Deep Learning Assessment 2 - Computer Vision Final report

Ronik Jayakumar
24680264

1.0 Introduction:

The Deep Learning Assessment 2 is a project based on Computer Vision. In this project, model assessment has been worked on to check the computers overall efficiency in image classification. We explore into transfer learning in the context of food classification using 3 popular Convolutional Neural Network architectures, namely - GoogLeNet, MobileNet V3, and NasNet.
The aim is to train these models onto our dataset and compare their overall performance and efficiency in accurately classifying images from the given dataset.

The overall project has been split into two parts:
Part A - Training all the three specified models using Transfer Learning
Part B - Fine tuning the best performing model from part A

2.0 Dataset:

The dataset used in this project is titled the Food101 dataset which has been downloaded off the internet. The link for the same is as follows - http://data.vision.ee.ethz.ch/cvl/food-101.tar.gz

The dataset consists of the image and the meta folders -
- The image folder consists of 101 types of food classified into their various subfolders. Each subfolder consist of 1000 images of that particular food type.
- The meta folder consists of text and JSON files specifying the class names along with the training and testing file names.

These have been preprocessed to link them together to create a directory for all images to be classified and trained.

3.0 Data Loading and Preparation:

Loading:

The coding platform used for the project was Google Colab. The reasoning for this was the versatility of Colab where it allows you to run bash commands on the platform. This reduced overall processing time. An overview of steps taken are as follows:

- Download the dataset onto colab using the !wget function.
- Load the dataset onto the notebook by using the !tar -xvf function
- Using the change directory (%cd) and list all function (!l), we list all the available image subfolders in the image folder.

Preparation:

Data preparation has been performed In a few steps. The performed steps have been highlighted below:

- The data is first split into its respective classes by associating the image folder with the classes.txt file available in the meta folder. This gives us the 101 folders representing 101 classes.
- The image folders are further split into training and testing data using the train.txt and test.txt files that are available in the meta folder. This gives us an overall split of 750 images per class as training data and 250 images per class as testing data.
- An Early Stopping class has also been created in the preparation phase as a precautionary measure during model training. This is to ensure that models that are not showing signs of

improvement automatically stop from going ahead with further epochs thereby saving time and computation resources.

4.0 Part A

Part A consists of the main transfer learning portion where pre-trained CNN models are imported to train on the given dataset. The 3 models that have been worked on will be explained systematically along with key learnings, problems faced, technical considerations, and final results.

Note, for part A, since we are merely doing a model assessment, **5 classes will be considered for training and testing**. Therefore we are using a total of 3,750 training images and 1,250 testing images split over 5 different class labels.

4.1 GoogLeNet

History:

The GoogLeNet model was created by a team at google consisting of Christian Szegedy along with his team of 8 in the year 2014 for the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The network was designed with computational efficiency and practicality in mind, so that inference can be run on individual devices including even those with limited computational resources. The network is 22 layers deep when counting only layers with parameters (or 27 layers if we also count pooling). The overall number of layers (independent building blocks) used for the construction of the network is about 100. [1]

The overall GoogLeNet architecture is given below. This has again been taken from the journal paper submitted by Christian Szegedy.

Technology:

The networks were trained using the DistBelief [2] distributed machine learning system. The model used asynchronous stochastic gradient descent with a momentum of 0.9, fixed learning rate schedule, and Polyak averaging [3] to create the final model used during inference. The main limitation of the GoogLeNet model was the memory usage when being trained using GPU's. [1]

Therefore, the resulting model had the following structure:
- An average pooling layer with 5×5 filter size and stride 3, resulting in an 4×4×512 output for the (4a), and 4×4×528 for the (4d) stage.
- A 1×1 convolution with 128 filters for dimension reduction and rectified linear activation.
- A fully connected layer with 1024 units and rectified linear activation.
- A dropout layer with 70% ratio of dropped outputs.
- A linear layer with softmax loss as the classifier (predicting the same 1000 classes as the main classifier, but removed at inference time). [1]

Modeling:

The steps taken for modeming are as follows:

- Data Transformation:
    - Resize images to 224 * 224
    - Convert the images into Tensors

- Normalise the images to mean = [0.485, 0.456, 0.406] and standard deviation = [0.229, 0.224, 0.225]

- Data creation:
  - The datasets have been defined using the ImageFolder function.
  - These datasets have been loaded using the Data Loader function. Smaller batch sizes of 32 have been taken to ensure top much load isn't put on the model. We have also set the pin memory hyper parameter to True to ensure fast memory transfers between the CPU and GPU.

