Fundamentals of Machine Learning Final Project

Bryan Etzine *(Author)*

Computer and Information Science Engineering

University of Florida

Gainesville, FL

bryanetzine@ufl.edu

Daniel Giraldo *(Author)*

Electrical and Computer Engineering

University of Florida

Gainesville, FL

danielgiraldo@ufl.edu

Dean Ward *(Author)*

Electrical and Computer Engineering

University of Florida

Gainesville, FL

dward2@ufl.edu

***Abstract*—In this paper, we discuss the steps we took to train a machine learning model to classify images of ten brand logos. We begin by discussing the motivation behind image classification and how it can be accomplished by a computer. We then discuss how we implemented a model to train and test our dataset. Next, we discuss the experimentation process and how we came about choosing and training our model for classifying the ten brand logos. Finally, we discuss the results of our model. We found that by utilizing transfer learning with the ResNet-50 architecture, our model could classify our test set of brand logo images with 95% accuracy.**

Keywords—Machine Learning, Deep Learning, Neural Networks, Convolutional Neural Networks (CNN), Image Classification

I. Introduction

Image classification is an important task that computers can be trained to perform with impact across many industries from manufacturing and healthcare, to agriculture and defense. Advancements in the field of artificial intelligence have led to the ability for us to train a computer to perform these classification tasks. [1] discusses and compares a multitude of machine learning approaches for the task of image classification, it finds that the Convolutional Neural Network is the best performing approach for image classification. We used these conclusions along with information from our class lectures to decide that the Neural Network was a great approach for our image classification task. [2] looks into a variety of pre-trained models for chest radiograph image classification and finds that the ResNet model architecture performed very well. Although we do not initially choose to use the ResNet model, we eventually decide to use it, specifically the ResNet-50 model.

We were tasked with designing a model that can classify images as one of ten brand logos. These include Nike, Adidas, Ford, Honda, General Mills, Unilever, McDonald's, KFC, Gator, and 3M. As a bonus challenge, which we took on, our model could also classify an image that is not one of the ten brand logos and correctly identify it as an unknown class.

II. Implementation

A. Train Function

This is the function that trains the model. The function has all imports already declared in the first cell of the function.

Note: This function expects there to be 10 classes. If you wish to train the model with a different number of classes, you must change the line of code `outputs = tf.keras.layers.Dense(10, activation='softmax')(x)`, change the '10' to the number of classes you wish to train on. This architecture was designed to classify 10 classes of specific image brand logos; we cannot speak to the performance of the architecture on a different number of classes or different genres of images.

Step 1: This function takes a numpy array data set in the form (Features x Number of Samples), specifically in this case it is expecting 270,000 features corresponding to a 300x300 RGB image. The default file path is input here in the form `data_train.npy` which should be in the same root directory as the `train.ipynb` file, please update if the file name is changed.

Step 2: The next step is to load the labels which is expected to be a one-dimensional array in the form (Number of Samples,). The default file path is input here in the form `labels_train_corrected.npy` which should be in the same root directory as the `train.ipynb` file, please update if the file name is changed.

Note 1: This is the file path to which the train_test_split will save the resulting test data and label data, for testing later using the `test` and `test_unknown` functions .

Step 3: The file path to the saved model is specified here, which by default is `best_model.h5`, please update if the file name is changed.

Step 4: This will load a previously saved model specified in Step 3, please comment out this line if there is no saved model (this would be the case if you are training the model for the first time).

B. Test Function

This function uses the previously trained model to make inference on the test data when there are no unknown classes.

Step 1: This function takes a numpy array test set in the form (Features x Number of Samples), specifically in this case it is expecting 270,000 features corresponding to a 300x300 RGB image. The default file path is input here in the form `x_test.npy` which should be in the same root directory as the `test.ipynb` file, please update if the file name is changed.

Step 2: The next step is to load the labels which is expected to be a one-dimensional array in the form (Number of Samples,). The default file path is input here as `t_test.npy` which should be in the same root directory as the `test.ipynb` file, please update if the file name is changed.

Step 3: The file path to the saved model is specified here, which by default is `best_model.h5`, please update if the file name is changed.

Note: This will output both a Classification Report and a Confusion Matrix. The classification report will return the metrics: precision, recall, f1-score, support and accuracy. The confusion matrix will return both the correctly and incorrectly classified samples based on their labels.

