CNN's Classification of High-Calorie Fast Food

Introduction

Food selection and consumption, which are crucial to human health and wellbeing, are greatly influenced by food recognition. As a result, it is important to the computer vision community and can support further multimodal and food-related vision tasks (Weiqing Min, 2021). Diet is crucial to human health. Our health depends on getting enough nutrition from regular meals. We frequently keep track of our regular meals in order to know what we eat. Typically, this type of meal recording is done manually using textual descriptions, however hand recording is tiresome and time-consuming. There have been attempts to use information technology to help with meal recording in order to get around this problem. Food recording could benefit from image recognition of various food products. The complexity of image-based food recognition is increased by the fact that many food products are similar in shape, size, and colour, and that cooking and food preparation methods vary by region. Traditional techniques for detecting food, like manually extracted features and image segmentation, have limits in terms of accuracy and effectiveness (Liu, Y., 2021). Convolutional neural networks (CNNs) have recently been demonstrated to perform well in a range of image recognition tasks, such as object detection and image classification. Nevertheless, the application of CNNs for food detection in photos has received little investigation. The main goal of this paper is to use convolutional neural networks to achieve high accuracy of detection and alert user of top 3 high calories fast food.

Literature Review

We reviewed a total of 5 research papers to strengthen and aid us in this journey before moving on to this study.

1. *CNN-Based Food Image Segmentation Without Pixel-Wise Annotation*

In this study, they suggested a DCNN-based method for segmenting food images without the need for pixel-by-pixel annotation. The suggested approach comprises of food region suggestions using bounding box clustering and selective search, back propagation-based saliency map estimation using the DCNN tuned with the UEC- FOOD100 dataset, GrabCut guided by the calculated saliency maps, and region integration via non-maximum suppression. In the studies, both the PAS-CAL VOC detection task and the food region detection task performed better using the suggested technique than RCNN (Wataru Shimoda, 2015).

UEC-FOOD100 mAP	100class (all)	53class (#item $\geq$ 10)	11class (#item $\geq$ 50)
R-CNN	26.0	21.8	25.7
proposed method	**49.9**	55.3	**55.4**

Figure 1. The results by RCNN and the proposed methods

2. *Efficient extraction of deep image features using convolutional neural network (CNN) for applications in detecting and analysing complex food matrices*

In this research, they claim that in classifying food photographs into several categories, their CNN-based method has an accuracy of over 90%. Furthermore, they demonstrate that the suggested strategy surpasses conventional ones in terms of precision and computational effectiveness. The suggested method entails obtaining deep image features from food photos using a CNN. The researchers employ a dataset of food photographs to fine-tune a pre-trained CNN model that serves as a feature extractor (Liu, Y., et. al., 2021).

3. *Food Detection and Recognition Using Convolutional Neural Network*

A CNN-based method for identifying and detecting food in photos was proposed in this study. A food-domain photos gathered from a food recording system available for public use was utilised in this study. A region-based CNN (R-CNN) is used by the food identification component to identify food regions in an image. The food recognition component divides the identified regions into various food groups using a fine-tuned CNN. The accuracy for food detection was reported to be 93.7%, while the accuracy for food recognition was reported to be 85.5% by the authors, who evaluated the system's performance using a dataset of food photos (Kagaya, H., et. al., 2014).

Method	Accuracy
Baseline	$89.7 \pm 0.73\%$
CNN	$\mathbf{93.8 \pm 1.39\%}$

Figure 2 - Comparison between CNN and the baseline method for the detection task.

4. *Food Detection With Image Processing Using Convolutional Neural Network (Cnn) Method*

The efficacy of CNNs for food image recognition and detection was examined by the researchers in this study. They created a food image dataset using pictures submitted by actual people, used CNN to identify 10 different foods, and assessed the results. They discovered that CNN outperformed conventional techniques using handcrafted characteristics significantly. The 80% training data and 20% test data partition yields the best results, with 100% accuracy and an average speed of less than 10 seconds (Ramdani, A., et. al., 2020).

(1)

Learning rate 0.0002, *Epoch* 9000	
Partition	Accuracy (%)
90% training data 10% testing data	69.01
80% training data 20% testing data	100
70% training data 30% testing data	69.43

(2)

Learning rate 0.0002	
Epoch	Accuracy (%)
1000	71.29
3000	63.88
5000	87.56
7000	78.7
9000	100

(3)

Epoch 9000	
Learning rate	Accuracy (%)
0.1	0
0.01	94.44
0.001	94.44
0.0001	83.33
0.0002	100

(4)

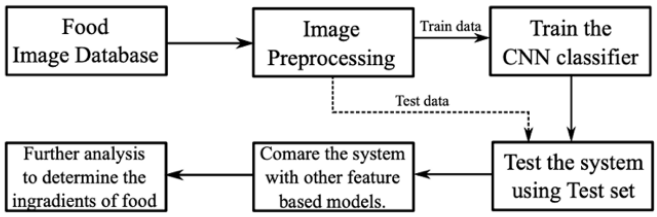
No	Class	Average Detection Speed (seconds)
1	1-2-3	5.90
2	1-2-4	4.88
3	1-2-5	4.7
4	1-2-6	7.10
5	1-6-3	7.22
6	1-6-4	4.64
7	1-6-5	4.72

Figure 3 - (1) Partition data, (2) Epoch, (3) Learning Rate, (4) Speed of Detection

5. *Food Classification from Images Using Convolutional Neural Networks*

A tensor of outputs is produced by using a 2D convolution layer to construct a convolution kernel and convolving it with the input of the layer in this study. They employed a number of pre-trained CNN architectures, including AlexNet, VGG16, and ResNet50. The food dataset was used by the authors to train the CNN model, and performance was assessed using common measures including classification accuracy. Additionally, they employ the Max-Pooling function for the data and train the network using the characteristics that are taken from this function. Using the suggested method, an accuracy of 86.97% for the classes of the FOOD-101 dataset is recognised. Additionally, they evaluated how well their CNN model performed in comparison to other established image classification techniques like SVM and k-NN (Attokaren., et. al., 2017).

(1)



(2)

Sl.No.	Model	Dataset	Accuracy (in %)
1.	SVM	Food-101	50.76
2.	Neural Networks	Food-101	56.40
3.	RFDC-based Approach	Food-101	56.76
4.	Resnet18	Food-101	67.23
5.	CNN	UEC-FOOD100	78.77
6.	CNN (ILSVRC)	Food-101	79.20
7.	CNN (Food-101)	EgocentricFood	90.90
8.	Proposed Approach	Food-101	**86.97**

Figure 4 - (1) Proposed methodology, (2) Comparison of top accuracies for different models and dataset.

Data

Convolutional Neural Networks are used in this study to recognise food using a collection of photos of fast food (CNN). This fast-food classification data collection includes pictures of ten distinct fast food items, including burgers, donuts, hot dogs, pizza, sandwiches, baked potatoes, crispy chicken, fries, tacos, and taquitos. This dataset was uploaded by Author DEEPNETS to Kaggle in 2022. It included 31,500 photos in total, with 15,000 images used for training, 15,000 for testing, and 1,500 for validation. In this file, each directory represents a class, and each class represents a sort of food. Tensorflow Records, Training Data, Validation Data, and Testing Data are the four categories into which the data set is divided.