- Model initialisation:
  - The GoogLeNet model has been initialised and the head of the model has been altered. A Global Average Pooling layer represented by AdaptiveAvgPool2d exists in the model head.
  - 3 Fully connected layers layers have been initiated with the 3rd layer giving us the output. We have also chosen to not freeze the model due to the large size of the overall dataset.

    - First Fully Connected Layer:
      - Consists of a Linear Layer with input size 1024 (GoogleNet input size) and an output of 512.
      - This is connected to a ReLU activation function layer to generate non linear transformation of the input features.
      - This Is finally connected to a dropout layer of probability 0.5 to perform regularisation and prevent overfitting.

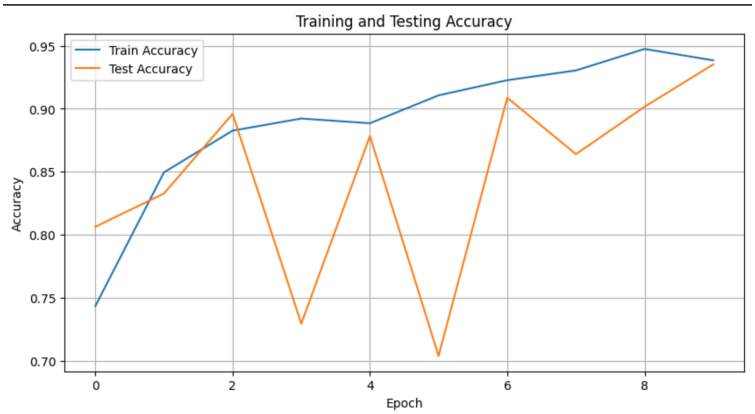
    - Second Fully Connected Layer:
      - Consists of a Linear Layer with input size 512 and an output of 256.
      - This is connected to another ReLU activation function layer.
      - This Is finally connected to another dropout layer of probability 0.5.

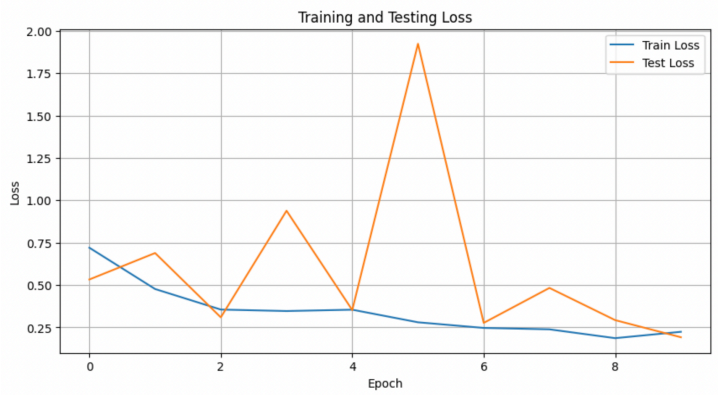
    - Third Fully Connected Output Layer:
      - The third and final FC layer consists of an input size of 256 and an output of 5 which represents the 5 classes taken in Part A.
      - This layer generates the final output of the model.

    - The Adam Optimiser and Cross Entropy Loss functions have been defined.

