Brand Logo Classification

Naga Akhil Belide
Computer and Information Sciences
University of Florida
Gainesville, United States
nagaakhil.belide@ufl.edu

Caleb Bean
Electrical and Computer Engineering
University of Florida
Gainesville, United States
caleb.bean@ufl.edu

Tejaswini Nandu
Electrical and Computer Engineering
University of Florida
Gainesville, United States
t.nandu@ufl.edu

***Abstract*—This paper proposes a brand logo classification system based on Convolutional Neural Networks (CNNs). The system aims to classify brand logos into one of ten pre-defined categories. The proposed CNN model is trained on a large dataset of brand logos. We implemented the model using TensorFlow, and Keras and tested it on a large dataset. The proposed system can be used in various applications, including brand recognition and marketing research. The results of this study demonstrate the effectiveness of CNNs in the context of brand logo classification and provide insights for future research in this field**

***Index Terms*—Convolutional Neural Networks, TensorFlow, Keras, Transfer Learning**

I. Introduction

We propose a brand logo classification system that utilizes convolutional neural networks (CNNs) to classify logos into one of ten pre-defined categories. CNNs have shown remarkable success in image classification tasks and have been widely used in various fields, including computer vision, natural language processing, and speech recognition.

The rest of this paper is organized as follows. In the next section, we describe the proposed CNN model architecture and the dataset used for training and testing. In the following section, we present the experimental results of our proposed approach. Finally, we conclude the paper and discuss the inference we learned from this project.

Transfer learning [1] is a popular technique used in deep learning for image classification tasks. In transfer learning, a pre-trained model is used as the base for a new classification task instead of training a new model from scratch. This allows the new model to leverage the knowledge learned by the pre-trained model on a large dataset, which can significantly reduce the amount of training data required and can lead to improved accuracy. In a multi-label classification project, where each image can be assigned to multiple categories, transfer learning can be particularly useful. The pre-trained model can be used to extract features from the images, and then these features can be used as input to a new classifier that is trained specifically for the multi-label classification task. This approach can help improve the accuracy of the classification task, as the pre-trained model can learn useful features that can be difficult to learn from scratch, especially when there is limited training data available.

II. Implementation

A. Dataset

The dataset used in this study contained an almost equal number of images for each of the ten label types. However, there were several misclassified image labels in the original dataset, which could negatively impact the accuracy of the model during training. To address this issue, we utilized label correction code to correct the labels, resulting in an improvement in the accuracy of the model on the validation set.

Brand Name	Label
Nike	0
Adidas	1
Ford	2
Honda	3
General Mills	4
Unilever	5
McDonald's	6
KFC	7
Gator	8
3M	9

Integer encoding for each brand class.

Given that the size of the dataset for training was not very large, we decided to employ augmentation techniques to increase its size. This technique involved making small changes to the images, such as orientation, flipping, and zooming, to create new training data. Additionally, we encoded the labels using one-hot encoding since we used softmax as the activation function with 10 units in the output layer.

The original size of each input image was (300,300,3); however, we resized them to (224,224,3) since the VGG-16 [7] base model performs better on this size of image. We then divided the dataset into training and validation sets in an 80:20 ratio to train and test the model. Overall, these preprocessing techniques helped to optimize the training process and improve the accuracy of our model.

B. Model Implementation

Our project was implemented using transfer learning, a popular technique in deep learning [2] that leverages pre-trained models to improve the accuracy and efficiency of new models. Specifically, we utilized the VGG-16 model, which had been pre-trained on a large dataset of images, as the base model for our project. We then trained all the weights of this base model with a very small learning rate to preserve its learned features and prevent overfitting.

To create a custom classifier for brand logo classification [3], we flattened the output of the base model. We added a hidden layer with 4096 units, with Rectified Linear Unit (ReLu) as the activation function. This hidden layer helped to extract more complex features from the input images and improve the discriminative power of the model. To classify the brand logos into the ten pre-defined categories, we added an output layer with 10 units, one for each category, and utilized the softmax activation function. This final layer provided the probability distribution of the input image across the ten categories, allowing us to make accurate predictions. We used the categorical-cross-entropy [6] as the loss function and Adam optimizer with a very small learning rate of 1e-5. Trained the entire model with a batch size of 64 for 150 epochs but added an early callback on the val-loss parameters with the patience of 10 epochs.