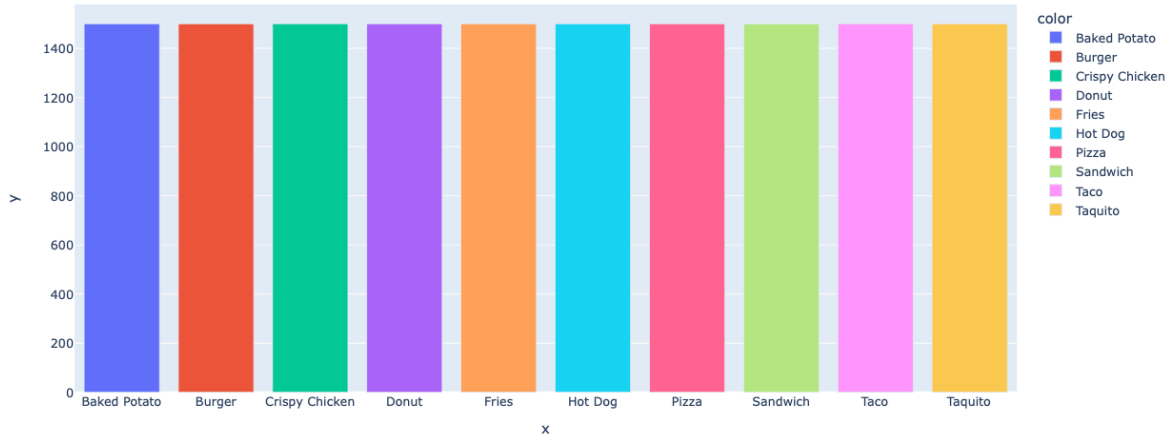


Figure 5 - Exploratory data analysis

- **Model**

The training dataset and the test dataset are the two components of the data. The model is trained using the training dataset, and its effectiveness is assessed using the test dataset. The loaded data was then divided by 255 to achieve normalisation. This was accomplished to place the images' pixel values inside the range of 0 and 1. Normalizing the data lowers the likelihood of overfitting and helps the model perform better during training.

In the training of CNN models for image classification, data augmentation is a crucial pre-processing step. Data augmentation is a collection of methods for creating additional data points from current data to artificially enhance the amount of data. This aids in lowering overfitting and enhancing the model's performance on unstructured data (Shah, 2023). In this case, the amount of the training dataset was adequate for training this model, hence data augmentation was not necessary.

- **Transfer Learning**

In this part of the study, data augmentation was applied to the training data using the ImageDataGenerator class from the Keras framework. Three instances of the class— training data, testing data, and validation data are created using the function. To improve training performance, the pixel values between 0 and 1 are normalised using the rescale function. For training and validation, the images are downsized to (256, 256) and divided into groups of size 64 and 32. The range of rotations to apply to the photos is specified by the rotation range argument, in this case 10 degrees and the dataset will be horizontally flipped, while still preserving their vertical alignment. Furthermore, the datasets are loaded utilising the flow from directory technique to read the photos directly from the directory and enhance them while the neural network model is training on the training data.

Modelling

We will be using an autoencoder as the benchmark CNN structure. Encoder and decoder are the two components that make up the autoencoder. The network's encoder section is in charge of applying numerous convolutional layers and max pooling layers to the input image to reduce its dimensionality. While the max pooling layers reduce the spatial dimension of the feature maps, the convolutional layers add filters to the input image to extract features.

(1)

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 64, 64, 3)]	0
conv2d (Conv2D)	(None, 64, 64, 256)	7168
max_pooling2d (MaxPooling2D)	(None, 32, 32, 256)	0
conv2d_1 (Conv2D)	(None, 32, 32, 256)	590080
conv2d_2 (Conv2D)	(None, 32, 32, 128)	295040
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
conv2d_4 (Conv2D)	(None, 16, 16, 64)	73792
conv2d_5 (Conv2D)	(None, 16, 16, 32)	18464
conv2d_6 (Conv2D)	(None, 16, 16, 16)	4624
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 16)	0
d1 (Conv2D)	(None, 8, 8, 16)	2320
d2 (Conv2D)	(None, 8, 8, 32)	4640
d3 (Conv2D)	(None, 8, 8, 64)	18496
d4 (Conv2D)	(None, 8, 8, 128)	73856
d5 (UpSampling2D)	(None, 16, 16, 128)	0
d6 (Conv2D)	(None, 16, 16, 128)	147584
d7 (Conv2D)	(None, 16, 16, 256)	295168
d8 (UpSampling2D)	(None, 32, 32, 256)	0
d9 (Conv2D)	(None, 32, 32, 256)	590080
d10 (UpSampling2D)	(None, 64, 64, 256)	0
d11 (Conv2D)	(None, 64, 64, 3)	6915

Total params: 2,275,811
Trainable params: 2,275,811
Non-trainable params: 0

(2)

Model: "model_1"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 64, 64, 3)]	0
conv2d (Conv2D)	(None, 64, 64, 256)	7168
max_pooling2d (MaxPooling2D)	(None, 32, 32, 256)	0
conv2d_1 (Conv2D)	(None, 32, 32, 256)	590080
conv2d_2 (Conv2D)	(None, 32, 32, 128)	295040
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_3 (Conv2D)	(None, 16, 16, 128)	147584
conv2d_4 (Conv2D)	(None, 16, 16, 64)	73792
conv2d_5 (Conv2D)	(None, 16, 16, 32)	18464
conv2d_6 (Conv2D)	(None, 16, 16, 16)	4624
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 16)	0

Total params: 1,136,752
Trainable params: 1,136,752
Non-trainable params: 0

(3)

Model: "model_2"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 8, 8, 16)]	0
d1 (Conv2D)	(None, 8, 8, 16)	2320
d2 (Conv2D)	(None, 8, 8, 32)	4640
d3 (Conv2D)	(None, 8, 8, 64)	18496
d4 (Conv2D)	(None, 8, 8, 128)	73856
d5 (UpSampling2D)	(None, 16, 16, 128)	0
d6 (Conv2D)	(None, 16, 16, 128)	147584
d7 (Conv2D)	(None, 16, 16, 256)	295168
d8 (UpSampling2D)	(None, 32, 32, 256)	0
d9 (Conv2D)	(None, 32, 32, 256)	590080
d10 (UpSampling2D)	(None, 64, 64, 256)	0
d11 (Conv2D)	(None, 64, 64, 3)	6915

Total params: 1,139,059
Trainable params: 1,139,059
Non-trainable params: 0

Figure 6 - (1) Autoencoder model, (2) Encoder model, (3) Decoder model

The benchmark model's selection of hyperparameters is based on the input picture size, which is indicated by the value IMG SIZE = (64,64,3). Convolutional layers (Conv2D), max-pooling layers (MaxPooling2D), and up-sampling layers are among the network's 11 total layers (UpSampling2D). The first convolutional layer in this model utilises the ReLU activation function and includes 256 filters with a 3x3 pixel size. The next max pooling layer reduces the spatial dimension of the feature maps by half with a pool size of 2x2. In the encoder portion of the network, multiple iterations of this pattern is a convolutional layer and a max pooling layer in a repeat form with the number of filters reducing as the spatial dimension of the feature maps increases.

Transposed convolutional layers and up-sampling layers are applied in the network's decoder section to give the feature maps' spatial dimensions a boost and create the reconstructed image. The last layer employs the sigmoid activation function, whereas the transposed convolutional layers utilise the ReLU activation function to create the reconstructed image, which has values between 0 and 1.

Since the input images are binary, the model is optimised using the Adam optimizer and the binary cross-entropy loss function. Additionally, the model's performance is assessed using the accuracy metric. The model is trained using a 256-batch training set across 5 epochs.