GoogLeNet model Results:

```
Epoch [1/10]
Train Loss: 0.7203, Train Accuracy: 74.35%
Test Loss: 0.5329, Test Accuracy: 80.64%
Epoch [2/10]
Train Loss: 0.4767, Train Accuracy: 84.96%
Test Loss: 0.6890, Test Accuracy: 83.28%
Epoch [3/10]
Train Loss: 0.3554, Train Accuracy: 88.27%
Test Loss: 0.3096, Test Accuracy: 89.60%
Epoch [4/10]
Train Loss: 0.3467, Train Accuracy: 89.23%
Test Loss: 0.9386, Test Accuracy: 72.96%
Epoch [5/10]
Train Loss: 0.3547, Train Accuracy: 88.85%
Test Loss: 0.3525, Test Accuracy: 87.84%
Epoch [6/10]
Train Loss: 0.2803, Train Accuracy: 91.07%
Test Loss: 1.9237, Test Accuracy: 70.40%
Epoch [7/10]
Train Loss: 0.2472, Train Accuracy: 92.27%
Test Loss: 0.2768, Test Accuracy: 90.88%
Epoch [8/10]
Train Loss: 0.2388, Train Accuracy: 93.04%
Test Loss: 0.4831, Test Accuracy: 86.40%
Epoch [9/10]
Train Loss: 0.1868, Train Accuracy: 94.75%
Test Loss: 0.2931, Test Accuracy: 90.16%
Epoch [10/10]
Train Loss: 0.2242, Train Accuracy: 93.84%
Test Loss: 0.1921, Test Accuracy: 93.52%
```





As we see above, the method consists of ten epochs. Initially, the model has a train accuracy of 74.35% and a test accuracy of 80.64%. As training advances, both train and test accuracies gradually improve, reaching 93.84% and 93.52%, respectively, by the tenth epoch. However, there are swings in test accuracy over epochs, indicating some level of overfitting. Overall, the model performs well on the test set, with a final accuracy of more than 93%, indicating that the model generalises effectively to new data despite minor volatility during training.

4.2 MobileNet V3:

History and Technology:

MobileNetV3 is tuned to mobile phone CPUs through a combination of hardware network architecture search (NAS) complemented by the NetAdapt algorithm and then subsequently improved through novel architecture advances. Overall, 2 MobileNet models were created - MobileNetV3-Large and MobileNetV3-Small which are targeted for high and low resource use cases.[4] In this project we will be using the MobileNetV3-Large. It was released in May 2019. [5] The mobileNet model consists of 28 layers where the output is given using a Softmax layer.

Modeling:

Most of the MobileNetV3 model head looks similar to the GoogleNet model.

- Data Transformation:
  - Resize images to 224 * 224
  - Convert the images into Tensors
  - Normalise the images to mean = [0.485, 0.456, 0.406] and standard deviation = [0.229, 0.224, 0.225]

- Data creation:
  - The datasets have been defined using the ImageFolder function.
  - These datasets have been loaded using the Data Loader function. Slightly bigger batch sizes of 64 have been taken due to the slightly smaller size of the MobileNetV3 model. We have also set the pin memory hyper parameter to True to ensure fast memory transfers between the CPU and GPU.

- Model initialisation:
  - The MobileNetV3 Large model has been initialised and the head of the model has been altered. A Global Average Pooling layer represented by AdaptiveAvgPool2d exists in the model head.
  - 3 Fully connected layers layers have been initiated with the 3rd layer giving us the output. We have also chosen to not freeze the model due to the large size of the overall dataset.

    - First Fully Connected Layer:

        - Consists of a Linear Layer with input size 576 (MobileNetV3 input size) and an output of 512.
        - This is connected to a ReLU activation function layer to generate non linear transformation of the input features.
        - This Is finally connected to a dropout layer of probability 0.5 to perform regularisation and prevent overfitting.

    - Second Fully Connected Layer:
        - Consists of a Linear Layer with input size 512 and an output of 256.
        - This is connected to another ReLU activation function layer.
        - This Is finally connected to another dropout layer of probability 0.5.

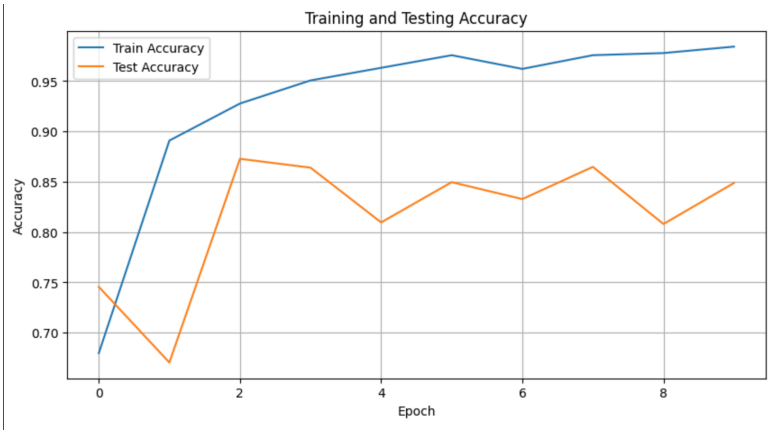
    - Third Fully Connected Output Layer:
        - The third and final FC layer consists of an input size of 256 and an output of 5 which represents the 5 classes taken in Part A.
        - This layer generates the final output of the model.

