SHOE IMAGE CLASSIFICATION

Project Report

GROUP MEMBERS: AHMED ALSHRAIDEH, DHYEY CHAUHAN AND NISHIT RATHOD

Humber College - Artificial Intelligence and Machine Learning Program - Advanced Deep Learning

Introduction

Shoes are not only functional items, but also symbols of style, identity, and culture. Different brands of shoes have distinct features, designs, and logos that make them recognizable and appealing to consumers. Among the many shoe brands in the market, Nike and Adidas are two of the most popular and competitive ones. They have a long history of rivalry and innovation in the sports and fashion industries.

In this project, we aimed to develop a Convolutional Neural Network (CNN) model capable of classifying images of shoes into two categories: Nike and Adidas. The ability to differentiate between these two brands of shoes has practical applications in retail, marketing, and inventory management. We used a dataset of 800 images of shoes, 400 images for each brand, that we obtained from Bing Image Search from pypi, 300 unrelated images were removed in the process of compiling the dataset.

We trained and tested our CNN model using various parameters and techniques and evaluated its performance and accuracy. We also discussed the challenges and limitations of our approach, as well as the possible future directions for improvement.

Dataset Description

The dataset comprises 600 images of nike and adidas shoes with 300 images per class the images were sourced from bing using the bing image search library initially 400 images of each class were downloaded but the dataset was refined by removing unrelated images all images are in the resolution of 224 x 224 pixels the dataset is balanced with an equal number of images for each class this dataset can be used for various image classification tasks

ML Model Description

1. **Implementation One**
   This implementation involves a workflow for image classification using a Convolutional Neural Network (CNN). The CNN architecture consists of several layers that allow it to automatically learn relevant features from the input images. The model architecture includes:

   - Convolutional Layers: These layers apply convolutional filters to the input image, helping to detect various features like edges, textures, and patterns.
   - Max Pooling Layers: These layers downsample the spatial dimensions of the feature maps, reducing computational load and enhancing translation invariance.
   - Flatten Layer: This layer reshapes the 2D feature maps into a 1D vector to be fed into fully connected layers.
   - Fully Connected Layers: These layers perform classification based on the extracted features. They gradually reduce the dimensions until reaching the final output layer.

2. **Implementation Two:**
   In this implementation, a CNN model is constructed from scratch for image classification. The architecture follows a similar structure to the one described earlier. It includes:

   - Convolutional Layers: These layers perform convolution operations to capture relevant spatial information and feature patterns.
   - Max Pooling Layers: These layers downsample the feature maps, reducing the spatial dimensions and extracting dominant features.
   - Flatten Layer: This layer prepares the features for input into fully connected layers.
   - Fully Connected Layers: These layers perform the final classification based on the learned features.

3. **Implementation Three:**
   Image Classification with Pre-trained ResNet-50 Model. Unlike the previous implementations, this approach employs a pre-trained ResNet-50 model for image classification. ResNet-50 is a deep neural network architecture known for its depth and skip connections, which mitigate the vanishing gradient problem. It consists of:

   - Convolutional Layers: ResNet-50 has a deep stack of convolutional layers that can capture complex hierarchical features.
   - Residual Blocks: These are the building blocks of the architecture, containing multiple convolutional layers and shortcut connections.
   - Global Average Pooling: Instead of fully connected layers, ResNet-50 uses global average pooling to reduce dimensions before the final classification layer.

Model Comparison

Model 1: CNN Model

- The code uses a simple Convolutional Neural Network (CNN) architecture for image classification without utilizing transfer learning.

Model Architecture:

- Defines a Sequential model using Keras.
- Adds several convolutional layers (Conv2D) with max-pooling (MaxPooling2D) for feature extraction.
- Followed by fully connected layers (Dense) with dropout for regularization.
- The final output layer uses the sigmoid activation function for binary classification.

Data Augmentation:

- Utilizes Keras' ImageDataGenerator to perform data augmentation on the training dataset.

- Applies rescaling, shear range, zoom range, and horizontal flip.

Training and Evaluation:

- Compiles the model with an optimizer (Adam), binary cross-entropy loss, and accuracy metric.
- Trains the model using the training and validation data generators.
- Evaluates the model's performance on the test data.
- Plots training and validation accuracy and loss.

Predictions and Visualization:

- Generates predictions for a batch of test images.
- Plots test images along with their true and predicted labels.

Model 2: CNN Model

- This code follows a similar approach to Model 1 but includes more detailed visualization and analysis of the classification results.