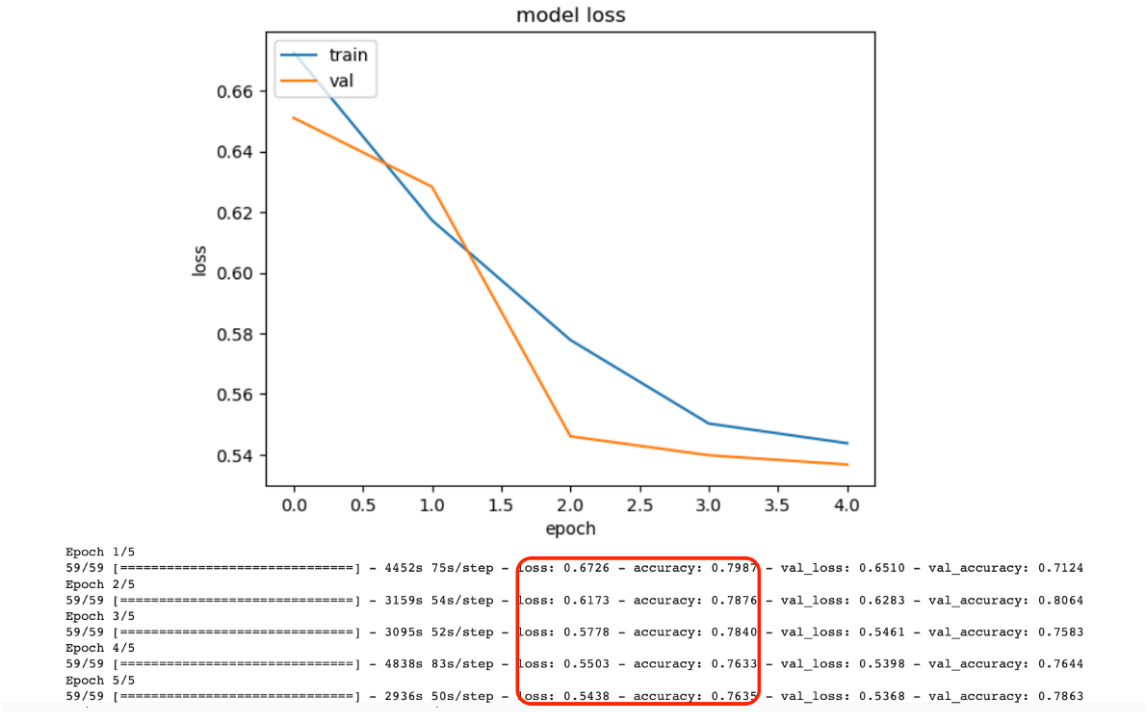


Figure 7 - Performance of proposed model

According to the training process findings, the accuracy increases while the loss decreases, indicating that the autoencoder is becoming more efficient and reaching a maximum accuracy of 76.35% on the fifth epoch. With a validation accuracy of 76.63% which indicates that the model generalised well to the validation data.

Application used

In this study, a Jupyter Notebook instance of type ml.t3.medium—the largest instance type offered under the free tier was used to construct a model for machine learning. It took about 18,480s – about 5 hours, to train the full dataset on the proposed model and 26,551s – about 7.3 hours to train on the transfer learning model, which was a computationally demanding process. Although the model operated at its best on a local Jupyter Notebook, it encountered difficulties when attempting to use the AWS website.

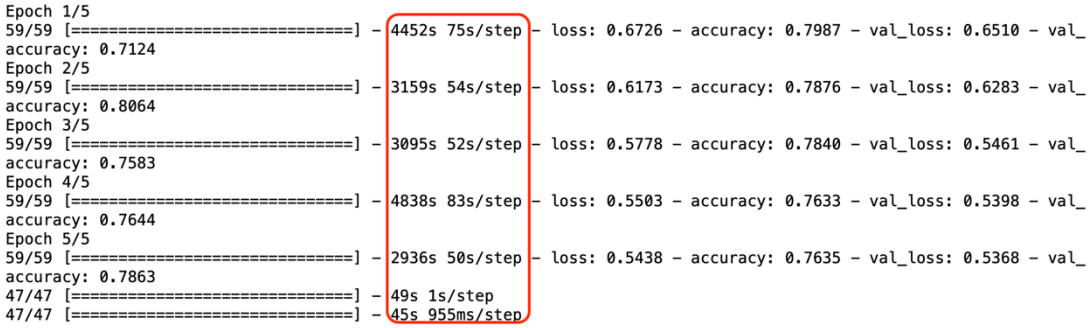


Figure 8 - Computational time on proposed model

```
Epoch 1/5
 98/235 [===========>..................] - ETA: 39:00 - loss: 0.5490 - accuracy: 0.8393
/opt/anaconda3/lib/python3.9/site-packages/PIL/Image.py:959: UserWarning:

Palette images with Transparency expressed in bytes should be converted to RGBA images

235/235 [==============================] - 6034s 26s/step - loss: 0.4287 - accuracy: 0.8748 - val_loss: 0.3981 - va
l_accuracy: 0.8846
Epoch 2/5
235/235 [==============================] - 5385s 23s/step - loss: 0.2824 - accuracy: 0.9165 - val_loss: 0.4063 - va
l_accuracy: 0.8851
Epoch 3/5
235/235 [==============================] - 5109s 22s/step - loss: 0.2851 - accuracy: 0.9155 - val_loss: 0.4085 - va
l_accuracy: 0.8817
Epoch 4/5
235/235 [==============================] - 5103s 22s/step - loss: 0.2835 - accuracy: 0.9173 - val_loss: 0.4061 - va
l_accuracy: 0.8831
Epoch 5/5
235/235 [==============================] - 4920s 21s/step - loss: 0.2819 - accuracy: 0.9173 - val_loss: 0.4008 - va
l_accuracy: 0.8820
```

Figure 9 - Computational time on transfer learning model

Transfer Learning

Transfer learning is the application of a model that has already been trained to solve a new issue. Since most of the real-world situations often do not have millions of labelled data points to train such complicated models, this is tremendously helpful in the data science sector (Donges, 2022). In our project, we used a pre-trained BiT model from TensorFlow Hub (https://tfhub.dev/google/bit/m-r50x1/1).

The pre-trained BiT model, an input layer, a dropout layer, and a dense layer with a softmax activation function and zero initializer make up the model architecture. The architecture and the number of parameters in each layer are displayed in the model's summary. The PiecewiseConstantDecay (PwCD) function was used to define a learning rate scheduler with a learning rate of 0.005. After 200, 300, and 400 epochs, the learning rate would drop by a factor of 10. Additionally, we have Stochastic Gradient Descent (SGD) with a momentum of 0.9 as the optimizer. When the validation loss did not decrease for 5 epochs, training was terminated using an EarlyStopping (ES) callback that tracked the loss.

```
Model: "Fast-Food-Classification-BiT"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 2048)              23500352

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 10)                20490

=================================================================
Total params: 23,520,842
Trainable params: 20,490
Non-trainable params: 23,500,352
_________________________________________________________________
```

Figure 10 - Transfer learning's model

Discussion

As evidenced by the greater accuracy score and lower loss value, transfer learning performs noticeably better than the benchmark model. In contrast to the benchmark model's accuracy of 76.35%, the transfer learning model's accuracy was 91.73%. This finding shows that transfer learning can efficiently use the information learnt from a large pre-trained model and transfer it to the target task, resulting in improved performance compared to training a model from scratch.