- The Adam Optimiser and Cross Entropy Loss functions have been defined.

MobileNet V3 Model Results:

The model was trained for ten epochs, with incremental gains shown in both training and testing performance. Initially, the model had a training accuracy of 67.97% and a testing accuracy of 74.56%. The model's accuracy increased dramatically as training went, peaking at 98.43% and tested at 84.88% by the last epoch. The model shows successful learning over time, as indicated by falling loss values and improving accuracy measures over epochs.

```
Epoch [1/10]
Train Loss: 0.8499, Train Accuracy: 67.97%
Test Loss: 1.0327, Test Accuracy: 74.56%
Epoch [2/10]
Train Loss: 0.3371, Train Accuracy: 89.09%
Test Loss: 1.7076, Test Accuracy: 67.04%
Epoch [3/10]
Train Loss: 0.2317, Train Accuracy: 92.77%
Test Loss: 0.4735, Test Accuracy: 87.28%
Epoch [4/10]
Train Loss: 0.1678, Train Accuracy: 95.07%
Test Loss: 0.4760, Test Accuracy: 86.40%
Epoch [5/10]
Train Loss: 0.1221, Train Accuracy: 96.32%
Test Loss: 1.0095, Test Accuracy: 80.96%
Epoch [6/10]
Train Loss: 0.0804, Train Accuracy: 97.57%
Test Loss: 0.7520, Test Accuracy: 84.96%
Epoch [7/10]
Train Loss: 0.1156, Train Accuracy: 96.21%
Test Loss: 0.8315, Test Accuracy: 83.28%
Epoch [8/10]
Train Loss: 0.0772, Train Accuracy: 97.57%
Test Loss: 0.6115, Test Accuracy: 86.48%
Epoch [9/10]
Train Loss: 0.0839, Train Accuracy: 97.79%
Test Loss: 0.8128, Test Accuracy: 80.80%
Epoch [10/10]
Train Loss: 0.0551, Train Accuracy: 98.43%
Test Loss: 0.7089, Test Accuracy: 84.88%
```



4.3 Nasnet

History and Technology:

NASNet is a CNN model architecture by Google Brain. The study presents a unique way to building convolutional architectures based on Neural Architecture Search (NAS). To reduce computational expenses, they recommend optimising models using a proxy dataset such as CIFAR-10 before moving them to ImageNet. This transferability is assisted via a search space known as "the NASNet search space," which is meant to keep architectural complexity similar regardless of network depth or input picture size. Using this technique, the study intends to efficiently optimise architectures for huge datasets such as ImageNet while minimising computing overhead. [6]

Modeling:

Most of the MobileNetV3 model head looks similar to the GoogleNet model.

- Data Transformation:
  - Resize images to 331 * 331 (NASNet image sizing)
  - Convert the images into Tensors
  - Normalise the images to mean = [0.485, 0.456, 0.406] and standard deviation = [0.229, 0.224, 0.225]

- Data creation:
  - The datasets have been defined using the ImageFolder function.
  - These datasets have been loaded using the Data Loader function. Batch sizes have once again been reduced to 32 to account for the larger NASNet model. We have also set the pin memory hyper parameter to True to ensure fast memory transfers between the CPU and GPU.

- Model initialisation:
  - The MobileNetV3 Large model has been initialised and the head of the model has been altered. A Global Average Pooling layer represented by AdaptiveAvgPool2d exists in the model head.
  - 3 Fully connected layers layers have been initiated with the 3rd layer giving us the output. We have also chosen to not freeze the model due to the large size of the overall dataset.