C. Test_Uknown Function

This function uses the previously trained model to inference on the test data when there is an unknown class or class not a part of the 10 logos our model was trained on.

Step 1: This function takes a numpy array test set in the form (Features x Number of Samples), specifically in this case it is expecting 270,000 features corresponding to a 300x300 RGB image. The default file path is input here in the form `x_test.npy` which should be in the same root directory as the `test.ipynb` file, please update if the file name is changed.

Step 2: The next step is to load the labels which is expected to be a one-dimensional array in the form (Number of Samples,). The default file path is input here as `t_test.npy` which should be in the same root directory as the `test.ipynb` file, please update if the file name is changed.

Step 3: The file path to the saved model is specified here, which by default is `best_model.h5`, please update if the file name is changed.

Note: This will output both a Classification Report and a Confusion Matrix. The classification report will return the metrics: precision, recall, f1-score, support and accuracy. The confusion matrix will return both the correctly and incorrectly classified samples based on their labels.

D. Handling Unknown Class

The way our model makes a prediction is by assigning probabilities to each class for every image. The class that has the highest probability is the one that the image is most likely a part of and the one that the model assigns class ownership to. We found that the model is typically very confident in the class that it assigns ownership to, very often it is more than 90% certain about the class the image is a part of. We uploaded a picture of a dog to see what the highest probability our model will assign to a given class is. With this knowledge, we can decide on a threshold for our model such that any images whose highest confidence value is less than the threshold will be classified as an unknown class. We were a bit surprised that with the dog image the highest confidence level was ~55% for the Nike class, we expected the highest probability to be much lower. On second look, we realized that the image of the dog did actually have characteristics of the Nike logo, with the angle and the way the dog was sitting, it was in the shape of the Nike logo! We uploaded a new dog picture and found the highest confidence was lower at ~40%, we were happier with this. However, we wanted to be conservative with our threshold. Just in case the test set with unknown class images had characteristics similar to some of our logos, we chose a threshold of 0.6. With this threshold, some of the classes that were actually a part of one of the brand logos were being classified as unknown, but we still achieved a 94% accuracy.

III. Experiments

The first steps we took were preprocessing the data. We noticed that some images in the dataset were labeled incorrectly, so we manually visualized all of the images, identified incorrectly labeled data, and correctly labeled them. We then normalized our data by dividing by 255, this is because pixels have a value between 0 and 255 and we wanted the data to be between 0 and 1.

We began our experimentation with a shallow Artificial Neural Network (ANN) to test what changes made the biggest difference to a simple model. This ANN consisted of an input (flattenting) layer, one dense layer with 300 units, another denser layer with 100 units and the final dense (output) layer

for the 10 samples. With this simple model we were able to successfully train, but with a subpar ~10% accuracy on both the training and validation data, regardless of how many epochs were completed.

This prompted a search through Tensorflow's documentation to see what could be done to increase the accuracy of the model along with direction based on the material taught in class. Improved quality and quantity of data is the best way to increase the accuracy of any model. We found that data augmentation, a technique to increase the diversity of the training set by applying random (but realistic) transformations, such as image rotation, was a way to improve performance. After implementing data augmentation, specifically random horizontal and vertical flips in addition to random rotations, the accuracy of both the training and validation data increased to ~60% accuracy after 10 epochs. This led to the conclusion that since more diverse data was successful with a shallow ANN then it would be extremely beneficial in a more complex model.

Since the performance of our shallow ANN was worse than the required specification of at least 90% accuracy, we chose to take a different approach. We turned to transfer learning, an approach in which you utilize a pre-trained model and fine-tune it with your dataset to reach a desired model. The next step was to search for a good pre-trained model that could work with our dataset. Searching through the internet for good image classification pre-trained models brought us to MobileNet-v2.

MobileNet-v2 is a CNN architecture of 53 layers, it is built as a small, fast, lightweight architecture designed especially for mobile devices (whose computational power is not as great as a supercomputer like HiPerGator or even a personal computer). Despite knowing it was especially designed for mobile devices, following Occam's razor principle, we figured we'd try out how well this simple model could perform on our dataset before trying out a more complex pre-trained model. The model performed better than our shallow ANN, with an accuracy of ~82%, still below our desired accuracy of at least 90%. We decided to keep searching for a good image classification pre-trained model and stumbled upon ResNet-50.

