**A blue shield with a cross and a book

Description automatically generated**

**Project 1 For Diploma in AI and its application in Business: Image Classification on Food101 Dataset**

**Project Report**

*Example Usage of Transfer Learning and Analysis on Learning Rate*

Prepared by: Chen Yijie

Supervisor: Zhou Changxin

Table of Contents

[Project Overview: 3](#_Toc163579323)

[Covering Concepts: 3](#_Toc163579324)

[Potential Use Case: 3](#_Toc163579325)

[Environment Used: 4](#_Toc163579326)

[Dataset Used: 4](#_Toc163579327)

[Preprocessing: 5](#_Toc163579328)

[Model Selection: 6](#_Toc163579329)

[Training Parameter: 7](#_Toc163579330)

[Initial Training: 8](#_Toc163579331)

[Overfitting Address: 9](#_Toc163579332)

[Intermediate Result: 11](#_Toc163579333)

[Learning Rate: 12](#_Toc163579334)

[Final Results: 16](#_Toc163579335)

[Potential Improvements to be Done: 17](#_Toc163579336)

[Conclusion: 17](#_Toc163579337)

## **Project Overview:**

The project aimed to utilize transfer learning with a ResNet model to predict food items from images. Transfer learning involves using pre-trained models on large datasets and fine-tuning them for specific tasks, thereby leveraging their learned features to improve performance on smaller datasets.

## **Covering Concepts:**

1. Convolutional Neural Network
2. Image Classification: Assigning labels to images based on their content.
3. Transfer Learning: Leveraging pre-trained models to boost performance on specific tasks.
4. ResNet: A deep convolutional neural network architecture known for its effectiveness in image classification tasks.
5. Overfitting Techniques
6. Learning Rates

## **Potential Use Case:**

A potential use case for this image classification model could be in a mobile application designed to assist individuals with dietary restrictions or preferences. For instance, consider a scenario where a person has specific dietary requirements due to health reasons, such as allergies or intolerances, or personal preferences like vegetarianism or veganism.

In this use case, the image classification model could be integrated into a mobile app that allows users to take photos of food items or scan barcodes of packaged products. The model would then analyze the images and provide instant feedback on whether the item aligns with the user's dietary restrictions or preferences.

For example, if a user with a gluten intolerance takes a photo of a meal, the model could quickly identify whether the dish contains any gluten-containing ingredients. Similarly, a user following a vegetarian diet could use the app to verify whether a product contains animal-derived ingredients.

This application would empower users to make informed decisions about their food choices, ensuring they adhere to their dietary requirements or preferences, even when faced with unfamiliar or ambiguous food items. Additionally, the app could provide personalized recommendations or alternative suggestions based on the user's dietary profile, further enhancing their dining experience and overall well-being.

## **Environment Used:**

Python programming language with:

1. tensorflow = ^2.15.0

2. keras = ^2.15.0

3. tensorflow\_datasets = ^4.9.4

4. tensorboard = ^2.15.1

5. numpy = ^1.26.3

6. matplotlib = ^3.8.2

7. tensorflow\_addons = ^0.23.0

## **Dataset Used:**

The dataset used for this project is the Food-101 dataset. The Food-101 dataset is a collection of 101,000 labeled food images, covering 101 different food categories. It serves as a benchmark dataset for food recognition and classification tasks in computer vision research. Each image in the dataset is associated with a single food category, providing a diverse range of food items for model training and evaluation.



Fig 1. Sample Images from the Dataset

## **Preprocessing:**

For preprocessing, images are standardized to a size of 256x256 pixels. Standardizing the image size ensures uniformity across the dataset, facilitating effective training and consistent performance. ResNet architectures require fixed-size inputs to maintain consistency in operations. A larger image size provides sufficient detail without overwhelming computational resources.