    - First Fully Connected Layer:
      - Consists of a Linear Layer with input size 4032 (NasNet input size) and an output of 512.
      - This is connected to a ReLU activation function layer to generate non linear transformation of the input features.
      - This Is finally connected to a dropout layer of probability 0.5 to perform regularisation and prevent overfitting.

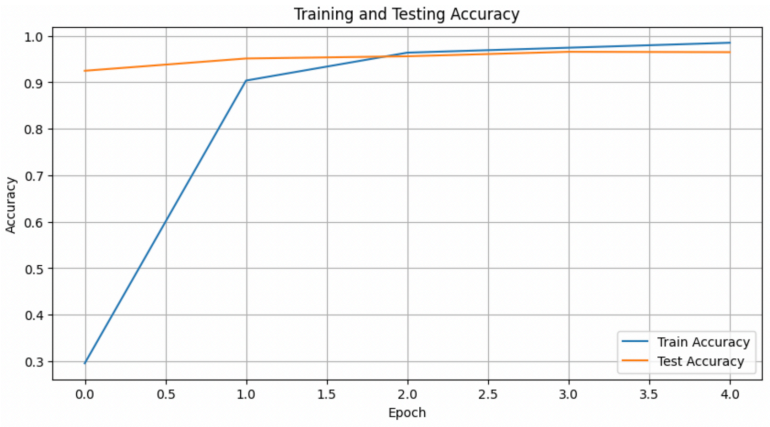
- Second Fully Connected Layer:
  - Consists of a Linear Layer with input size 512 and an output of 256.
  - This is connected to another ReLU activation function layer.
  - This Is finally connected to another dropout layer of probability 0.5.

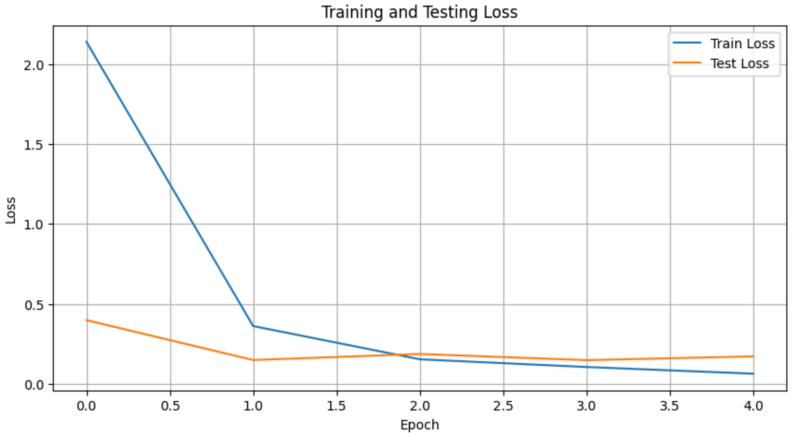
- Third Fully Connected Output Layer:
  - The third and final FC layer consists of an input size of 256 and an output of 5 which represents the 5 classes taken in Part A.
  - This layer generates the final output of the model.

- The Adam Optimiser and Cross Entropy Loss functions have been defined.

Results:

```
Epoch [1/5]
Train Loss: 0.6986, Train Accuracy: 78.72%
Test Loss: 0.1957, Test Accuracy: 93.84%
Epoch [2/5]
Train Loss: 0.1942, Train Accuracy: 94.21%
Test Loss: 0.1790, Test Accuracy: 94.24%
Epoch [3/5]
Train Loss: 0.0969, Train Accuracy: 97.12%
Test Loss: 0.1992, Test Accuracy: 94.80%
Epoch [4/5]
Train Loss: 0.0726, Train Accuracy: 98.21%
Test Loss: 0.2408, Test Accuracy: 93.76%
```





The NasNet model exhibits steady increase in both training and testing performance over the duration of training.

1. Training: The training loss gradually falls from 0.6986 to 0.0726, suggesting that the model is learning well from the training data. Simultaneously, training accuracy gradually improves, reaching 98.21% by the fourth epoch.
2. Testing: Throughout training, the model maintains good testing accuracy, with values ranging from 93.84% to 94.80%. This shows that the model generalises effectively to previously unknown data while maintaining strong performance.
3. Generalisation: The test loss remains reasonably low over epochs, showing that the model's performance is steady and does not overfit the training data.