ResNet-50 is also a CNN architecture but has 50 layers, it is a more computationally expensive architecture but the trade-off is that it performs very well. This is the architecture we later decided to stick with as without testing different hyperparameters we got well over 90% accuracy. The weights used on our ResNet architecture are based on the ImageNet dataset, a dataset with millions of images.

At the start, one of the difficulties was getting the data into a format that ResNet-50 recognized. We figured out that using the "Dataset.from_tensor_slices" function allowed us to convert our data into the tensor data structure/type. The problem was that due to the greater computational resources demanded by ResNet-50 (and our bottleneck of only 16GB of RAM), our kernel kept crashing. After discussing with the professor and fellow classmates, we tried memory management techniques that allowed our kernel to run and our model to train. These techniques included deleting unused variables, converting our images to the 'float16' data type, closing all other kernels, and finally we found out that the input to the tensorflow model could be numpy arrays and we did not have to convert them to a tensor which saved some memory.

Now that we had a functioning model and one that the kernel could handle without crashing, it was time to experiment with hyperparameters and overfitting mitigation techniques. Some of the hyperparameters we chose to experiment with were the learning rate optimizer, learning rate, loss function, batch size, and dropout percentage for mitigating overfitting.

We did not use a rigorous technique like grid search for choosing hyperparameters because it was too computationally expensive to do so, instead we made a validation set and manually changed values to see what had the best accuracy and validation accuracy. If our validation accuracy was far off of our train set accuracy, then we knew we were overfitting. We found that the best hyperparameters for us were: Nadam learning rate optimizer, learning rate of 0.001, categorical cross entropy loss function, batch size of 16, and a dropout of 0.4. We used callback to choose the best model weights with these hyperparameters, the best model weights were saved with an accuracy of 0.9995 and validation accuracy of 0.9853.

Now that the weights for our best model were saved, it was time to see how it performed on the test set. The test set outputted a 95% accuracy, we completed our goal of achieving at least a 90% accuracy for classifying the ten brand logos. Next, we took on the unknown class challenge.

In the Implementation section we discussed how we took on the unknown class challenge. The experiments we performed in this step consisted of choosing a threshold value that determined whether a class was classified as unknown or as one of the ten brand logos. There was a trade-off between correctly classifying an unknown class and incorrectly classifying a brand logo image as an unknown class. If we made the threshold too high, brand logo images were being classified as unknown. If the threshold was too low, then some images which are not a part of the brand logos were still being classified as one of the brand logos. We discussed that with our dog image, our model was ~55% confident that it was a Nike logo, likely due to the dog having characteristics of the logo. We decided to be conservative in our threshold and set it a bit on the higher end in the event that the unknown class images have similar characteristics to our logos like our dog image had. With this threshold set, we found that the accuracy of our model was still hitting the 90% accuracy goal that we had.

IV. Conclusions

This project resulted in a machine learning system that was trained to accurately classify ten different brand logos with the resulting model. Through the use of a preprocessing pipeline that reshaped the data into a form that our model could use, in addition to the use of the pre-trained model ResNet-50 we were able to achieve an accuracy of over 90%, specifically, we achieved 95% accuracy on our test set. This accuracy is the

result of tuning hyperparameters and allowing the base model to train its parameters. This could likely be improved upon with further tuning of hyperparameters and techniques beyond this class.

References

[1] S. Devi, V. Kumar, and P. Sivakumar, "A Review of image Classification and Object Detection on Machine learning and Deep Learning Techniques," IEEE, December 2021 [2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA)]

[2] K. Bressem, L. Adams, C. Erxleben, B. Hamm, S. Niehues and J. Vahldiek, "Comparing different deep learning architectures for classification of chest radiographs", Nature, Scientific Reports, vol. 10, 2020

[3] S. Cloud, "How to Implement ResNet in TensorFlow for Deep Learning",saturncloud.io. https://saturncloud.io/blog/how-to-implement-resnet-in-tensorflow-for-deep-learning/ (accessed Jul., 2022)

[4] M. Abadi et al., "tf.keras.applications.resnet50.ResNet50", tensorflow.org. https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50 (accessed Jul., 2022)

[5] M. Abadi et al., "Transfer learning and fine-tuning", tensorflow.org. https://www.tensorflow.org/tutorials/images/transfer_learning (accessed Jul., 2022)

[6] The MathWorks, Inc., "mobilenetv2", mathworks.com. https://www.mathworks.com/help/deeplearning/ref/mobilenetv2.html (accessed Jul., 2022)