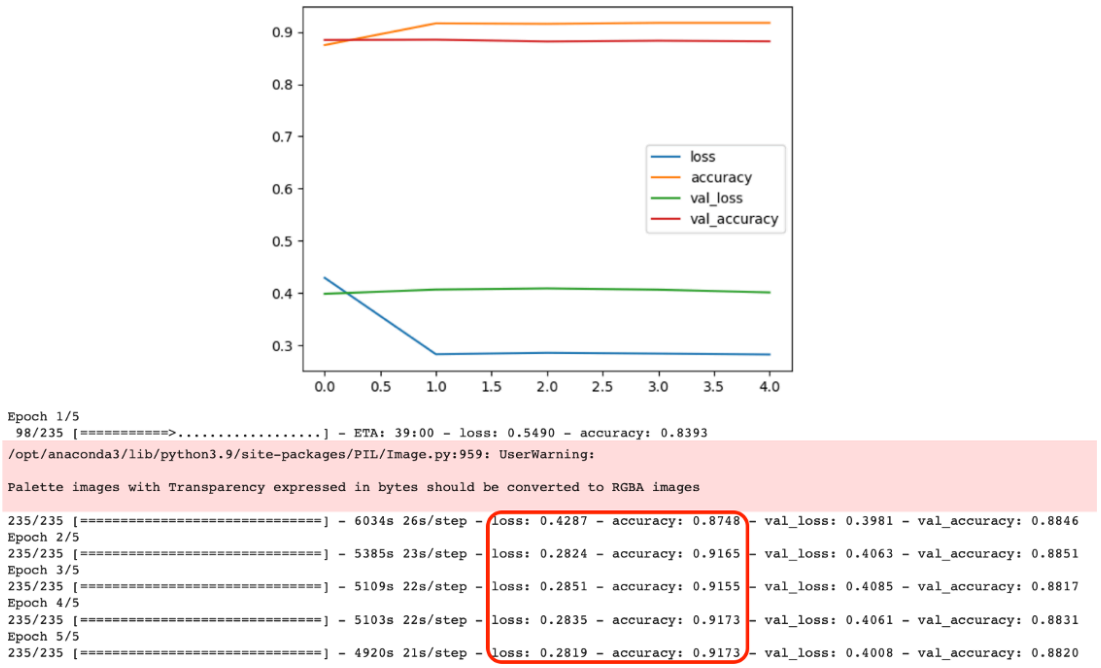


Figure 11 - Performance of transfer learning model

Transfer learning has several advantages, such as reducing training time, improving neural network performance, and does not require large data. Even though transfer learning outperforms alternative methods in this study, it may not always be the best option in all projects since it is highly dependent on the availability of a trained model that is similar to the problem at the moment. Training a model from scratch may still be required in situations where the target task differs significantly from the previously learned model.

The information in this table was created from a variety of web resources and followed by the lowest amount of calories for each food item by the smallest serving size due to numerous distinct images that were used in this dataset.

Items	**Estimated Calories**
Burger	120
Donuts	125
Hot Dogs	150
Pizza	847
Sandwiches	252
Baked Potatoes	156
Crispy Chicken	140
Fries	143

Tacos	141
Taquitos	58

Using the higher accuracy model, we have predicted of the class and confidence of an image as well as randomly detect imagines and bolded the words with the top 3 highest calories fast food such as baked potato, sandwich, and pizza referring to the figure below.

Figure 12 – Final testing's result

Limitation

In this study, the model was trained over a subset of 5 epochs. Given the size of the dataset being used, this was done to reduce the processing time and memory needs. The model is trained on the data for fewer iterations when fewer epochs are employed, which makes it more computationally possible for the system being used. Furthermore, the model would be ineffective if we increased the number of epochs because the accuracy did not change much, and each epoch takes about 2 hours to run. Despite its drawbacks, this method nevertheless

offers a good foundation for further training and testing, and the study's findings could offer insightful information.

References

Weiqing Min, Z. W. (2021, March 30). *Large Scale Visual Food Recognition*. Retrieved from Cornell University : https://arxiv.org/abs/2103.16107

Yao Liu, H. P.-W. (2021, July). *Efficient extraction of deep image features using convolutional neural network (CNN) for applications in detecting and analysing complex food matrices*. Retrieved from Science Direct: https://www.sciencedirect.com/science/article/abs/pii/S0924224421003022

Wataru Shimoda, K. Y. (2015). *CNN-Based Food Image Segmentation Without Pixel-Wise Annotation.* Tokyo: Department of Informatics, The University of Electro-Communications.

Yao Liu, H. P.-W. (2021, July). *Efficient extraction of deep image features using convolutional neural network (CNN) for applications in detecting and analysing complex food matrices*. Retrieved from Science Direct: https://www.sciencedirect.com/science/article/abs/pii/S0924224421003022

Shah, D. (2023, February 2). *BLOG MACHINE LEARNING The Essential Guide to Data Augmentation in Deep Learning* . Retrieved from v7 Labs: https://www.v7labs.com/blog/data-augmentation-guide

Donges, N. (2022, August 25). *What Is Transfer Learning? Exploring the Popular Deep Learning Approach*. Retrieved from Builtin: https://builtin.com/data-science/transfer-learning

Liu, Y., Pu, H., & Sun, D. W. (2021). Efficient extraction of deep image features using convolutional neural network (CNN) for applications in detecting and analysing complex food matrices. *Trends in Food Science & Technology*, *113*, 193-204.

Kagaya, H., Aizawa, K., & Ogawa, M. (2014, November). Food detection and recognition using convolutional neural network. In *Proceedings of the 22nd ACM international conference on Multimedia* (pp. 1085-1088).

Ramdani, A., Virgono, A., & Setianingsih, C. (2020, July). Food detection with image processing using convolutional neural network (CNN) method. In *2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)* (pp. 91-96). IEEE.

Attokaren, D. J., Fernandes, I. G., Sriram, A., Murthy, Y. S., & Koolagudi, S. G. (2017, November). Food classification from images using convolutional neural networks. In *TENCON 2017-2017 IEEE Region 10 Conference* (pp. 2801-2806). IEEE.

Appendix

Assessment 4: Capstone Research

Food selection and consumption, which are crucial to human health and wellbeing, are greatly influenced by food recognition. As a result, it is important to the computer vision community and can support further multimodal and food-related vision tasks. The main goal of this paper is to use convolutional neural networks to achieve high accuracy of detection and alert user of top 3 high calories fast food.

Download dataset

This fast food classification data collection includes pictures of ten distinct fast food items. The dataset was uploaded by Author DEEPNETS to Kaggle in 2022.

In [1]:
```
# importing image dataset
import os
os.environ["KAGGLE_USERNAME"] = "sallypangshueyan"
os.environ["KAGGLE_KEY"] = "968442fea61b821a0a9b2831b153d214"
```

In [2]:
```
# !pip install kaggle
!kaggle datasets download utkarshsaxenadn/fast-food-classification-dataset
```

```
Downloading fast-food-classification-dataset.zip to /Users/sallypang/Library/CloudStorage/OneDrive-JamesCookUniversity/LAST BUT NOT LAST YET
100%|████████████████████████████████████████| 820M/821M [01:06<00:00, 21.9MB/s]
100%|████████████████████████████████████████| 821M/821M [01:06<00:00, 12.9MB/s]
```

In [3]:
```
!unzip fast-food-classification-dataset.zip
```

```
Archive:  fast-food-classification-dataset.zip
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_1.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_10.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_100.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_11.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_12.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_13.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_14.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_15.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_16.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_17.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_18.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_19.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_2.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_20.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_21.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_22.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_23.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato_24.tfrecord
  inflating: Fast Food Classification V2/TFRecords/Test/Baked Potato 25.tfrecord
```

Define test, train, valid set directory

In [5]:
```
train_dir = 'Fast Food Classification V2/Train/'
test_dir = 'Fast Food Classification V2/Test/'
valid_dir = 'Fast Food Classification V2/Valid/'
IMG_SIZE = (64,64,3)
```

In [6]:
```
def walk_through_dir(dir_path):

    for dirpath, dirnames,filenames in os.walk(dir_path):
        print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'")
```

In [7]:
```
walk_through_dir('Fast Food Classification V2/Train/')
```

```
There are 10 directories and 0 images in 'Fast Food Classification V2/Train/'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Sandwich'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Donut'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Crispy Chicken'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Baked Potato'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Fries'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Burger'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Pizza'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Hot Dog'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Taquito'
There are 0 directories and 1500 images in 'Fast Food Classification V2/Train/Taco'
```