Overall, the NasNet model displays good learning and generalisation skills, with high accuracy on both training and testing datasets.

5.0 Part B (Model Fine Tuning):

Part B consists of Fine Tuning a selected model that performed well. The chosen model is MobileNetV3 Large. The reasoning for this selection is the comparatively linear behaviour of the model as compared to the other two tuned model.

For Part B, the entire Food101 dataset has been used to train the model. The same preprocessing steps from Part A have been undertaken. Randomly selected images from 10 randomly selected classes are displayed to make sure the images have been imported correctly.

5.1 Fine Tuning Process:

Training the model:

In the fine tuning process, the following steps have been taken:

- Loading the model and the Food101 dataset.

- Freeze all layers.
  - Iterates through the entire model
  - The reasoning for this is to leverage the knowledge and features learned from a large diverse dataset while pre training.
  - Freezing prevents the parameters of these layers from being updated during training, thus retaining the pre-trained weights.

- Unfreeze Layers:
  - Specifies list of layers to be unfrozen which in our case are -
    - ‘features.11’- 12th convolutional layer within the feature extraction portion
    - ‘classifier.0' - First classification layer
    - ‘classifier.3' - Third Classification layer
  - Iterates only through the named parameters.
  - Unfreezing allows parameters of these layers to be updated during subsequent iterations.

- Optimiser is defined with the parameters of the unfrozen layers only. This allows the model to adapt to the dataset without significantly altering the learned representations in the learned layer. It also reduces computation and makes the overall training more efficient.

- Model Training.

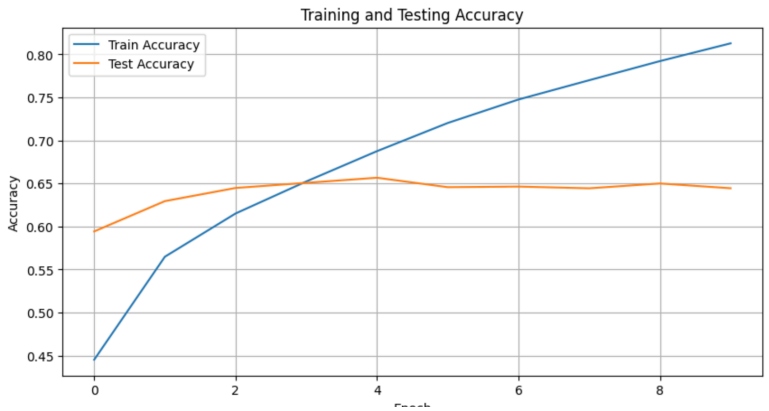
Results:

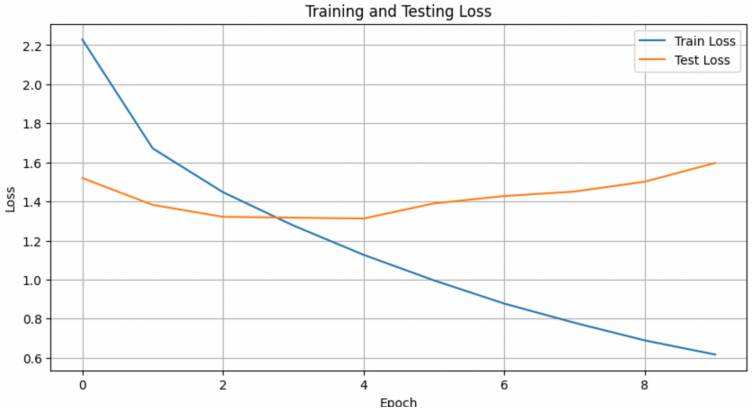
```
Epoch [1/10]
Train Loss: 2.2287, Train Accuracy: 44.52%
Test Loss: 1.5197, Test Accuracy: 59.43%
Epoch [2/10]
Train Loss: 1.6719, Train Accuracy: 56.49%
Test Loss: 1.3824, Test Accuracy: 62.95%
Epoch [3/10]
Train Loss: 1.4473, Train Accuracy: 61.52%
Test Loss: 1.3211, Test Accuracy: 64.48%
Epoch [4/10]
Train Loss: 1.2770, Train Accuracy: 65.24%
Test Loss: 1.3169, Test Accuracy: 65.08%
Epoch [5/10]
Train Loss: 1.1267, Train Accuracy: 68.76%
Test Loss: 1.3125, Test Accuracy: 65.67%
Epoch [6/10]
Train Loss: 0.9959, Train Accuracy: 72.03%
Test Loss: 1.3898, Test Accuracy: 64.57%
Epoch [7/10]
Train Loss: 0.8771, Train Accuracy: 74.76%
Test Loss: 1.4276, Test Accuracy: 64.64%
Epoch [8/10]
Train Loss: 0.7787, Train Accuracy: 77.00%
Test Loss: 1.4505, Test Accuracy: 64.44%
Epoch [9/10]
Train Loss: 0.6885, Train Accuracy: 79.22%
Test Loss: 1.5013, Test Accuracy: 65.00%
Epoch [10/10]
Train Loss: 0.6162, Train Accuracy: 81.28%
Test Loss: 1.5971, Test Accuracy: 64.45%
```

