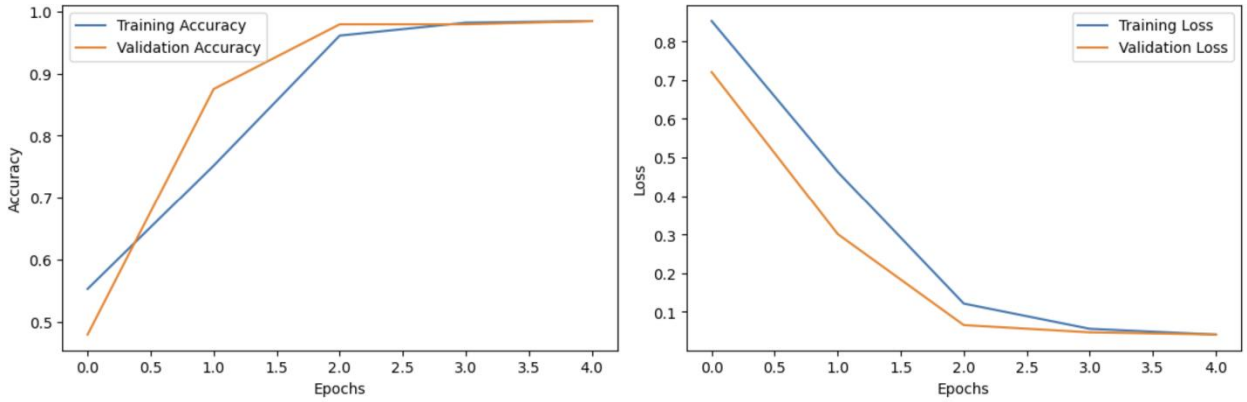
```
6/6 [==============================] - 10s 2s/step - loss: 0.0108 - accuracy: 0.9948
Test Accuracy: 0.9947916865348816
Test Loss: 0.010844740085303783
```



PyTorch

Implementation can be found in ./code/pytorch-classification.ipynb.

```
Test Accuracy: 0.984375
Test Loss: 0.04753050572859744
```



The above is a sample output of my results from PyTorch. The same method was used to program this classification like Keras; there are no significant differences in the method used.

Comparison to Google Bard LLM

```
Google Bard Accuracy
    Defective - 70% accuracy
    Non-Defective - 10% accuracy

Keras Accuracy - 100%
PyTorch Accuracy - 100%
```

Keras:

```
1/1 [==============================] - 1s 1s/step - loss: 0.0457 - accuracy: 1.0000
Test Accuracy: 1.0
Test Loss: 0.04565506428480148
```

PyTorch:

```
Test Accuracy: 1.0
Test Loss: 0.10267154127359390
```

I used Google Bard as my LLM to compare the results of my model. I took the first ten images of the Defective and Not Defective directories from each the Train and Test directories. I uploaded the pictures one by one into Bard and saw the results were nowhere near the results of my model. I asked the LLM “Is this chip defective?” for each image I uploaded. The results from the LLM were 70% accurate for the defective images and 10% accurate for the non-defective images. As you can see, my model was 100% accurate for both the Keras and PyTorch implementations, concluding that my models are more accurate than Bard.

Final Results

Classification using Keras:

Run	Accuracy	Run Time
1	94.79%	7m 7.1s
2	96.35%	7m 31.9s
3	94.27%	7m 13.6s
4	95.31%	7m 11s
Average	95.18%	7m 15.9s

Classification using PyTorch
Pre-Optimization

Run	Accuracy	Run Time
1	98.96%	31m 10s
2	97.92%	24m 49.3s
3	98.44%	27m 35s
4	98.96%	25m 7.7s
Average	98.57%	27m 10.5s

Post-Optimization

Run	Accuracy	Run Time
1	97.92%	13m 31.4s
2	98.44%	13m 36.2s
3	98.44%	11m 29.5s
4	97.92%	9m 32.9s
Average	98.18%	12m 2.5s

Above are the results I achieved from training my classification method four times. Initially, the PyTorch solution took an extensive amount of time to run, almost 30 minutes. After my fourth run, I thought about how I could optimize my solution so that it would run faster. Turns out, I made an error when coding, I initially had my solution evaluate my model during the for loop where it was training, and then again after the training completed. This resulted in my model taking double the amount of time than what it should've taken. Nevertheless, I decided to keep my results from my old model to compare accuracies with the possibility of them being any different. However, in the end the accuracies remained relatively the same with under 1% difference.

In conclusion, based on my two models, PyTorch seems to perform better in terms of accuracy with about a 3% difference. However, the Keras model runs almost two times faster than the PyTorch model. From a manufacturing standpoint, it can be concluded that the PyTorch model is better to use when using machine learning to detect defects in products, as the higher the percentage of accuracy, the less waste and cost the manufacturer will incur. Inversely, if time is more of a concern, then the Keras model is the better option to go with.

Related Work

Industrial Quality Control of Packages

This dataset is another example of classifying a product on whether it is defective or not. This one pertained to whether packages could be identified as damaged or not. This solution takes into account whether the package can also be correctly classified based on the top, bottom or side view. This project is very similar to mine as it attempts to solve the same problem as I am, however it takes more variables into account. Here is a classification solution somebody created using Keras, but only achieving 50% accuracy and 69.31% loss as opposed to my solution achieving an average of 95.18% accuracy and 0.02% loss.

casting product image data for quality inspection

This dataset is an example of quality control in the metal casting industry. Although the problem is the same as mine, this dataset is used to attempt to figure out the faults in the machinery used to produce these products. Unlike a chip being defective (burnt) due to heat, defects in metal can be caused by various machines used to cast metal. These machines can cause defects such as pinholes, shrinkage defects, pouring metal defects, etc. Nevertheless, the problem is the same being whether the product is a defect or not. Here is a classification solution created by someone achieving 98.86% accuracy and 75% test loss. Although his accuracy was better than mine, their loss was more deficient than mine.

Step By Step Guide for Binary Image Classification in Tensorflow

This is a solution with the use of binary classification on images on X-rays from this dataset, something not product related, but utilizes the same procedure as my model. This dataset contains x-ray images of healthy chests (non-defects) and pneumothorax-infected chests (defects). This solution also uses the sequential model from the Keras library. This test also achieves a higher accuracy than that of mine with an accuracy of 99.04%, however the number of epochs is 30. The solution doesn't explicitly state the loss, but based on the graphs, it looks to be about 0.03% loss. Based on these results, this model performs a little better than mine in achieving a good accuracy.