In [8]:
```
walk_through_dir('Fast Food Classification V2/Test/')
```

```
There are 10 directories and 0 images in 'Fast Food Classification V2/Test/'
There are 0 directories and 200 images in 'Fast Food Classification V2/Test/Sandwich'
There are 0 directories and 200 images in 'Fast Food Classification V2/Test/Donut'
There are 0 directories and 100 images in 'Fast Food Classification V2/Test/Crispy Chicken'
There are 0 directories and 100 images in 'Fast Food Classification V2/Test/Baked Potato'
There are 0 directories and 100 images in 'Fast Food Classification V2/Test/Fries'
There are 0 directories and 200 images in 'Fast Food Classification V2/Test/Burger'
There are 0 directories and 200 images in 'Fast Food Classification V2/Test/Pizza'
There are 0 directories and 200 images in 'Fast Food Classification V2/Test/Hot Dog'
There are 0 directories and 100 images in 'Fast Food Classification V2/Test/Taquito'
There are 0 directories and 100 images in 'Fast Food Classification V2/Test/Taco'
```

In [9]:
```
walk_through_dir('Fast Food Classification V2/Valid/')
```

```
There are 10 directories and 0 images in 'Fast Food Classification V2/Valid/'
There are 0 directories and 300 images in 'Fast Food Classification V2/Valid/Sandwich'
There are 0 directories and 300 images in 'Fast Food Classification V2/Valid/Donut'
There are 0 directories and 400 images in 'Fast Food Classification V2/Valid/Crispy Chicken'
There are 0 directories and 400 images in 'Fast Food Classification V2/Valid/Baked Potato'
There are 0 directories and 400 images in 'Fast Food Classification V2/Valid/Fries'
There are 0 directories and 300 images in 'Fast Food Classification V2/Valid/Burger'
There are 0 directories and 300 images in 'Fast Food Classification V2/Valid/Pizza'
There are 0 directories and 300 images in 'Fast Food Classification V2/Valid/Hot Dog'
There are 0 directories and 400 images in 'Fast Food Classification V2/Valid/Taquito'
There are 0 directories and 400 images in 'Fast Food Classification V2/Valid/Taco'
```

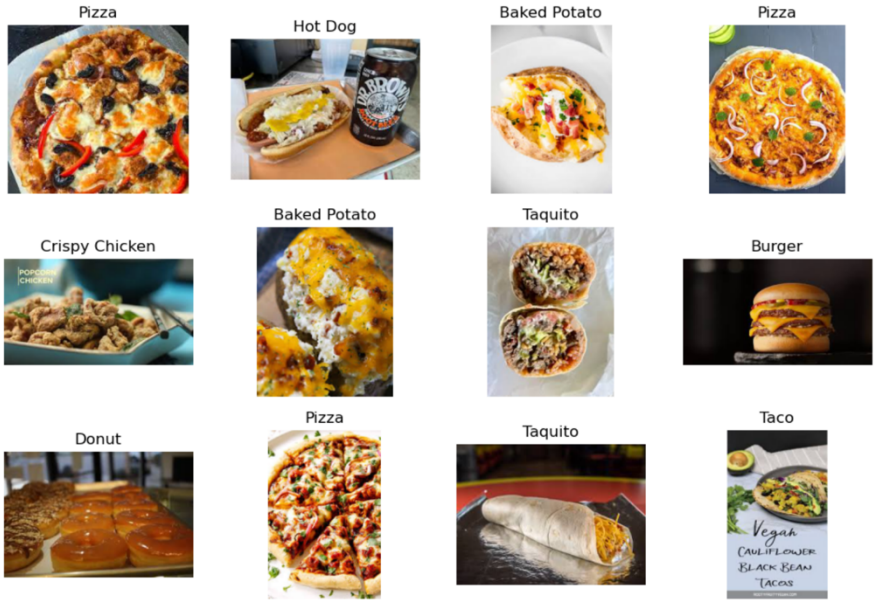
The dataset included 31,500 photos in total, with 15,000 images used for training, 15,000 for testing, and 1,500 for validation.

Plot Some Random Images from training and test set

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import random
import pathlib

data_dir = pathlib.Path('Fast Food Classification V2/Train')
class_names = np.array(sorted([item.name for item in data_dir.glob('*')]))

def view_rand_img(target_dir):
    plt.figure(figsize = (12,8)) # create the figure size

    for i in range(12): # loop to show 12 images at a time

        ax = plt.subplot(3, 4, i+1) # show the chosen 12 images in a 3 * 4 grid

        rand_class = random.choice(class_names) # choose a random class
        target_folder = 'Fast Food Classification V2/' + target_dir + "/" + rand_class # create the directory to the images
        # target_folder = 'Fast Food Classification V2/' + target_dir + "/Fries" # create the directory to the images

        rand_img = random.sample(os.listdir(target_folder), 12) # choose the 12 images randomly
        img = mpimg.imread(target_folder + "/" + rand_img[i]) # read the images
        plt.imshow(img) # show the images
        plt.title(rand_class) # set title
        plt.axis(False) # hide the axis
```

In [12]:
```
# keep runing this cell to see more random images from the training data
view_rand_img('Train')
```

In [13]:
```
# keep runing this cell to see more random images from the test data
view_rand_img('Test')
```



Exploratory data analysis

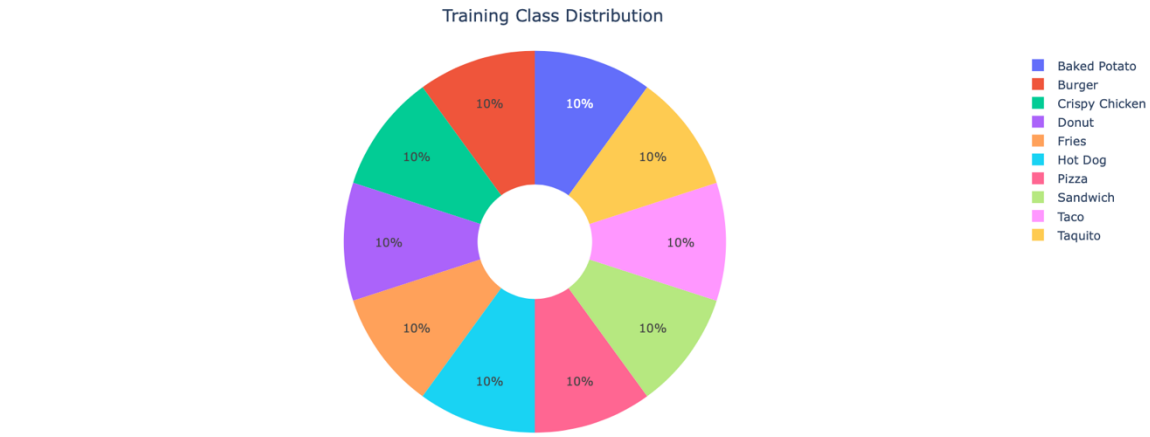
There are ten distinct fast food items, including:

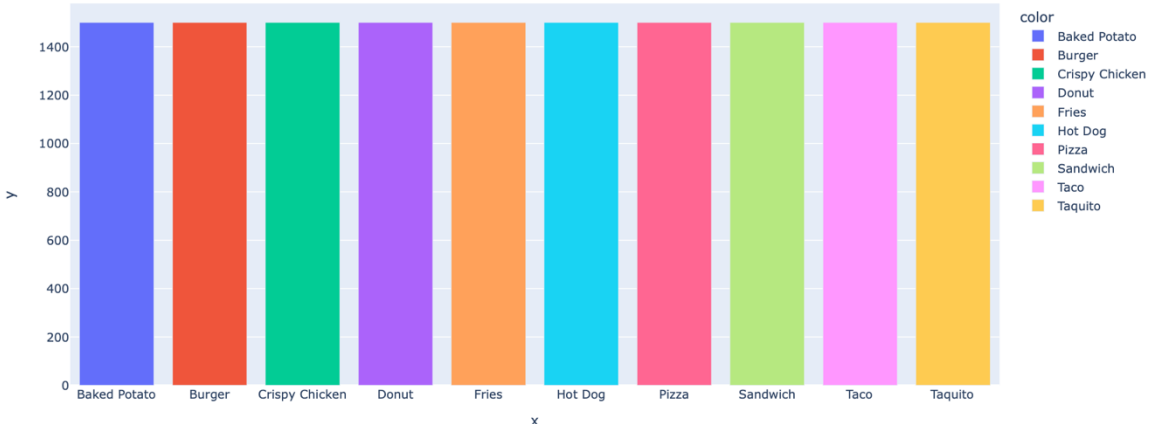
- burgers
- donuts
- hot dogs
- pizza
- sandwiches
- baked potatoes
- crispy chicken
- fries
- tacos
- taquitos

In [14]:
```
# Collecte the class names for training set.
class_names = sorted(os.listdir(train_dir))
n_classes = len(class_names)