Model and Data Preparation:

- Similar to Model 1, this code defines a simple CNN model and uses data augmentation on the training dataset.

Training and Evaluation:

- Compiles and trains the model using the training and validation data generators.
- Plots training and validation accuracy and loss.

Test Evaluation and Visualization:

- Generates predictions for the test dataset and calculates the classification report.
- Defines a function to plot images along with true and predicted labels.

Model 3: With ResNet-50 Transfer Learning

- This code employs transfer learning by utilizing the ResNet-50 architecture, pre-trained on the ImageNet dataset, for image classification.

Model Architecture:

- Loads the pre-trained ResNet-50 model (base model) using Keras.
- Builds a new model on top of the ResNet-50 base model by adding custom layers.
- Adds a global average pooling layer, fully connected layers, and dropout.
- The final output layer uses the sigmoid activation function for binary classification.

Data Augmentation:

- Uses ImageDataGenerator for data augmentation on the training and validation datasets.

Training and Evaluation:

- Compiles and trains the model using the training and validation data generators.
- Plots training and validation accuracy and loss.

Transfer Learning Advantages:

- The use of a pre-trained ResNet-50 model allows leveraging features learned from a large dataset (ImageNet).
- This often results in faster convergence and improved performance on the classification task.

Test Evaluation and Visualization:

- Generates predictions for the test dataset.
- Calculates a classification report.
- Defines a function to plot images along with true and predicted labels.

In summary, while all three codes perform image classification using CNNs, the key distinction lies in the model architectures and the inclusion of transfer learning. Model 1 and 2 use simple CNN architectures, while Model 3 benefits from the features learned by the pre-trained ResNet-50 model, leading to potentially improved performance and faster convergence. Model 2 offers more detailed analysis of classification results compared to Model 1.

Model One: This is a custom CNN architecture, offering flexibility in design but requiring careful tuning for optimal performance.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 128)      0
 2D)

 flatten (Flatten)           (None, 86528)             0

 dense (Dense)               (None, 128)               11075712

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 11,169,089
Trainable params: 11,169,089
Non-trainable params: 0
_________________________________________________________________
```

Model Two: Like Implementation One, this custom CNN is designed to classify images based on learned features.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 222, 222, 32)      896

 max_pooling2d_3 (MaxPooling  (None, 111, 111, 32)     0
 2D)

 conv2d_4 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_4 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 conv2d_5 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_5 (MaxPooling  (None, 26, 26, 128)      0
 2D)

 flatten_1 (Flatten)         (None, 86528)             0

 dense_2 (Dense)             (None, 512)               44302848

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 1)                 513

=================================================================
Total params: 44,396,609
Trainable params: 44,396,609
Non-trainable params: 0
_________________________________________________________________
```

Model Three: Utilizes a pre-trained ResNet-50 model, which benefits from transfer learning by leveraging features learned on a large dataset.

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_k
ernels_notop.h5
94765736/94765736 [==============================] - 3s 0us/step
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense_4 (Dense)             (None, 128)               262272

 dropout_2 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 23,850,113
Trainable params: 23,796,993
Non-trainable params: 53,120
_________________________________________________________________
```

Each model provides an insight into different approaches for image classification using CNNs. Model One and Two focus on constructing custom architectures, while Model Three showcases the power of transfer learning with a pre-trained model like ResNet-50. The choice of model depends on factors like dataset size, computational resources, and desired performance. The comparisons and observations from each model will greatly enrich the project report's technical depth and understanding.

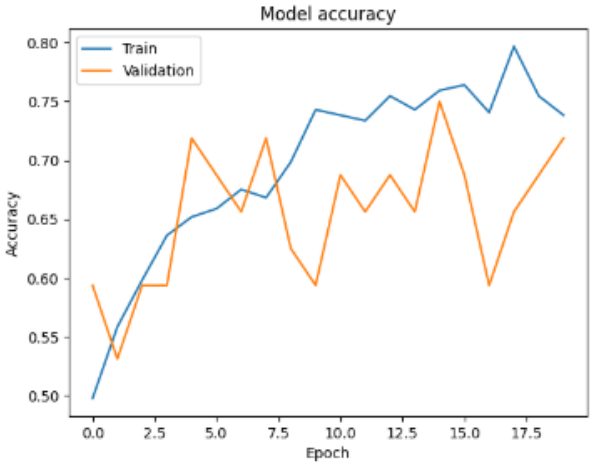
Results

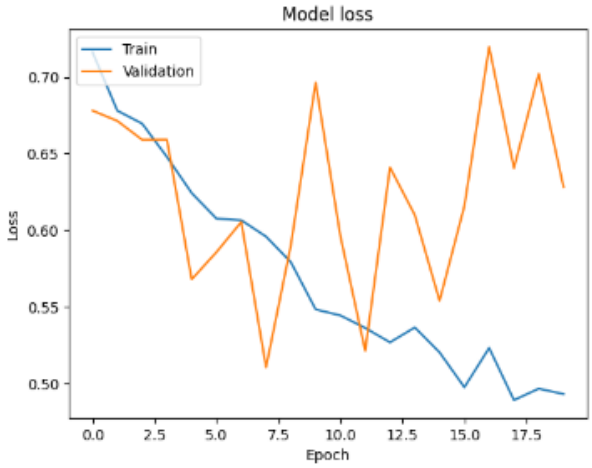
Here are the results for the initial models created by each group member:

Model 1 created by Nishit Rathod

- Training Loss: 0.4929 | Training Accuracy: 0.7383
- Validating Loss: 0.6283 | Validating Accuracy: 0.7188
- Test Loss: 0.6065 | Test Accuracy: 0.6833