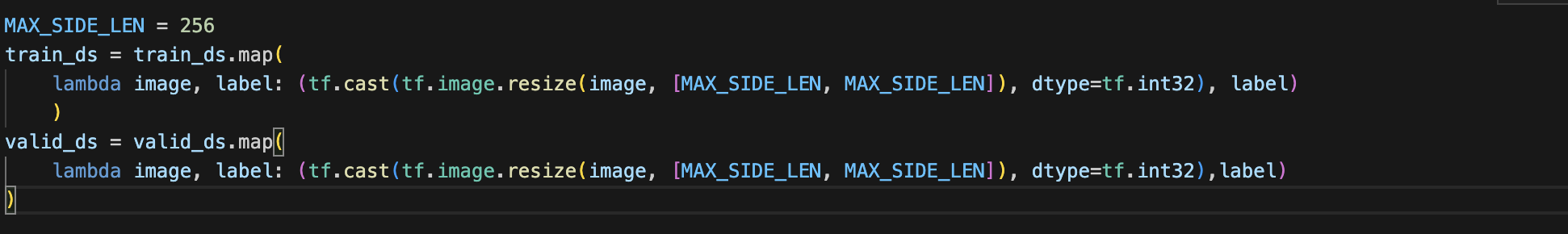


Fig 2. Codes Used to Preprocess the Data

## **Model Selection:**

For the model selection, a pre-trained ResNet from ImageNet with a fully connected dense layer for prediction has been chosen. ResNet, short for "Residual Network," is a deep convolutional neural network architecture renowned for its effectiveness in image classification tasks. One of its notable features is the incorporation of residual connections, which facilitate the training of extremely deep networks by mitigating the vanishing gradient issue. These connections enable the gradients to flow more effectively during backpropagation by allowing shortcuts to bypass one or more layers.

Utilizing a pre-trained ResNet model from ImageNet offers several advantages. ImageNet is a vast dataset comprising millions of labeled images across thousands of categories. By pretraining the ResNet model on ImageNet, it can learn generic features and patterns from a diverse range of images. Leveraging this pre-trained model allows us to capitalize on the knowledge gained during its training, which can then be fine-tuned for our specific food classification task.

In addition to the ResNet backbone, a fully connected dense layer is appended for prediction. This dense layer acts as the classifier, taking the features extracted by the ResNet backbone and mapping them to the output classes corresponding to the different food categories. The inclusion of this dense layer enables the model to learn task-specific representations tailored to our food classification problem. For the consideration of balancing computing resources, only the fully connected dense layer will be trained.

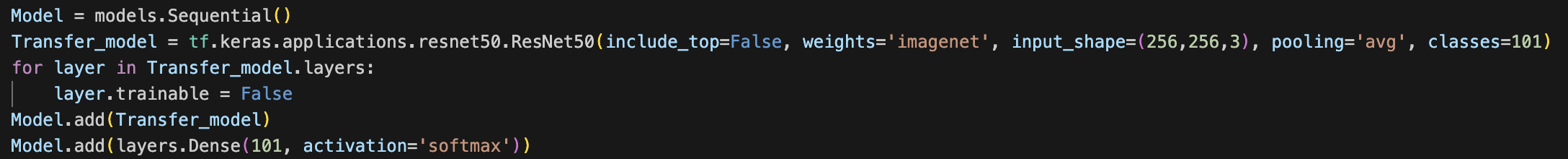
In summary, the combination of a pre-trained ResNet from ImageNet with a fully connected dense layer for prediction offers a strong foundation for image classification tasks, particularly for the Food-101 dataset. The structure's ability to extract relevant features, leverage transfer learning, fine-tune for specific tasks, and perform accurate classification makes it well-suited for the challenges posed by food image recognition.

Fig 3. Codes that Sets up the Model Calling ResNet

A diagram of a computer

Description automatically generated

Fig 4. ResNet-backboned Model Structure

## **Training Parameter:**

A batch size of 16 strikes a balance between stochasticity and stability during training, promoting efficient gradient estimation and generalization. Autotune prefetch feature will be incorporated into data pipeline, Autotune prefetch dynamically adjusts the number of input data samples to prefetch, optimizing data loading and processing parallelism on the GPU.

Pooling methods used for ResNet Structure is set to average instead of max pooling. Average pooling samples all pixel values within a filtered window, smoothing spatial variations and providing more generalized representations, this ensures maximum retaining of pixel information in sacrifice of feature extraction ability which is not as important in

These training parameter choices aim to create a stable and effective training environment, facilitating the model's ability to learn relevant features from food images while ensuring computational efficiency and reproducibility.

A black background with white text

Description automatically generated

Fig 5. Codes that Specify Batch Size and Setting Prefetch

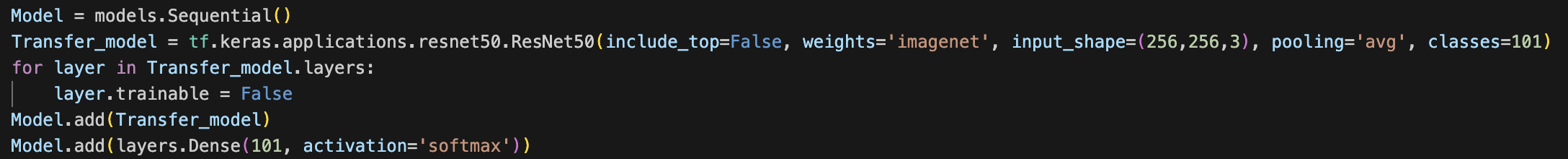


Fig 6. Codes that Specify Hyperparameters in the Model

## **Initial Training:**

The model achieved a baseline accuracy of approximately 56.26% after 4 episodes of training, and it depletes to an accuracy of 55.07% after all 15 episodes of training.

A graph of a line

Description automatically generated

Fig 7. Initial Training Results

The validation result fluctuates and even decreases towards the end of training while training accuracy goes higher and higher. The decrease in performance suggests the occurrence of overfitting. Overfitting happens when a model learns to memorize the training data instead of generalizing well to unseen data. In the initial stages of training, the model's performance on the training data steadily improves as it learns to capture relevant patterns and features. However, as training progresses, the model may start to overfit the training data, becoming overly specialized and losing its ability to generalize to new, unseen examples.

## **Overfitting Address:**

To address the overfitting problem, a block that augments data containing multiple layers is introduced before the ResNet Structure. The block consists of the following:

1. RandomFlip("horizontal\_and\_vertical"): This augmentation randomly flips images horizontally and vertically, making the model invariant to image orientation, reducing overfitting.
2. RandomRotation(0.2): Randomly rotating images up to 0.2 radians introduces viewpoint variations, aiding the model in recognizing objects from different angles and mitigating overfitting.
3. RandomZoom(0.2): Randomly zooming into or out of images by a factor of 0.2 alters object scales, helping the model focus on relevant features at different resolutions, reducing overfitting.
4. RandomContrast(0.2): Randomly adjusting image contrast by 0.2 enhances or diminishes feature visibility, reducing sensitivity to brightness levels and mitigating overfitting.
5. lambda(preprocess\_input): Normalizes input images by subtracting the mean and dividing by the standard deviation, preparing them for compatibility with ResNet models.

These data augmentation techniques introduce diversity and variability to the training data, helping the model generalize better to unseen examples and mitigate overfitting. By exposing the model to a wider range of variations during training, data augmentation encourages the learning of more robust and invariant representations, ultimately improving the model's performance on unseen data.

A screen shot of a computer program

Description automatically generated

Fig 8. Code Implementation on Data Augmentation Block

A diagram of a function

Description automatically generated

Fig 9. Updated Model’s Structure With a Data Augmentation Block Before ResNet

## **Intermediate Result:**

The model achieved an accuracy of 0.6832 after training over 10 episodes, which is a huge improvement.

A graph with blue and orange lines

Description automatically generated

Fig 10. Training Result for Training with Data Augmentation

From the graphs, we can tell that the model is converging to a point from the decreasing gradient. To continue finetuning the model and for the purpose of learning, we will further explore the effect of learning rate on model training.

## **Learning Rate:**

The learning rate is a crucial hyperparameter in training deep learning models that controls the size of the step taken during gradient descent optimization. It determines the magnitude of parameter updates at each iteration, influencing the convergence speed and stability of the training process.

The optimizer used for the project is Adam optimizer from Keras, the default learning rate is 1e-4. To understand the effect of learning rates, adjustments, and callback techniques are used to continue training from the new model.

From episode 11 to episode 15, an even lower learning rate of 1e-5 is selected. The result shows that the validation accuracy increases steadily from 0.6864 to 0.6892. While the result is still increasing, the convergence is happening too slowly.



Fig 11. Code Implementation to Set Learning Rate

For the next 10 episodes of training, learning rate callbacks are implemented to increase the learning rate constantly using the call-back function:A computer screen with text

Description automatically generated

Fig 12. Code Implementation Utilizing Callbacks to Set Learning Rate

The observation is such that the validation accuracy shows a further increase from 0.6892 to 0.6926 in the first 4 episodes, and afterward the validation result starts to oscillate and does not achieve any higher accuracy. This shows that the model is close to convergence, but it is unable to due to the high learning rate. A screenshot of a computer program

Description automatically generated

Fig 13. Oscillating Training History Visualization / Training History Visualization to Compare the Effect of Different Learning Rates

From episode 26 to 30, a fixed learning rate of 8.9125e-5 is used to continue training. The learning rate is selected among the previous 10 episodes as the better-performing learning rate striking a balance between converging speed and addressing the issue of oscillation. The model accuracy increases from 0.6929 to 0.6978 but shows trends of oscillation during the last 2 episodes, showing that the learning rate is no longer appropriate as the model converges further.

A screenshot of a computer program

Description automatically generated

Fig 14. Oscillating Training History Towards the End

For the final 10 episodes, an even lower learning rate of 1e-5 is used again to fine-tune the performance of the model. Model accuracy further increases from 0.6979 to 0.7017.