# Print
print("No. Classes : {}".format(n_classes))
print("Classes     : {}".format(class_names))
```

```
No. Classes : 10
Classes     : ['Baked Potato', 'Burger', 'Crispy Chicken', 'Donut', 'Fries', 'Hot Dog', 'Pizza', 'Sandwich', 'Taco', 'Taquito']
```

In [15]:
```
import plotly.express as px

# Calculate the class distribution
class_dis = [len(os.listdir(train_dir + name)) for name in class_names]

# Virtualize class distribution
fig = px.pie(names=class_names, values=class_dis, hole=0.3)
fig.update_layout({"title":{'text':"Training Class Distribution", 'x':0.48}})
fig.show()

fig = px.bar(x=class_names, y=class_dis, color=class_names)
fig.show()
```





Import libraries

In [16]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import cv2
import numpy as np
import glob
from pathlib import Path
from tqdm import tqdm

from operator import itemgetter

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, Conv2D, UpSampling2D, MaxPooling2D
from tensorflow.keras.models import Model
```

```
2023-02-01 04:31:16.499538: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Pre-processing data

In [17]:
```
def load_data(path):
    data = []

    path = Path(path)
    path_content = [x for x in path.iterdir() if x.is_dir()]

    for i, directory in enumerate(path_content):
        print(str(i+1) + "/" + str(len(path_content)))

        for file in directory.iterdir():
            if file.is_file():
                data.append(read_img(str(file)))

    return np.array(data)
```

In [18]:
```
data_train = load_data(train_dir)
data_train.shape
```

```
1/10
2/10
3/10
4/10
5/10
6/10
7/10
8/10
9/10
10/10
```

Out[18]:
```
(15000, 64, 64, 3)
```

In [19]:
```
data_test = load_data(test_dir)
data_test.shape
```

```
1/10
2/10
3/10
4/10
5/10
6/10
7/10
8/10
9/10
10/10
```

Out[19]:
```
(1500, 64, 64, 3)
```

Normalisation

In [20]:
```
X_train = data_train.astype('float32') / 255.
X_test = data_test.astype('float32') / 255.
```

Model - autoencoder

In [22]:
```
input_img = Input(shape=IMG_SIZE)

# Encode
encoded = Conv2D(256, (3, 3), activation='relu', padding='same')(input_img)
encoded = MaxPooling2D(pool_size=(2, 2))(encoded)

encoded = Conv2D(256, (3, 3), activation='relu', padding='same')(encoded)
encoded = Conv2D(128, (3, 3), activation='relu', padding='same')(encoded)
encoded = MaxPooling2D(pool_size=(2, 2))(encoded)

encoded = Conv2D(128, (3, 3), activation='relu', padding='same')(encoded)
encoded = Conv2D(64, (3, 3), activation='relu', padding='same')(encoded)
encoded = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
encoded = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
encoded = MaxPooling2D(pool_size=(2, 2))(encoded)

# Decode
decoded = Conv2D(16, (3, 3), name='d1' , activation='relu', padding='same')(encoded)
decoded = Conv2D(32, (3, 3), name='d2' , activation='relu', padding='same')(decoded)
decoded = Conv2D(64, (3, 3), name='d3' , activation='relu', padding='same')(decoded)
decoded = Conv2D(128, (3, 3), name='d4' , activation='relu', padding='same')(decoded)
decoded = UpSampling2D(size=(2,2), name='d5')(decoded)

decoded = Conv2D(128, (3, 3), name='d6' , activation='relu', padding='same')(decoded)
decoded = Conv2D(256, (3, 3), name='d7' , activation='relu', padding='same')(decoded)
decoded = UpSampling2D(size=(2,2), name='d8')(decoded)

decoded = Conv2D(256, (3, 3), name='d9' , activation='relu', padding='same')(decoded)
decoded = UpSampling2D(size=(2,2),name='d10')(decoded)

decoded = Conv2D(3, (3, 3), name='d11' , activation='sigmoid', padding='same')(decoded)

autoencoder = Model(input_img, decoded)
autoencoder.summary()
```

```
2023-02-01 04:32:42.836369: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Libr
ary (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 64, 64, 3)]       0

 conv2d (Conv2D)             (None, 64, 64, 256)       7168

 max_pooling2d (MaxPooling2D  (None, 32, 32, 256)      0
 )

 conv2d_1 (Conv2D)           (None, 32, 32, 256)       590080

 conv2d_2 (Conv2D)           (None, 32, 32, 128)       295040

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       147584

 conv2d_4 (Conv2D)           (None, 16, 16, 64)        73792

 conv2d_5 (Conv2D)           (None, 16, 16, 32)        18464

 conv2d_6 (Conv2D)           (None, 16, 16, 16)        4624

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 16)         0
 2D)

 d1 (Conv2D)                 (None, 8, 8, 16)          2320

 d2 (Conv2D)                 (None, 8, 8, 32)          4640

 d3 (Conv2D)                 (None, 8, 8, 64)          18496

 d4 (Conv2D)                 (None, 8, 8, 128)         73856

 d5 (UpSampling2D)           (None, 16, 16, 128)       0

 d6 (Conv2D)                 (None, 16, 16, 128)       147584

 d7 (Conv2D)                 (None, 16, 16, 256)       295168

 d8 (UpSampling2D)           (None, 32, 32, 256)       0

 d9 (Conv2D)                 (None, 32, 32, 256)       590080

 d10 (UpSampling2D)          (None, 64, 64, 256)       0

 d11 (Conv2D)                (None, 64, 64, 3)         6915

=================================================================
Total params: 2,275,811
Trainable params: 2,275,811
Non-trainable params: 0
_________________________________________________________________
```

In [23]:
```
encoder = Model(input_img, encoded)
encoder_output_shape = encoder.output_shape[1:]
encoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 64, 64, 3)]       0

 conv2d (Conv2D)             (None, 64, 64, 256)       7168

 max_pooling2d (MaxPooling2D  (None, 32, 32, 256)      0
 )

 conv2d_1 (Conv2D)           (None, 32, 32, 256)       590080

 conv2d_2 (Conv2D)           (None, 32, 32, 128)       295040

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 16, 16, 128)       147584

 conv2d_4 (Conv2D)           (None, 16, 16, 64)        73792

 conv2d_5 (Conv2D)           (None, 16, 16, 32)        18464

 conv2d_6 (Conv2D)           (None, 16, 16, 16)        4624

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 16)         0
 2D)