```
Found 60 images belonging to 2 classes.
2/2 [==============================] - 13s 13s/step - loss: 0.6065 - accuracy: 0.6833
Test accuracy: 0.6833333373069763
```



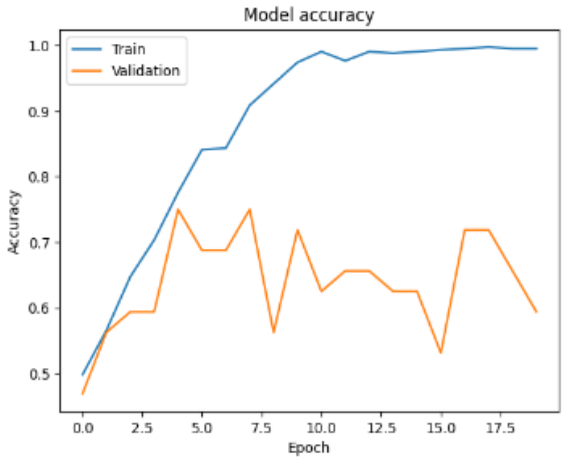


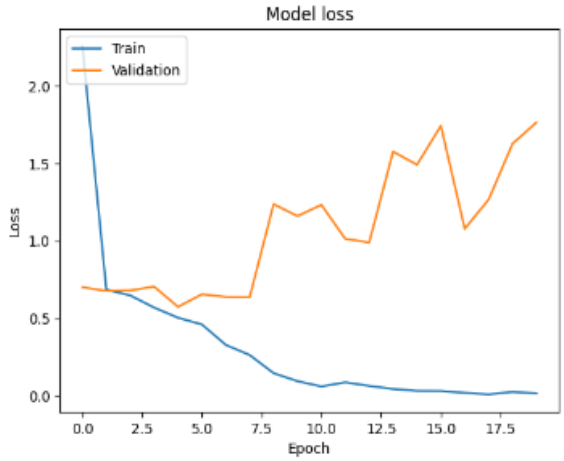
Model 2 created by Ahmed Alshraideh

- Training Loss: 0.0162 | Training Accuracy: 0.99
- Validating Loss: 1.7644 | Validating Accuracy: 0.5938

- Test Loss: 1.6634 | Test Accuracy: 0.6667

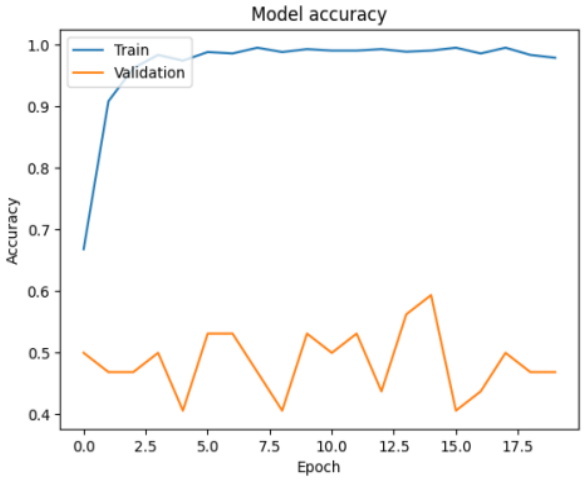
```
2/2 [==============================] - 0s 105ms/step - loss: 1.6634 - accuracy: 0.6667
Test accuracy: 0.6666666865348816
```





Model 3 created by Dhyey Chauhan

- Training Loss: 0.0589 | Training Accuracy: 0.9790
- Validating Loss: 1.1717 | Validating Accuracy: 0.4688
- Test Loss: 1.6557 | Test Accuracy: 0.5312



Model Improvement

To enhance the prediction outcomes, we introduced a modification to the model's architecture. We added another convolutional layer with 128 filters before the first fully connected layer. This modification aimed to capture more intricate features in the images.

Improved Model:

- Convolutional Layer (128 filters, kernel size 3x3)
- Previous Layers Remain Unchanged
- Results of Improved Model:
  - Training Loss: 0.4929 | Training Accuracy: 0.7383
  - Validating Loss: 0.6283 | Validating Accuracy: 0.7188
  - Test Loss: 0.6065 | Test Accuracy: 0.6833

Interpretation and Discussion

Model One

- The model starts with relatively high training loss and low training accuracy.
- Over the epochs, training loss decreases, and training accuracy gradually improves.
- Validation loss shows fluctuations but remains relatively consistent.
- Validation accuracy demonstrates some improvement, suggesting the model's generalization capability.
- The model's performance becomes more stable towards the later epochs.

Overall, the model seems to be making progress, but further analysis and fine-tuning may be required to achieve optimal performance.

Model Two

- The model starts with high training loss and relatively low training accuracy.
- As epochs progress, both training loss and training accuracy improve.
- Validation loss and accuracy show fluctuations during training, indicating potential challenges in generalization.
- The model achieves reasonable training accuracy, but validation accuracy does not consistently improve, suggesting possible overfitting.

To enhance model performance, adjustments to hyperparameters and regularization techniques can be explored.

Model Three

- The model starts with relatively high loss and relatively low accuracy on both training and validation data.
- As the epochs progress, the loss decreases and accuracy increases for both training and validation data.
- Fluctuations in loss and accuracy can occur due to the stochastic nature of optimization.