A screenshot of a computer program

Description automatically generated

Fig 15. Final Training History with 1e-5 Learning Rate

A graph with numbers and a line

Description automatically generated

Fig 16. Learning Rate used Over All Training Episodes

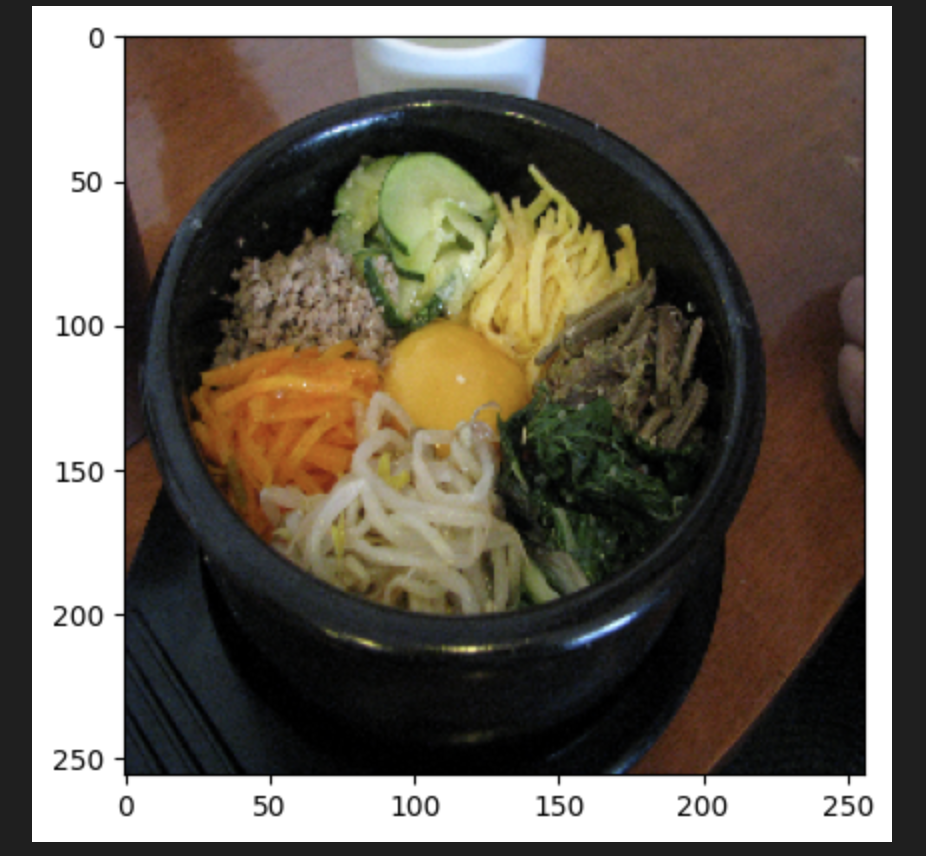
A graph with a line

Description automatically generated

Fig 17. Validation Accuracy Across All Training Episodes

## **Final Results:**

The model achieved a validation accuracy of 70.17%, surpassing the baseline. Testing on a random food image from the internet resulted in correct classification within 971ms, suitable for mobile app integration. However, for real-time classification, exploring faster architectures like VGG or MobileNet could enhance the performance on video classification.



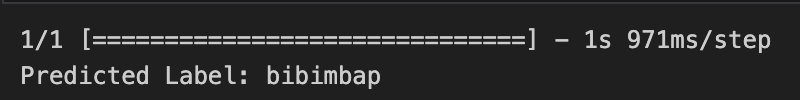


Fig 18&19. Model Testing with Online Photo

## **Potential Improvements to be Done:**

1. Experiment with different pre-trained models: Explore the performance of other architectures like VGG or Inception or a more complex ResNet like ResNet101. Architecture like MobileNet can also be explored to achieve faster prediction in sacrifice of accuracy.
2. Hyperparameter tuning: Further optimize model performance by adjusting parameters such as batch size, and optimizer choice.
3. Open the last few layers of ResNet to training instead of only training the dense layer to finetune the model for specific tasks.
4. Use an even smaller learning rate to further finetune the model, until no obvious improvements can be observed.

## **Conclusion:**

In conclusion, this project successfully utilized transfer learning with a ResNet model to predict food items from images, achieving a significant improvement in classification accuracy on the Food-101 dataset. By leveraging the pre-trained ResNet model from ImageNet and employing data augmentation techniques, overfitting was effectively mitigated, leading to better generalization performance. The exploration of learning rates further optimized model convergence, demonstrating the importance of hyperparameter tuning in deep learning tasks. Despite achieving a classification accuracy of 70.17% on the validation set, there remain opportunities for improvement, such as experimenting with different pre-trained models, fine-tuning hyperparameters, and expanding the dataset. Overall, this project provides valuable insights into the application of transfer learning and deep learning techniques for food image classification tasks, paving the way for future advancements in this field. Top of Form

Bottom of Form