=================================================================
Total params: 1,136,752
Trainable params: 1,136,752
Non-trainable params: 0
_________________________________________________________________
```

In [24]:
```
encoded_input = Input(shape=encoder_output_shape)
decoder_layer = autoencoder.get_layer('d1')(encoded_input)
decoder_layer = autoencoder.get_layer('d2')(decoder_layer)
decoder_layer = autoencoder.get_layer('d3')(decoder_layer)
decoder_layer = autoencoder.get_layer('d4')(decoder_layer)
decoder_layer = autoencoder.get_layer('d5')(decoder_layer)
decoder_layer = autoencoder.get_layer('d6')(decoder_layer)
decoder_layer = autoencoder.get_layer('d7')(decoder_layer)
decoder_layer = autoencoder.get_layer('d8')(decoder_layer)
decoder_layer = autoencoder.get_layer('d9')(decoder_layer)
decoder_layer = autoencoder.get_layer('d10')(decoder_layer)
decoder_layer = autoencoder.get_layer('d11')(decoder_layer)
decoder = Model(encoded_input, decoder_layer)
decoder.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 8, 8, 16)]        0

 d1 (Conv2D)                 (None, 8, 8, 16)          2320

 d2 (Conv2D)                 (None, 8, 8, 32)          4640

 d3 (Conv2D)                 (None, 8, 8, 64)          18496

 d4 (Conv2D)                 (None, 8, 8, 128)         73856

 d5 (UpSampling2D)           (None, 16, 16, 128)       0

 d6 (Conv2D)                 (None, 16, 16, 128)       147584

 d7 (Conv2D)                 (None, 16, 16, 256)       295168

 d8 (UpSampling2D)           (None, 32, 32, 256)       0

 d9 (Conv2D)                 (None, 32, 32, 256)       590080

 d10 (UpSampling2D)          (None, 64, 64, 256)       0

 d11 (Conv2D)                (None, 64, 64, 3)         6915

=================================================================
Total params: 1,139,059
Trainable params: 1,139,059
Non-trainable params: 0
_________________________________________________________________
```

The input image is used as the input to the autoencoder and the output is the decoded image. The autoencoder model is defined using the Model class from Keras. The encoder component of the autoencoder is described as a distinct model with the encoded feature representation as its output and the same input. The encoded feature representation serves as the input and the decoded image serves as the output for the decoder component, which is defined as a distinct model. By linking the previously specified layers of the autoencoder in reverse order, the decoder is produced. This model uses shape images with inputs of (None, 64, 64, 3), where 64 and 64 are the images' height and width, respectively, and 3 is the number of color channels (RGB). Convolutional layers (Conv2D), max-pooling layers (MaxPooling2D), and up-sampling layers are among the network's 11 total layers (UpSampling2D).

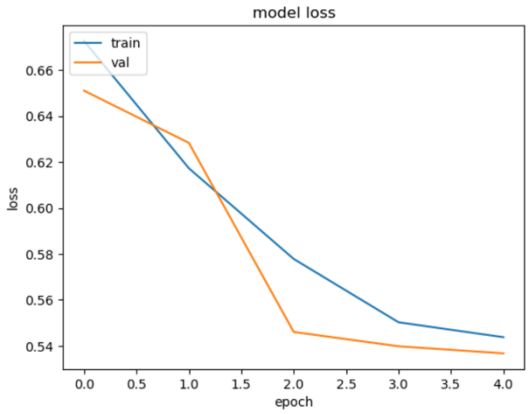
In [25]:
```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = []
```

In [26]:
```
history.append(
    autoencoder.fit(X_train, X_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(X_test, X_test)))
encoded_imgs = encoder.predict(X_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

```
Epoch 1/5
59/59 [==============================] - 4452s 75s/step - loss: 0.6726 - accuracy: 0.7987 - val_loss: 0.6510 - val_accuracy: 0.7124
Epoch 2/5
59/59 [==============================] - 3159s 54s/step - loss: 0.6173 - accuracy: 0.7876 - val_loss: 0.6283 - val_accuracy: 0.8064
Epoch 3/5
59/59 [==============================] - 3095s 52s/step - loss: 0.5778 - accuracy: 0.7840 - val_loss: 0.5461 - val_accuracy: 0.7583
Epoch 4/5
59/59 [==============================] - 4838s 83s/step - loss: 0.5503 - accuracy: 0.7633 - val_loss: 0.5398 - val_accuracy: 0.7644
Epoch 5/5
59/59 [==============================] - 2936s 50s/step - loss: 0.5438 - accuracy: 0.7635 - val_loss: 0.5368 - val_accuracy: 0.7863
47/47 [==============================] - 49s 1s/step
47/47 [==============================] - 45s 955ms/step
```

In [68]:
```
def plot_history(hitories):
    data = {
        "loss" : [],
        "val_loss" : []
    }

    for h in hitories:

        [data["loss"].append(x) for x in h.history['loss']]

        [data["val_loss"].append(x) for x in h.history['val_loss']]

    plt.plot(data["loss"])
    plt.plot(data["val_loss"])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val','y=0.5'], loc='upper left')
    plt.show()
```

In [27]:
```
plot_history(history)
```



The accuracy rises and the loss drops during training, showing that the autoencoder is getting more accurate. After the last epoch, the values obtained for the final loss and accuracy are 0.5368 and 0.7863, respectively (5th epoch).

In [28]:
```
autoencoder.save('autoencoder.h5')
```

Transfer Learning

Pre-processing data

The pixel values between 0 and 1 are normalised using the rescale function. For training and validation, the images are downsized to (256, 256) and divided into groups of size 64 and 32. The range of rotations to apply to the photos is specified by the rotation range argument, in this case 10 degrees and the dataset will be horizontally flipped, while still preserving their vertical alignment.

In [34]:
```
from keras.preprocessing.image import ImageDataGenerator as IDG

# Initialize image data generator
train_gen = IDG(rescale=1/255, rotation_range=10, horizontal_flip=True, vertical_flip=False)
test_gen = IDG(rescale=1/255, rotation_range=10, horizontal_flip=True, vertical_flip=False)
valid_gen = IDG(rescale=1/255)

# Load the datasets
train_ds = train_gen.flow_from_directory(train_dir, shuffle=True, batch_size=64, target_size=(256,256), class_mode='binary')
test_ds = test_gen.flow_from_directory(test_dir, shuffle=True, batch_size=64, target_size=(256,256), class_mode='binary')
valid_ds = test_gen.flow_from_directory(valid_dir, shuffle=True, batch_size=32, target_size=(256,256), class_mode='binary')
```

```
Found 15000 images belonging to 10 classes.
Found 1500 images belonging to 10 classes.
Found 3500 images belonging to 10 classes.
```

Import libraries

In [35]:
```
#!pip install tensorflow_hub
# Common
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from IPython.display import clear_output as cls

# Data Loading
from keras.preprocessing.image import ImageDataGenerator as IDG

# Data Visualization
import plotly.express as px
import matplotlib.pyplot as plt

# Model
from keras.models import Sequential
from tensorflow_hub import KerasLayer as KL
from keras.layers import Dense, InputLayer, Dropout

# Optimizer
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers.schedules import PiecewiseConstantDecay as PwCD

# Callbacks
from keras.callbacks import EarlyStopping as ES, ModelCheckpoint as MC
```

Model

The pre-trained BiT model, an input layer, a dropout layer, and a dense layer with a softmax activation function and zero initializer make up the model architecture.

In [36]:
```
# Model URL
url = "https://tfhub.dev/google/bit/m-r50x1/1"

# Load Model
bit = KL(url)

# Model Name
model_name = "Fast-Food-Classification-BiT"

# Model Architecture
model = Sequential([
    InputLayer(input_shape=(256, 256, 3)),
    bit,
    Dropout(0.2),
    Dense(n_classes, activation='softmax', kernel_initializer='zeros')
], name=model_name)

# Model Summary
model.summary()
```

```
Model: "Fast-Food-Classification-BiT"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 2048)              23500352

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 10)                20490

=================================================================
Total params: 23,520,842
Trainable params: 20,490
Non-trainable params: 23,500,352
_________________________________________________________________
```

The PiecewiseConstantDecay (PwCD) function was used to define a learning rate scheduler with a learning rate of 0.005. After 200, 300, and 400 epochs, the learning rate would drop by a factor of 10.