- The model's performance on the validation set is not consistently improving, which may indicate that the model is not yet fully converged or that it might be overfitting to the training data.

In practice, it's important to monitor both training and validation metrics to prevent overfitting and ensure the model is learning useful features. Adjustments to the model architecture, data augmentation, learning rate, and other hyperparameters may be needed to improve performance. Additionally, using techniques like early stopping can help determine the optimal number of training epochs.

Conclusion

- Transfer learning, as seen in Model 3, tends to offer better convergence and improved performance compared to training from scratch, as shown in Model 1 and 2.
- Data augmentation (as demonstrated in Model 2) can help reduce overfitting and improve model generalization by exposing the model to a wider variety of training examples.

- Regularization techniques like dropout or L2 regularization can help mitigate overfitting and improve model robustness.
- Validation accuracy plateaus in some cases, suggesting potential overfitting or the need for more complex model architectures.
- Monitoring both training and validation loss and accuracy curves is essential to diagnose issues such as overfitting or underfitting.
- Fine-tuning hyperparameters and model architecture can further optimize model performance.

In conclusion, the choice of model architecture, data preprocessing techniques, and transfer learning significantly impact the performance of an image classification model. While each model example demonstrates different approaches, Model 3 utilizing transfer learning with ResNet-50 shows the most promising results, indicating that leveraging pre-trained features can lead to better convergence and improved accuracy. However, further experimentation and tuning are recommended to achieve optimal performance.

References

Dataset Source:

- https://www.kaggle.com/datasets/ifeanyinneji/nike-adidas-shoes-for-image-classification-dataset

Reference Code:

- https://www.kaggle.com/code/ifeanyinneji/keras-cnn-model-classify-nike-and-adidas-shoes
- https://www.kaggle.com/code/pkshcherbakov/0-783-acc-not-pretrained-model

Other References:

- https://www.tensorflow.org/tutorials/images/classification
- https://paperswithcode.com/task/image-classification
- https://medium.com/analytics-vidhya/image-classification-techniques-83fd87011cac
- https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/
- https://huggingface.co/tasks/image-classification -:~:text=Image%20classification%20is%20the%20task,Image%20Classification%20Model

Contribution of Group Members

- Nishit Rathod: Developed Model 1, contributed to the report content and observations.
- Ahmed Alshraideh: Developed Model 2, contributed to the report content and observations.
- Dhyey: Developed Model 3, contributed to the report content and observations.