Overall, by leveraging transfer learning and adapting the pre-trained VGG-16 model to the specific task of brand logo classification, we were able to achieve high accuracy and efficiency in our model.

III. Experimentations

A. Support Vector Machines

In our initial attempts to predict the labels of the images in our dataset we used Support Vector Machines (SVM) we found that the performance was unsatisfactory, with an accuracy of just 18%. Since the original image size was (300,300,3) which is computationally expensive to process, we decided to downscale the images to (64,64,3). We then used PCA to reduce the dimensionality of our dataset, which resulted in a significant reduction in the computational cost. However, since we could not find enough resources to train the model with the original image size while maintaining at least 90% explained variance, we decided to use KernelPCA with 1000 components, as it can handle non-linear data more effectively than PCA.

B. Convolutional Neural Network

We attempted to implement a Convolutional Neural Network (CNN) from scratch. To do this, we began by fine-tuning the different hyperparameters of the network. Our initial architecture consisted of 5 layers of Conv2D, each of which was followed by a MaxPool2D layer. To vary the number of filters and their sizes, we experimented with different values and configurations for each layer. We also tried different activation functions such as ReLu and LeakyReLu and tried GlobalAveragePool2D and Flatten layers after the CNNs. Added a hidden layer with various units like 512, 256, and 128, and then an output layer with a softmax activation function. While we were able to achieve some promising results with our CNN with accuracies being between 65-77% for different models, we found that the network was not performing as well as we had hoped. After further analysis, we realized that our CNN architecture was too simple to effectively capture the underlying features of the images to get an accuracy over the threshold of 90%. So, then decided to experiment with transfer learning.

The project utilized VGG16 as a base model for a classification task, where the number of classes for the project varied from the original architecture and so the top layer of the model was excluded, and the weights of the base model were frozen. We experimented with various configurations, such as the number of hidden layers, the number of neurons in each layer, and the method used to vectorize the CNN output. Additionally, the number of epochs and batch size for training were varied.

Using the GlobalAveragePool2D layer to vectorize the CNN output and adding two dense layers with 512 and 128 units with ReLu activation before the output layer resulted in an accuracy of 86% with the Adam optimizer with a learning rate of default 0.01. However, on just adding 1 Dense layer with 256 units increased the accuracy to 88.5%, while keeping all other hyperparameters constant. Further experimentation involved using a Flatten layer to vectorize the CNN output, which resulted in improved results. The model used a dense layer with 4096 units before the output layer and trained all layers except the last five with an Adam learning rate of 0.01, achieving an accuracy of 91.2% on the validation set. When training all layers with a learning rate of 1e-5, the accuracy improved to 94.62%. To prevent overfitting, early stopping with patience of 10 on the val loss parameter was employed, and the ”restore best weights” parameter was set to True. The final saved model stored the best weights.

C. Model Evaluation

We used accuracy as a metric to evaluate the model and for a multi-label classification problem, it is computed slightly differently than in a single-label classification problem. In a multi-label classification problem, each sample can be associated with multiple labels, so the accuracy is calculated based on the number of correctly predicted labels across all the samples in the test dataset. To compute accuracy in a multi-label classification problem, we first make predictions on the test dataset using the trained model. We then compare the predicted labels to the actual labels of the test dataset and calculate the number of correctly predicted labels across all the samples. Finally, we divide the total number of correctly predicted labels by the total number of labels in the test

dataset to obtain the accuracy score

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

On substituting the values of True Positive(TP), True Negatives(TN), False Positive(FP), and False Negatives(FN) we can calculate the accuracy of the model. TP, TN represents the number of correctly predicted labels

D. Results

The graph below illustrates how four different parameters, namely loss, accuracy, val-loss, and val-accuracy, changed as the number of epochs increased. It can be observed that as the number of epochs increased, the loss and val-loss decreased, while the accuracy and val-accuracy increased. However, after a certain point, the val-loss started to increase, and the val-accuracy started to decrease. This is a common occurrence when the model begins to overfit the training data, and further training leads to poor performance on the validation set. To address this issue, the early callback option was utilized, which helps stop the training process when the performance on the validation set does not improve for a certain number of epochs, thus preventing overfitting. The early callback option has a "patience" parameter, which specifies the number of epochs to wait before stopping the training process. This approach can be particularly helpful when training deep learning models, where overfitting can occur frequently, and it is essential to prevent it to achieve the best performance on the test set.

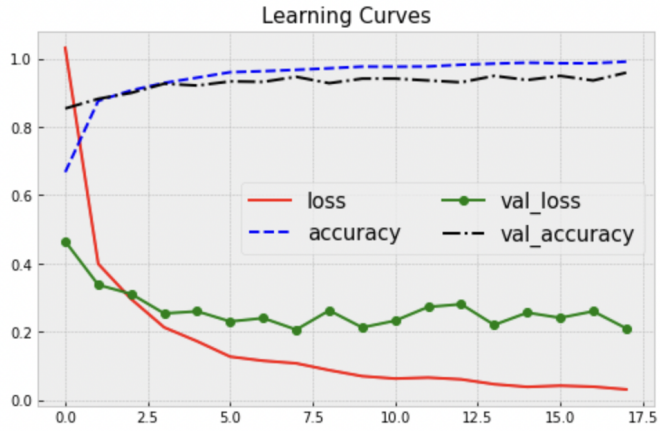

Fig. 1. Model Performance vs epoch

In addition to the previous graph, we have plotted another graph that clearly shows the accuracy and val-accuracy as the number of epochs increases. It can be observed that the accuracy on the training set continues to increase as the number of epochs increases, while the val-accuracy on the validation set initially increases, but after a certain point, it starts to decrease. This is a clear indication that the model is overfitting the training data, and further training can lead to poor performance on the validation set. It is important to note that achieving high accuracy on the training set alone does not necessarily mean that the model is performing well on unseen data. Therefore, it is crucial to monitor the performance on the validation set and prevent overfitting to achieve the best performance on the test set. The graph serves as a visual representation of how the accuracy and val-accuracy vary with epochs and can help in determining the appropriate number of epochs to train the model for optimal performance.

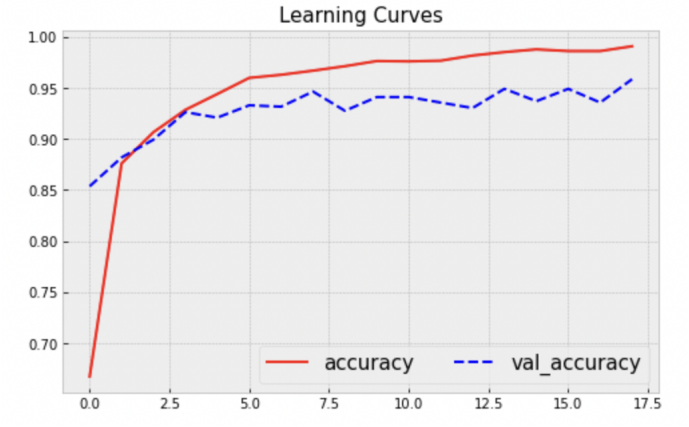

Fig. 2. Model Performance vs epoch

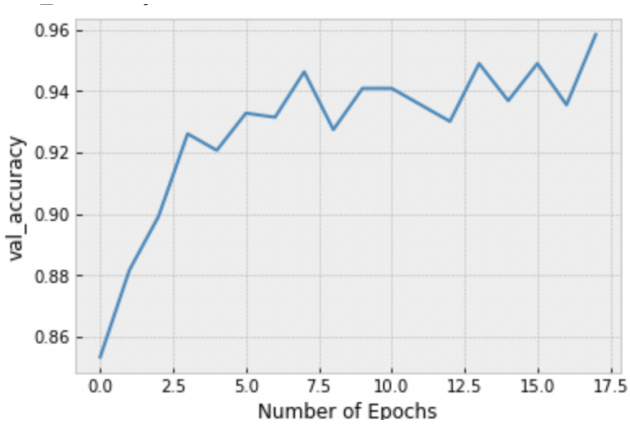

Fig. 3. val-accuracy vs epoch

To validate the performance of the final model on unseen data, we split a portion of the given data as a test set, which accounted for 20% of the total data. We then split the remaining data for training and validation purposes. Upon evaluating the model on the test data, we achieved an accuracy of approximately 94%, which suggests that the model is performing well on previously unseen data. To further analyze the performance of the model, we have provided a classification report for the test data, which is illustrated in the attached image. The classification report provides insights into the precision, recall, and F1-score for each class in the multi-label classification problem. It is a useful tool for evaluating the performance of the model on each individual class and identifying any potential issues or imbalances in the data. Overall, the high accuracy achieved on the test data suggests

that the model is performing well and can be considered reliable for predicting the labels of new, previously unseen data.

	precision	recall	f1-score	support
Nike	0.88	0.97	0.93	76
Adidas	0.98	0.85	0.91	74
Ford	0.97	0.95	0.96	75
Honda	0.97	0.96	0.97	74
General Mills	0.97	0.97	0.97	74
Unilever	0.95	1.00	0.97	75
McDonalds	0.86	0.96	0.91	73
KFC	0.97	0.93	0.95	74
Gator	1.00	0.97	0.99	74
3M	0.93	0.91	0.92	75
accuracy			0.95	744
macro avg	0.95	0.95	0.95	744
weighted avg	0.95	0.95	0.95	744

Fig. 4. Classification report on test data

IV. Conclusion

In conclusion, we have successfully applied transfer learning using the VGG16 model for a multi-label classification problem. Through our experiments with various hyperparameters, we learned how the choice of the number of layers, activation functions, learning rate, and method for vectorizing the CNN output can have a significant impact on the accuracy of the model. We also observed that early stopping on the val-loss parameter helped prevent overfitting and improve the generalization performance of the model. By training the model, we were able to achieve a high accuracy of 94.62% on the validation dataset, demonstrating the effectiveness of our approach. Overall, this project has provided us with valuable insights into the process of applying transfer learning to solve real-world multi-label classification problems and involved experimentation with various configurations, hyperparameters, and optimization techniques to improve the accuracy of the model. It highlights the importance of careful hyperparameter tuning and the need to balance model complexity and generalization performance. The skills and knowledge gained from this project will be useful in future endeavors in the field of deep learning and computer vision.

References

[1] "Transfer learning" by Y. Bengio, J. Dean, and M. O. Rabinovich (2012)

[2] "Transfer Learning for Image Classification with Few Training Examples" by K. Saenko, B. Kulis, M. Fritz and T. Darrell (2010)

[3] S. Saha, S. Chowdhury, and M. Nasipuri, "Logo Detection and Recognition Using Deep Convolutional Neural Network," in Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA), vol. 496 of Advances in Intelligent Systems and Computing, pp. 391–402, Springer, New Delhi, India, 2016.

[4] N. Zahid, A. M. Mirza, and J. Ahmad, "Deep Learning Based Logo Detection and Recognition Using Transfer Learning," in Proceedings of the 2019 2nd International Conference on Computer Applications And Information Security (ICCAIS), pp. 1–5, IEEE, Rawalpindi, Pakistan, 2019.

[5] S. Mohapatra and S. Kumar, "Logo Detection and Classification using Deep Convolutional Neural Network," in Proceedings of the 2018 4th International Conference on Computing Communication Control and Automation (ICCUBEA), pp. 342–347, IEEE, Dehradun, India, 2018.

[6] "On the Usefulness of Cross-Entropy Loss for Classification Problems" by A. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio and S. Lacoste-Julien (2017)

[7] "Very Deep Convolutional Networks for Large-Scale Image Recognition" by K. Simonyan and A. Zisserman (2015): This is the original paper that introduced the VGG16 architecture for image classification.

[8] "Visualization of Deep Convolutional Neural Networks for Classification of Histology Images" by K. Sirinukunwattana, J. P. Pluim, H. Chen, X. Qi, P. Heng and Y. Guo (2016): This paper uses VGG16 to classify histology images and also visualizes the intermediate feature maps to gain insights into the inner workings of the network.