1. Stability and Reliability: The testing dataset results reveal a more linear behaviour than the training dataset, implying that the fluctuations in accuracy, which might indicate overfitting, have been reduced. This shows that the method of freezing and unfreezing layers has aided in addressing the overfitting problem, resulting in more trustworthy and consistent outcomes.

2. Performance difference: Despite a considerable increase in dataset size from 5 to 101 classes, and a corresponding increase in picture count from 5,000 to 101,000, the MobileNet model performed well. This indicates the model architecture's resilience and scalability, since it performs well across a variety of dataset sizes and complexity.

3. Test Loss and Accuracy Trend: The test loss immediately diminishes, showing good learning and generalisation. However, it then grows, suggesting probable overfitting. Despite this, test accuracy increases across epochs, but it plateaus at the end, showing the need for optimisation to avoid overfitting and improve generalisation.





6.0 Challenges Faced:

1. Computational Power: The biggest challenge faced during the project is the overall computational power available. The sizes of the dataset coupled with the large models took high amounts of computation to deliver results.

2. Overfitting and Generalisation: Overfitting in the model is another challenge that was faced which needs to be overcome in the next steps using steps such as data augmentation.

3. Time: The biggest challenge faced was the overall time that it takes to run a deep learning model.

4. Model Interpretability: Understanding the layers and the networks of the deep learning model was another key challenge faced.

7.0 Future Steps and Recommendations:

1. Data Augmentation: To increase model resilience and generalisation, supplement the dataset using techniques like as rotation, flipping, and scaling.

2. More models: Explore into more transfer learning models that could be of higher relevance to our dataset.

3. Domain Specific Pre training: Explore into pre training methods of models into the domain of Food image classification.

4. Hyperparameter tuning exploration: Perform more hyperparameter tuning in terms of relevant batch sizes, choices in optimisers etc.

5. User Feedback Integration: Work on systems to ensure feedback given by users are considered and adapted into the ever changing models.

References:

[1] : Going Deeper with Convolutions - Christian Szegedy et.al. - 17th Sep 2014 - https://arxiv.org/pdf/1409.4842.pdf

[2] : Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In P. Bartlett, F.c.n. Pereira, C.j.c. Burges, L. Bottou, and K.q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1232–1240. 2012

[3] : B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. SIAM J. Control Optim., 30(4):838–855, July 1992.

[4] : Andrew Howard et.al. - Searching for MobileNetV3 - 20th Nov 2019 - https://arxiv.org/pdf/1905.02244.pdf

[5] : The Evolution of Google's MobileNet Architectures to Improve Computer Vision Models - Jesus Rodriguez - Feb 17 2021 - https://medium.com/dataseries/the-evolution-of-googles-mobilenet-architectures-to-improve-computer-vision-models-ffb483ffcc0a#:~:text=In%202017%2C%20Google%20introduced%20MobileNets,and%20MobileNetV3%20in%20May%202019%20.

[6] : Learning Transferable Architectures for Scalable Image Recognition - Barret Zoph et.al. - 11th April 2018 - https://arxiv.org/pdf/1707.07012.pdf