In [37]:
```
# Learning Rate Scheduler
lr = 5e-3

lr_scheduler = PwCD(boundaries=[200,300,400],values=[lr*0.1, lr*0.01, lr*0.001, lr*0.0001])

opt = SGD(learning_rate=lr_scheduler, momentum=0.9)

# Compile
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=opt,
    metrics=['accuracy']
)
```

In [38]:
```
# Callbacks
cbs = [ES(patience=5, restore_best_weights=True), MC(model_name+".h5", save_best_only=True)]
```

In [39]:
```
# Training
history = model.fit(train_ds, validation_data=valid_ds, epochs=5, callbacks=cbs)
```

```
Epoch 1/5
 98/235 [===========>..................] - ETA: 39:00 - loss: 0.5490 - accuracy: 0.8393
/opt/anaconda3/lib/python3.9/site-packages/PIL/Image.py:959: UserWarning:

Palette images with Transparency expressed in bytes should be converted to RGBA images

235/235 [==============================] - 6034s 26s/step - loss: 0.4287 - accuracy: 0.8748 - val_loss: 0.3981 - val_accuracy: 0.8846
Epoch 2/5
235/235 [==============================] - 5385s 23s/step - loss: 0.2824 - accuracy: 0.9165 - val_loss: 0.4063 - val_accuracy: 0.8851
Epoch 3/5
235/235 [==============================] - 5109s 22s/step - loss: 0.2851 - accuracy: 0.9155 - val_loss: 0.4085 - val_accuracy: 0.8817
Epoch 4/5
235/235 [==============================] - 5103s 22s/step - loss: 0.2835 - accuracy: 0.9173 - val_loss: 0.4061 - val_accuracy: 0.8831
Epoch 5/5
235/235 [==============================] - 4920s 21s/step - loss: 0.2819 - accuracy: 0.9173 - val_loss: 0.4008 - val_accuracy: 0.8820
```

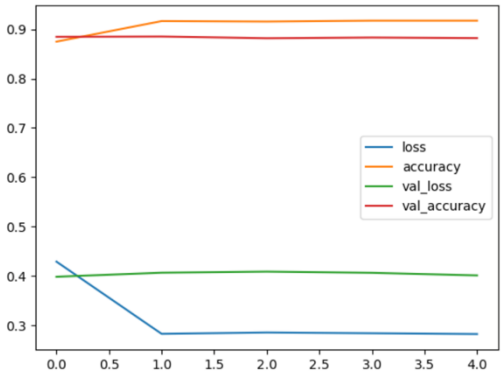
In [40]:
```
lc = pd.DataFrame(history.history)
lc
```

Out[40]:

	loss	accuracy	val_loss	val_accuracy
0	0.428736	0.874800	0.398092	0.884571
1	0.282358	0.916467	0.406278	0.885143
2	0.285095	0.915533	0.408502	0.881714
3	0.283545	0.917333	0.406052	0.883143
4	0.281918	0.917333	0.400788	0.882000

In [43]:
```
lc.plot()
```

Out[43]:
```
<AxesSubplot:>
```



After five training iterations, the model's accuracy was between 0.9165 and 0.9173, and its validation accuracy ranged between 0.8817 and 0.8851, indicating good performance. The model's loss was gradually decreasing, reaching a low of 0.2819 in the last epoch.

Testing Model

Employ a higher accuracy model to a prediction of the class and confidence of an image.

In [48]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Read the image
image = plt.imread('/Users/sallypang/Library/CloudStorage/OneDrive-JamesCookUniversity/LAST BUT NOT LAST YET /Fast Food Classification V2/Train/Baked Pot

# Convert image to a tensor
image = tf.convert_to_tensor(image)

# Preprocess the image
image = tf.image.resize(image, (224, 224))
image /= 255.0

# Expand the dimensions of the image to match the model input shape
image = tf.expand_dims(image, axis=0)

# Predict the class
prediction = model.predict(image)
prediction = np.squeeze(prediction)
pred_index = np.argmax(prediction)
pred_class = class_names[pred_index]
pred_confidence = prediction[pred_index]

# Print the results
print("Class:", pred_class)
print("Confidence:", pred_confidence)
plt.imshow(image[0])
plt.axis("off")
plt.show()
```

```
1/1 [==============================] - 3s 3s/step
Class: Baked Potato
Confidence: 0.9985417
```

In [49]:
```
def show_images(data, class_names, model=None, GRID=[5,6], SIZE=(25,25)):

    # Plot Configurations
    n_rows, n_cols = GRID
    n_images = n_cols * n_rows
    plt.figure(figsize=SIZE)

    # Iterate Through the data
    i = 1
    for images, labels in iter(data):

        # Select data Randomly
        id = np.random.randint(len(images))
        image, label = tf.expand_dims(images[id], axis=0), class_names[int(labels[id])]

        # Show Image
        ax = plt.subplot(n_rows, n_cols, i)
        plt.imshow(image[0])

        # make Prediction
        if model is not None:
            prediction = model.predict(image)[0]
            score = np.round(max(prediction), 2)
            pred = class_names[np.argmax(prediction)]
            title = f"True : {label}\nPred : {pred}"
        else:
            title = label

        # Add calorie information to the title
        if title.startswith("True : Baked Potato"):
            title += f"\nCalories : ≥ 156"
        elif title.startswith("True : Sandwich"):
            title += f"\nCalories : ≥ 252"
        elif title.startswith("True : Pizza"):
            title += f"\nCalories : ≥ 847"

        # Bold the title and increase the font size if the true class is one of the top 3 highest fast food
        if title.startswith("True : Baked Potato") or title.startswith("True : Pizza") or title.startswith("True : Sandwich"):
            plt.title(title, fontweight='bold', fontsize=14, color='red')
        else:
            plt.title(title)
        plt.axis('off')

        # Break Loop
        i+=1
        if i>n_images:
            break

    # Show Final Plot
    plt.show()

show_images(data=valid_ds, class_names=class_names, model=model)
```

```
1/1 [==============================] - 1s 793ms/step
1/1 [==============================] - 1s 569ms/step
1/1 [==============================] - 1s 589ms/step
1/1 [==============================] - 1s 514ms/step
1/1 [==============================] - 1s 587ms/step
1/1 [==============================] - 1s 518ms/step
1/1 [==============================] - 1s 507ms/step
1/1 [==============================] - 1s 572ms/step
1/1 [==============================] - 1s 607ms/step
1/1 [==============================] - 1s 627ms/step
1/1 [==============================] - 1s 551ms/step
1/1 [==============================] - 1s 581ms/step
1/1 [==============================] - 1s 547ms/step
1/1 [==============================] - 1s 636ms/step
1/1 [==============================] - 1s 715ms/step
1/1 [==============================] - 1s 634ms/step
1/1 [==============================] - 1s 688ms/step
1/1 [==============================] - 0s 496ms/step
1/1 [==============================] - 1s 552ms/step
1/1 [==============================] - 0s 491ms/step
1/1 [==============================] - 1s 552ms/step
1/1 [==============================] - 1s 516ms/step
1/1 [==============================] - 1s 560ms/step
1/1 [==============================] - 1s 533ms/step
1/1 [==============================] - 1s 531ms/step
1/1 [==============================] - 1s 581ms/step
1/1 [==============================] - 0s 463ms/step
1/1 [==============================] - 1s 607ms/step
1/1 [==============================] - 0s 469ms/step
1/1 [==============================] - 1s 518ms/step
```

Discussion

This research implemented transfer learning to increase model accuracy from 76.35% to 91.73%. Due to the size of the dataset and the computer's restricted memory, the model was only trained on a finite number of 5 epochs. Despite its limitations, transfer learning has advantages like shorter training times, better performance, and a lack of big data requirements. Furthermore, the study emphasizes the top 3 most calorie fast food items and forecasts image class and confidence (baked potato, sandwich, and pizza).

In [ ]: